Model Transformations for Fun & Profit

Richard Paige
(with Dimitris Kolovos)
@richpaige, @dskolovos, @epsilonnews
Department of Computer Science, University of York, UK
Structure of Lectures

1. Foundations of Model Driven Engineering
   - Motivation; definitions.
   - What is it; why should we care; principles?

2. Overview of Model Transformations
   - Characteristics and features
   - Model-to-model and model-to-text transformations.

3. Advanced Model Transformations
   - Update-in-place
   - Migration transformations
   - Merging transformations

4. Applications.
Structure of Lectures

1. Foundations of Model Driven Engineering
   - Motivation; definitions.
   - What is it; why should we care; principles?

2. Overview of Model Transformations
   - Characteristics and features
   - Model-to-model and model-to-text transformations.

3. Advanced Model Transformations
   - Update-in-place
   - Migration transformations
   - Merging transformations

4. Applications.
What is MDE?
It's Not Really...

```
AddAppointment

BookingSystem
BookingSystem'

customer? : Customer
timeSlot? : TimeSlot

timeSlot? ∈ workingTimeSlots \ dom appointments
appointments' = appointments ∪ \{timeSlot? ↦ custom
workingTimeSlots' = workingTimeSlots
```

```
dtmc

module Queue

s : [0..3] init 0;

[ ] s=0 -> (s'=1);
[ ] s=1 -> 0.01;(s'=1) + 0.01;(s'=2) + 0.98;(s'=3);
[ ] s=2 -> (s'=0);
[ ] s=3 -> (s'=3);
```

```java
public class Account {
    private /*@ spec_public @*/ int balance;
    //@ invariant balance >= 0;

    /*@ requires amount > 0;
    @ assignable balance;
    @ ensures balance == \old(balance) - amount
    @
    && \result == balance;
    @ signals (OverdraftEx) balance == \old(balance);
    @*/
    public int withdraw(int amount) throws OverdraftEx {
        if (amount <= balance) {return (balance -= amount);} 
        else {throw new OverdraftEx(...);} 
    }
    ...
}
```
Conceptually, MDE’s ultimate goal is the same as that of formal methods.

- i.e., build more reliable, robust, acceptable, available, etc, systems.
- Reliance on abstraction and separation of concerns.
- Reliance on tools to construct, manipulate and validate descriptions.

MDE mechanisms for implementation differ from those of formal methods.
It's not...

class MySession : public SNMPSession {
  public:
    virtual void processTrap(const SNMPObject & obj) {
      std::cout << obj.getDisplayInformation() << "\n";
    }
};

MySession session;
SNMPRemoteAgent ragent(host, community, 0, &session);
SNMPTrap trap(oid, &ragent);
trap.enable();
Though...

- Code generation (model-to-text) is a legitimate, if obvious, scenario of use.
- There are many other legitimate, valuable and important scenarios
  - (and we shall see some later).
Model Driven Engineering

- A principled approach to system engineering
- Promotes *models* to first-class artefacts
- More than documentation
- Models are structured and living entities that are amenable to automated processing
  - Validation, transformation, comparison, merging, refactoring, code generation etc.
  - They are structured in very specific ways.
Dependability?

- What has this got to do with dependability?
  - Automation of repetitive, error-prone engineering tasks.
  - Constructing accurate and acceptable descriptions of phenomena of interest.
  - Mechanisms for relating engineering artefacts (largely automatically)
    - cf traceability
    - to feed in to audit, certification, validation...
A (simplified) MDE scenario
A Transport Project

- A transport organisation has a legacy railway interlocking model in an old version of xUML.
- They want to do the following:
  - Load the legacy xUML model (do not underestimate this)
  - Migrate it to UML 2.x
  - Do:
    - Validate the model;
    - Generate a simulation model;
    - Generate an HTML report;
    - Apply some refactorings
  - Until false
MDE is all about managing and manipulating models.
Foundations of Model Management
Models ≠ UML diagrams

- UML is just one modelling language
  - Though a very popular one.

- Most domains have different abstractions/semantics
  - Domain Specific Modelling Languages (DSMLs)
  - ... but also general-purpose languages as well.

- Models ≠ Pictures
  - Models can be graphical or textual
  - ... And are often both
Model of application communications
Model of an Astute submarine
Model of a Missile Controller
Model of a Castle
Model of an Actor
Model of Friend Relationships
Model of Mongo

MAP OF THE PLANET MONGO

MONGO IS APPROXIMATELY ONE HALF THE DIAMETER OF EARTH BUT HAS A GRAVITATIONAL DENSITY THAT IS ONLY SLIGHTLY LESS. IT IS A RELATIVELY YOUNG WORLD WITH TOWERING MOUNTAINS NOT YET WORN SMOOTH BY TIME AND MANY AREAS OF VOLCANIC ACTIVITY. ITS VEGETATION IS STILL LIMITED TO ISOLATED AREAS OF BOTANICAL GIANTS. BIOLOGICALLY, IT IS STILL IN THE ERA OF REPTILIAN GIANTS. MAN EVOLVED FAST INTO DIVERSE RACES, MANY OF WHICH POSSESS AMAZINGLY ADVANCED TECHNOLOGY WHILE OTHERS STILL LIVE IN PRIMITIVE AND UNEXPLORED REGIONS.
Maps

Canada circ. 1955

Apocalypse map

The System

Beer Map

Largest « Thing » Map

Example idea by Jean Bezivin
Every map has a legend

Same visual notation, different context, different meaning (Thick red dotted lines for bicycle lanes)

The legend is called a *metamodel*.
Synthesis

- Support for different languages is critical in MDE.
  - General purpose, domain specific, obsolete...
  - Sometimes we need to support all of the above in one project.

- We use structure (metamodels) to enable model management.

- We’ll see more on metamodels and metamodelling shortly.
Synthesis - Implementation?

- How are models and metamodels typically implemented?
  - EMF/Ecore is the most popular (graphs).
  - MDR/MOF (graphs).
  - XML [schema-ful and schema-less] (trees)
  - Proprietary formats (graphs, DBs).

- You shouldn’t have to care about this when managing your models.
  - Except perhaps when things get REALLY big.
Synthesis - Semantics

- What’s the semantics of these models?
- Plausible answers:
  - Use mathematics as we normally do.
  - Via transformations to something we understand.
  - Wrong question; what do you want to do with your models?
- Semantics is a modelling problem.
- Semantics in MDE is purpose-driven.
- Do not mistake UML’s weak semantics (for verification) as a general illness!
Metamodelling
What is a metamodel?

- A description of a language.
  - Models are *instances* of this language.
  - (Sentences, in EBNF terminology.)
- Most typically, a metamodel is a description of the *abstract syntax* of a language.
  - Concepts, structures and constraints.
  - Not usually the tokens, lexemes, symbols, blobs...
What is a metamodel?

- It’s also a model.
- This is the so-called unification property of MDE: everything’s a model.
  - So, in principle, models, metamodels and lots of other things can be implemented and managed using one set of tools.
  - In practice this is mostly true.
  - However, it’s convenient and pragmatic to take short-cuts if you want to build big systems.
Metamodelling

- Metamodelling is at the heart of MDE.
  - Without a metamodel, we cannot automatically manipulate models.
- So how do we construct metamodels?
- What do they look like?
- What’s a typical process?
- Example.
A Model
When to Metamodel?

- If we’re constructing a one-off model, there’s no point in constructing a metamodel.

- For example:
  - if we’re only interested in describing Elvis, there’s little point investing any effort in constructing a *metamodel* that allows us to describe Elvis.

- However, if we’re interested in different musicians, a metamodel could be useful.
A Metamodel for Musicians

- A metamodel for musicians will let us describe Elvis.
  - And other musicians; all are instances.
- It will let us express the concepts and constraints of musicians that are important.
  - For some purpose(s).
  - E.g., animation, simulation, comparison, reasoning.
- In essence it will define a language for talking about (and manipulating) musicians.
Instances
Abstract Syntax

- Metamodelling starts with thinking about:
  - What are the key concepts of musicians?
- It may help to think about what you want to do with the models that you will ultimately produce – i.e., their *purpose*.
- Key concepts:
  - Name, style, behaviour, do they play instruments, do they sing?
- Purposes: animation, simulation, comparison
Finding Concepts?

- Michael mentioned noun-verb analysis yesterday.
  - Use it as a first-pass approximation.
  - It can find useless or redundant concepts.
  - ... and can miss some of the important ones.

