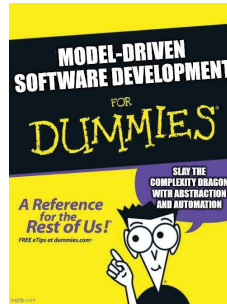


CISC 844: Models in Software Development: Methods, Techniques and Tools



Topic: Domain Specific Languages

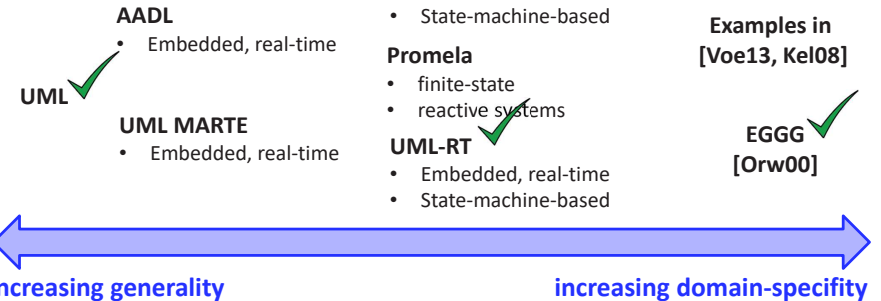
Juergen Dingel
Feb/March 2026

CISC844, Winter 2026

DSLs

1

Modeling Languages



[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

CISC844, Winter 2026

DSLs

2

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. DSL tools
 - EMF, GMF, Graphiti, **Xtext**

CISC844, Winter 2026

DSLs

3

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. DSL tools
 - EMF, GMF, Graphiti, Xtext

CISC844, Winter 2026

DSLs

4

DSLs: Examples

Not a new idea

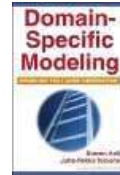
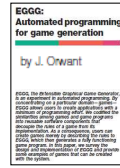
- [MHS05]: BNF (1959), HTML, Latex, make, SQL, VHDL, TXL, ADLs, ...
- EGGG: The Extensible Graphical Game Generator [Orw00]

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.dsifin.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

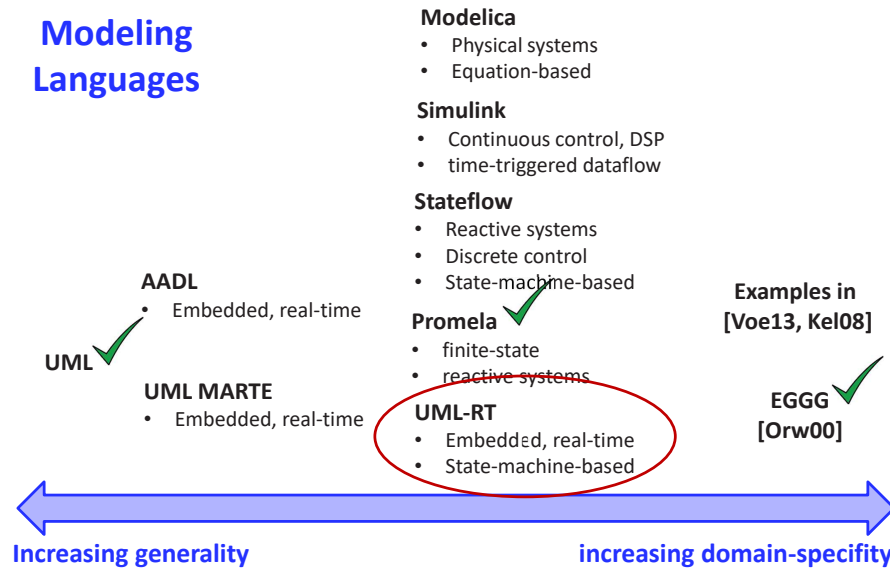
- UML profile for Modeling and Analysis of Real-time and Embedded systems
- supports performance analysis
- tools: Papyrus
- <http://www.omgmarTE.org/>

Stateflow/Simulink

Esterel/Scade

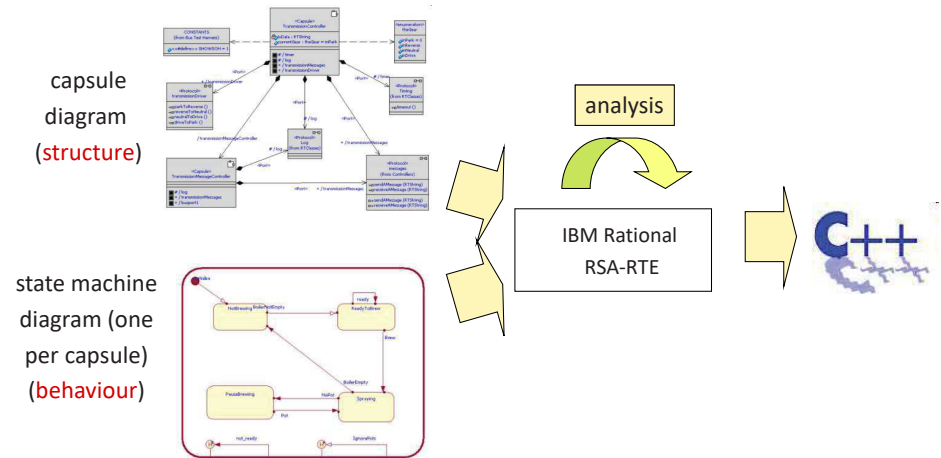
- <http://www.esterel-technologies.com/products/scade-suite/>

