Poster: CrossEcore: An Extendible Framework to Use Ecore and OCL across Platforms

Simon Schwichtenberg
Paderborn University
Paderborn, Germany
simon.schwichtenberg@upb.de

Ivan Jovanovikj
Paderborn University
Paderborn, Germany
ivan.jovanovikj@upb.de

Christian Gerth
Hochschule Osnabrück
Osnabrück, Germany
c.gerth@hs-osnabrueck.de

Gregor Engels
Paderborn University
Paderborn, Germany
engels@upb.de

ABSTRACT

Today, model-driven approaches are a cornerstone in modern software development. The Eclipse Modeling Framework (EMF) is highly adopted in practice and generates Java code from platform-independent models with embedded Object Constraint Language (OCL) expressions. However, applications that target multiple platforms like Android, iOS, Windows, web browsers usually need to be implemented in different programming languages. Feature-complete Ecore and OCL runtime APIs are not available for all these platforms, such that their functionality has to be re-implemented. In this paper, we present CrossEcore: A multi-platform enabled modeling framework that generates C#, Swift, TypeScript, and JavaScript code from Ecore models with embedded OCL. An OCL compiler translates OCL expressions into expressions of the target language. The Ecore and OCL API can be consistently used across platforms, which facilitates application portability. CrossEcore is also extendible and can be easily adopted for new programming languages.

ACM Reference Format:
Simon Schwichtenberg, Ivan Jovanovikj, Christian Gerth, and Gregor Engels. 2018. Poster: CrossEcore: An Extendible Framework to Use Ecore and OCL across Platforms. In *ICSE '18 Companion: 40th International Conference on Software Engineering Companion, May 27-June 3, 2018, Gothenburg,Sweden.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3183440.3194976

1 INTRODUCTION

Today, model-driven software development approaches have proven its worth in practice as they increase the productivity of developers. For example, EMF [13] is highly adopted in industry. EMF bases on Ecore class models. EMF's Ecore API provides functionality that is often needed in model-driven software, e.g., notifications on model changes, reflection, persistence, maintenance of referential integrity, validation etc. OCL [1] can be used to describe constraints on model elements. In EMF, OCL can be embedded into Ecore models to specify derived attributes, operation bodies, operation pre- and postconditions, and class invariants. EMF's code generator transforms the platform-independent Ecore models with embedded OCL expressions into platform-specific Java classes.

However, software applications often target multiple platforms, while each platform uses different technologies and programming languages. For example, native Android apps are usually written in Java/Kotlin, iOS apps in Objective-C/Swift, Windows apps in C#, and web apps in JavaScript. The original Ecore and OCL APIs are written in Java. In fact, there are code generators that generate C# [7], C++ [5, 9], JavaScript [6], or Python [4] code from Ecore models. However, none of these code generators supports OCL. OCL compilers are only available for Java [2, 8, 12, 15] and C# [14]. In addition, Akehurst et al. [3] present a mapping of OCL to C#. Thus, a universal approach to support Ecore and OCL on multiple platforms is missing.

2 CROSSECORE

CrossEcore[1] is a multi-platform modeling framework that generates C#, TypeScript, JavaScript, Swift code from Ecore models with embedded OCL. It includes an Ecore and OCL API that can be used consistently across platforms. An OCL compiler translates OCL expressions into executable expressions of the target programming language, which are compiled ahead-of-time. CrossEcore's code generator increases the productivity of software engineers, because redundant implementations can be avoided. The consistent API increases the traceability across implementations on different platforms, which facilitates portability.

CrossEcore's components and how its artifacts needs to be adjusted to support new target languages is explained in the following (c.f. Figure 1). To support a new language, the following four steps have to be done only once:

(1) Create Code Templates: The code Generator has nine code templates per target platform written in Xtend, which have to be adjusted for the syntax of the new language. A primitive type mapping needs to be defined between platform-independent Ecore/OCL data types and platform-specific types. Eight of the code templates are easy to adjust. Only one code template requires more effort, because it generates some program logic that is necessary for referential integrity, notifications, etc.

[1] http://www.crossecore.org/

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3194976

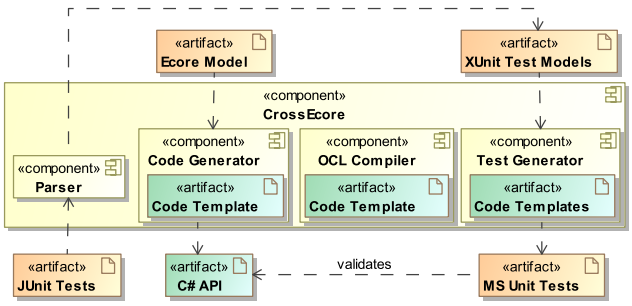

Figure 1: CrossEcore Components

(2) Create OCL Templates: The code generator calls the OCL compiler for every embedded OCL expression. CrossEcore's OCL compiler is based on the MDT/OCL [15] implementation. The MDT/OCL parser has handler methods for different kinds of OCL expressions which have to be implemented once.

(3) Implement Base Functions: Most of the classes of the Ecore API can entirely be generated. Some of the Ecore API classes need to be completed manually: For example, the API functions to maintain referential integrity, trigger notifications, etc. have to be inserted in the base class `BasicEObjectImpl` and collection classes (`OrderedSet` etc.). Also the OCL API has to be implemented manually.