- But it’s fine to start with.
  - Better to write something down and shout about it than to shout about nothing...
Constraints

- Ask what *constraints* you want to capture of your descriptions.
  - In other words, *are there restrictions on the concepts?*
  - This will help clarify constraints on your models and operations on your models.
  - E.g., Elvis can’t simultaneously move his left leg in and his left leg out.
Metamodel (First-Pass)

context Leg inv: self.in <> self.out
A metamodel (adding ops)
What are the next steps?

- Concrete syntax:
  - Usually derived from abstract syntax.
  - Graphical or textual?
  - Depends on what you want to do with models.
What are the next steps?

- Operations applied to models.
  - Simulation
  - Transformation
  - Producing text
  - Comparison
- Model management
Model Management Tasks

(What do you want to do to your models?)
What tasks?

- Transforming models
- Generating text from models
- Refactoring models
- Merging models
- Validating models
- Comparing models
- Migrating models as a metamodel changes
- Querying and modifying models
- Chains
How are model management tasks supported?
Model Management

- Manipulate your models directly.
  - Invariably, XML/XMI manipulation.
  - Write XSLT, Java...
  - Build an API...

- Use standard, general purpose MDE languages.
  - E.g., Object Constraint Language.

- Use task-specific languages, e.g., ATL, QVTo, Tefkat, KerMeta, ...
Languages for MDE

- Inconsistent syntaxes
  - Different expression dialects
  - Different ways to perform model navigation/modification
  - End up writing the same code in many languages

- Poor integration and interoperation
  - E.g. validation -> M2M -> M2T

- Recurrence of bugs / missing features
Example: Checking for a UML stereotype

<<persistent>>
User
name: String
address: Address
OCL (Model validation)

```ocm
package uml

class Element

def Operations:
  let hasStereotype(s : String) : Boolean
    = getAppliedStereotypes() ->
      exists(st | st.name = s)

endpackage
```
helper context UML2!Element def :
    hasStereotype(s : String): Boolean
        self.getAppliedStereotypes()
        ->exists(st | st.name = s);
MOFScript (Code Generation)

```csharp
uml.Element::

    hasStereotype(s : String): Boolean {
        result = self.getAppliedStereotypes()
        ->exists(st | st.name = s);
    }
```
Languages for MDE
There is hope...
epsilon: a family of integrated programming languages for managing models
• Extensible.
• Interdependent.
• Task-specific.
• Technology agnostic.
• Scalable.
- Mature project
  - Under Eclipse.org since 2006
- Well-documented
  - Examples, articles, screencasts, book
- Substantial user base
  - 1000s of posts in the forum
Used in
Architecture of Epsilon

Core Language (EOL)

- EMF
- XML
- MDR
- M2T (EGL)
- Validation (EVL)
- M2M (ETL)
- Refactoring (EWL)
- Merging (EML)
- Comparison (ECL)
- Evolution (Flock)
Features

- Languages for a range of model management tasks
- Languages have consistent syntaxes
- Can manage models from different metamodels / modelling technologies
- Can call methods of Java objects
- Strong integration with EMF and GMF
- Eclipse-based development tools
  - Editors, Launching facilities
- Download
  - [www.eclipse.org/epsilon](http://www.eclipse.org/epsilon)
- Documentation
  - [www.eclipse.org/epsilon/doc](http://www.eclipse.org/epsilon/doc)
- Screencasts
  - [www.eclipse.org/epsilon/cinema](http://www.eclipse.org/epsilon/cinema)
- Twitter: [@epsilonnews](http://@epsilonnews)
Synthesis – Pub Talk

- Model versus specification?
  - No real difference.
- Model versus program?
  - Again, no real difference.
  - They are both abstractions of something, created for a purpose.
- MDE is all about enabling the construction of languages that are fit for specific purposes.
Schematically

Load xUML model → Migrate model to UML 2.x → Validate model

Generate HTML report ← Generate simulation model
Next time

- Model transformation.
  - Classification.
  - Examples.
  - Model transformation with Epsilon.
- Advanced model transformation.
- Applications.
# MDE versus Formal Methods?

<table>
<thead>
<tr>
<th></th>
<th>MDE</th>
<th>Formal Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Language syntax</strong></td>
<td>• Emphasis on abstract syntax</td>
<td>• Emphasis on concrete syntax</td>
</tr>
<tr>
<td><strong>Language implementation</strong></td>
<td>• Uses standardised infrastructure.</td>
<td>• Some commonly used data structures/algorithms.</td>
</tr>
<tr>
<td><strong>Language semantics</strong></td>
<td>• Defined for specific purposes, potentially governed by constraints.</td>
<td>• Defined for analysis, soundness, completeness, ...</td>
</tr>
<tr>
<td></td>
<td>• Mathematics or transformation</td>
<td>• Mathematics</td>
</tr>
<tr>
<td><strong>Tools</strong></td>
<td>• For modelling &amp; model management.</td>
<td>• For modelling &amp; analysis.</td>
</tr>
<tr>
<td></td>
<td>• The first priority.</td>
<td>• Historically came second; not today.</td>
</tr>
</tbody>
</table>
Model Transformations for Fun & Profit

Richard Paige
(with Dimitris Kolovos)
@richpaige, @dskolovos, @epsilonnews
Department of Computer Science, University of York, UK
Structure of Lectures

1. Foundations of Model Driven Engineering
   - Motivation; definitions
   - What is it; why should we care; principles?

2. Overview of Model Transformations
   - Characteristics and features
   - Model-to-model and model-to-text transformations.

3. Advanced Model Transformations
   - Update-in-place
   - Migration transformations
   - Merging transformations

4. Applications.
Recap

- **Models:**
  - Abstractions of *something*, created for a purpose.
  - Amenable to automated processing by tools.
  - Created using metamodelling technology.

- **Metamodels:**
  - “Models of models”.
  - Define languages used for modelling.
  - Focus on abstract syntax.
  - Basis for automated processing of models.

- **Model management:**
  - Tasks we want to carry out on models.
Model Transformation

- “The heart and soul of MDE.”

- Basic scenario:
  - a model (in one language) is transformed into a model in a (possibly) different language.
  - Multiple inputs, multiple outputs also possible.

- Obvious MDE workflow:
  - Construct an abstract model.
  - Successively transform it until a sufficiently detailed model is produced.
  - Generate code from the detailed model.
Applications of MT

- **Elaboration**: generating detailed models or code from less detailed models.
- **Synchronisation**: ensuring consistency between models at the same or different levels of abstraction.
- **View creation**: producing query-based views.
- **Model evolution** (including refactoring)
- **Abstraction**: generating less detailed models from more detailed ones.
Transformations are not ...

- ... necessarily semantics preserving.
  - They can be, but there are useful transformations that are “lossy”.

- ... necessarily refinements.
  - They can be (especially update-in-place transformations) but many useful ones aren’t.

- ... necessarily specified in a way that allows interesting properties to be checked of them.
  - Sometimes they must be transformed!
Standards and Tools

- OMG Query/Views/Transformations (QVT).
  - Operational, Relational and Core languages.
  - Reference implementations still developing.
  - Lots of complexity and ambiguity!

- Industrial-strength and mature MT tools:
  - VIATRA2, Tefkat, GReAT, GReTL, ATL, Epsilon, KerMeta, ...
  - We will illustrate transformations using Epsilon (selfishly).
Basic Concept

Source metamodel \(\text{Conforms to}\) Source model \(\text{Reads}\) Transformation engine \(\text{Executes}\) Target model \(\text{Writes}\) Target metamodel

