CISC836: Models in Software Development: Methods, Techniques and Tools

Topic 5: Domain Specific Languages

Juergen Dingel
Feb, 2017
Modeling Languages

**Modelica**
- Physical systems
- Equation-based

**Simulink**
- Continuous control, DSP
- Time-triggered dataflow

**Stateflow**
- Reactive systems
- Discrete control
- State-machine-based

**AADL**
- Embedded, real-time

**UML**

**UML MARTE**
- Embedded, real-time

**Promela**
- Finite-state
- Reactive systems

**UML-RT**
- Embedded, real-time
- State-machine-based

**Modelica**
- Physical systems
- Equation-based

**Simulink**
- Continuous control, DSP
- Time-triggered dataflow

**Stateflow**
- Reactive systems
- Discrete control
- State-machine-based

**AADL**
- Embedded, real-time

**UML**

**UML MARTE**
- Embedded, real-time

**Promela**
- Finite-state
- Reactive systems

**UML-RT**
- Embedded, real-time
- State-machine-based

Increasing generality  increasing domain-specificity

---

**Examples in**
- [Voe13, Kel08]

**EGGG**
- [Orw00]

---


Expressing SW models: Overview (Cont’d)

Domain-specific languages (DSLs)

1. Intro and examples (EGGG, CPML, UML-RT)
2. Pros and cons
3. Defining DSLs
   - abstract syntax
     - CFGs in BNF
     - meta models
       - MOF, ECore and OCL
   - concrete syntax
   - semantic mapping
     - Denotational, operational, axiomatic, translational
4. Defining DSLs using UML
   - semantic variation points, profiles
5. DSL tools
   - EMF, GMF, Graphiti, Xtext
Expressing SW models: Overview (Cont’d)

Domain-specific languages (DSLs)

1. Intro and examples (EGGG, CPML, UML-RT)
2. Pros and cons
3. Defining DSLs
   - abstract syntax
     - CFGs in BNF
     - meta models
       - MOF, ECore and OCL
   - concrete syntax
   - semantic mapping
     - Denotational, operational, axiomatic, translational
4. Defining DSLs using UML
   - semantic variation points, profiles
5. DSL tools
   - EMF, GMF, Graphiti, Xtext
## Domain-Specific Languages

### General purpose vs. Domain-specific

<table>
<thead>
<tr>
<th>General purpose</th>
<th>Domain-specific</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>Language size</td>
</tr>
<tr>
<td>large</td>
<td>small</td>
</tr>
<tr>
<td>always</td>
<td>Turing completeness</td>
</tr>
<tr>
<td>years to decades</td>
<td>Lifespan</td>
</tr>
<tr>
<td>large, anonymous</td>
<td>User community</td>
</tr>
<tr>
<td>slow</td>
<td>Evolution</td>
</tr>
</tbody>
</table>

- **Productivity**
- **Quality**
- **Analysis capabilities**
- **Communication**
- **Need for specific expertise**
- **Maintainability**
- **Integratibility**

### Potential for Impact

- **Potential for positive impact**
- **Potential for negative impact**
DSLs: Examples

- **Web development**
  - WebDSL: [Vis08], http://webdsl.org

- **Robotics**
  - RobotML: http://robotml.github.io

- **Train signaling**
  - Graphical language and analysis [ECM+08, SHM+12]

---


DSLs: Examples (Cont’d)

- **Financial industry**
  - RISLA: a DSL for describing financial products (e.g., mortgages, interest rate swaps) [vDe97]
  - DSLFin’13 Workshop [DSLFin13]
    - Resource page lists 22 DSLs

- **Healthcare**
  - Clinical decision support system [MLN+09]

- **Home automation In [Jimenez et al 09]:**
  - Home automation system [SJR+11]

---

[vDe97] van Deursen, Domain-Specific Languages versus Object-Oriented Languages. 1997
DSLs: Examples (Cont’d)

- Real-time embedded
  - UML-RT (see below)
    - UML profile for real-time, concurrent, embedded systems
    - tool: IBM Rational RoseRT (IBM Rational RSA-RT)
  - UML MARTE
    - UML profile for Modeling and Analysis of Real-time and Embedded systems
    - supports performance analysis
    - tools: Papyrus
  - Stateflow/Simulink
  - Esterel/Scade
DSLs: Examples (Cont’d)

- In [KT08], http://www.dsmbook.org
  - IP telephony and call processing
  - insurance products, home automation
  - mobile phone applications using a Python framework
  - digital wristwatch
- In [Voe13], http://dslbook.org
  - Component architecture
  - Refrigerator configuration
  - Pension plans
- **EGGG:** The Extensible Graphical Game Generator [Orw00]
When to use DSLs?

