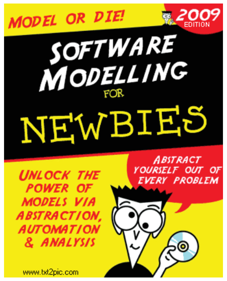
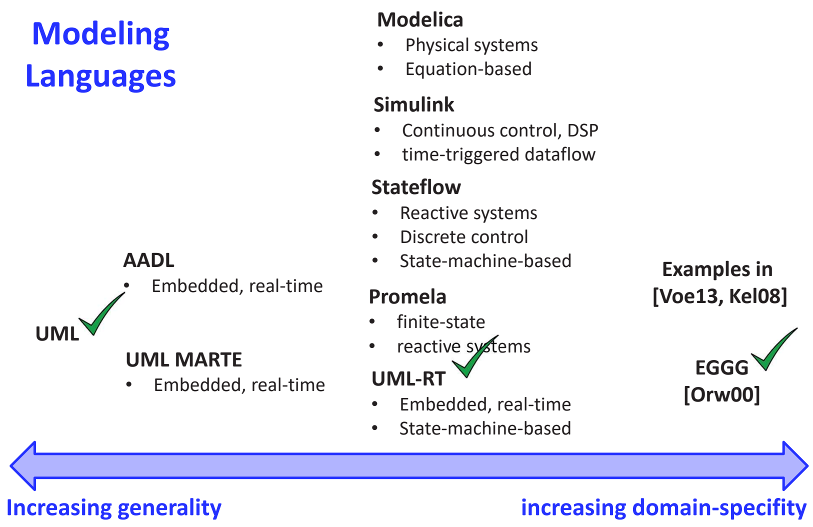
CISC836: Models in Software Development: Methods, Techniques and Tools


MODEL OR DIE!
2009 EDITION
SOFTWARE MODELLING FOR NEWBIES
UNLOCK THE POWER OF MODELS VIA ABSTRACTION, AUTOMATION & ANALYSIS
ABSTRACT YOURSELF OUT OF EVERY PROBLEM
www.txt2pic.com

Topic: Domain Specific Languages

Juergen Dingel
Feb/March 2021

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

Examples in [Voe13, Kel08]

EGGG ✓ **[Orw00]**

Increasing generality

increasing domain-specifity

[**Orw00**] J. Orwant. EGGG: Automated programming for game generation. IBM Systems Journal 39(3&4):782-794, 2000.

[**Voe13**] M.Voelter. DSL Engineering: Designing, Implementing and Using Domain-Specific Languages. CreateSpace Independent Publishing Platform. 2013.

[**KT08**] S. Kelly and J.-P. Tolvanen. Domain-Specific Modeling: Enabling Full Code Generation. Wiley. 2008

Expressing SW models: Overview (Cont'd)

Domain-specific languages (DSLs)

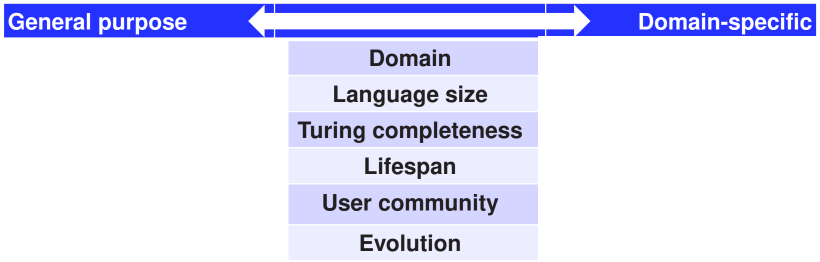
1. Intro and examples (EGGG, CPML, UML-RT)
2. Pros and cons
3. Defining DSLs
   - abstract syntax
     - CFGs in BNF
     - meta models
       - MOF, ECore and OCL
   - concrete syntax
   - semantics
     - Denotational, operational, axiomatic, translational
4. Defining DSLs using UML
   - semantic variation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

Domain-Specific Languages



Productivity
Quality
Analysis capabilities
Communication

Potential for positive impact

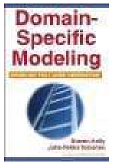
Need for specific expertise
Maintainability
Integratebility

Potential for negative impact

DSLs: Examples

- Not a new idea
  - [MHS05]: BNF (1959), HTML, Latex, make, SQL, VHDL, TXL, ADLs, …
  - EGGG: The Extensible Graphical Game Generator [Orw00]

- 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

[MHS05] Mernik, Heering, Sloane. When and how to develop domain-specific languages. ACM Computing Surveys 37(4):316-344. 2005

DSLs: Examples (Cont'd)

- Web development
  - WebDSL: [Vis08], http://webdsl.org
- Robotics
  - RobotML: http://robotml.github.io
- Train signaling
  - Graphical language and analysis [ECM+08, SHM+12]
- Financial industry
  - RISLA: a DSL for describing financial products (e.g., mortgages) [vDe97]
  - DSLFin'13 Workshop [DSLFin13]
- Healthcare
  - Clinical decision support system [MLN+09]
- Home automation In [Jimenez et al 09]:
  - Home automation system [SJR+11]
- Software development
  - Model transformation (Xtend, Epsilon, ATL, …)
  - Software architecture description languages

DSLs: Examples (Cont'd)

[vDe97] van Deursen, Domain-Specific Languages versus Object-Oriented Languages. 1997

[Vis08] Visser. WebDSL: A Case Study in Domain-Specific Language Engineering. GTTSE. LNCS 5235, 291-373. 2008

[DSLFin13] http://www.dslfin.org/resources.html

[ECM+08] J. Endresen, E. Carlson, T. Moen, K. J. Alme, O. Haugen, G K. Olsen, A. Svendsen. Train control language teaching computers interlocking. Computers in Railways XI. WITPress. 2008. pages 651 - 660.

[MLN+09] Mathe, Ledeczi, Sztipanovits, et al. A Model-Integrated, Guideline-Driven, Clinical Decision-Support System. IEEE Software. 2009

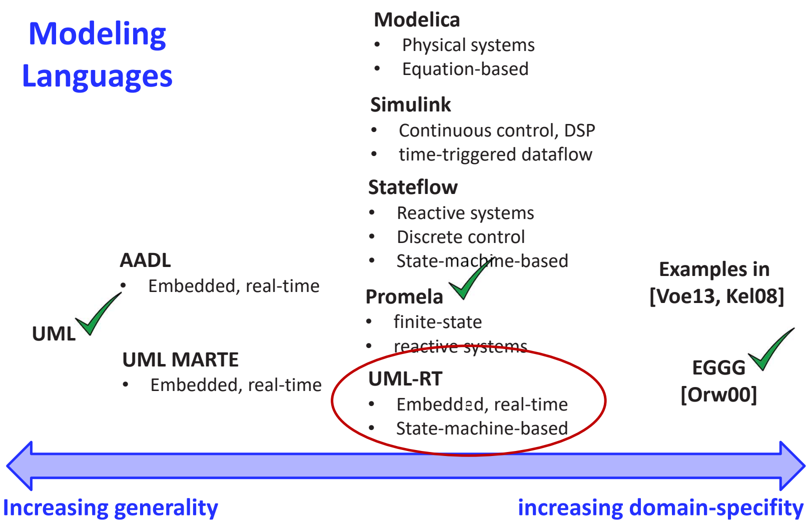
[SHM+12] A. Svendsen, O. Haugen, B.Moeller-Pedersen. Synthesizing Software Models: Generating Train Station Models Automatically. SDL 2011: Integrating System and Software Modeling. LNCS Volume 7083, 2012, pp 38-53.

[SJR+11] P. Sanchez, M. Jimenez, F. Rosique, B. Alvarez, A. Iborra. A framework for developing home automation systems: From requirements to code. JSS 84(6). 2011

DSLs: Examples (Cont'd)

- Real-time embedded
  - UML-RT
    - 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
    - http://www.omgmarte.org/
  - Stateflow/Simulink
  - Esterel/Scade
    - http://www.esterel-technologies.com/products/scade-suite/

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

Examples in [Voe13, Kel08]

EGGG ✓ **[Orw00]**

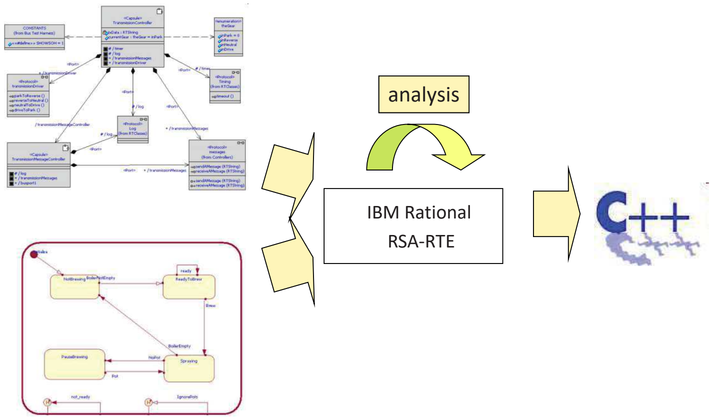
Increasing generality **increasing domain-specifity**