Transformation definition \(\text{Refers to}\)
Example Transformation

(a) Simple UML metamodel

(b) Simple RDBMS metamodel
Example Transformation

1. Every UML package should be mapped to a RDBMS schema with the same name.
2. Every (persistent) class should be mapped to a table with the same name.
   1. Table should have a primary key column with the type NUMBER and the name being that of the class with _tid appended.
3. UML attributes should be mapped to appropriate columns (related via foreign key definitions).
Part of an ETL Example

```java
rule Class2Table
  transform c : UML!Class
to t : DB!Table
extends ModelElement2NamedElement {
  guard : c.hasStereotype('table') and
    Sys.user.confirm('Transform ' +
    c.name + '?')

  t.database ::= c.namespace;
  var idCol := new DB!Column;
  idCol.name := 'id';
  t.columns.add(idCol);
}
What is this?

- It’s a model transformation written using the Epsilon Transformation Language (ETL).
  - A task-specific language.
  - Part of the Epsilon platform.
- It implements one of the rules mentioned earlier.
- It demonstrates a specific type of model transformation.
  - Mapping.
  - There are many others!
Classification of MT

- Czarnecki and Helsen wrote a seminal paper on classifying model transformations.
  - Rigorous approach, using feature diagrams.
- Covered all important top-level features (circa 2006) of transformations.
  - Didn’t elaborate some parts in detail.
- Consider parts of the classification.
Top-level Features
Some Key Features

- **Specification:**
  - Some approaches mandate a particular specification mechanism (e.g., pre/post).
  - Some specifications may be executable (e.g., functional ones); others may be full relations and are not executable (e.g., original QVTr).

- **Transformation Rules:**
  - The “smallest unit” of transformation.
  - E.g., rewrite rules, function/procedure implementing a transformation step.
Some Key Features

- **Rule application control:**
  - How are transformation rules scheduled and executed?
  - How are salient *locations* of the model determined, against which rules are executed?

- **Directionality:**
  - Can rules be executed in one direction only, or in multiple directions (e.g., bidirectional)?
route_reservation
  any r where
  r ∈ R \ resrt
  rtbl^−1[{r}] ∩ resbl = ∅
then
  resrt := resrt ∪ {r}
  rsrtbl := rsrtbl ∪ rtbl ▷ {r}
  resbl := resbl ∪ rtbl^−1[{r}]
end
Rule Scheduling

- Can engineers indicate order in which rules are executed?
- Are there phases?
- Are there iteration mechanisms?
Types of Transformations

- Two broad categories:
  - Model-to-Model
    - Results are instances of metamodels.
  - Model-to-Text
    - Results are strings.

- Many specialisations of each.
Model-to-Model Approaches

- **Direct manipulation:**
  - Have some API (e.g., JMI) that lets you directly manipulate model representations.
  - You have to implement all your features from scratch (often).

- **Structure-driven:**
  - Two-phase approaches.
  - 1) Create hierarchical structure of target model; 2) populate its attributes and references.
Model-to-Model

- **Operational approaches:**
  - Use an operational language with metamodelling support to transform models.
  - E.g., KerMeta, QVTo, EOL.

- **Relational approaches:**
  - Use a declarative language with metamodelling support to transform models.
  - E.g., QVTr, Tefkat, AMW, ...
Model-to-Model

- **Graph transformation:**
  - Treat models as attribute graphs.
  - Graph transformation rules (e.g., TGGs)
  - E.g., VIATRA, AGG, AToM3, GReAT, GReTL.

- **Hybrid:**
  - Combining declarative and operational approaches.
  - E.g., ATL, ETL (Epsilon in general)
Model-to-Text

- **Visitor-based approaches:**
  - Provide a visitor mechanism to traverse internal model structure and produce text to a stream.
  - Need to mess with internal structures.

- **Template-based approaches:**
  - Most popular.
  - Have static and dynamic regions; static regions copied, dynamic regions generate output.
  - E.g., MOFscript, JET, oAW, EGL, ...
Illustrations

- We’ll illustrate some of these model transformation concepts by examples of languages from the Epsilon platform.
  - Motivation for Epsilon.
  - Conceptual architecture.
  - Core concepts (and their relationship to the classification).
  - Model-to-Model
  - Model-to-Text.
epsilon: a family of integrated programming languages for managing models
Architecture of epsilon

Core Language (EOL)

- EMF
- XML
- Comparison (ECL)
- Merging (EML)
- Evolution (Flock)
- Refactoring (EWL)
- Validation (EVL)
- M2M (ETL)
- M2T (EGL)
EPSILON

Core Languages : The Epsilon Object Language
EOL: Overview

- Dynamically and strongly typed
- Object-oriented
- Modular
- Primitives, collections and model elements are objects

Play with EOL in your browser:
www.eclipse.org/epsilon/live
Design for EOL

- What do model management operations have in common?
  - i.e., transformation, query, merge, generate code, validate, etc.

- After some reflection:
  1. They all require the ability to **navigate** models (e.g., to go from class to class, node to node, line of code to line of code).
  2. Many require the ability to **modify** models.
Navigation

- Navigation is something that the OMG’s Object Constraint Language (OCL) is really good at.
  - It’s abstract and declarative.
  - It allows very concise navigation expressions to be written.
  - e.g., `self.processor_rack.process`
- But it’s restricted to the OMG’s standards.
  - How would it apply to Z specs, or a blob-and-line variant?
- It also doesn’t allow model modification.
  - ... and there are other substantial difficulties...
Borrows navigation expressions (and some basic operations) from OCL.
Borrows conceptually from Javascript.
Adds assignment statements, sequencing, and multiple model access.
... plus some other stuff.
EOL: Types

- Four primitive types:
  - String, Integer, Real and Boolean
- Four collection types:
  - Bag, Sequence, Set and OrderedSet

- Universal type: Any
  - isDefined(), isUndefined()
  - isOfType(type:Type), isKindOf(type:Type)
EOL: Operations

- Collection types provide (side-effect free) higher-order operations
  - select(), reject()
  - collect(), exists()

- Ability to define custom operations
  - Can have context, i.e. called using dot notation
  - Optional return type
EOL: Other

- User input: `System.getUser()`
  - `.inform()`, `.choose()`, `.chooseMany()`, `.confirm()`, `.prompt()`, `.promptInteger()`, `.promptReal()`

- Platform independent.
Basic Features of EOL

```javascript
var model : UML!Model;
model = new UML!Model;
model.name = 'TestModel';
var i : Integer;
for (i in Sequence{1..3}){
    var class : new UML!Class;
    class.name = 'TestClass' + i;
    class.visibility = UML!VisibilityKind#vk_public;
    class.namespace = model;
}
```

- Define a model variable
- Instantiate the model variable
- Assign a name to the model
- Define an iterator for i = 1 to 3
- Define and instantiate a class variable
- Assign a name to the class
- Set the class visibility to public
- Set the namespace of the class to be the model
Access to Multiple Models

- Many model management tasks, such as transformation or comparison, require simultaneous access to multiple models.
- To support this, EOL employs the `<Model Name>!<Meta-class>` syntax.

```java
for (class in UML!Class.allInstances()){
    if (DBMS!Table.allInstances().exists(table|table.name == class.name)){
        (‘Found matching table for class ‘ + class.name).println();
    }
    else {
        (‘Not found matching table for class ‘ + class.name).println();
    }
}
```
EOL Summary

- Effectively, all model management can be done in EOL.
  - (Or Java... but...)
- Operational model transformations can (and have) been written in EOL.
- But EOL doesn’t possess task-specific constructs for transformations.
  - There are repeated patterns that arise with any transformation.
EPSILON

The Essential Languages: Transformation, Generation
Friends

isFriendOf

acknowledges
Enemies

- Enemies
- isEnemyOf
- tolerates

- Valley of the Dolls
- International Organization for Standardization
Language Definitions

Language FriendMap

Language EnemyMap
EPSILON

Core Languages: The Epsilon Generation Language

```java
public class Circle {
    private Square square;
    public Circle(Square thisSquare) {
        this.squares = thisSquare;
    }
}
```
EGL: Overview

- Model-to-text transformation language
- Two types of sections
  - Static: content appears verbatim in generated text
  - Dynamic: executable code (EOL)
- Templates
  - Generate files
- Protected regions
- Beautification
EGL – A Template Language

- EGL is a template language (e.g. PHP)

```
[% for (i in Sequence{1..5}) %]
i is [%= i %]
[% } %]
```

- Dynamic sections: contents executed
- Static sections: contents appear verbatim in output
EGL – Preprocessor for EOL

- EGL is minimally derived from EOL

```
[\% for (i in Sequence\{1..5\}) \{ \%]
i is [\%= i \%]
[\% } \%]
```

becomes:

```
for (i in Sequence\{1..5\}) {
    out.print('i is ');
    out.println(i);
}
```
EGL – Feature Summary

- Common M2T language features:
  - Support for defining and utilising protected regions
  - Beautification
  - Traceability

- Novel / uncommon features
  - Co-ordination engine: encourages decoupling
  - Strong integration with other model management languages
EGL – Readability

- Templates should be readable
  - But so should generated text

- Philosophy: make templates readable
  - And run a post-processor on the generated text

- Beautifiers provided for Java and XML
  - Extensible; invoked via Epsilon workflow
  - Similar concept available in Xpand
EGL - Co-ordination

- Encourages decoupling of destination and content
  - No “file” construct in EGL

- Instead, templates are types in the language
  - File-generating template:
    - Can be stored to disk, supports merging
  - Socketed template:
    - Contents written directly to a network socket
Example: Nixon’s Enemy List

[% var rmn = EnemyMap!Enemy.all().
    select( e | e.name = 'Richard Nixon' ); %]

Richard M. Nixon’s Enemy List
---------------------------------------

[% for ( e in rmn.isEnemyOf ) { %]
    I hate [%= e.name %]
[% } %]

will produce

Richard M. Nixon’s Enemy List
---------------------------------------

I hate Dick Dastardly
I hate the Novels of Jacqueline Susann
EPSILON

Core Languages: The Epsilon Transformation Language
ETL: Overview

- Model-to-model transformation language
- Hybrid language (declarative and imperative parts)
- Arbitrary number of source/target models
- Traceability
ETL: Overview

- Rule-based
  - Optional guards
  - Reuse via rule extension
  - Abstract, primary, lazy annotations
  - Can be interactive
- Pre and post blocks
ETL: Overview

- Execution
  - Pre blocks
  - Non-abstract, non-lazy (applicable) rules
  - Post blocks
- `.equivalents()` and `.equivalent()`
  - Resolves source elements to their target counterparts
  - Invokes both lazy and non-lazy rules
  - Shorthand ::=
ETL Example

rule EnemyBecomesFriend
    transform e : EnemyMap!Enemy
to f : FriendMap!Friend {

    guard: UserInput.confirm('Is your enemy ' + e.name + ' now a friend?')

    f.name = e.name;
    f.acknowledges ::= e.tolerates;
}
Some M2M approaches take the view that the transformation should (provably) preserve desirable properties.

- “Correctness” or “Consistency” is a favourite.

We take the perspective that:

- A transformation does the transformation.
- A validation (e.g., OCL, EVL, ...) checks that your model obeys properties.

Separation of concerns.
Semantics?

- What do transformations mean?
  - Good question!
  - A M2M transformation defines a relation between source and target model.
  - In fact, Epsilon generates these relations (traces!) automatically – cf Manfred’s “dynamic traces”.
  - Use this to reason about/validate transformations.

- Started ongoing work on formalisation via UTP.
Open Research Areas?

- Semantics of transformations.
  - Generic patterns/templates and specific ones.
- Engineering processes for transformations.
- Validation of transformations.
- Coverage measures for testing transformations.
- “Learning” transformations from metamodels and examples.
Fun Epsilon Facts!

- Biggest Epsilon programs?
  - 7KLOC for acquisition support
  - 20KLOC for validation of TDL model
  - 4KLOC for interlocking transformation in ETL
  - 3KLOC for bidirectional transformation
  - 1.2KLOC for EuGENia

- Strangest program so far?
  - Twitter client written in EOL.
  - Super Awesome Fighter.
  - (In progress...) Dancing Robot Elvis.
Program vs Model Transform?

- Many of the concepts of program transformation apply to model transformation.
  - Program transf typically applies to tree structures.

- Model transformation:
  - Applies to graphs (in a standardised format)
  - Multi-way transformation.
  - Traceability from sources to targets.
  - Multi-directionality.
Example
The Epsilon Object Language : Animating a flowchart
Flowcharts: Example
EOL: Example

```java
var flowchart : Flowchart := Flowchart.all.first(); // Get flowchart; grab initial node
var state : Node = flowchart.nodes.select(n | n.incoming.size() == 0).first();
state.name.println('-');
while (state.outgoing.size() > 0) {
    if (state.isTypeOf(Decision)) {
        var tran:Transition=System.getUser().chooseMany(state.name, state.outgoing).first();
        if (tran.isUndefined()) { break; }
        tran.name.println('--');
        state = tran.target;
    } else if (state.isTypeOf(Action)) {
        state = state.outgoing.first().target;
    }
    state.name.println('-'); // Print new node name
}
'Simulation complete.'println();
```
EPSILON

Core Languages: The Epsilon Validation Language
Model-to-text classification

- Not refined/detailed in Czarnecki et al’s paper.
- See Rose et al, MiSE 2012 proceedings.
EVL: Overview

- Specify and evaluate constraints on models
- Context: specifies the type over which the invariants will be evaluated
  - Optional guard
- Invariant – constraint vs critique
  - Can be lazy
  - Optional guard
  - Fix
  - Message
- .satisfies(), .satisfiesAll(), .satisfiesOne()
- Pre and post blocks
EVL: Overview

- Execution
  - Pre blocks
  - Each context evaluated
  - User presented with any failure messages and asked to select a fix
  - Post blocks
Epsilon Validation Language

- **Applications include:**
  - Checking that a model obeys essential properties.
  - Critiquing a model.
  - Repairing a model that is ill-formed or that has been updated improperly.
  - Checking that different models are consistent.

- **Some features of EVL are:**
  - Separation between critical and non-critical constraints
  - Context-aware human-friendly messages when constraints fail
  - Dependencies among constraints
  - Ability to repair inconsistencies
context FriendMap!Friend {
    constraint MyFriendIsNotMyEnemy {
        guard: self.name<>"
        check: not EnemyMap!Enemy.all.exists(e|e.name=self.name)
        message: ‘My friend ‘ + self.name + ‘ is also my enemy.’
    }

    fix {
        title: ‘I welcome my former enemy ‘+self.name
        do {
            var formerEnemy: EnemyMap!Enemy;
            formerEnemy = EnemyMap!Enemy.all.selectOne(e|
                e.name=self.name);

            delete formerEnemy;
        }
    }

    fix {
        title: ‘I shun my former friend ‘ +self.name
        do { delete self; }
    }
}
Continued...

@abstract
rule ModelElement2NamedElement
  transform s : UML!ModelElement
to t : DB!NamedElement {

  t.name := s.getDBName();
}

@cached
operation UML!NamedElement getDBName() : String {
  if (self.isTypeOf(UML!Class))
    return 'T_' + self.name;
  else
    return self.name;
}
Features

- Languages for a range of model management tasks
- Languages have consistent syntaxes
- Can manage models from different metamodels / modelling technologies
- Can call methods of Java objects
- Strong integration with EMF and GMF
- Eclipse-based development tools
  - Editors, Launching facilities
before epsilon...
Languages for MDE

- Inconsistent syntaxes
  - Different dialects of OCL
  - Different ways to perform model navigation/modification
  - End up writing the same code in many languages

- Poor integration and interoperation
  - E.g. validation -> M2M -> M2T

- Recurrence of bugs / missing features
- **Domain**: that part of a rule responsible for accessing one of the source models.
  - Includes ways of specifying whether source models are read-only, etc.
- **Syntactic separation**: are domains kept syntactically separate (e.g., rewrite rules)?
Model Transformations for Fun & Profit

Richard Paige
(with Dimitris Kolovos)
@richpaige, @dskolovos, @epsilonnews
Department of Computer Science, University of York, UK
Structure of Lectures

1. Foundations of Model Driven Engineering
   - Motivation; definitions.
   - What is it; why should we care; principles?

2. Overview of Model Transformations
   - Characteristics and features
   - Model-to-model and model-to-text transformations.

3. Advanced Model Transformations
   - Update-in-place
   - Migration transformations
   - Merging transformations

4. Applications.
Recap

- MDE and model transformations.
- Classification of different kinds of transformation:
  - Model-to-model
  - Model-to-text
- Illustrations using Epsilon.
The ‘Debate’

- Manfred and I were debating two options:
  1. Prove your transformation is semantics preserving (e.g., weak bisimilarity, equivalence, refinement...) viz Hulsbusch, Rensink et al.
  2. Run-time checks of equivalence between source/target models, viz Karsai et al.
- Both approaches arguably are needed.
Why?

- Some transformations must be correct (engineered to the highest quality).
  - Largest examples in literature? Varro (SC2PN), Hulsbusch et al (about 5 rules).
- Some can be acceptable and useful without a full correctness proof.
- Some transformations are so complex that a correctness proof is impractical.
  - So we monitor via tracebility.
Update-in-Place
Update-in-Place

- Model-to-model transformations come in a number of flavours:
  - Mappings: from a source to target model expressed in different languages.
    - Usually when languages are similar.
  - Update: perform in-place modifications to a model (source/target languages are identical)
    - -in-the-large: apply to large sets of elements calculated using well-defined rules.
    - -in-the-small: user-driven
Update-in-Place

- In general, these transformations automatically create, update or delete model elements.
  - Information needed is obtained from users.
- Actions taken are generally referred to as wizards (to distinguish them from rules).
Typical Requirements

- Wizards must be able to specify:
  - Actions to apply to model elements
  - Selection of applicable model elements
  - Labelling
  - Some means of connecting wizards with a user interface.
    - Easiest if a relatively standard architecture is used, e.g., MVC.
Epsilon Wizard Language (EWL)

- Wizards have names, a guard, an executable title (more soon), and a set of statements.
- All reused from EOL.
Example Wizard