- Need lots of expertise about domain, problem and how to solve it (e.g., relevant domain concepts, modeling and code patterns, etc)
- E.g., Orwant’s game generator was made possible by a very careful classification of games with respect to several criteria and properties [Orw99]

“We need to know what we are doing before we can automate it. A DSM solution is implausible when building an application or a feature unlike anything developed earlier”

[KT08, p18]

Modeling Languages

Modelica
- Physical systems
- Equation-based

Simulink
- Continuous control, DSP
- Time-triggered dataflow

Stateflow
- Reactive systems
- Discrete control
- State-machine-based

AADL
- Embedded, real-time

UML
- Embedded, real-time

UML MARTE
- Embedded, real-time

Promela
- Finite-state
- Reactive systems

UML-RT
- Embedded, real-time
- State-machine-based

Examples in
[Voe13, Kel08]

EGGG
[Orw00]

Increasing generality

Increasing domain-specificity
DSML Example: UML-RT

UML profile for (soft) real-time, embedded systems

capsule diagram (structure)

state machine diagram (one per capsule) (behaviour)

analysis

IBM Rational RSA-RTE

C++
DSML Example: UML-RT (Cont’d)

Hierarchical Composition in UML-RT

Port
- concrete realization of an interface by means of an object
- equipped with a protocol
  - list of incoming/outgoing messages/signals
  - (signal flow)
- the aid for “encapsulation” and “separation of concerns”

Capsule
- active object or “passive” container
- communication with the environment
  - signal-based (asynchronous message exchange)
  - exclusively via interface objects (ports)
- supports hierarchical composition

Connector
- communication link between ports
- “drives” protocol
DSML Example: UML-RT (Cont’d)

Example: UML-statecharts

```
h.lock/l.down  l.ready/r.down  r.ready/h.done
wld          wrd

UNLD

r.ready/h.done  l.ready/r.up  h.unlock/l.up
wru          wlu

LCKD

«capsule» :MotorControl

h l r
```

February 4, 2003
© Ingrid H. Krueger
CSC836, Winter 2017

DSMLs
Signal-Based Communication

- Capsules receive and send *signals* via their *ports*
- Signals, which cannot be processed immediately, are stored in a *queue*
DSL example: EGGG

game is poker
turns alternate clockwise
Discard means player removes 0..3 cards or 4 cards if Ace
Fold means player loses
2..6 players
game is Shuffle(deck) and Deal(cards, 5) and (bet(money) or Fold)
    and Discard(hand, N) and Deal(cards, 5-N)
    and (bet(money) or Fold) and compare(cards)

StraightFlush is (R, S) and (R-1, S) and (R-2, S) and (R-3, S) and (R-4, S)
FourKind is (R, s) and (R, s) and (R, s) and (R, s)
FullHouse is (R, s) and (R, s) and (R, s) and (Q, s) and (Q, s)
Flush is (R, S) and (R, S) and (R, S) and (R, S) and (R, s)
Straight is (R, s) and (R-1, s) and (R-2, s) and (R-3, s) and (R-4, s)
ThreeKind is (R, s) and (R, s) and (R, s)
TwoPair is (R, s) and (R, s) and (Q, s) and (Q, s)
Pair is (R, s) and (R, s)
HighCard is (R, s)

hands are [StraightFlush, FourKind, FullHouse, Flush, Straight,
            ThreeKind, TwoPair, Pair, HighCard]

hand is five cards
goal is highest(hand)

P13: [Orw00]
Advantages of DSLs

- Allow solution to be expressed at level of abstraction of problem
  - artifacts more likely to be
    - concise, self-documenting
    - understood, validated, modified, developed by domain experts
- Enhance productivity, reliability, maintainability & portability
- Embody domain knowledge
  - facilitate conservation and reuse
Disadvantages of DSLs

- Costs of
  - designing, implementing, maintaining, evolving a DSL
    - relevant concepts and abstractions? proper scope? effective syntax?
      - supporting tooling? domain stable enough?
  - integrating DSLs into
    - each other
    - existing workflows, processes, and legacy code
  - education, training
Expressing SW models: Overview (Part 2)