Modeling Languages



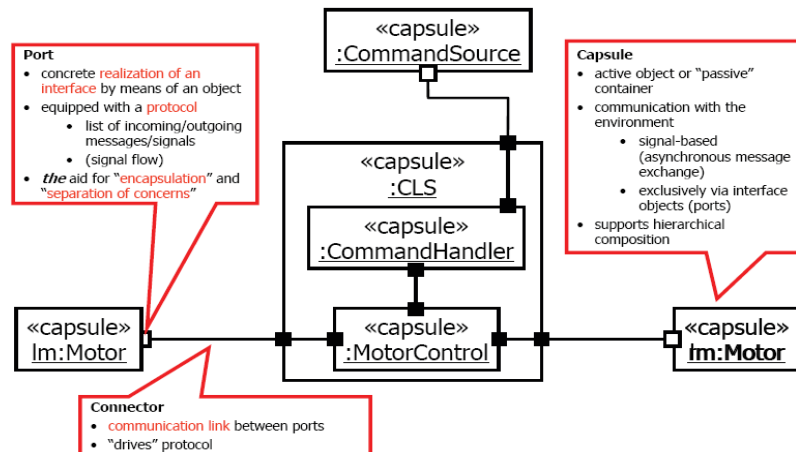
DSML Example: UML-RT

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



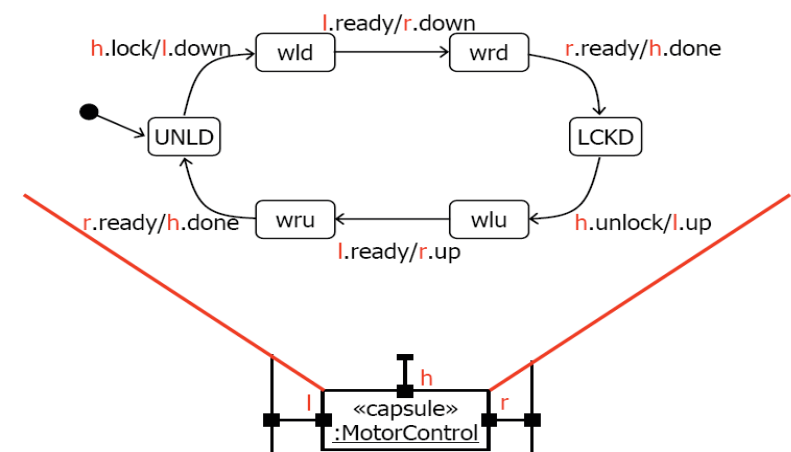
DSML Example: UML-RT (Cont'd)

Hierarchical Composition in UML-RT



DSML Example: UML-RT (Cont'd)

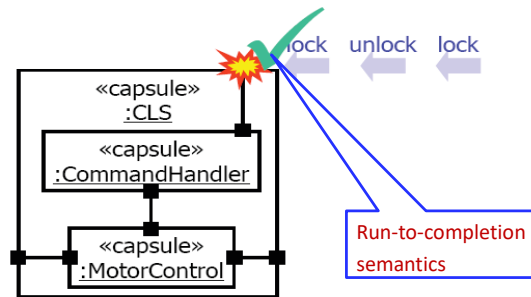
Example: UML-statecharts



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

CISC844, Winter 2026

DSLs

13

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) 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



[Orw00]

CISC844, Winter 2026

DSLs

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

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

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]