```plaintext
wizard ExtractInterface {
  guard : self.isKindOf(Class)
  title : 'Extract interface I' + self.name
  do {
    var i : new Interface;
    self.owner.packagedElement.add(i);
    i.name = 'I' + self.name;

    var g : new Generalization;
    self.generalization.add(g);
    g.general = i;
  }
}
```
Example Wizard (2)

```java
for (p : Property in Property.allInstances().select(p|p.type = self)) {
    p.type = i;
}

for (o : Operation in self.ownedOperation.clone()) {
    i.ownedOperation.add(o);
}
```
UI Integration

- Inherently, executing wizards is user-driven.
- Have integrated EWL with the MVC architecture of various modelling tools.
  - ArgoUML, Eclipse UML, general GMF editors.
- This is done via Epsilon’s model connectivity architecture, by producing tool-specific drivers.
UI Integration
Some Applications

- Implementing refactoring patterns [TOOLS’07].
- Supporting a refinement method for a hybrid statecharts language [MBED’11].
- “Faking” bidirectional transformation [Commercial].
  - Define consistency rules (in Epsilon) between source and target languages.
  - Define EWL wizards on source and target models.
  - Whenever models violate consistency rules, run one or more EWL wizards to re-establish consistency.
Semantics Preserving?

- Yes indeed!
- Graph transformations are a particularly good representation for these.
- Can also use run-time verification.
Migrating Models
Metamodels Change Over Time
Changed Metamodel
Model Migration Process

Evolve metamodel

Create migration strategy

Install new version of metamodel

Find conformance problems

Migrate models

Metamodel developer

Modeller (metamodelling user)
A Migration Strategy

• Ports with only incoming Connectors become InputPorts.

• Ports with only outgoing Connectors become OutputPorts.

• Other Ports are split into both an InputPort and an OutputPort.
Original Model

System

Telephone

Exchange

Loudspeaker

First

Third

Second

Fourth

Mic

Speaker
Evolved Model
In Parallel
Approaches

- **Manual**
  - [Sprinkle 2004]
  - [Hussey 2006]
  - [Rose 2010]

- **Operators**
  - [Wachsmuth 2007]
  - [Herrmannsdörfer 2009]

- **Matching**
  - [Cicchetti 2008]
  - [Garcés 2009]

Increased automation
Reduced flexibility
Epsilon Flock

- Transformation language tailored for migration:
  - Model elements that have *not been affected* by metamodel evolution are automatically copied.
  - Model elements that *have been affected* are transformed with migrate rules and retyping rules (or are deleted).

- Extension point for integration with EMF.
- Does not constrain the evolution process.

- ICMT’10, very recent SOSYM paper.
Epsilon Flock

`delete` Port `when`: not (original.isInput() xor original.isOutput())

`retype` Port `to` InputPort `when`: original.isInput()
`retype` Port `to` OutputPort `when`: original.isOutput()

`migrate` Connector {
    migrated.'in' = original.from.equivalent();
    migrated.out = original.'to'.equivalent();
}

`operation` Original!Port isInput() : Boolean {
    return Original!Connector.all.exists(c|c.from == self);
}

`operation` Original!Port isOutput() : Boolean {
    return Original!Connector.all.exists(c|c.'to' == self);
}
Migration Transformations

- These won’t be semantics preserving in general.
  - Constructs can be deleted from a language, or semantics changed completely.
  - A sub-transformation may be semantics preserving (i.e., the ‘copying’ part).

- Migration is a big problem when working with standards.
  - E.g., going from UML 1.x to 2.x
  - Versions of GMF.
Merging Models
Model merging

- **Model merging** is about combining two models of arbitrary languages into a single model that:
  - does not contain redundant information
  - preserves desirable properties of source models.
- Sometimes called model composition, model unification, model integration.
- There is extensive literature on Database Schema merging, an area very closely related to model merging.
Why merge models?

- Popular scenario: model versioning.
- Distributed teams.
- To support problem decomposition.
  - Sometimes it’s just easier to carry out tasks on small models and combine the results.
  - E.g., merge state machines.
- Product line engineering.
  - Merge is used for configuration/instantiation.
- Batch performance analysis.
Phases of Model Merging

- **Compare**
  - Discover the corresponding concepts in the source models

- **Conform**
  - Resolve conflicts and align models to make them compatible for integration

- **Merge**
  - Merge common concepts of the source models and port non-matching concepts

- **Restructure**
  - Restructure the merged model so that it satisfies desired properties
The Epsilon Merging Language (EML) is a language that supports most of these phases.

EML reuses EOL as an infrastructure language.
- Specifically to implement the behaviour of merging rules.

Therefore it can be used to merge different types of models (EMF, MDR, CZT, XML, ...)
Structure of an EML Program

- An EML program consists of merge rules.
- It can also use transform rules (from ETL).
  - Some model elements in a source model don’t need to be merged, they just need to be transformed.
- It also contains a pre and a post block that are executed before and after the merging (respectively) to perform tasks that are not pattern-based.
- It assumes that you have already matched.
So how do you match?

- Many approaches can be used for matching models.
  - It’s an important MDE scenario in and of itself.
- Quick overview of the main conceptual approaches for matching.
Persistent identifiers
Overview

- Every model element has a persistent ID.
  - Compare them.
- No effort from the user
- Fast
- Inflexible
- Only applies to homogeneous models
- Models must share a common parent
Signature-based comparison
Overview

- Calculate a signature for each element
- ... then compare the signatures
- Relatively little effort
  - Define the signature functions
- Fast
  - Is often reduced to string comparison
- Mainly useful for tree-like models
- Not resilient to significant structural changes
Similarity-based comparison
Overview

- Assign weights to features and compare elements based on the aggregated similarity
- Little effort from the user (set weights)
- Sophisticated algorithms (e.g. similarity flooding)
- Not particularly flexible (all vs all)
- Cannot exploit metamodel semantics
- Can compare only homogeneous models
- Can get false positives
Epsilon Comparison Language

... a language tailored for model matching
Examples: Matching heterogeneous models
Match class with table

rule ClassWithTable
  match c : OO!Class
  with t : DB!Table {

    guard : not c.abstract

    compare :
      ("T_" + c.name).toUpperCase() == t.name.name.toUpperCase()

  }

Match guest with room

```java
rule GuestWithRoom
    match g : Agent!Guest
    with r : Hotel!Room {

        compare {
            return g.budget >= r.price
            and g.reqStars >= r.hotel.stars;
        }
    }
```
Match bolt with nut

```plaintext
rule BoltWithNut
    match b : Bolts!Bolt
    with n : Nuts!Nut {

    compare {
        var math : new Native('utils.MathUtils');
        var difference : Real;
        difference = n.perimeter - 2*math.pi*b.diameter;
        return difference >= 0 and difference <= 0.1;
    }
}
```
After matching...

- Model elements are partitioned into ones that match and ones that don’t.
- Matched elements are stored in an internal model called a match-trace.
  - Analogous to the monitors in Klaus’s lectures.
  - Trace-links conform to a simple traceability metamodel.
- How is the match-trace exposed to humans/other operations?
Back to merging...

- Elements that are matching will be merged.
  - The specification of merging is defined in a **Merge Rule**

- Elements not matching (but that have been compared) will be transformed into model elements compatible with the target metamodel.
  - The specification of transformation is defined using ETL rules.

- Any other elements indicate an error or incomplete match rule set.
Example (in EMFatic)

class System {
    val Entity[*]#system entity;
}

class Entity {
    attr String name;
    ref System#entity system;
    attr Boolean inDomain;
}

class Vocabulary {
    val Term[*] term;
}

class Term {
    attr String name;
    val Alias[*] alias;
}

class Alias {
    attr String name;
}
Compare Vocab/Entity Models

rule MatchSystemWithVocabulary
  match s : Source!System
  with v : Vocabulary!Vocabulary {
    compare { return true; }
  }

rule MatchEntityWithTerm
  match s : Source!Entity
  with t : Vocabulary!Term {
    compare {
      return s.name = t.name or
      t.`alias`.exists(a|a.name = s.name);
    }
  }
Merge Models – Merge Rules

**rule** MergeEntityWithTerm

```plaintext
merge s : Source!Entity
with t : Vocabulary!Term
into m : Target!Entity {
  m.name = t.name;
  m.inDomain = true;
}
```

**rule** MergeSystemWithVocabulary