2. Domain-specific languages
   1. Intro and examples (Risla, EGGG, CPML, UML-RT)
   2. Pros and cons
   3. Defining DSLs
      ° abstract syntax
         - CFGs in BNF
         - meta models
            ● MOF, ECore and OCL
      ° concrete syntax
      ° semantic mapping
         - Denotational, operational, axiomatic, translational
   4. Defining DSLs using UML
      ° semantic variation points, profiles, and meta model extensions
   5. DSL tools
      ° EMF, GMF, Graphiti, Xtext
Definition of (domain-specific) languages

A DSL is a 7-tuple:

1. abstract syntax
2. concrete syntax
3. abstract-to-concrete-syntax mapping
4. serialization syntax
5. abstract-to-serialization mapping
6. semantic domain
7. abstract-to-semantic mapping

syntax
(a.k.a., “static semantics”)

semantics
(a.k.a., “dynamic semantics”)

concrete syntax

abstract syntax

serialization syntax

semantic domain
Abstract Syntax

- **In programming languages:**
  - defines language elements and rules for composing them [GS04]
  - defines parse trees, *abstract syntax trees* (ASTs)

- **In MDD:**
  - defines concepts, relationships, integrity constraints (“well-formedness rules”, “static semantics”) [Kle09]
  - defines *abstract syntax graphs* (ASGs)

- **Does not** define how to render language elements to the user as, e.g., linear strings or 2D drawings (that is what the concrete syntax is for)

- **Ways to define abstract syntax:** E.g.,
  1. **Regular expressions** (regular grammars)
  2. **Context-free grammars** (CFGs) (expressed using Backus-Naur Form (BNF))
     - e.g., ITU’s ASN.1 [ITU09] (as compared to OMG’s MOF)
  3. **Meta models**
### Regular expressions

\[
\begin{align*}
<\text{Var}> & ::= <\text{Letter}> + ( <\text{Letter}> )* \\
<\text{Letter}> & ::= a + b + c + \ldots + z + A + B + \ldots + Z
\end{align*}
\]

### BNF

\[
\begin{align*}
<\text{Exp}> & ::= <\text{Num}> \mid <\text{Var}> \mid <\text{Exp}> <\text{BinOp}> <\text{Exp}> \mid <\text{UnOp}> <\text{Exp}> \\
<\text{BinOp}> & ::= + \mid - \mid * \mid / \\
<\text{UnOp}> & ::= - \\
<\text{Var}> & ::= <\text{Letter}> \mid <\text{Letter}> <\text{Var}> \\
<\text{Letter}> & ::= a \mid b \mid c \mid \ldots \mid z \mid A \mid B \mid \ldots \mid Z
\end{align*}
\]

### Abstract Syntax Tree (AST)

Parse tree

Which expression does this AST belong to?
How exactly does a BNF define a language?

Example:

• Consider the CFG $G$
  
  \[
  <S> ::= ab \mid ab <S>
  \]

• Let $N = \{<S>\}$ and $T = \{a, b\}$

• Then, $L(G)$ can be characterized in two ways:
  1. $L(G) = \{w \in T^* \mid <S> \rightarrow w\}$
     where $\rightarrow \subseteq (N \cup T)^* \times (N \cup T)^*$ is the smallest relation satisfying
        1. $<S> \rightarrow ab$ (i.e., $(<S>, ab) \in \rightarrow$), and
        2. if $<S> \rightarrow w$, then $<S> \rightarrow abw$ for all $w \in T^*$
  2. $L(G)$ smallest set $X \subseteq T^*$ such that $X = F(X)$ where
     
     \[
     F(X) = \{ab\} \cup X \cup \{abw \mid w \in X\}
     \]
     i.e., $L(G)$ is smallest “fixed point” of $F: T^* \rightarrow T^*$

• Note that, in this case, the grammar is unambiguous, i.e., every $w \in L(G)$ has exactly one parse tree (i.e., Abstract Syntax Tree, AST)
How exactly does a BNF define a language? (Cont’d)

