Agile MDE

Program Translation using MDE

Kevin Lano

Legacy software causes major social and economic problems:

- ***US Government, 2020***: national COVID measures delayed by legacy software problems
- ***Commonwealth Bank of Australia***: modernising applications from COBOL to Java cost $750 million & 5 years
- ***JP Morgan***: 2+ years to modernise Athena from Python 2 to Python 3

Agile MDE

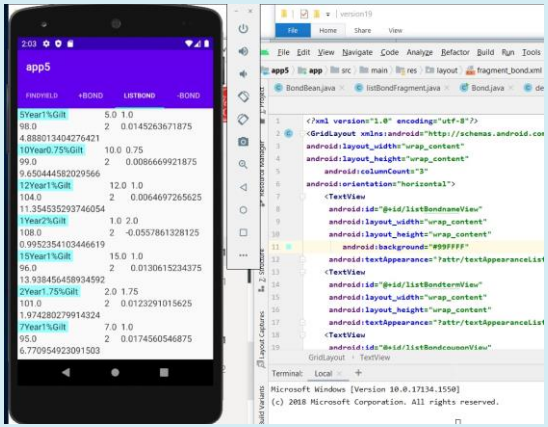
Program Translation using MDE

Our solution is to abstract legacy code to a specification, then forward-engineer:

Specifications

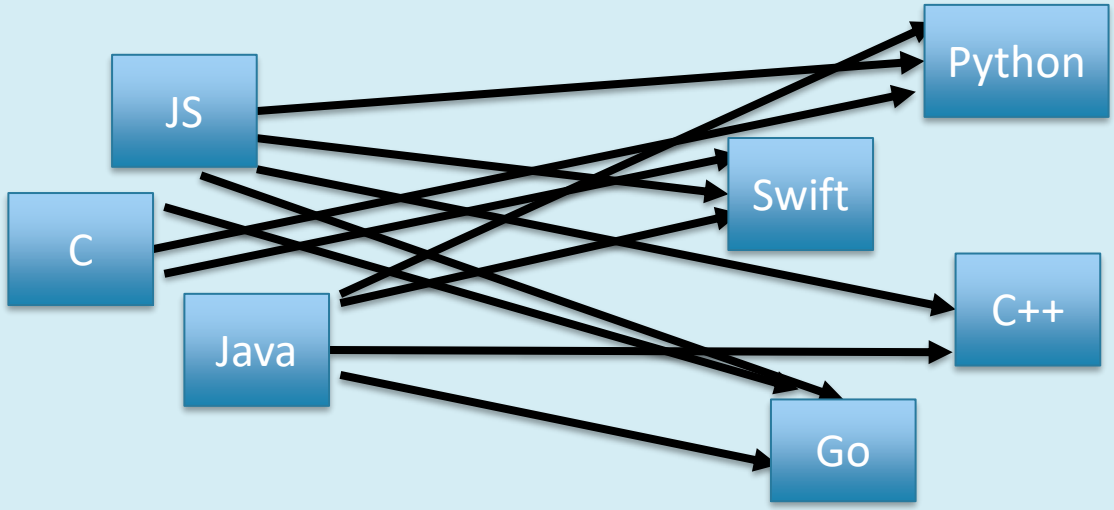
Modernised software for new platforms

Old software; old platforms

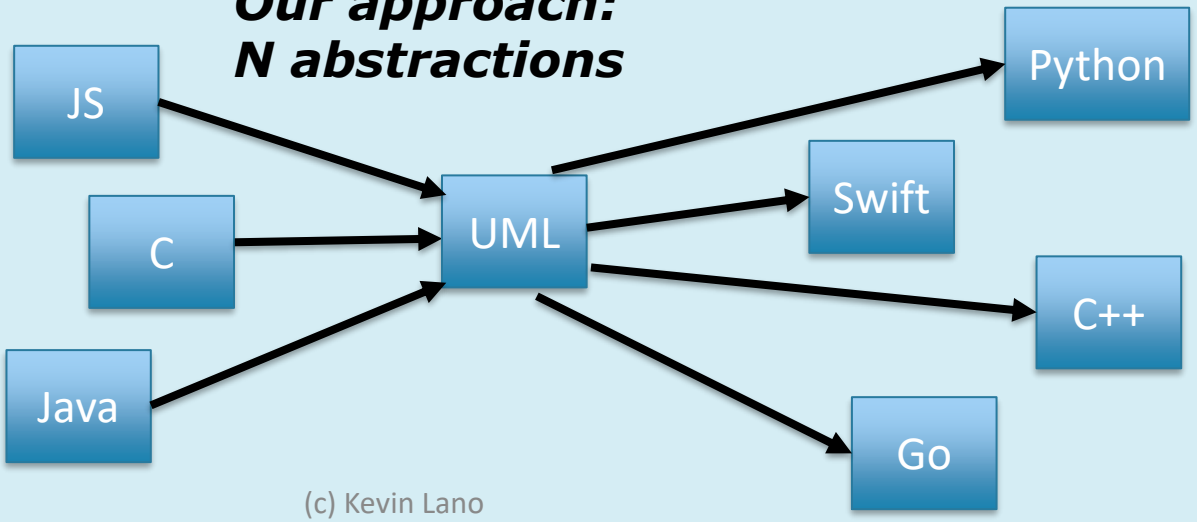


Agile MDE

Existing rule-based approaches need N*M individual translations



Our approach: N abstractions



Innovation:

- translation via intermediate language (UML) provides *specification* of system
- reduces number of translations needed
- increases flexibility.