DSML Example: UML-RT

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

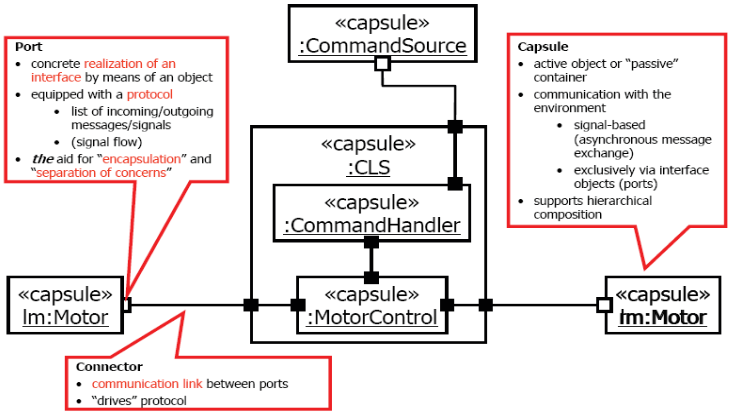
capsule diagram (structure)

state machine diagram (one per capsule) (behaviour)



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"

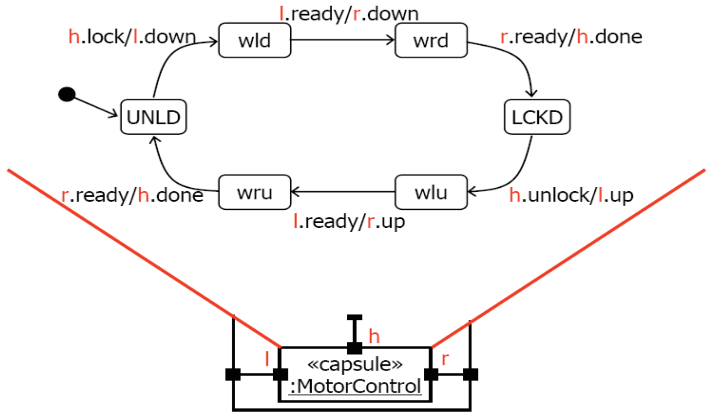
Capsule
- active object or "passive" container
- communication with the environment
  - signal-based (asynchronous message exchange)
  - exclusively via interface objects (ports)
- supports hierarchical composition

Connector
- communication link between ports
- "drives" protocol

February 4, 2003 © Ingolf H. Krueger CSE/CAL·(IT)² 7

DSML Example: UML-RT (Cont’d)

Example: UML-statecharts

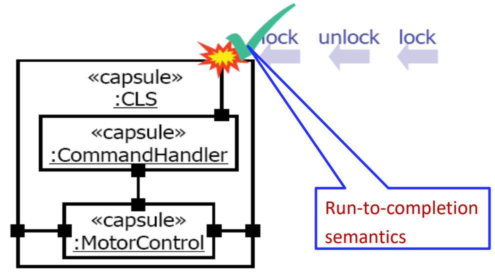


February 4, 2003 © Ingolf H. Krueger CSE/CAL·(IT)² 8

DSML Example: UML-RT (Cont’d)

Signal-Based Communication

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

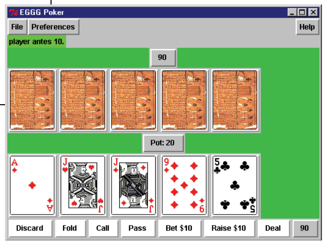


February 4, 2003 © Ingolf H. Krueger CSE/CAL·(IT)² 9

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)
```

automatically generated using EGGG



player antes 10.

[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 communication and reuse

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]

[KT08] S. Kelly and J.-P. Tolvanen. Domain-Specific Modeling: Enabling Full Code Generation. Wiley. 2008

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 with
    - 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

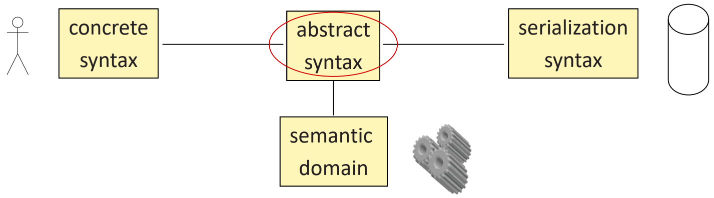
Definition of (domain-specific) languages

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"*)



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, BNF, and parse trees

Regular expressions

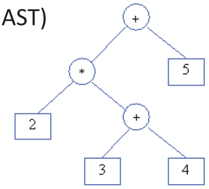
```
<Var>     ::=  <Letter> + (<Letter>)*
<Letter>   ::= a + b + c + … + z + A + B + … + Z
```

BNF

```
<Exp>     ::= <Num> | <Var> | <Exp> <BinOp> <Exp> | <UnOp> <Exp>
<BinOp>    ::= + | - | * | /
<UnOp>    ::= -
<Var>     ::=  <Letter> | <Letter> <Var>
<Letter>   ::= a | b | c | … | z | A | B | … | Z
```

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 | ab <S>
- Let N = {<S>} and T = {a, b}
- Then, L(G) can be characterized in two ways:
  1. $L(G) = \{w \in T^* \mid \text{<S>} \rightarrow w\}$

     where $\rightarrow \subseteq (N \cup T)^* \times (N \cup T)^*$ is the smallest relation satisfying

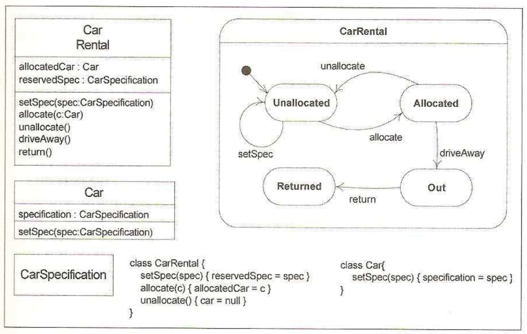
     1. $\text{<S>} \rightarrow ab$ (i.e., $(\text{<S>}, ab) \in \rightarrow$), and
     2. if $\text{<S>} \rightarrow w$, then $\text{<S>} \rightarrow abw$ for all $w \in T^*$
  2. L(G) smallest set $X \subseteq T^*$ such that $X = F(X)$ where

     $F(X) = \{ab\} \cup X \cup \{abw \mid w \in X\}$

     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)

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:



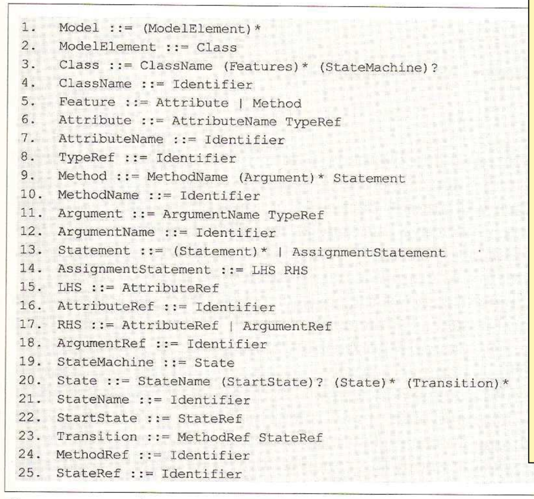
```
class CarRental {
   setSpec(spec) { reservedSpec = spec }
   allocate(c) { allocatedCar = c }
   unallocate() { car = null }
}
```

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

Figure 8.2 Example OSL model

[GS04] J. Greenfield, K. Short. Software Factories. Wiley. 2004

Defining OSL using a BNF