- Suppose CFG $G$ is not unambiguous
- Then, better to think of $L(G)$ as set of all ASTs of $G$:
  - $L(G) = \{ t \mid \text{“}t \text{ is AST of } G\text{”}\}$ where
  - $t$ is AST of $G$ iff
    - root of $t$ is the start symbol of $G$
    - all leaves in $t$ are terminal (i.e., $\in T$)
    - a node $n$ in $t$ has the children
      $$n_1, \ldots, n_m$$
      iff
      $$n ::= n_1 \ldots n_m$$
      is a production in $G$
Describing abstract syntax of a modeling language using CFGs: An Example

- Want to define modeling language OSL (Our Simple Language) such that following is well-formed OSL model:

![Diagram of OSL model]

**Figure 8.2** Example OSL model

Defining OSL using a BNF

1. Model ::= (ModelElement)*
2. ModelElement ::= Class
3. Class ::= ClassName (Features)* (StateMachine)?
4. ClassName ::= Identifier
5. Feature ::= Attribute | Method
6. Attribute ::= AttributeName TypeRef
7. AttributeName ::= Identifier
8. TypeRef ::= Identifier
9. Method ::= MethodName (Argument)* Statement
10. MethodName ::= Identifier
11. Argument ::= ArgumentName TypeRef
12. ArgumentName ::= Identifier
13. Statement ::= (Statement)* | AssignmentStatement
14. AssignmentStatement ::= LHS RHS
15. LHS ::= AttributeRef
16. AttributeRef ::= Identifier
17. RHS ::= AttributeRef | ArgumentRef
18. ArgumentRef ::= Identifier
19. StateMachine ::= State
20. State ::= StateName (StartState)? (State)* (Transition)*
21. StateName ::= Identifier
22. StartState ::= StateRef
23. Transition ::= MethodRef StateRef
24. MethodRef ::= Identifier
25. StateRef ::= Identifier

Notes:
- Need **explicit names** (e.g., StateRef) to refer to other elements
- **Not every instance well-formed** OSL model: E.g.,
  - a state has at most one parent state
  - a transition connects two states in the same state machine
- These **additional constraints** are enforced by context analysis by parser

) BNF alone incomplete specification of OSL

**Figure 8.3** BNF abstract syntax

An OSL model as AST

Notes:

- Represents **well-formed** element of OSL
- **Can be derived** from start symbol of grammar using only grammar’s productions (i.e., $2 L(Model)$) and satisfies any additional constraints

Figure 8.4  AST for car rental model  [GS04]
Meta models

- A **meta model** \( MM \) is a model (a specification) of a set of models (i.e., a modeling language \( L(MM) \))
- An **instance** \( M \) of meta model \( MM \) is a well-formed model in modeling language \( L \) (i.e., \( M \in L(MM) \))

**Languages for expressing meta models**

- **Meta Object Facility (MOF):**
  - OMG standardized language for defining modeling languages
  - subset of UML class diagrams: types (classes, primitive, enumeration), generalization, attributes, associations, operations
- **ECore:**
  - Eclipse version of MOF; used by Xtext
- **Object Constraint Language (OCL):**
  - declarative language to express well-formedness rules (e.g., “the inheritance hierarchy is acyclic”)
Meta Object Facility (MOF)

- OMG standard http://www.omg.org/mof
- A standardized model for meta modeling (i.e., a metameta model):
  - “simplest set of concepts required to capture metamodels” [MSUW04]
  - DSL for the development of meta models (i.e., DSL for the definition of the abstract syntax of modeling languages)
  - Example: UML2 meta model (i.e., the UML2 specification) is expressed using MOF
- Main goal: interoperability
- Question:
  - How to define MOF? Using MOF!
Meta Object Facility (Cont’d)

- **Example:** How is UML defined with MOF?

UML2 meta model/specification, MOF model of UML2 (M2)

UML2 model/user model (M1)

- MOF uses a subset of UML class diagrams: types (classes, primitive, enumeration), generalization, attributes, associations, operations

```
<table>
<thead>
<tr>
<th>Class</th>
<th>Association</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Person</th>
<th>Car</th>
</tr>
</thead>
<tbody>
<tr>
<td>metamodel</td>
<td></td>
</tr>
</tbody>
</table>
```

or, more precisely

```
<table>
<thead>
<tr>
<th>Class name</th>
<th>attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>emp</th>
<th>enrolment()</th>
</tr>
</thead>
<tbody>
<tr>
<td>name:</td>
<td>operation</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>emp#</th>
<th></th>
</tr>
</thead>
</table>
```
Example: Specifying generalization in UML using MOF