```plaintext
merge s : Source!System
with v : Vocabulary!Vocabulary
into t : Target!System {
  t.entity = s.entity.equivalent();
}
```
Putting the pieces together?

- Also need a transform rule to transform source/target entities that aren’t matched but need to be kept.
- How do we take the results of the match (ECL) and use them in the merge (EML)?
  - Global variables?
  - Magic?
- Workflow and orchestration.
  - General mechanism.
Orchestration and Coordination
Epsilon ANT tasks

- To enable developers to combine MDE with classical tasks, a workflow solution for Epsilon is implemented atop ANT.

- The Epsilon Workflow provides
  - ANT tasks for loading & disposing of models
  - ANT tasks for executing Epsilon programs
  - A common model repository accessible to all the tasks in a workflow
  - Features for importing/exporting variables between different Epsilon programs (e.g., trace information).
  - Existing ANT tasks (e.g., for visualisation, code generation, profiling) can be used.
For our ECL/EML example

```xml
<target name="compare">
  <epsilon.ecl src="Comparison.ecl" exportmatchtrace="eclMatchTrace">
    <model ref="Source"/>
    <model ref="Vocabulary"/>
  </epsilon.ecl>
</target>

<target name="merge">
  <epsilon.eml src="Merging.eml" usematchtrace="eclMatchTrace">
    <model ref="Source"/>
    <model ref="Vocabulary"/>
    <model ref="Target"/>
  </epsilon.eml>
</target>
```
Workflows

- These workflows need not be restricted to MDE/Epsilon tasks.
- Any ANT task can be executed.
- E.g., compile, repository access, debug.
- MDE tasks are typically not executed in a vacuum.
  - Particularly important for working with legacy.
What have we seen?

- Specialised types of model transformation.
- All of these could be implemented using EOL or ETL.
- We have done so.
  - To gather requirements and to convince ourselves that it was a bad idea.
- You get lots of repetitive code which is hidden with these specialist languages.
  - Less error prone to work with specialist languages.
What haven’t we seen?

- Validation of models.
  - Epsilon Validation Language.
  - Also supports inter-model consistency checking, model repair.
- Testing of model management tasks.
  - EUnit.
- Adding new modelling repositories.
- All described in the Epsilon book
  - (bestselling, better than Harry Potter, free).
Epsilon Workflow Example

```xml
<project default="main">
  <target name="main" depends="load,validate,transform">
  </target>
  <target name="load">
    <epsilon.loadModel name="A">...</epsilon.loadModel>
    <epsilon.loadModel name="B">...</epsilon.loadModel>
  </target>
  <target name="validate">
    <epsilon.evl src="AConstraints.evl">
      <model ref="A"/>
    </epsilon.evl>
  </target>
  <target name="transform">
    <epsilon.etl src="A2B.etl">
      <model ref="A"/>
      <model ref="B"/>
    </epsilon.etl>
  </target>
</project>
```
Merge Workflow

1. Execute a match:
   - Use Epsilon Comparison Language (or anything else that produces match-traces).

   ```
   rule Models
   match l : Left!Model with r : Right!Model {
     compare : true
   }
   
   rule Class match l : Left!Class with r : Right!Class {
     compare : l.name = r.name
   }
   ```
Execute Merge Workflow

2. Check the generated match for consistency.
   - Epsilon’s validation language is used for this.

```plaintext
context SimpleOO!Class {
  constraint BothAbstractOrNot {
    guard : self.getMatching().isDefined()
    check : self.getMatching().isAbstract = self.isAbstract
    message : 'Inconsistent value in feature
               "abstract" ' + 'of class ' + self.name
  }
}
```
Execute Merge Workflow

3. Merge models using EML.
   - See previous code.
   - To execute these MDE tasks in sequence, we use the Epsilon workflow.
EML Program

rule MergeModel
merge l : Left!Model with r : Right!Model
into t : Target!Model {
    t.name := l.name + ' and ' + r.name;
    t.contents ::= l.contents + r.contents;
}

rule MergeClass
merge l : Left!Class with r : Right!Class
into t : Target!Class {
    t.name := l.name;
    t.isAbstract := l.isAbstract;
}
EML Program (2)

rule CopyModel
  
  transform s : Source!Model to t : Target!Model {
    t.contents ::= s.contents;
  }

rule CopyClass
  
  transform s : Source!Class to t : Target!Class {
    t.name := s.name;
    t.isAbstract := s.isAbstract;
  }
Model Transformations for Fun & Profit

Richard Paige
(with Dimitris Kolovos)
@richpaige, @dskolovos, @epsilonnews
Department of Computer Science, University of York, UK
Structure of Lectures

1. Foundations of Model Driven Engineering
   - Motivation; definitions.
   - What is it; why should we care; principles?

2. Overview of Model Transformations
   - Characteristics and features
   - Model-to-model and model-to-text transformations.

3. Advanced Model Transformations
   - Update-in-place
   - Migration transformations
   - Merging transformations

4. Applications.
This is you
Recent Applications

1. Eating your own dog food.
   - Practical application.

2. Search-related applications.
   - Acquisition of capability.
   - Super Awesome Fighter (and variations).

3. No time.
   - Sensitivity analysis.
   - Transformations of MARTE to Zot.
   - Transformations of xUML to Promela.
EuGENia: GMF for mortals
Aim: Implement a graphical editor for a DSL
Technologies: Eclipse, EMF, GMF
Our Metamodel (in EMF/XMI)
How EMF Works

Metamodel → Generator Model → Code Generator
How GMF Works

Generator Model

Mapping Model

Graph Model

Tooling Model

Meta Model

Code Generator

Generator Model
How GMF Works

- Meta Model
- Graph Model
- Mapping Model
- Tooling Model

Generator Model → Code Generator
The GMF Tooling Model
How GMF Works

Meta Model

Tooling Model

Mapping Model

Graph Model

Generator Model

Code Generator
The GMF Graph Model
How GMF Works

- Meta Model
- Tooling Model
- Graph Model
- Mapping Model

Generator Model → Code Generator
The GMF Mapping Model
Mapping Model Wizard
How GMF Works

Meta Model

Graph Model

Mapping Model

Tooling Model

Generator Model

Code Generator
The Generator Model
How GMF Works

- Meta Model
- Graph Model
- Tooling Model
- Mapping Model

Generator Model

Code Generator

Diagram showing the process flow from Meta Model to Generator Model to Code Generator.
Powerful
Configurable
Labour intensive
Hard to master
Error prone
EuGENiA
How EuGENia Works

- Graph Model
- Tooling Model
- Mapping Model

Metamodel + Annotations

GMF
Our Metamodel
Our Metamodel (in Emfatic)
Our annotated metamodel

```java
@namespace(uri="filesystem", prefix="filesystem")
package filesystem;

class Filesystem {
    val Drive["*"] drives;
    val Sync["*" ] syncs;
}

class Drive extends Folder {
}

class Folder extends File {
    @gmr.compartment
    val File["*" ] contents;
}

class Shortcut extends File {
    @gmr.link(target.decoration="arrow", style="dash")
    ref File target;
}

class Sync {
    ref File source;
    ref File target;
    attr String lastSync;
}

class File {
    attr String name;
}
```
A closer look...
@gmfdiagram
class Filesystem {
    val Drive[*] drives;
    val Sync[*] syncs;
}

Filesystem
File

@gmf.node(label="name")

class File {
    attr String name;
}

Shortcut

@gmf.node(label="name")
class Shortcut extends File {
    attr String name;
    @gmf.link(target.decoration="arrow",
        style="dash")
    ref File target;
}

Folder

```java
@gmf.node(label="name")
class Folder extends File {
    attr String name;
    @gmf.compartment
    val File[*] contents;
}
```
Drive

```java
@gmf.node(label="name")
class Drive extends Folder {
    attr String name;
    @gmf.compartment
    val File[*] contents;
}
```
Sync