MDE Process for Language Translation

Translation steps:

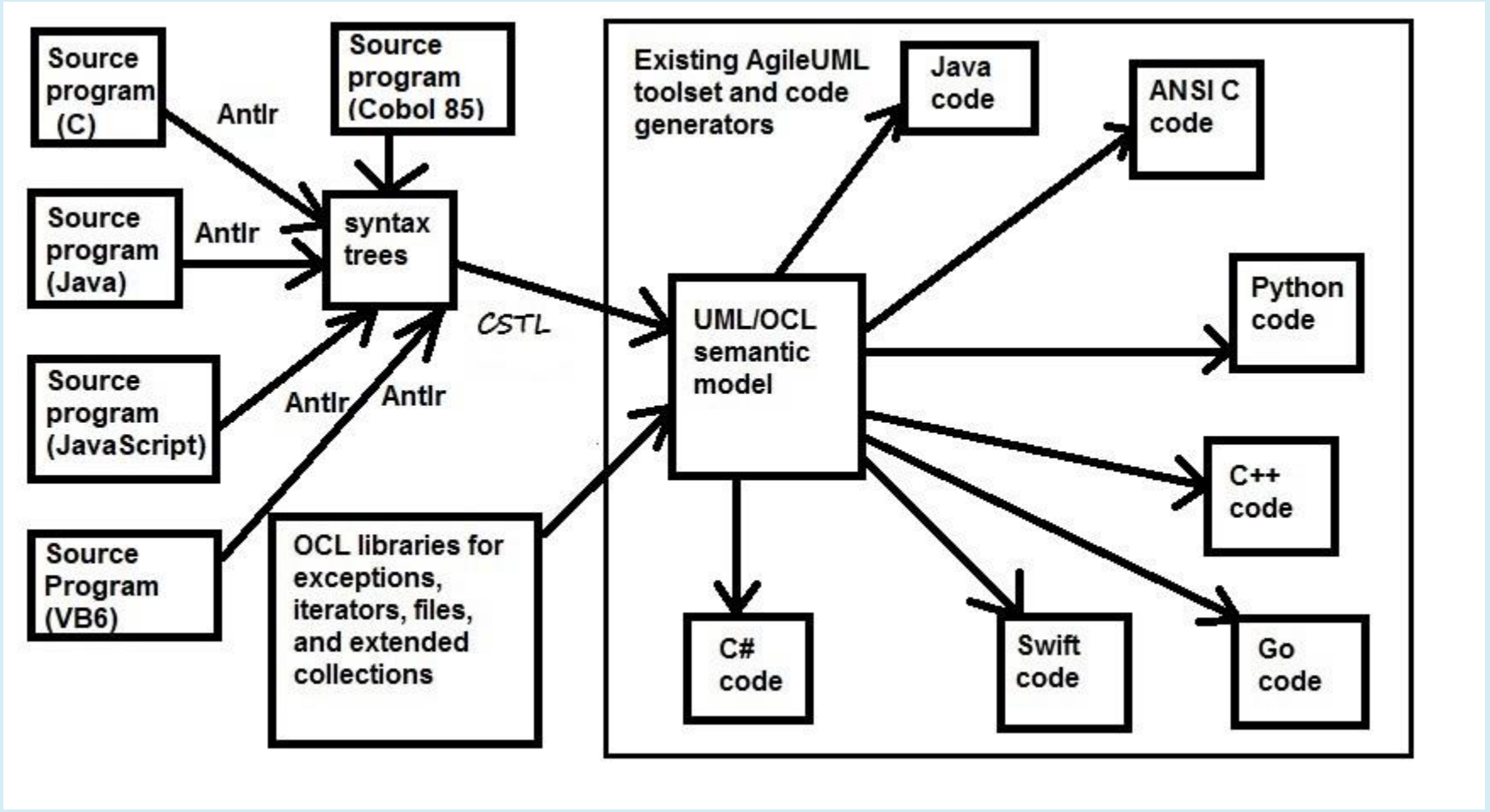
- *Abstraction of source program into a UML/OCL specification – captures precise semantics*
- *Forward engineering of specification into target language.*

Many MDE tools already provide forward engineering from UML/OCL into multiple target languages.

Thus main new work is reverse-engineering step + program representation in UML and OCL.