UML2 meta model/specification, MOF model of UML2 (M2)

UML2 model/user model (M1)

Excerpt of UML 2.1.2 Metamodel (Class Diagrams)

Figure 7.12 - Classes diagram of the Kernel package

formal/2007-11-02. 2007
CISC836, Winter 2017

DSls
EMF and ECore

- **Eclipse Modeling Framework (EMF)**
  - modeling framework and code generation facility for building tools and other applications based on a structured data model
  - http://eclipse.org/modeling/emf/

- **Ecore**
  - Version of MOF using in EMF
  - Runtime support
    - change notification
    - persistence w/ XMI serialization
    - API for manipulation

http://eclipse.org/Xtext/documentation.html#emf_integration
Describing abstract syntax of a modeling language using meta modeling: An Example

- Suppose want to define modeling language OSL (Our Simple Language) such that following is well-formed:

```
class CarRental {
    setSpec(spec) { reservedSpec = spec }
    allocate(c: Car) { allocatedCar = c }
    unallocate() { car = null }
}

class Car {
    setSpec(spec) { specification = spec }
}
```

![Diagram of Car and CarRental classes](image)

**Figure 8.2** Example OSL model

J. Greenfield, K. Short. Software Factories. Wiley. 2004
Meta model contains more constraints than BNF, but not all
Express all missing constraints in separate constraint language
Typically, the **Object Constraint Language (OCL)** is used for this purpose

Notes
Object Constraint Language (OCL)

- Declarative language for describing well-formedness rules of models
- May be used with any MOF-based meta model

**Examples:**

- "The source & target states of transition belong to same machine"

  **Transition**
  
  \[
  \text{target.root().machine} = \text{source.root().machine}
  \]

  where `root()` is
  
  ```
  \text{State::root() : State} \\
  \quad \text{if parent = null then self else parent.root()}
  ```

- "The left-hand side and the right-hand side of an assignment have the same type"

  **AssignmentStatement**
  
  \[
  \text{lhs.type} = \text{rhs.type}
  \]
An OSL model as ASG

- Abstract Syntax Graph (ASG)
  - Is UML Object Diagram
- This ASG G satisfies all constraints expressed in OSL meta model

Figure 8.6  Car rental model as metamodel instance

J. Greenfield, K. Short. Software Factories. Wiley. 2004
Example of 4-layer meta model hierarchy in UML

Figure 7.8 - An example of the four-layer metamodel hierarchy

How exactly does a meta model define a language?

- If language $L(MM)$ is described by some meta model $MM$, then $L(MM)$ can be thought of as the set of all ASGs of $MM$:
  - $L(MM) = \{ g \mid \text{"g is ASG of MM"} \}$
  - $g$ is ASG of $MM$ iff
    - $g$ satisfies all the constraints expressed in $MM$
CFGs vs Meta models

**CFGs**

- textual
- well-researched with excellent tool support
- references must be encoded via, e.g., ids (e.g., `StateRef`)
- no name spaces
- no place to put additional constraints

**Meta Models**

- graphical
- relatively novel
- attributes aid readability
- elements can be referred to directly
- classes define a namespace
- OCL can be used for additional constraints
- harder to define semantic mappings

19. `StateMachine ::= State`
20. `State ::= StateName (StartState)? (State)* (Transition)*`
21. `StateName ::= Identifier`
22. `StartState ::= StateRef`
23. `Transition ::= MethodRef StateRef`
24. `MethodRef ::= Identifier`
25. `StateRef ::= Identifier`
Comparing Abstract Syntax Systems

Technology #1  Technology #2  Technology #3  Technology #4
(formal grammars  (MOF + OCL)  (XML Meta-Language)  (Ontology engineering)
attribute grammars,
etc.)

M3

EBNF

MOF

A XML DTD
Or Schema

Upper Level
Ontologies

M2

Pascal Language
Grammar

The UML
meta-Model

A XML document

KIF
Theories

M1

A specific
Pascal Program

A Specific
UML Model

A XML
document

+Description
Logics

+Xlink, Xpath, XSLT

+ RDF, OIL, DAML

+etc.