```
Model ::= (ModelElement)*
ModelElement ::= Class
Class ::= ClassName (Features)* (StateMachine)?
ClassName ::= Identifier
Feature ::= Attribute | Method
Attribute ::= AttributeName TypeRef
AttributeName ::= Identifier
TypeRef ::= Identifier
Method ::= MethodName (Argument)* Statement
MethodName ::= Identifier
Argument ::= ArgumentName TypeRef
ArgumentName ::= Identifier
Statement ::= (Statement)* | AssignmentStatement
AssignmentStatement ::= LHS RHS
LHS ::= AttributeRef
AttributeRef ::= Identifier
RHS ::= AttributeRef | ArgumentRef
ArgumentRef ::= Identifier
StateMachine ::= State
State ::= StateName (StartState)? (State)* (Transition)*
StateName ::= Identifier
StartState ::= StateRef
Transition ::= MethodRef StateRef
MethodRef ::= Identifier
StateRef ::= Identifier
```

Figure 8.3 BNF abstract syntax

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

[GS04] J. Greenfield, K. Short. Software Factories. Wiley. 2004

An OSL model as AST

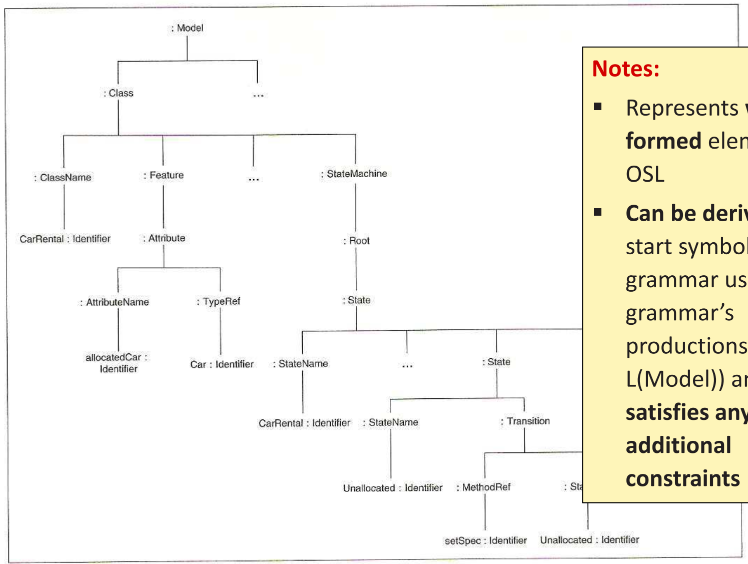


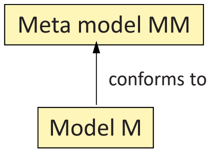
Figure 8.4 AST for car rental model [GS04]

Notes:

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

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 ∈ 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!

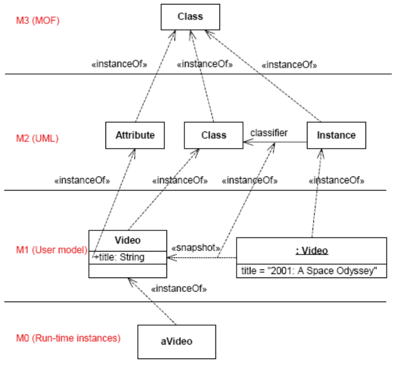


Figure 7.8 - An example of the four-layer metamodel hierarchy

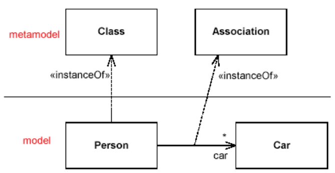
OMG Unified Modeling Language, Infrastructure, Version 2.2. Number: formal/2009-02-04, http://www.omg.org/spec/UML/2.2/Infrastructure. pages 16-19

Meta Object Facility (Cont'd)

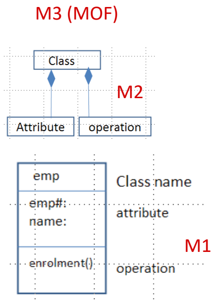
- Example: How is UML defined with MOF?

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

UML2 model/ user model (M1)



or, more precisely



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

Example: Specifying generalization in UML using MOF

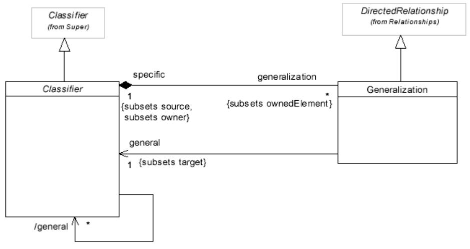


Figure 3-32. The elements defined in the Generalizations package.

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

UML2 model/ user model (M1)

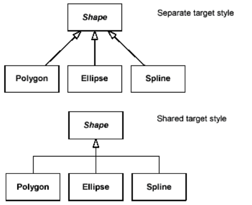


Figure 3-33. Examples of generalizations between classes.

[OMG07] Object Management Group. UML Superstructure specification. Version 2.1.2. formal/2007-11-02. 2007

Excerpt of UML 2.1.2 Metamodel (Class Diagrams)

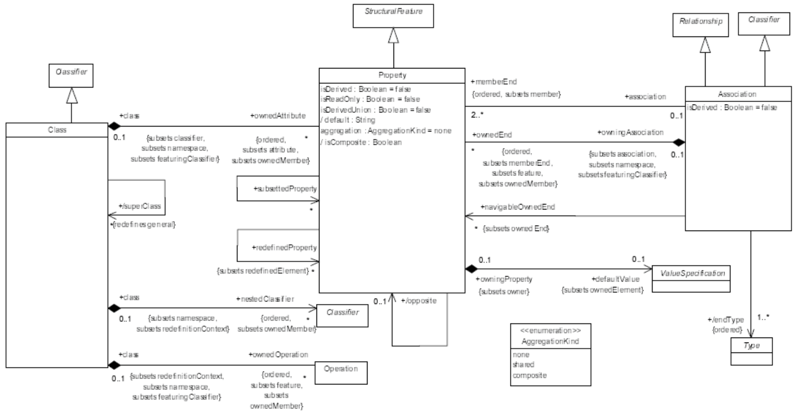
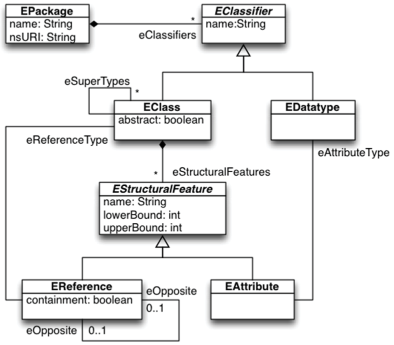

StructuralFeature
Relationship
Classifier
Classifier
Property
isDerived : Boolean = false
isReadOnly : Boolean = false
isDerivedUnion : Boolean = false
/ default : String
aggregation : AggregationKind = none
/ isComposite : Boolean
Association
isDerived : Boolean = false
Class
+memberEnd
{ordered, subsets member}
+association
2..*
0..1
+class
+ownedAttribute
0..1
{subsets classifier, subsets namespace, subsets featuringClassifier}
{ordered, subsets attribute, subsets ownedMember}
*
+ownedEnd
+owningAssociation
*
{ordered, subsets memberEnd, subsets feature, subsets ownedMember}
{subsets association, subsets namespace, subsets featuringClassifier}
0..1
+subsettedProperty
*
+/superClass
*{redefines general}
+navigableOwnedEnd
* {subsets owned End}
+redefinedProperty
{subsets redefinedElement} *
0..1
0..1
+owningProperty
{subsets owner}
+defaultValue
{subsets ownedElement}
ValueSpecification
+class
+nestedClassifier
0..1
{subsets namespace, subsets redefinitionContext}
{ordered, subsets ownedMember}
*
Classifier
0..1
+/opposite
+/endType
{ordered}
1..*
Type
<<enumeration>>
AggregationKind
none
shared
composite
+class
+ownedOperation
Operation
0..1
{subsets redefinitionContext, subsets namespace, subsets featuringClassifier}
{ordered, subsets feature, subsets ownedMember}
*

Figure 7.12 - Classes diagram of the Kernel package

[OMG07] Object Management Group. UML Superstructure specification. Version 2.1.2. formal/2007-11-02. 2007

EMF and ECore

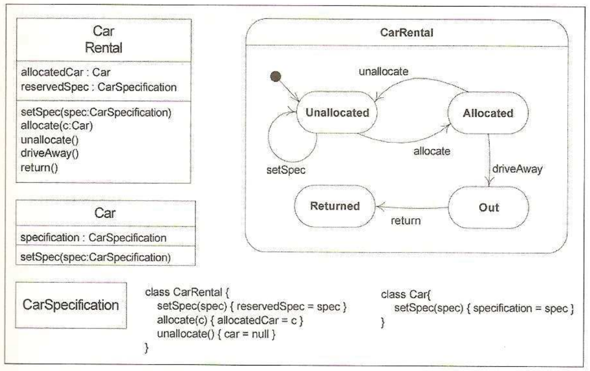
- Eclipse Modeling Framework (EMF)
  - modeling framework and code generation facility for building tools and other application based on a structured data model
  - http://eclipse.org/modeling/emf/
- Ecore
  - Version of MOF in EMF
  - Runtime support
    - change notification
    - persistence w/ XMI 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) { allocatedCar = c }
   unallocate() { car = null }
}
```

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

Figure 8.2 Example OSL model

J. Greenfield, K. Short. Software Factories. Wiley. 2004

A meta model for OSL

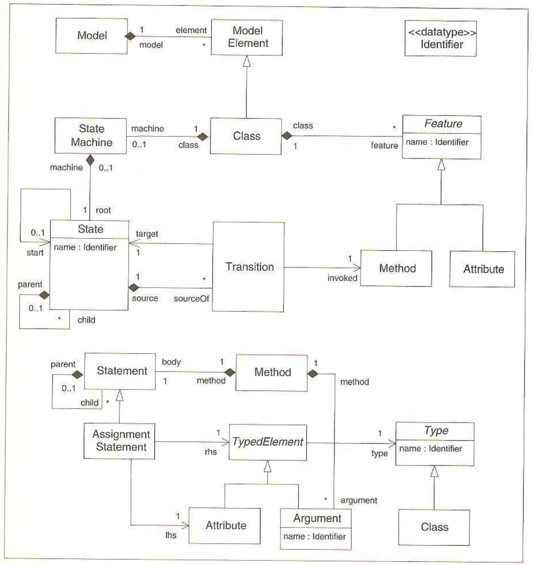


Figure 8.5 Metamodel abstract syntax [GS04]