```java
@gmf.link(label="lastSync", source="source",
          target="target", style="dot", width="2")

class Sync {
    ref File source;
    ref File target;
    attr String lastSync;
}
```
How EuGENia Works

Metamodel + Annotations

Graph Model
Tooling Model
Mapping Model

GMF
Good stuff

- Easy
- High level
- Hides GMF details
- Change resilient
- Can target different editor frameworks in the future
  - Graphiti, Splash
Not so good stuff

- Not 1:1 GMF mapping
  - Intentionally (obviously)
  - But we are adding features
  - Further customization with EOL

- Pollutes metamodel
  - Trade-off for usability
Implementation Notes

- 2 Model-to-Model Transformations
- 1235 Lines of Code
- Transformations in EOL
  - Good example of a model-to-model transformation problem
  - where declarative (mapping) approaches are (extremely) impractical.
What’s Next?

- We are working on EuGENia Live!
- Build editors in your browser.
- Don’t have to worry (much) about that pesky metamodel stuff.
  - Just work with concrete syntax.
  - Don’t have to deal with the separation between models/metamodels that exists in Eclipse.
- Export to Ecore to bootstrap the Eclipse process.
Figur e 1: U M L class diagr am of t he dom ain m odel of EuG EN ia L ive.
(a) The DrawingEditor view.
(b) The PaletteEditor view.

Figur e 2: U sing t he views of EuG EN ia L ive t o define a seat ing plan diagr am and it s language definit ion.
Acquisition of Capability
Capability

- We define capability as:
  - 'the measure of the abilities of an entity to achieve its objectives, especially in relation to its overall mission’ [The Business Dictionary]

- Capability is about being able to solve problems
- It is assessed based on how well the problems are solved in the real world.
Capability-Based Management

- Some governments (including the UK) are moving away from
  - Management of projects in terms of equipment to
  - Management of projects in terms of capabilities.
- In the UK, this is one of the goals of the MoD.
- It means moving from defining problems in terms of concrete solutions to defining problems in terms of abstract needs.
- Why is this useful?
Example

- Previously, MOD procurers might have defined a problem in terms of a need for artillery pieces.
- Defined as a requirement for a capability of firing at range we can consider a set of possible solutions.
  - E.g., bombers, destroyers.
Example

- Each of the solutions (e.g., bomber) proposed on the previous slide satisfies the same need.
- However, they differ in terms of their own individual requirements, their cost, and their original purposes.
- In other words, solutions come with problems.

Modelling can help us understanding problems, solutions, interdependencies, and contexts!
It’s easier than you think to make things worse.
Why?

- Let’s say I buy a set of long-range missiles to solve a military problem.
- Purchasing the missiles has a number of side-effects:
  - I have to store them, maintain them, train people to use them, purchase support equipment, update doctrines, ...
  - And I may scare someone else, who then buys their own long-range missiles, ...
  - ... and then I need further capability.
Defence Lines of Development

- These ideas are inherent in the MoD’s Defence Lines of Development (DLoD).
- DLoDs are used to make up a capability:
  - Training
  - Equipment
  - Personnel
  - Information
  - Doctrine and Concepts
  - Organisation
  - Infrastructure
  - Logistics
Supporting Tradeoffs

- Since the same capability can be produced in many ways there are trade-offs, e.g.,
  - Better training for operators to read a sonar screen vs a better, more easily read sonar screen.
  - Existing levels of personnel and equipment vs fewer personnel and tanks with clearer information and rules of engagement.
  - Better network infrastructure vs better video compression.

- Organisations only have a finite budget so what are the best trade-offs to make?
Introducing CATMOS

- The “Complex Acquisition Tool using Multi-Objective Search”.
  - A generic tool for decision support, inspired by, but not dependent on DLoDs and TLCM.
- Integrates modelling, model transformation, search-based software engineering, and optimisation.
- Implemented using Epsilon (EOL, ETL, Flock).
  - ~ 7K of Epsilon code.
Happy Scenario

- John & Jane are looking to purchase a house in York.
- What kinds of decisions may they need to take?
  - Obviously, which house they want to buy and move into.
- This is clearly trivial.
WRONG!
Alas!

- There are many factors to consider.
- When viewing a house, both John & Jane have an opinion; both views must be considered.
- John is a librarian working in York’s city center; Jane is a lecturer near Osbaldwick (3.5km away), and both need to get to work each day.
- Different houses cost different amounts each month.
It gets worse!

- If they have a cheaper house, they have more disposable income.
  - And, e.g., can afford a car, more entertainment, ..
- Other considerations: what’s the local area like, things to do, etc.
- How can we help support John & Jane in their decision?
Model of Decomposed Goals

- House Decision Making
  - Required Measurements
    - Travelling Time
  - Required Measurements
    - Travel Time
  - Required Measurements
    - Time to get to Library
  - Required Measurements
    - Library Distance
  - Library Distance
    - Speed
  - Required Measurements
    - Speed
  - John Happy with House
    - Number of Rooms
  - House Age
  - Required Measurements
    - John's Opinion
    - Like
  - Dislike
  - Okay
  - Jane Happy with House
    - Jane's Opinion
    - Bad
    - Reasonable
    - Good

- Bakery Distance
- Transport
- Local Leisure Activities
- Activities Quality
- Local Area
- Quality
Model of Acquirable Things
We can search for results
The algorithm is multi-objective random search.

It calculates optimal options (between heterogeneous things) and presents them.

It makes clear the dependencies between capabilities and components.

It combines both quantitative and qualitative optimization.

But it doesn’t tell you which option to choose!
Tool-chain

Provision/Dependency Model → Aggregation Model → Constraint Tree Input Model

High-Level Goal Model → Full Goal Model → Constraint Tree Output Model
Status

- Currently the modelling approach and toolset finds solutions, gives quantitative guidance.
  - Can take into account combinations of quantitative fitness functions and qualitative ones.
- GUI/interface needs more work.
- Applied to a number of examples: real estate, search and rescue, crisis management, next-release problem, ...
- Now taking into account temporal properties.
Super Awesome Fighter
Super Awesome Fighter

- How do you explain modelling to high school students?
  - Who may want to come to university to study the awesomeness that is Software Engineering?

- What do they understand?
  - Language: they may have worked with HTML, PHP, Java, C, ...
  - Stupid video games.
Super Awesome Fighter

- Over several brainstorming sessions, we developed a number of DSLs for describing the behaviour of players in a fighting game.
  - These DSLs (and their evolution) are interesting by themselves, but not the real focus here.
- We also built a game engine, which would take player descriptions and (using Epsilon), interpret them and fight.
One of the most widely used modelling frameworks is the Eclipse Modeling Framework (EMF) [17], part of the Eclipse website: http://www.eclipse.org.

The Fighter Description Language is made up of several elements:

- **Bot**
  - name : String

- **Personality**

- **Behaviour**

- **Characteristic**
  - punchReach : Int
  - punchPower : Int
  - kickReach : Int
  - kickPower : Int

- **Rule**

- **MoveAction**
  - type : MoveActionType

- **FightAction**
  - type : FightActionType

- **Condition**
  - type : ConditionType

- **MoveActionType**
  - walk_towards
  - walk_away
  - run_towards
  - run_away
  - jump
  - crouch
  - stand

- **FightActionType**
  - block_low
  - block_high
  - punch_low
  - punch_high
  - kick_low
  - kick_high

- **ConditionType**
  - always
  - near
  - far
  - much_stronger
  - stronger
  - even
  - weaker
  - much_weaker
Example Character in FDL

JackieChan{
    kickPower = 7
    punchPower = 5
    kickReach = 3
    punchReach = 9
    far [run_towards punch_high]
    near [choose (stand crouch) kick_high]
    much_stronger [walk_towards punch_low]
    weaker [run_away choose (block_high block_low)]
    always [walk_towards block_high]
}
Some details

- We implemented FDL using Xtext.
  - Great tool for rapid development of DSLs.
- Fighter characteristics (power, reach, speed – values between 0..9) represent tradeoffs.
  - A stronger character moves more slowly.
- Behaviour rules specify how fighters act in certain conditions.
  - E.g., choose between block high or block low
- Could introduce high school students to things like sequencing and nondet.
This is cool!

- Kids can write their own players, without knowing anything about DSLs, game engines, etc.
  - But we can slip these ideas in as we go.
- We can construct fun animations.
- We can introduce some MDE concepts, e.g., using EOL to do the simulations, health calculations, validation, etc.
But wait, there’s more...

- We got fed up with losing.
  - More accurately, I got fed up with losing.
- If a high school kid specifies a fighter using FDL, can we determine an “ideal” opponent for them?
  - i.e., one that regularly defeats them?
- More precisely, given a specification of a player, can we determine good opponents, e.g., ones who win $\geq 80\%$ of the time.
Search

- Search-based software engineering is about using optimization techniques to find solutions.
- We implemented a search algorithm for Super Awesome Fighter.
  - The algorithm is based on grammatical evolution (which we implemented in EOL)
  - We combined this with a metaheuristic search algorithm to identify the ‘good’ fighters.
Grammatical Evolution

- Calculate sentences ('programs') from descriptions specified in BNF (or equivalent).
- The main idea in GE is the \textit{genotype to phenotype} mapping.
  - A genotype (e.g., an integer) is mapped to a phenotype, which is a valid sentence in the BNF.
1. Integer string is translated into a Genotype model

2a. Genotype model passed to transformation

2b. Xtext definition of the FDL grammar parsed into a model and passed to transformation

3. Script outputs the fighter as a string