[XMI=MOF+XML+OCL]

(From J. Bézivin)

© J.-M. Jézéquel, 2008
Concrete Syntax

- Need to decide how AST or ASG is displayed to and input by the user
- The abstract-to-concrete mapping assigns elements of abstract syntax to some concrete syntax

**Examples:**
1. Linear concrete syntax

   **Abstract Syntax**

   ![Abstract Syntax Diagram]

   **Concrete Syntax**

   (examples)

   - “2*3 + 4 + 5”
   - “(2*(3+4))+5”
   - “+[*[2,+[3,4]],5]”

Which of these is bad?
Example 2: Graphical concrete syntax

Figure 8.7 A state machine in graphical concrete syntax

Figure 8.8 Annotated metamodel for state diagram view

J. Greenfield, K. Short. Software Factories. Wiley. 2004
Example 2: Graphical concrete syntax (Cont’d)

Figure 8.9 Concrete syntax for ai

Figure 8.10 Annotated metamodel for class notation
Other examples: Graphical concrete syntax

http://scratch.mit.edu

Lego Mindstorms’ NXT-G language
Abstract and concrete syntax: summary

- Definitions of abstract and concrete syntax of modeling language L
  - place *constraints* on the shape, form, and display of model M
  - define when M and its presentation to user is *well-formed*

1. Format of abstract syntax constraints:
   - context-free grammars, meta models, OCL

2. Format of concrete syntax constraints:
   - annotations
Serialization Syntax

- In which format should a model be persisted (i.e., saved)?
- The abstract-to-serialization-mapping maps elements of the abstract syntax to some serialization syntax
- Typically done using **Extensible Markup Language (XML)**
- **Two ways:**
  1. Define your own **XML Schema Definition (XSD)**
  2. If meta model is expressed using **Meta-Object Facility (MOF)**, then can use **XML Metadata Interchange (XMI)**
- **Another relevant standard:**
  - **XMI**: OMG standard for exchanging metadata information via XML
  - most common use of XMI is as an interchange format for UML models, but it can also be used for serialization of models of other languages
Serialization syntax: an example

Figure 8.11  XML for ASG fragment of car rental model

J. Greenfield, K. Short. Software Factories. Wiley. 2004
Expressing SW models: Overview (Part 2)

2. Domain-specific languages

1. Intro and examples (e.g., Risla, EGGG, CPML, UML-RT)
2. Pros and cons
3. Defining DSLs
   - abstract syntax
     - CFGs in BNF
     - meta models
       - MOF, ECore and OCL
   - concrete syntax
     - semantic mapping
       - Denotational, operational, axiomatic, translational
4. Defining DSLs using UML
   - semantic variation points, profiles
5. DSL tools
   - EMF, GMF, Graphiti, Xtext
Techniques for the definition of semantics

- **Denotational**
  Meaning of program given by *mathematical function* operating on a formalization of state (e.g., Alloy)

- **Operational**
  Meaning of program given by collection of formal *execution rules* operating on a formalization of state (e.g., Promela, F#, Petri nets)

- **Axiomatic**
  Meaning of program given *by proof system* describing effect of program statements on assertions

- **Translational**
  Meaning of program given by *translation* (implicit or explicit) to equivalent program in another, known language (e.g., Promela)

Translational semantics

- Semantics of elements of a modeling language $L$ is given by translating them into another (better understood) language $L'$

- **Examples:**
  - **Generic Modeling Environment (GME) [BGKSN06]:**
    - “A DSML’s dynamic semantics can be enforced by applying model interpreters written using a traditional programming language like C++, Python, or Java and registered with GME.”
  - **IBM Rational RoseRT:**
    - UML-RT models are translated into C++ code
  - **[MC01]:**
    - Translate UML to Promela
[MC01] Translating UML into Promela

Figure 3. A few of the homomorphic mappings of components from the UML metamodel to the Promela metamodel.