Notes

- 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

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"*

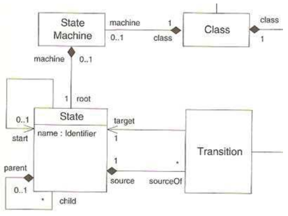
    <u>Transition</u>

    target.root().machine =

    source.root().machine

    where **root()** is

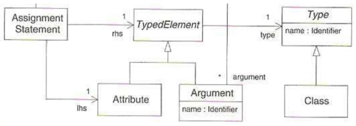
    ```
    State::root() : State {
        if parent = null then self else parent.root()
    }
    ```



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

    <u>AssignmentStatement</u>

    lhs.type = rhs.type



An OSL model as ASG

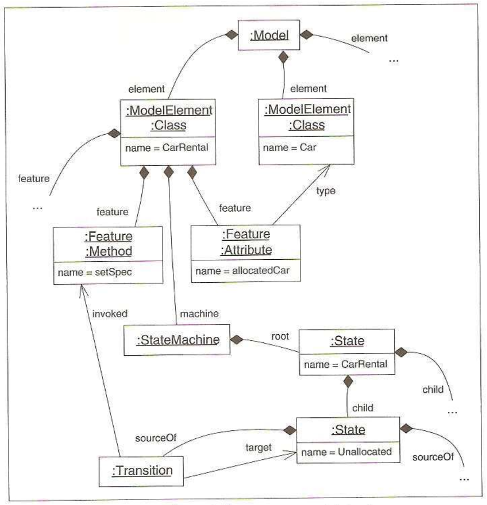
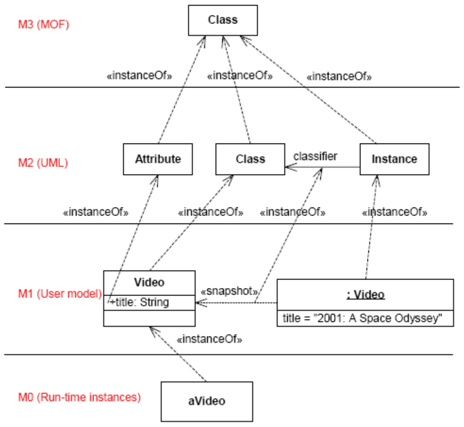


Figure 8.6 Car rental model as metamodel instance

J. Greenfield, K. Short. Software Factories. Wiley. 2004

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

Example of 4-layer meta model hierarchy in UML



UML Specification used to call the UML meta model the “static semantics” of UML

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

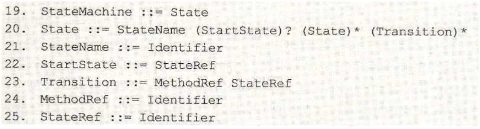
OMG Unified Modeling Language, Infrastructure, Version 2.2. Number: formal/2009-02-04, http://www.omg.org/spec/UML/2.2/Infrastructure. pages 16-19

How exactly does a meta model define a language?

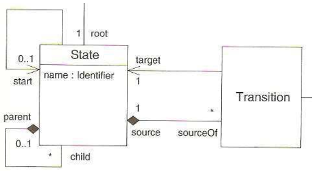
- If language L(MM) is described by some meta model MM, then L(MM) can be thought of as the set of all ASGs of MM:
  - L(MM) = {g | *“g is ASG of MM”}*
  - g is ASG of MM iff
    - g satisfies all the constraints expressed in MM

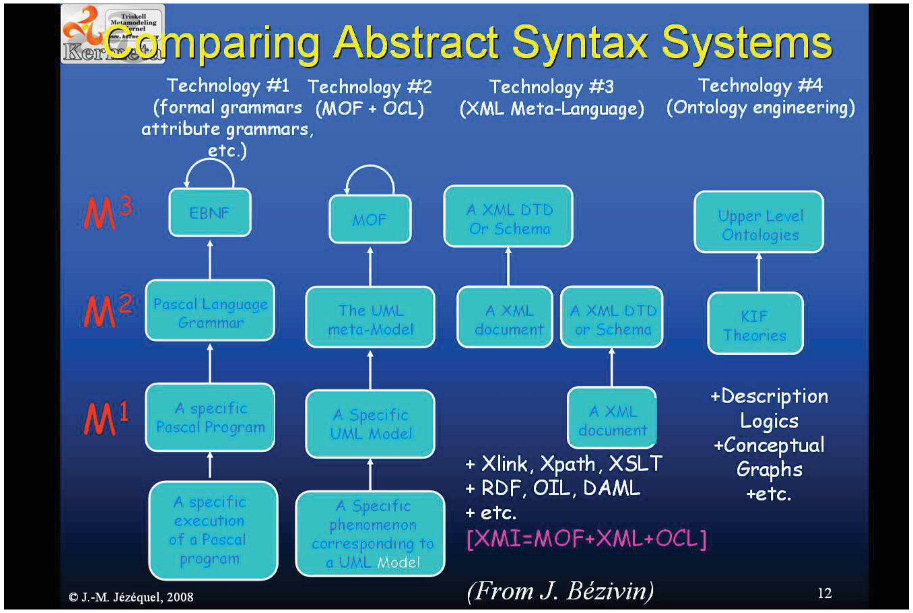
CFGs vs Meta models

CFGs	Meta Models
• 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	• 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



```
StateMachine ::= State
State ::= StateName (StartState)? (State)* (Transition)*
StateName ::= Identifier
StartState ::= StateRef
Transition ::= MethodRef StateRef
MethodRef ::= Identifier
StateRef ::= Identifier
```





Comparing Abstract Syntax Systems

Triskell Metamodeling Kernel
Kermeta
Technology #1 (formal grammars attribute grammars, etc.)
Technology #2 (MOF + OCL)
Technology #3 (XML Meta-Language)
Technology #4 (Ontology engineering)
M3
M2
M1
EBNF
Pascal Language Grammar
A specific Pascal Program
A specific execution of a Pascal program
MOF
The UML meta-Model
A Specific UML Model
A Specific phenomenon corresponding to a UML Model
A XML DTD Or Schema
A XML document
A XML DTD or Schema
A XML document
Upper Level Ontologies
KIF Theories

+ Xlink, Xpath, XSLT
+ RDF, OIL, DAML
+ etc.
[XMI=MOF+XML+OCL]

+Description Logics
+Conceptual Graphs
+etc.

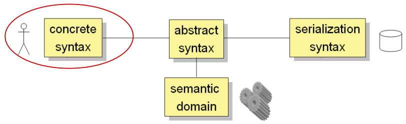
© J.-M. Jézéquel, 2008

(From J. Bézivin)

12

Can We Describe BNF with BNF?

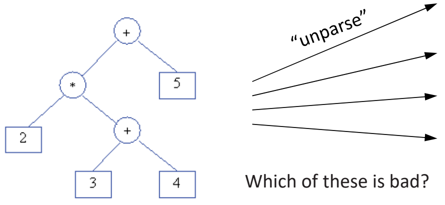
Concrete Syntax



- 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

Concrete Syntax (examples)



~~**"2*3 + 4 + 5"**~~

"(2*(3+4))+5"

"+[*[2,+[3,4]],5]"

**"+
*
2
+
3
4
5"**

Which of these is bad?

Example 2: Graphical concrete syntax

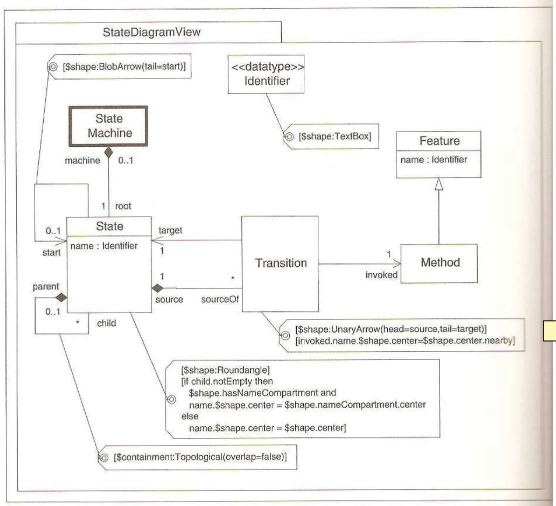

StateDiagramView
[$shape:BlobArrow(tail=start)]
<<datatype>> Identifier
[$shape:TextBox]
State Machine
machine
0..1
1
root
Feature
name : Identifier
0..1
start
State
name : Identifier
target
1
Transition
1
invoked
Method
parent
0..1
1
*
source
sourceOf
*
child
[$shape:UnaryArrow(head=source,tail=target)]
[invoked.name.$shape.center=$shape.center.nearby]
[$shape:Roundangle]
[if child.notEmpty then
$shape.hasNameCompartment and
name.$shape.center = $shape.nameCompartment.center
else
name.$shape.center = $shape.center]
[$containment:Topological(overlap=false)]