Agile MDE

Program Translation using MDE



Program Translation using MDE

MDE Process for Language Translation

- *Antlr* parser for source language *L*1 used to produce parse trees, input to abstraction transformation for *L*1, written using *CSTL*
- Output is text UML/OCL specification, consisting of class specifications with data features + operations.
- OCL libraries added to represent files, processes, reflection, dates, exceptions and iterators, not present in standard OCL.
- Forward engineering using *AgileUML* to map abstracted specification to target language *L*2.

Other MDE tools could be used for forward engineering.

Program abstraction

Examples of CSTL abstraction rules for Java expressions & statements:

expression::
_1 ? _2 : _3 |-->if _1 then _2 else _3 endif
(_1) _2 |-->_2->oclAsType(_1)
_1 instanceof _2 |-->_1->oclIsKindOf(_2)

statement::
if (_1) { _2 } |--> if _1 then _2 else skip
for (_1 _2 : _3) _4 |--> for _2 : _3 do _4

OCL extension types and libraries

- Function, map and reference types *Function*(*S,T*), *Map*(*T*) and *Ref* (*T*).
- *OclDate*: dates and times.
- *MathLib*: Byte processing and bitwise operators.
- *OclType* extended to provide full reflection capabilities.
- *OclException*.
- *OclIterator* represents iterators for collections, *generator* functions and database result sets.
- *OclFile* represents files and streams.
- *OclProcess* models threads and OS processes.
- *OclDatasource* models SQL databases, TCP sockets and HTTP connections.

Program Translation using MDE

Issues and problems

- Programming languages use side-effecting expressions, whilst OCL does not: separately represent pre/post side-effects and query semantics
- Language libraries can be very extensive (eg., Java): only the core datatypes and features can be represented
- Unstructured control flow: goto, switch, etc.: replace by structured statements.
- No translation of GUI code

Program Translation using MDE

Evaluation and comparison

- Abstraction mappings to UML/OCL from Java 6/7, ANSI C, JavaScript, Cobol85 and VB6 have been developed.

CSTL Script	Size (LOC)	Effort
Java2UML	2955	5pm
C2UML	2160	2.5pm
JS2UML	793	2pm
VB2UML	1230	2pm
Cobol85	3284	2pm
Averages	*2084*	*2.7pm*
About 35LOC/pd		

Program Translation using MDE

Evaluation and comparison

Grammar coverage:

Language	Abstraction rules/Grammar cases	Coverage
Java	345/386	89%
C	138/153	90%
JavaScript	258/324	80%
VB6	320/412	77%

For C, 158 of 179 library operations are abstracted (88%).
For JavaScript, 39 of 53 library components are abstracted (74%).
For VB6, 197 of 229 built-in elements (86%).

Evaluation and comparison

Functional correctness evaluated by tests on source & target versions of translation cases. Compute percentage of test results which agree.

- This measure of accuracy is *computational accuracy*.
- For Java 6/7 mappings to Python, Swift, C#, C++ and Go, used 100 evaluation cases.
- For C mappings to Swift, C# and Go, used 70 cases.
- For JavaScript mapping to Python, used 100 cases.
- For mapping VB6 to JavaScript, used 100 cases.
- For mapping Cobol85 to Java, used 60 cases.

Program Translation using MDE

Evaluation and comparison: accuracy

Target language	From Java 6/7	From C	From JavaScript	From VB6	From Cobol85
Python	93%		95%	83%	
Swift	96%	84%			
C#	96%	90%			
Go	90%	91%			
Java 8	98%				77%
C++	93%				
C	86%				

Program Translation using MDE

Evaluation and comparison: accuracy

Compared to other translation tools:

- *Java2python* only achieves an accuracy of 38.3% on similar dataset of Java to Python examples
- *Transcoder* achieves 68.7% accuracy for Java to Python
- Our results are significantly better than these scores.

Program Translation using MDE

Related work:

Main approaches for program translation:

- Heuristic manually-created rules
- Machine learning approaches inducing implicit rules from examples in source and target languages

For these, translator construction must be repeated for each pair of languages under consideration.

Second approach requires large datasets of program examples + learned knowledge is only implicit.

Program Translation using MDE

Related work:

Other MDE approaches include:

- MoDisco and REMICS: structural representation of legacy applications for migration and modernisation.
- Gra2Mol and GReTL text-to-model languages to perform program abstraction to models: ATL-style languages – more complex than *CSTL*.
- Reverse-engineering of Java bytecode to statemachines (Sen et al, 2016).
- Formal semantic approaches also used for reverse-engineering of database schemas.

Our contributions

- Detailed semantic model of source programs, to enable semantically-correct translations + reduce retesting cost
- Precise and explicit abstraction rules which can be edited and configured
- Set of OCL library components which can be used for OCL specification of new applications
- Systematic procedure for building program abstractors based on language grammars.

Conclusions and future work

- Shown that program translation approach using MDE can be effective for practical program translation tasks.
- Approach enables users to customise translation rules used, provides rigorous semantically-based abstraction & forward-engineering process.
- Future work includes using symbolic machine learning to automate construction of abstraction mappings.