Uses of formal semantics

- For debugging & clarifying notations:
  - van der Beeck’s paper on UML-RT [vdB06]
  - Crane’s paper on state machines [CD07]
  - Gessenharter paper on associations [Ges08]
  - Tenzer and Stevens paper on state machines [TS03]

- For generative purposes:
  - ASF+SDF [vdBK05]
  - Using “template semantics”:
    - model checker generation for state machine variants [DAN03]
    - semantically configurable code generation [PADS12]
  - static analyzer generation for assembler code via transition system language (TSL) [LR13]
Expressing SW models: Overview (Part 2)

2. Domain-specific languages
   1. Intro and examples (Risla, EGGG, CPML, UML-RT)
   2. Pros and cons
   3. Defining DSLs
      ° abstract syntax
         - CFGs in BNF
         - meta models
            - MOF, ECore and OCL
      ° concrete syntax
      ° semantic mapping
         - denotational, operational, axiomatic, translational
   4. Defining DSLs using MOF or UML
      ° Semantic variation points, profiles
   5. DSL tools
      ° EMF, GMF, Graphiti, Xtext
Using UML or MOF to define DSLs

- **UML offers two customization mechanisms [FGDT06]**
  1. semantic variation points (see below)
  2. profiles (see below)

- **Build a new language using MOF [MSUW04]**
  - MOF concepts: types (classes, primitive, enumeration), generalization, attributes, associations, operations
  - UML and MOF use same concrete syntax
  - Building a MOF model is like building UML class diagram


**Semantic variation points**

“Semantic Variation Points” explicitly identify areas where semantics are intentionally under-specified to provide leeway for domain-specific refinements of general UML semantics” [UML 2.4.1, p16]

- **Small adjustments**, not completely new language
- **Examples** (from UML 2.4.1)
  - “Precise semantics of shared aggregation varies by application area and modeler” (page 36)
  - “The order and way in which part instances in a composite are created is not defined.” (page 38)
  - “The behavior of an invocation of an operation when a precondition is not satisfied is a semantic variation point” (page 107)
Profiles

- Consist of two concepts
  - Stereotypes
    - add labels (e.g., <<capsule>>) to UML elements (e.g., classes)
    - add tags (attributes)
  - Constraints
    - express rules possibly involving the new tags (attributes)
    - using OCL
- Many different UML profiles already exist
  - UML-RT, SysML, UML-MARTE, UML-SPT, UML-XML, UML$_{sec}$
  - many of them proprietary
Profiles: Example

Simple EJB profile

UML 2.5 Specification, page 277
Expressing SW models: Overview (Part 2)

2. Domain-specific languages

1. Intro and examples (Risla, EGGG, CPML, UML-RT)
2. Pros and cons
3. Defining DSLs
   - abstract syntax
     - CFGs in BNF
     - meta models
       - MOF, ECore and OCL
   - concrete syntax
   - semantic mapping
     - Denotational, operational, axiomatic, translational
4. Defining DSLs using UML
   - semantic variation points, profiles
5. DSL tools
   - EMF, GMF, Graphiti, Xtext
DSL tools

- Eclipse, EMF, GMF, Graphiti
- Xtext [Assignment 3]
- Generic Modeling Environment (GME) (Vanderbilt)
- Spoofax
- JetBrains Meta Programming System (MPS)
- MetaEdit+ (MetaCase)
- IBM RSA (UML based)
- MS Visual Studio
- Kermit
- ...
EMF + X

EMF = modeling framework and code generation facility

ECore = Eclipse version of MOF

http://www.eclipse.org/modeling/emf/

Meta model

used to define

used to generate

abstract syntax

EMF

used to create

Model

Textual Editor

concrete syntax

CISC836, Winter 2017

DSLs
EMF + X

EMF = modeling framework and code generation facility
http://www.eclipse.org/modeling/emf/

ECore = Eclipse version of MOF
http://www.eclipse.org/modeling/emf/

Meta model

EMF + X

Graphical Editor

used to create

Model

concrete syntax

EMF

used to generate

abstract syntax

Textual Editor

used to define

where

- X = Graphiti, https://eclipse.org/graphiti, or

CISC836, Winter 2017

DSLs

62
Efforts related to DSLs

- Software Factories (Microsoft, [GS04])
- Intensional Programming ([Sim01], [ADKdMRS98])
- Language-oriented programming ([MPS09], [LOP09])
- Language workbench ([Fow09])
Xtext

- Eclipse-based open-source framework for development of programming languages and domain-specific languages

- Offers
  - Parser generator
  - Editor plugin generator supporting
    - Syntax highlighting
    - Well-formedness checking (validation) w/ error markers and quick fixes
    - Background parsing
    - Auto-completion with content assist
    - Hyperlinking connecting uses with declarations
    - Hovering
    - Folding and outline view
  - Support for
    - Code generation (using Xtend, a variant of Java)
    - Interpretation, translation to Java
  - Large user community, http://www.eclipse.org/Xtext/community.html