Figure 8.8 Annotated metamodel for state diagram view

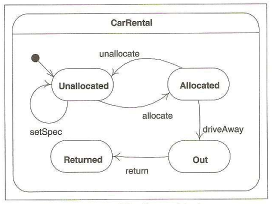


Figure 8.7 A state machine in graphical concrete syntax

J. Greenfield, K. Short. Software Factories. Wiley. 2004

Example 2: Graphical concrete syntax (Cont'd)

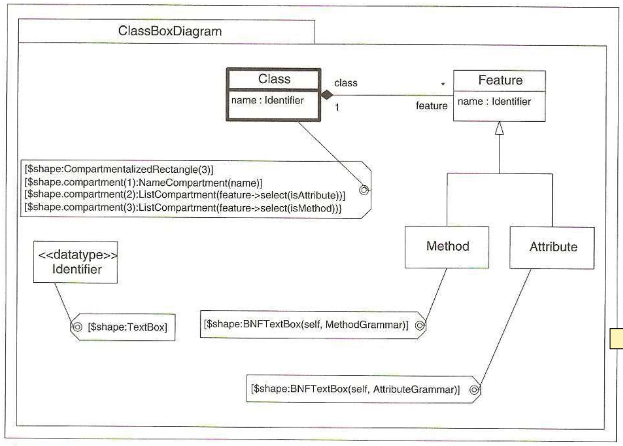

ClassBoxDiagram
Class
name : Identifier
class
1
*
feature
Feature
name : Identifier
[$shape:CompartmentalizedRectangle(3)]
[$shape.compartment(1):NameCompartment(name)]
[$shape.compartment(2):ListCompartment(feature->select(isAttribute))]
[$shape.compartment(3):ListCompartment(feature->select(isMethod))}
Method
Attribute
<<datatype>> Identifier
[$shape:TextBox]
[$shape:BNFTextBox(self, MethodGrammar)]
[$shape:BNFTextBox(self, AttributeGrammar)]

Figure 8.10 Annotated metamodel for class notation

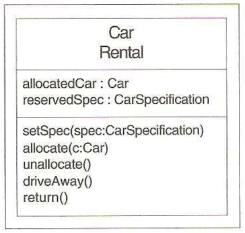
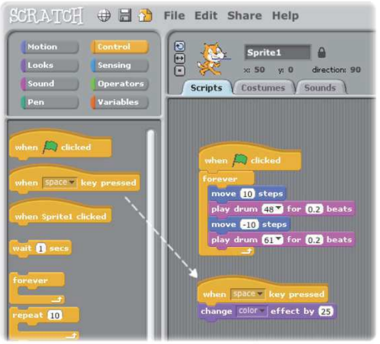


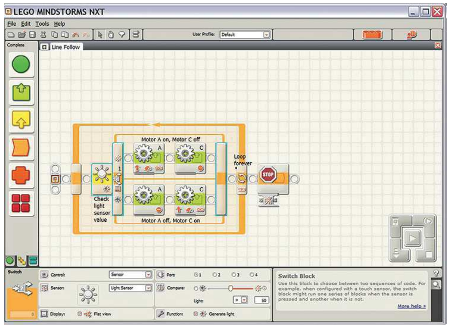
Figure 8.9 Concrete syntax for ar

J. Greenfield, K. Short. Software Factories. Wiley. 2004

Other examples: Graphical concrete syntax



Scratch (http://scratch.mit.edu)



Lego Mindstorms' NXT-G language

How about another dimension?

- UML state machines in Second Life: https://www.youtube.com/watch?v=mkiXRzZ_mJ0
- X3D-UML [MHS08]: https://www.youtube.com/watch?v=gcgQajTXVrA

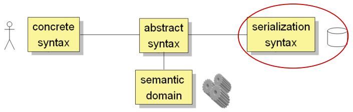
[MHS08] MacIntosh, Hamilton, Schyndel. X3D-UML: 3D UML State Machine Diagrams. MODELS'08. 2008

Abstract and concrete syntax: summary

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

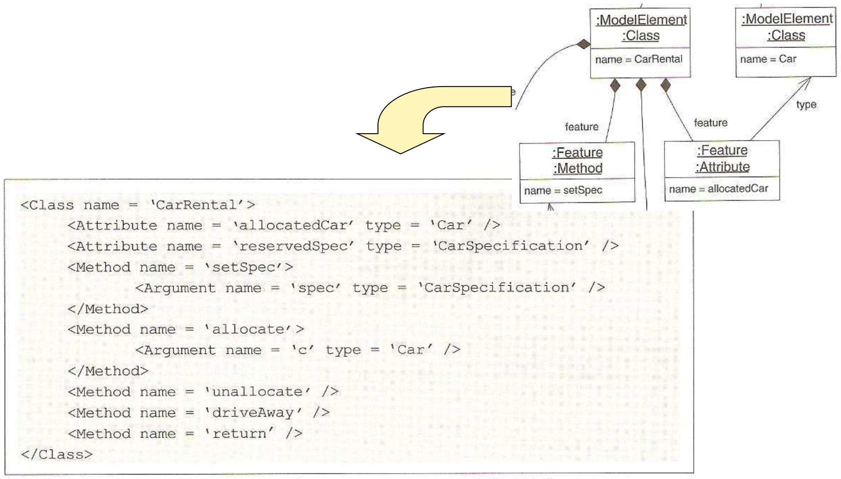
1. Format of abstract syntax constraints:
   - context-free grammars, meta models, OCL
2. Format of concrete syntax constraints:
   - annotations

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
  - Mostly used as interchange format for UML models, but can also be used for serialization of any MOF-based models

Serialization syntax: an example



```
<Class name = 'CarRental'>
      <Attribute name = 'allocatedCar' type = 'Car' />
      <Attribute name = 'reservedSpec' type = 'CarSpecification' />
      <Method name = 'setSpec'>
               <Argument name = 'spec' type = 'CarSpecification' />
      </Method>
      <Method name = 'allocate'>
               <Argument name = 'c' type = 'Car' />
      </Method>
      <Method name = 'unallocate' />
      <Method name = 'driveAway' />
      <Method name = 'return' />
</Class>
```

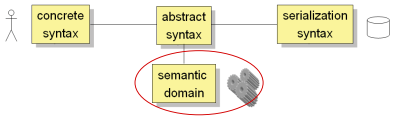
Figure 8.11 XML for ASG fragment of car rental model

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
   - semantics
     - 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

- Most practically relevant
  - Translational

    Meaning of program given by translation (implicit or explicit) to equivalent program in another, known language
  - Operational/interpretative

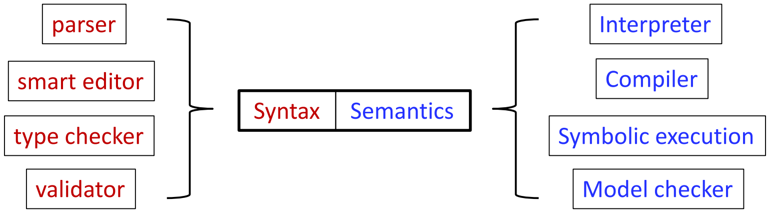
    Meaning of program given by collection of execution rules operating on a formalization of state
    - Execution rules may be implemented in interpreter
- Less practically relevant
  - Denotational

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

    Meaning of program given by logical statements describing effect of program statements on assertions

Implicitly vs explicitly given semantics descriptions

- Implicit:
  - E.g, execution/translation rules deeply embedded, intertwined in interpreter/translator
  - Hard to leverage description for other purposes
- Explicit:
  - E.g., execution/translation rules separated out in processable fashion
  - Easier to use description for generation of supporting tooling ("semantics engineering")



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
   - semantics
     - 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

- Using UML [FGDT06]

  Two customization mechanisms

  1. semantic variation points (see below)
  2. profiles (see below)

- 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

[MSUW04] Mellor, Scott, Uhl, Weise. MDA Distilled: Principles of Model-Driven Architecture. Addison Wesley. 2004.