```plaintext
Sample {
  kickPower =6
  punchReach=4
  near[stand_punch_low]
  stronger[run_towards block_high]
  always[jump kick_high]
}
```
What remains?

- Connect our Epsilon GE implementation with a metaheuristic search algorithm.
- Define a fitness metric to assess whether a ‘found’ fighter is difficult to beat.
  - Based this on the difference between number of fights won by a candidate against a seed set, and a target number of winning fights.
  - Numerous settings for running experiments, see our paper at SSBSE’11 for details.
Conclusions?

- Unbeatable fighters can be derived; a simple GA finds such examples ~70% of the time.
  - This also suggests that a human could build an unbeatable fighter in FDL quite easily.
  - Suggests we need to evolve FDL.
- Very easy to develop ‘strong’ fighters, winning 80% of the time.
- The search exposed shortcomings in FDL, e.g., that it needed a ‘completeness’ clause.
  - Also helped debugging fighters.
What’s next?

- Evolution of FDL, including working with different dialects.
  - Modelling fighter behaviour via state machines.

- Other games, including “choose-your-own adventure”.

- “Barely Adequate Fighter”.
The Thrill of Victory!
The Agony of Defeat!
Uncertainty & Sensitivity Analysis

“I took a test in Existentialism. I left all the answers blank and got 100.”
Uncertainty

- All software engineering suffers from degrees of uncertainty.
- Any form of modelling is subject to different levels of uncertainty.
  - Errors of measurement or interpretation.
  - Incomplete information
  - Poor or partial understanding of domain.
- When applying operations to models, uncertainty can lead to unexpected behaviours.
Sensitivity Analysis

- A means to explore how changes to a model affects the output of a model management operation.
  - E.g., a transformation.
- Sensitivity analysis can provide modellers with greater confidence in the adequacy of their model.
- Highlighting sensitive parts of a model can provide insight into execution of an operation.
Types of MDE Uncertainty

- Data uncertainty.
  - E.g., types of values/attributes of classes, transitions, multiplicities.
- Structural uncertainty.
  - E.g., types of relationships between model elements; usage of patterns, or instances of patterns.
- Behavioural uncertainty.
  - E.g., the operating context in which an operation has been developed.
MDE Sensitivity Analysis

- Use an *uncertainty model* to capture data uncertainty in an input model.
- Pass to an input space sampler – a model generator that selects variants of a model according to a sampling method.
- Execute against operation and produce a report based on a domain-specific response measure (e.g., effect on small part of a model).
Clever Parts

- The model generator is the tricky part.
- Based on a lightweight and simple way of representing arbitrary models (of arbitrary metamodels) as integer strings.
Metamodel Finitisation

(a) Step 1: Assigning identifiers to the metamodel.

(b) Step 2: Data finitisation.

(c) Step 3: Mapping a segment to an object.
Example

- We have applied sensitivity analysis framework to CATMOS.
  - Tool was instantiated to calculate optimal acquisition decisions for an airport crisis management system.

- The analysis identified capability that had no effect on a system goal – response time to reach a fire.
  - Such components can be removed: they have a cost, but don’t contribute.
Example (2)

- CATMOS calculates a Pareto front.
  - Solutions that are optimal in some attribute.

- CATMOS found a solution on the first non-dominating rank that appeared to be (on average) better than solutions on the Pareto front.
  - However, this solution is much more sensitive to uncertainty than ones on the Pareto front.
  - So it might be less desirable to engineers.
What’s Next?

- Working on structural uncertainty.
  - Trickier, because greater dependency between changes may exist.
  - E.g., change an attribute type may require changes to metamodel itself.
- Also are adding support for probability distributions to uncertainty.
Enabling Verification

The Epsilon Team
University of York
Enabling verification

- We have used MDE in several projects to enable verification of dependable systems.
  - Connect domain-expert friendly languages
  - To powerful analysis languages and tools.
  - “Formal methods under-the-hood”.

- Two (brief) examples:
  - Railway interlocking.
  - Embedded systems verification.
The INESS Project

- **INtegrated European Signalling System (INESS)**
  - Integration of different signalling systems within Europe

- One of the objectives is to define models of this integrated signalling system
  - Defined in Executable UML (Artisan xUML tool)
  - Models can be analysed via simulation (Cassandra tool)
Formal Verification Strategy

- Generate PROMELA from xUML
  - xUML also used to model (safety) properties of interest.

- We implemented a multi-step transformation of xUML to PROMELA, whereafter verification can take place

- We also automatically generate counter-examples for properties with a false result during verification using model transformations
Sample xUML of Horrors
Automatic Generation of Counter-Examples

- A false verification result produces text at the abstraction level of the target verification tool
- In order to represent the result in an abstraction compatible with the models, we automatically generate UML sequence diagrams
- Four different transformation steps are defined which include the generation of:
  - A counter-example model
  - A trace-sequence model
  - A graphical trace-sequence model
  - The UML files representing the sequence diagram
MADES...

Model-based methods and tools for Avionics and surveillance embedded systems

STREP project of the FP7...
Verification in MADES
Verification
in the context of MADES...

Describe **operational behaviours**, by modelling explicitly the notion of **time** and expressing time **constraints**.
Is the system able to complete Task X within $t$ time units?
Does Event E always precede Event F?
If Event E occurs, will Event F occur within t time units?
(\&\&

;; CLASS DIAGRAM

;; OBJECTDIAGRAM

;; OBJECT brakeS
Hmmm...
No prior experience with formal methods.

Tools not integrated with those that the developers already use.

No prior experience with Lisp.
So our goal is to enable the use of formal verification ...
... by hiding complexity...
... and by integrating formal verification tools with standard modelling tools.
Toolchain

MODELLING
- UML
- MARTE
- SysML
- MADELS Language
- UML Modelling Tool

TRANSFORMATION
- XMI (system model)
- Epsilon
- Java
- JVM
- TRIO formulae

TRACEABILITY
- Traceability tool
- Verification results

VERIFICATION
- SAT/SMT solvers
- Zot (model checker)
- Plug-ins
MADES modeling...
Supported diagrams...

- Class diagrams
- Object diagrams
- Interaction overview diagrams
- Sequence diagrams
- State diagrams
Time constraints...

@now-@braking.enter = 5

Textual DSL
Time properties...
Transformation component...

XMI

Epsilon → Java → JVM

TRIO
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version="2.1" xmlns:xmi="http://schema.omg.org/spec/XMI/2.1" xml
<ulm:Model xmi:id="_eHaaULrTEeGLIomrb6Rm7g" name="CCAS_Paper">
  <eAnnotations xmi:id="_eHaaUbrTEeGLIomrb6Rm7g" source="Objing">
    <contents xmi:type="uml:Property" xmi:id="_eHaaUrrTEeGLIomrb6Rm7g" name
      <defaultValue xmi:type="uml:LiteralString" xmi:id="_eHaaU7rTEeGLIomrb
    </contents>
  </eAnnotations>
  <ownedComment xmi:id="_eHaaVbrTEeGLIomrb6Rm7g">
    <eAnnotations xmi:id="_eHaaVrrTEeGLIomrb6Rm7g" source="Objing">
      <contents xmi:type="uml:Property" xmi:id="_eHaaV7rTEeGLIomrb6Rm7g" na:
Initialization and definition of available operations for each class

```java
ClassDiagram classDiagram = ClassDiagram.getInstance("", madesModel);
Class Radar = Class.getInstance("_GfBnH3MyeGQ35u-3v15yg", "Radar", classDiagram);
Class CAN = Class.getInstance("_GfBnSHMyeGQ35u-3v15yg", "CAN", classDiagram);
Operation Op_sendSensorDistance_CAN = Operation.getInstance("_GfBnVXMyeGQ35...
Operation Op_sendBrakeCommand_CAN = Operation.getInstance("_GfBnVnMyeGQ35u-...

Class Controller = Class.getInstance("_Ge9VrXMyeGQ35u-3v15yg", "Controller";
Operation Op_notifyDistance_Controller = Operation.getInstance("_GfBnG3MyeG...
Class BrakingSystem = Class.getInstance("_GfBnNHMyeGQ35u-3v15yg", "BrakingS...
Operation Op_notifyBrake_BrakingSystem = Operation.getInstance("_GfBnRXMyeGq...
```
Initialization and definition of available operations for each class

47  (;; OPERATIONS SEMANTICS
48  (<-> (P- OBJctrl0POp_notifyDistance_Controller) (P- $MSG_GfBnx3MyEeGQ35u-3v
49  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
50  (;; SEQUENCE DIAGRAM SDSendBrakeCommand
51  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
52  (<-> (P- SDDSSendBrakeCommand) (11 (P- SDDSSendBrakeCommandSTART) (since_e
53  
54  (;; CLOSE WORLD SEMANTICS
55  
56  (;; START SEMANTICS
Verification component...

Lisp
Traceability component...

EGL Traceability

Hyperlinks
Traceability editor...