“A language is only as good as its supporting tooling”
[B. Selic]
Xtext: Supporting technology

- Parser generation
  - Antlr (www.antlr.org)
  - lex, flex and yacc, bison (dinosaur.compilertools.net)

- Eclipse
  - Generated editor is an Eclipse plugin
    - Release engineering
    - Git

- Eclipse Modeling Framework (EMF)
  - Modeling framework and code generation facility for building tool based on structured data
  - Ecore for describing and implementing modeling languages

- Java/Xtend
ANTLR (Another Tool for Language Recognition) is a powerful parser generator for reading, processing, executing, or translating structured text or binary files

```
grammar Expr;
prog: (expr NEWLINE)* ;
expr: expr ('*'|'/')
   | expr ('+','-')
expr : INT
    | '(' expr ')' 
NEWLINE : [\r\n]+ ;
INT : [0-9]+ ;
```

```bash
$ antlr4 Expr.g4
$ javac Expr*.java
$ grun Expr prog -gui
100+2*34
^D
```

From [www.antlr.org](http://www.antlr.org):
Xtend

From [eclipse.org/xtend](http://eclipse.org/xtend):

“Xtend is a flexible and expressive dialect of Java, which compiles into readable Java 5 compatible source code”

**Some features:**

- More defaults
- Optional semicolons
- Implicit returns
- Type inference
- Better support for code generation
- Extension methods
- Lambda expressions
- Multiple dispatch
- Shorthands for getters and setters
Using Xtext

0. Installation instructions etc on Assignment 3 page
1. Create Xtext project
   In Package Explorer: “New | Project ...” then “Xtext Project”
Using Xtext (Cont’d)

2. Create grammar `.xtext` in folder “src/<project name>”

3. Generate Xtext artifacts
   - in “src-gen” folder: `.java`
   - in “model/generated” folder: `.ecore, .genmodel`
Using Xtext (Cont’d)

```plaintext
grammar myDsl.MyDsl with org.eclipse.xtext.common.Terminals

generate myDsl "http://www.MyDsl.myDsl"

Model:
greetings +=Greeting;

Greeting:
'Hello' name=ID '!' ;
```
Using Xtext (Cont’d)

4. Start editor
   Right-click project, “Run As | Eclipse Application”

5. Input text, validate, etc

6. Inspect generated output
Using Xtext (Cont’d)

6. Implement custom validation rules

In folder “src/<project name>/validation/<language name>.xtend”
Using Xtext (Cont’d)

7. Implement interpreter
   - in “src/<project name>/interpreter”
Using Xtext (Cont’d)

8. Implement code generator
   • in “src/<project name>/generator”
   • implement “doGenerate” and “compile” using “filter”
   • integrate into Eclipse build mechanism
   • allow for invocation from command line
A3: Urml

- Textual modeling language for reactive systems

- Support for
  - structural modeling via
    - Classes
    - Composite structures (connectors, ports, protocols)
  - behavioural modeling via
    - State machines
    - Simple, imperative action language

- Inspired by UML-RT

- Keith Yip’s 2014 MSc
  
  [https://qspace.library.queensu.ca/handle/1974/12274](https://qspace.library.queensu.ca/handle/1974/12274)
A3: Urml (Cont’d)

```java
/* A simple example that consists of a producer and a consumer */
model handshake {
  root capsule Handshake {
    capsuleInstance sender : Originator
    capsuleInstance receiver : Receiver
    connector sender.hand and receiver.hand
  }
  capsule Originator {
    external port hand : HandshakeProtocol
    logPort logger
    stateMachine {
      state start
      final state end
      transition init : initial -> start {
      }
      transition doHandShake : start -> end {
        action {
          send hand.shake()
          log logger with "sent a handshake"
        }
      }
    }
  }
  capsule Receiver {
    external port ~hand : HandshakeProtocol
    logPort logger
    stateMachine {
      state start
      final state end
      transition init : initial -> start {
      }
      transition receiveHandshake : start -> end {
        triggeredBy hand.shake()
        action {
          log logger with "received a handshake"
        }
      }
    }
  }
  protocol HandshakeProtocol {
    outgoing {
      shake()
    }
  }
}
```
A3: Urml (Cont’d)
A3: Urml (Cont’d)