[FGDT06] France, Ghosh, Dinh-Trong. Model-Driven Development Using UML 2.0: Promises and Pitfalls. IEEE Computer 39(2), Feb. 2006

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]*

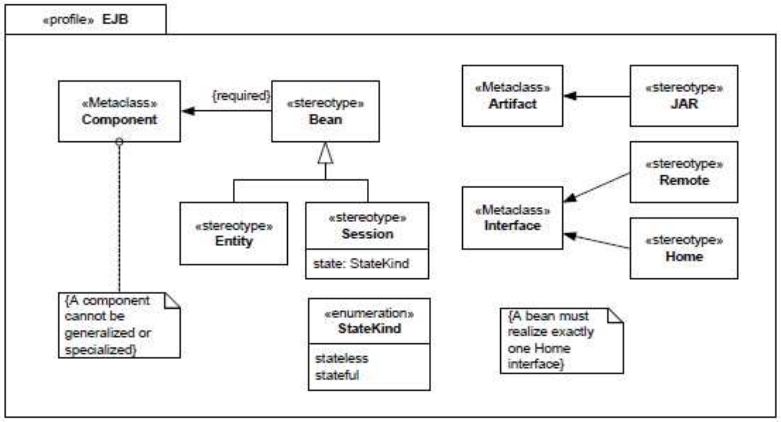
- Small adjustments, not completely new language
- Examples (from UML 2.4.1)
  - *"Precise semantics of shared aggregation varies by application area and modeler" (page 36)*
  - *"The order and way in which part instances in a composite are created is not defined." (page 38)*
  - *"The behavior of an 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)

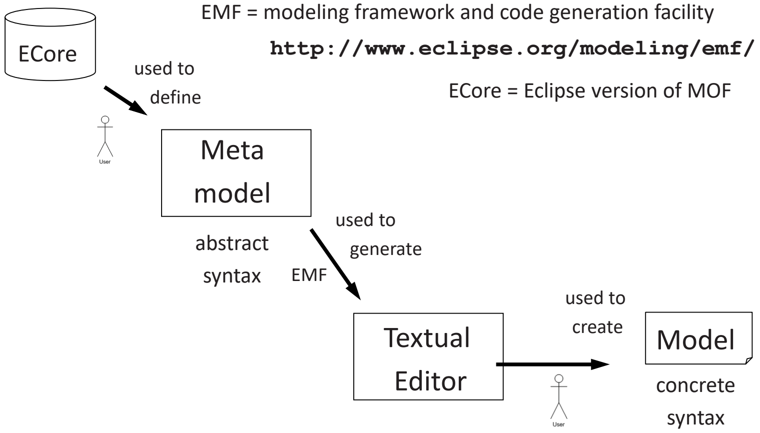
2. Domain-specific languages

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, Sirius
- Xtext [Assignment 3]
- JetBrains Meta Programming System (MPS)
- Spoofax
- MetaEdit+ (MetaCase)
- IBM RSA (UML based)
- Generic Modeling Environment (GME) (Vanderbilt)
- MS Visual Studio
  - Visualization and Modeling SDK (DSL Tools)
  - https://code.msdn.microsoft.com/Visualization-and-Modeling-313535db

EMF + X

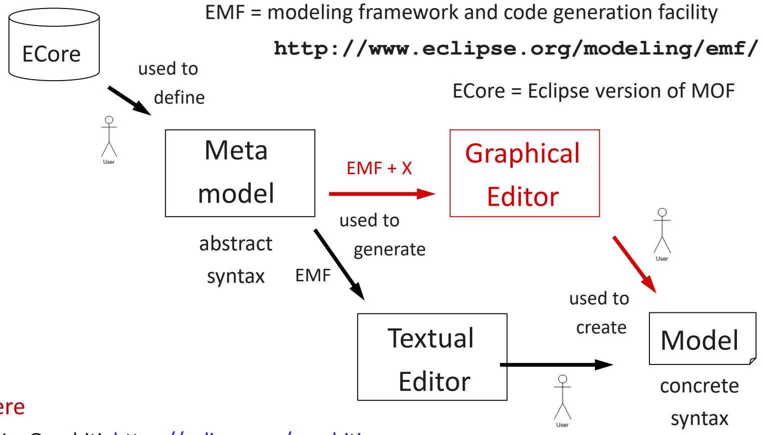


EMF = modeling framework and code generation facility

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

ECore = Eclipse version of MOF

EMF + X



EMF = modeling framework and code generation facility

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

ECore = Eclipse version of MOF

where

- X = Graphiti, https://eclipse.org/graphiti, or
- X = GMF, http://eclipse.org/modeling/gmp
- X = Sirius, https://www.eclipse.org/sirius

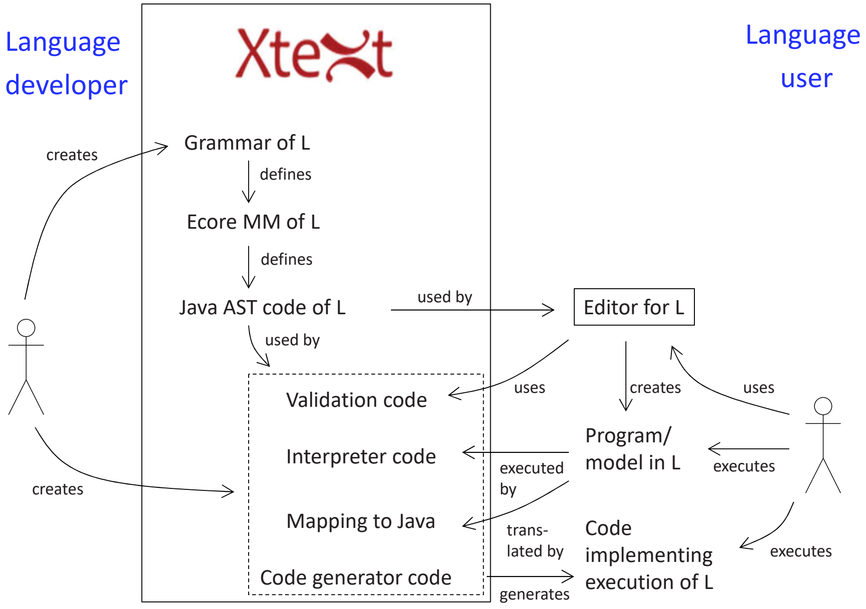
Efforts related to DSLs

- Software Factories (Microsoft, [GS04])
- Intensional Programming ([Sim01], [ADKdMRS98])
- Language-oriented programming ([MPS09], [LOP09])
- Language workbench ([Fow09])
- Language Workbench Challenge 2016
  - https://2016.splashcon.org/track/lwc2016

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]



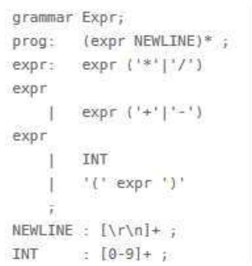
Xtext: Supporting technology

- 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

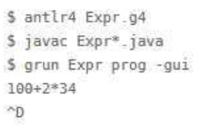
ANTLR

From www.antlr.org:

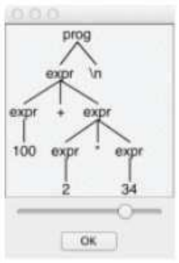
> *"ANTLR (Another Tool for Language Recognition) is a powerful parser generator for reading, processing, executing, or translating structured text or binary files"*



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



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



Xtend

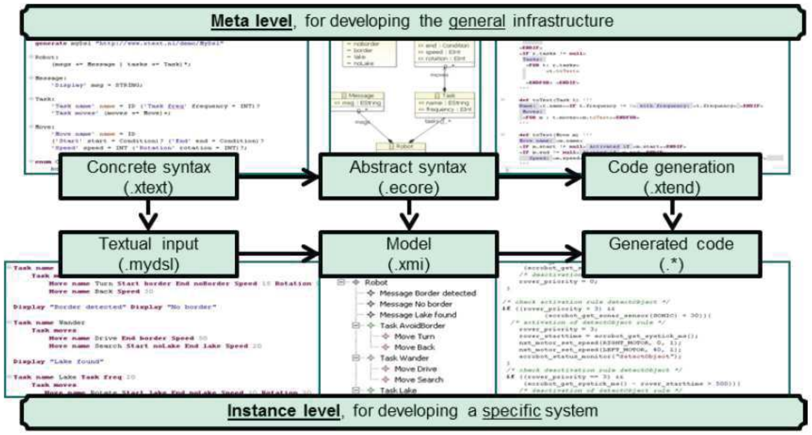
From eclipse.org/xtend:

> *"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

Overview of key Xtext artifacts



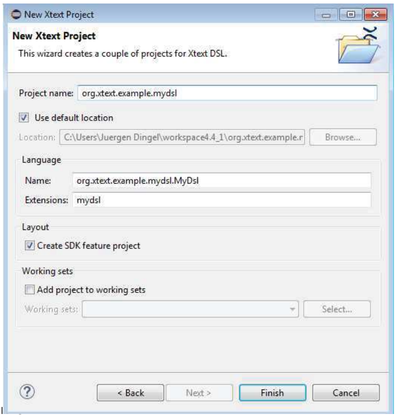
From: A. Mooiji, J. Hooman. Creating a Domain Specific Language (DSL) with Xtext. Version 2.14. Available at http://www.cs.kun.nl/J.Hooman/DSL/

Using Xtext

0. Installation instructions etc on Assignment 4 page
1. Create Xtext project

   In Package Explorer: "New | Project …" then "Xtext Project"

FREE DEMO TRY IT NOW!



New Xtext Project

New Xtext Project

This wizard creates a couple of projects for Xtext DSL.