(4) Validate Migration: The migration of the Ecore and OCL API to the target language is validated through software testing [10]: Existing JUnit test cases from public EMF and MDT/OCL code repositories are parsed into platform-independent XUnit test models [11]. The code templates of the test generator has to be adjusted before test cases can be generated for the target programming language and test environment, e.g., C# and MS Unit Test Framework. Listing 1 and Listing 2 show an original JUnit test and the migrated MS Unit Test.

```
@Test
public void testCollectionAppend() {
ocl.assertQueryResults("Sequence{'a','b','c'}", "Sequence
    ↪ {'a','b'}->append('c')");
}
```

Listing 1: Original JUnit Test

```
[TestMethod]
public void testCollectionAppend(){
var expected = new Sequence<string>{"a","b","c"};
var actual = new Sequence<string>{"a","b"}.append("c");
ocl.assertQueryResults(expected, actual);
}
```

Listing 2: Generated MS Unit Test

3 EVALUATION AND DISCUSSION

CrossEcore has its origin in the context of an industrial project where product managers with limited programming skills shall be able to phrase modeling constraints that can be executed on different target platforms. CrossEcore was successfully deployed in several prototypes on different platforms.

In terms of the industrial project, we conduct the migration validation step of our migration method for C#. Out of 4000 test cases for OCL, contained in 13 different JUnit test suites, 92% could be automatically migrated to the Microsoft Unit Test Framework. The remaining tests are regression tests that have an irregular structure, which complicates an automation.

As presented in this paper, CrossEcore has the capability to be easily extended to other programming languages. However, the migration is currently semi-automatic and includes manual steps. Therefore, it remains an open question to what degree the migration can be further automatized.

4 CONCLUSION

CrossEcore is a multi-platform modeling framework that generates C#, TypeScript, JavaScript, Swift code from Ecore models with embedded OCL. The generated code provides functions that are frequently used in model-driven software development like reflection, persistence, notifications, etc. An OCL compiler automatically translates invariants, derived attributes, and operation bodies into expressions of the target languages, such that they do not have to be implemented manually. All functions are provided through an API that can be used consistently across different target platforms. This consistent API increases traceability and facilitates portability across different platforms. Due to its extendible design, CrossEcore can be easily adopted for further new programming languages. To show a wider generality of our approach, we will adopt CrossEcore for programming languages that are not object-oriented, e.g., Rust.

REFERENCES

[1] 2014. Object Constraint Language Version 2.4. (2014). http://www.omg.org/spec/OCL/2.4/

[2] David Akehurst and Octavian Patrascoiu. 2004. OCL 2.0 - Implementing the Standard for Multiple Metamodels. *Electronic Notes in Theoretical Computer Science* 102 (2004), 21 – 41. https://doi.org/10.1016/j.entcs.2003.09.002

[3] David H Akehurst, W Gareth J Howells, Markus Scheidgen, and Klaus D McDonald-Maier. 2008. C# 3.0 makes OCL redundant. *Electronic Communications of the EASST* 9 (2008).

[4] Vincent Aranega and Mike Pagel. 2018. PyEcore. (2018). Retrieved March 19, 2018 from https://github.com/pyecore

[5] Andrés Senac González, Diego Sevilla Ruiz, and Gregorio Martinez Perez. 2010. EMF4CPP: a C++ Ecore Implementation. *DSDM 2010 - Desarrollo de Software Dirigido por Modelos, Jornadas de Ingenieria del Software y Bases de Datos* (2010).

[6] Guillaume Hillairet. 2018. Ecore.js. (2018). Retrieved March 19, 2018 from https://github.com/emfjson/ecore.js

[7] Georg Hinkel. 2016. *NMF: A Modeling Framework for the .NET Platform.* Technical Report. Karlsruhe Institute of Technology. http://nbn-resolving.org/urn:nbn:de:swb:90-537082

[8] Heinrich Hußmann, Birgit Demuth, and Frank Finger. 2002. Modular architecture for a toolset supporting OCL. *Science of Computer Programming* 44, 1 (2002), 51–69. https://doi.org/10.1016/S0167-6423(02)00032-1

[9] S. Jäger, R. Maschotta, T. Jungebloud, A. Wichmann, and A. Zimmermann. 2016. An EMF-like UML generator for C++. In *2016 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*. 309–316.

[10] Ivan Jovanovikj and Stefan Sauer. 2017. Towards a Framework for Constructing Context-Specific Migration Methods for Test Cases. *Softwaretechnik-Trends, Proceedings of the 19th Workshop Software-Reengineering & Evolution (WSRE) & 8th Workshop Design for Future (DFF)* 37, 2 (2017), 50–51.

[11] Gerard Meszaros. 2007. *xUnit test patterns: Refactoring test code.* Pearson Education.

[12] Octavian Patrascoiu. 2005. *Model Driven Language Engineering.* Ph.D. Dissertation. University of Kent.

[13] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. 2008. *EMF: eclipse modeling framework.* Pearson Education.

[14] Tamás Vajk, Gergely Mezei, and Tihamer Levendovszky. 2013. Incremental semantic analysis for OCL compilers. *ISSE* 9, 3 (2013), 147–162. https://doi.org/10.1007/s11334-013-0218-7

[15] Edward D. Willink. 2010. Re-engineering Eclipse MDT/OCL for Xtext. *ECEASST* 36 (2010). http://journal.ub.tu-berlin.de/index.php/eceasst/article/view/444