Project name: org.xtext.example.mydsl

☑ Use default location

Location: C:\Users\Juergen Dingel\workspace4.4_1\org.xtext.example.r

Language

Name: org.xtext.example.mydsl.MyDsl

Extensions: mydsl

Layout

☑ Create SDK feature project

Working sets

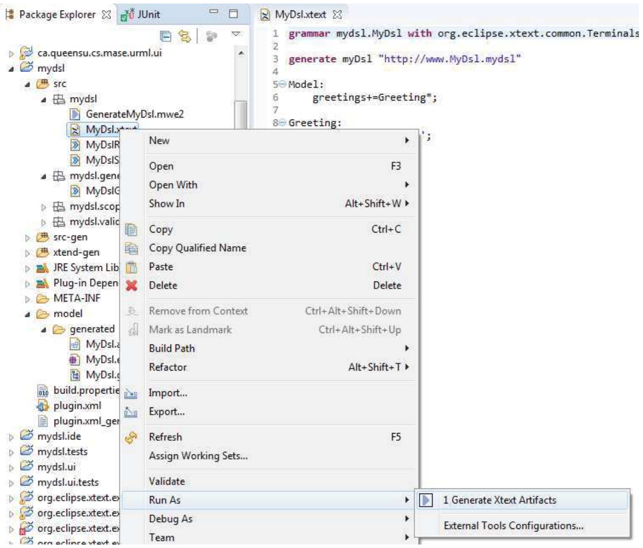
☐ Add project to working sets

Working sets:

Using Xtext (Cont'd)

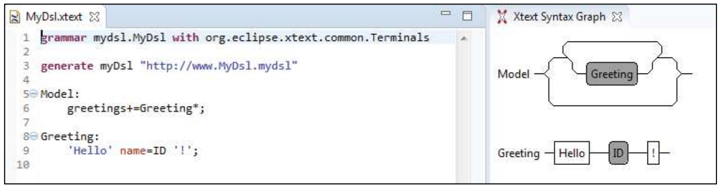
2. Create grammar .xtext in folder "src/<project name>"
3. Generate Xtext artifacts

- in "src-gen" folder: .java
- in "model/generated" folder: .ecore, .genmodel

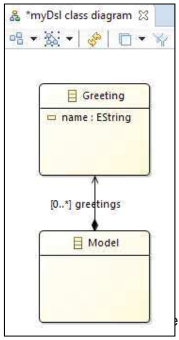


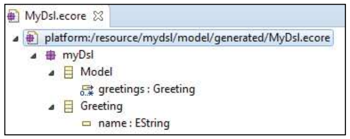
```
grammar mydsl.MyDsl with org.eclipse.xtext.common.Terminals

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

Model:
    greetings+=Greeting*;

Greeting:
```

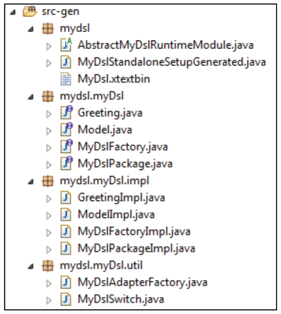
Using Xtext (Cont'd)



```
grammar mydsl.MyDsl with org.eclipse.xtext.common.Terminals

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

Model:
    greetings+=Greeting*;

Greeting:
    'Hello' name=ID '!';
```



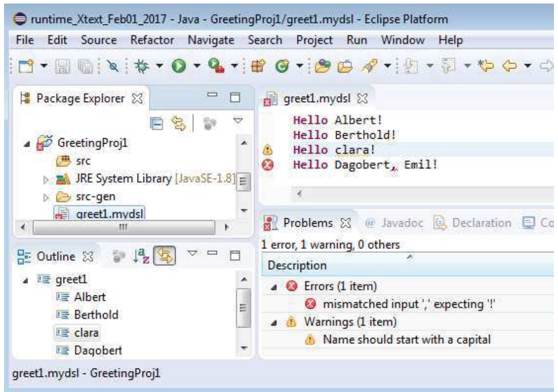




Using Xtext (Cont'd)

4. Start editor

   Right-click project, "Run As | Eclipse Application"
5. Create new Java project
6. Input text, validate, etc
7. Inspect generated output
8. Run generated code



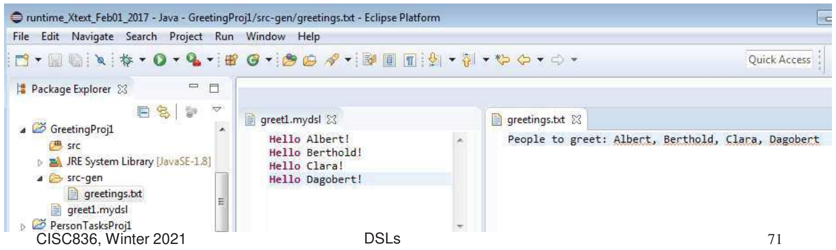
runtime_Xtext_Feb01_2017 - Java - GreetingProj1/greet1.mydsl - Eclipse Platform

```
Hello Albert!
Hello Berthold!
Hello clara!
Hello Dagobert, Emil!
```

1 error, 1 warning, 0 others

Errors (1 item)
mismatched input ',' expecting '!'
Warnings (1 item)
Name should start with a capital

greet1.mydsl - GreetingProj1



runtime_Xtext_Feb01_2017 - Java - GreetingProj1/src-gen/greetings.txt - Eclipse Platform

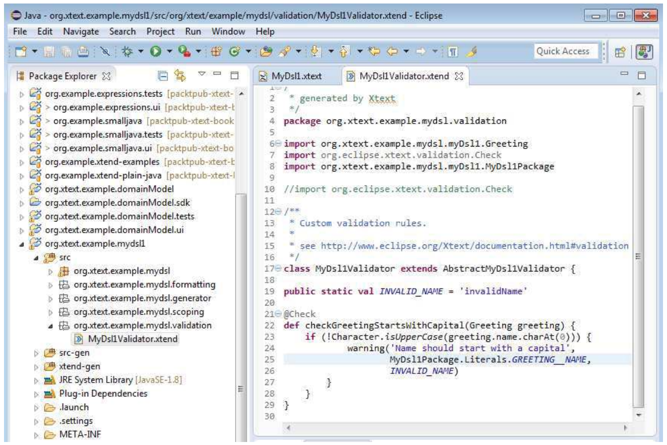
```
Hello Albert!
Hello Berthold!
Hello Clara!
Hello Dagobert!
```

```
People to greet: Albert, Berthold, Clara, Dagobert
```

Using Xtext (Cont'd)

6. Implement custom validation rules

   In folder "src/<project name>/validation/<language name>.xtend"



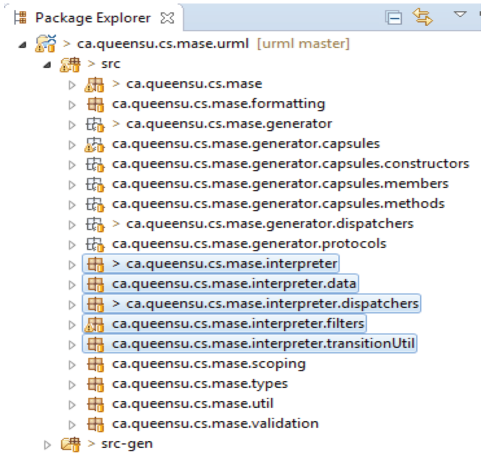
Java - org.xtext.example.mydsl1/src/org/xtext/example/mydsl/validation/MyDsl1Validator.xtend - Eclipse

```
 * generated by Xtext
 */
package org.xtext.example.mydsl.validation

import org.xtext.example.mydsl.myDsl1.Greeting
import org.eclipse.xtext.validation.Check
import org.xtext.example.mydsl.myDsl1.MyDsl1Package

//import org.eclipse.xtext.validation.Check

/**
 * Custom validation rules.
 *
 * see http://www.eclipse.org/Xtext/documentation.html#validation
 */
class MyDsl1Validator extends AbstractMyDsl1Validator {

public static val INVALID_NAME = 'invalidName'

@Check
def checkGreetingStartsWithCapital(Greeting greeting) {
    if (!Character.isUpperCase(greeting.name.charAt(0))) {
            warning('Name should start with a capital',
                    MyDsl1Package.Literals.GREETING__NAME,
                    INVALID_NAME)
        }
    }
}
```

File
Edit
Navigate
Search
Project
Run
Window
Help
Quick Access
Package Explorer
org.example.expressions.tests [packtpub-xtext-
> org.example.expressions.ui [packtpub-xtext-b
> org.example.smalljava [packtpub-xtext-book
> org.example.smalljava.tests [packtpub-xtext-
> org.example.smalljava.ui [packtpub-xtext-bo
org.example.xtend-examples [packtpub-xtext-b
org.example.xtend-plain-java [packtpub-xtext-
org.xtext.example.domainModel
org.xtext.example.domainModel.sdk
org.xtext.example.domainModel.tests
org.xtext.example.domainModel.ui
org.xtext.example.mydsl1
src
org.xtext.example.mydsl
org.xtext.example.mydsl.formatting
org.xtext.example.mydsl.generator
org.xtext.example.mydsl.scoping
org.xtext.example.mydsl.validation
MyDsl1Validator.xtend
src-gen
xtend-gen
JRE System Library [JavaSE-1.8]
Plug-in Dependencies
.launch
.settings
META-INF
MyDsl1.xtext
MyDsl1Validator.xtend

Using Xtext (Cont'd)

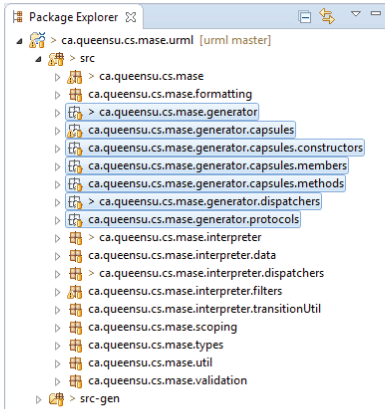
7. Implement interpreter
   - in "src/<project name>/interpreter"



```
Package Explorer
◢ > ca.queensu.cs.mase.urml [urml master]
  ◢ > src
    ▷ > ca.queensu.cs.mase
    ▷ ca.queensu.cs.mase.formatting
    ▷ > ca.queensu.cs.mase.generator
    ▷ ca.queensu.cs.mase.generator.capsules
    ▷ ca.queensu.cs.mase.generator.capsules.constructors
    ▷ ca.queensu.cs.mase.generator.capsules.members
    ▷ ca.queensu.cs.mase.generator.capsules.methods
    ▷ > ca.queensu.cs.mase.generator.dispatchers
    ▷ ca.queensu.cs.mase.generator.protocols
    ▷ > ca.queensu.cs.mase.interpreter
    ▷ ca.queensu.cs.mase.interpreter.data
    ▷ > ca.queensu.cs.mase.interpreter.dispatchers
    ▷ ca.queensu.cs.mase.interpreter.filters
    ▷ ca.queensu.cs.mase.interpreter.transitionUtil
    ▷ ca.queensu.cs.mase.scoping
    ▷ ca.queensu.cs.mase.types
    ▷ ca.queensu.cs.mase.util
    ▷ ca.queensu.cs.mase.validation
  ▷ > src-gen
```

Using Xtext (Cont'd)

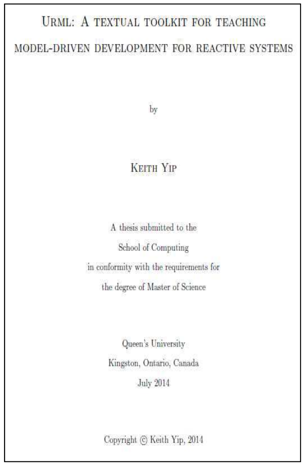
8. Implement code generator
   - in "src/<project name>/generator"
   - implement "doGenerate" and "compile" using "filter"
   - integrate into Eclipse build mechanism
   - allow for invocation from command line



```
Package Explorer
◢ > ca.queensu.cs.mase.urml [urml master]
  ◢ > src
    ▷ > ca.queensu.cs.mase
    ▷ ca.queensu.cs.mase.formatting
    ▷ > ca.queensu.cs.mase.generator
    ▷ ca.queensu.cs.mase.generator.capsules
    ▷ ca.queensu.cs.mase.generator.capsules.constructors
    ▷ ca.queensu.cs.mase.generator.capsules.members
    ▷ ca.queensu.cs.mase.generator.capsules.methods
    ▷ > ca.queensu.cs.mase.generator.dispatchers
    ▷ ca.queensu.cs.mase.generator.protocols
    ▷ > ca.queensu.cs.mase.interpreter
    ▷ ca.queensu.cs.mase.interpreter.data
    ▷ > ca.queensu.cs.mase.interpreter.dispatchers
    ▷ ca.queensu.cs.mase.interpreter.filters
    ▷ ca.queensu.cs.mase.interpreter.transitionUtil
    ▷ ca.queensu.cs.mase.scoping
    ▷ ca.queensu.cs.mase.types
    ▷ ca.queensu.cs.mase.util
    ▷ ca.queensu.cs.mase.validation
  ▷ > src-gen
```

A4: 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



URML: A TEXTUAL TOOLKIT FOR TEACHING MODEL-DRIVEN DEVELOPMENT FOR REACTIVE SYSTEMS

by

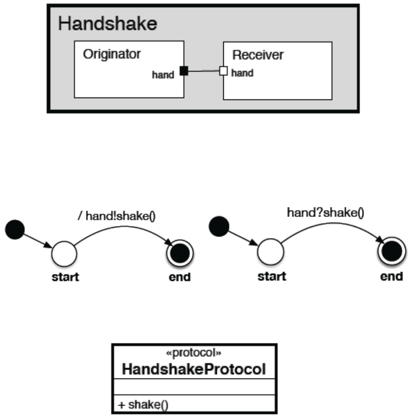
KEITH YIP

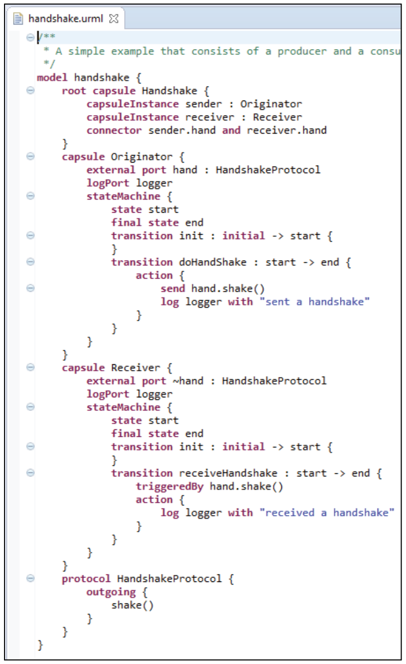
A thesis submitted to the
School of Computing
in conformity with the requirements for
the degree of Master of Science

Queen's University
Kingston, Ontario, Canada
July 2014

Copyright © Keith Yip, 2014

A4: Urml (Cont'd)

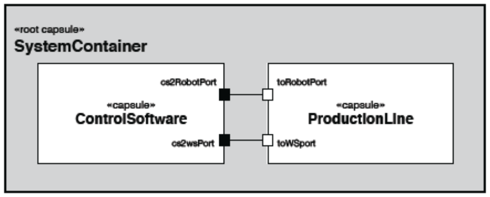


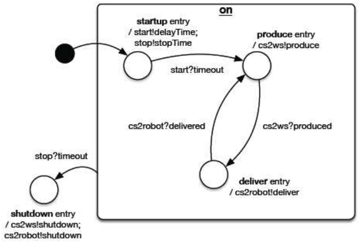


handshake.urml

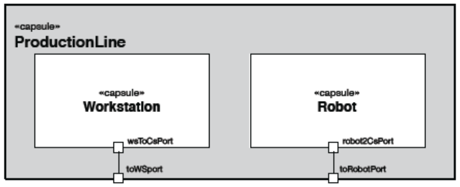
```
/**
 * A simple example that consists of a producer and a consu
 */
model handshake {
    root capsule Handshake {
        capsuleInstance sender : Originator
        capsuleInstance receiver : Receiver
        connector sender.hand and receiver.hand
    }
    capsule Originator {
        external 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()
        }
    }
}
```

A4: Urml (Cont’d)

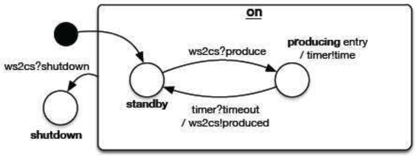


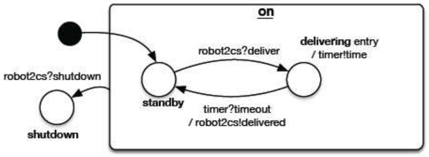


«protocol»
WorkstationProtocol
+ widgetProduced()
+ produceWidget()
+ shutdown()

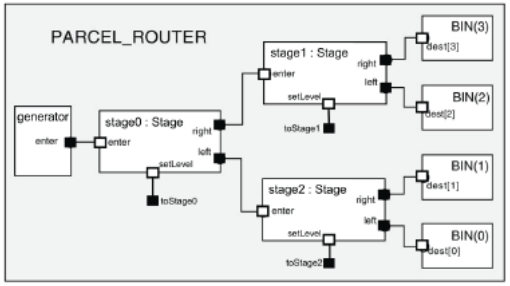


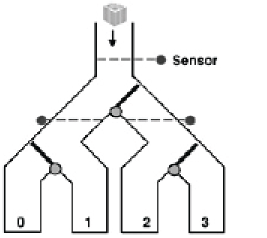
«protocol»
RobotProtocol
+ widgetDelivered()
+ deliverWidget()
+ shutdown()

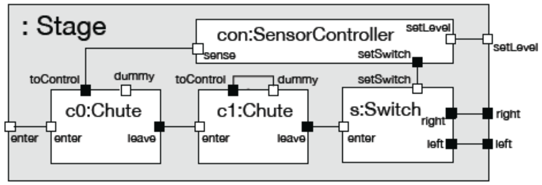




A4: Urml (Cont’d)


PARCEL_ROUTER
generator
enter
stage0 : Stage
enter
right
left
setLevel
toStage0
stage1 : Stage
enter
right
left
setLevel
toStage1
stage2 : Stage
enter
right
left
setLevel
toStage2
BIN(3)
dest[3]
BIN(2)
dest[2]
BIN(1)
dest[1]
BIN(0)
dest[0]


Sensor
0
1
2
3


: Stage
con:SensorController
setLevel
sense
setSwitch
setLevel
toControl
dummy
c0:Chute
enter
leave
enter
toControl
dummy
c1:Chute
enter
leave
setSwitch
s:Switch
enter
right
left
right
left