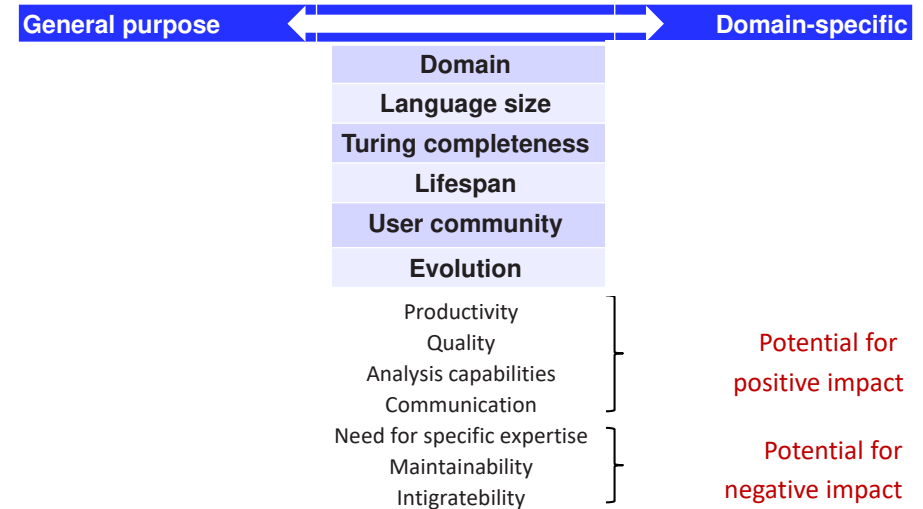
[KT08] S. Kelly and J.-P. Tolvanen. Domain-Specific Modeling: Enabling Full Code Generation. Wiley. 2008

CISC844, Winter 2026

DSLs

17

GPLs vs DSLs



CISC844, Winter 2026

DSLs

18

Expressing SW models: Overview (Part 2)

2. Domain-specific languages

- Intro and examples (Risla, EGGG, CPML, UML-RT)
- 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. DSL tools

- EMF, GMF, Graphiti, Xtext

CISC844, Winter 2026

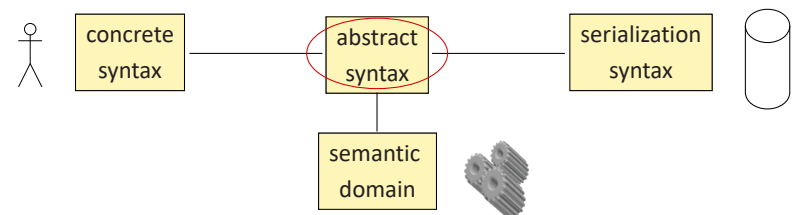
DSLs

19

Definition of (domain-specific) languages

A DSL is a 7-tuple:

- abstract syntax
 - concrete syntax
 - abstract-to-concrete-syntax mapping
 - serialization syntax
 - abstract-to-serialization mapping
 - semantic domain
 - abstract-to-semantic mapping
- syntax (a.k.a., "static semantics")
- semantics (a.k.a., "dynamic semantics")



CISC844, Winter 2026

DSLs

20

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”)
 - 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)
- 2 ways to define abstract syntax:
 1. Context-free grammars (CFGs)
 - expressed using (Extended) Backus-Naur Form (BNF)
 - but often also contains elements of concrete syntax
 2. Meta models
 - expressed as [class diagrams](#)

CISC844, Winter 2026

DSLs

21

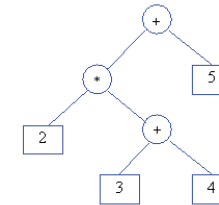
BNF and parse trees

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?

CISC844, Winter 2026

DSLs

22

How exactly does a BNF define a language?

Example:

- Consider the CFG G

<S> ::= ab ab <S>
--
- Here:
 - Grammar **G = (T, N, start, R)**
 - Non-terminal symbols **N = {<S>}**
 - Start symbol **start = <S>**
 - Terminal symbols **T = {a, b}**
 - Rules **R = {(<S>, ab), (<S>, ab<S>)}**
- Then, **L(G) = {w ∈ T* | start →* w}** where $\rightarrow \subseteq (NUT)^* \times (NUT)^*$ is the smallest relation satisfying
 1. $\langle S \rangle \rightarrow ab$ (i.e., $(\langle S \rangle, ab) \in \rightarrow$), and
 2. if $\langle S \rangle \rightarrow w$, then $\langle S \rangle \rightarrow abw$ for all $w \in T^*$

CISC844, Winter 2026

DSLs

23

Ambiguity

Example:

- Consider the CFG G

<S> ::= ab ab <S>
--
- G is **unambiguous**, i.e., every $w \in L(G)$ has exactly one derivation: **start** → ... → w
- Here's an **ambiguous** version G' with $L(G') = L(G)$

<S> ::= ab ab <S> <S> ab

- Or

<Exp>	::= <Num> <Exp> <BinOp> <Exp>
<Num>	::= 0 1 ... 9
<BinOp>	::= + - * /

CISC844, Winter 2026

DSLs

24

Examples of the use of CFGs to define abstract syntax

1. Java

- www.cs.au.dk/~amoeller/RegAut/JavaBNF.html
- Mixes abstract and concrete syntax

2. ANTLR's grammar repo

- lab.antlr.org and github.com/antlr/grammars-v4
- Java: github.com/antlr/grammars-v4/tree/master/java/java
 - Lexer (201 LoC): from characters to tokens
 - Parser (640): from tokens to ASTs
- C++:
 - Lexer (238), parser (638)
 - Context-free? stackoverflow.com/questions/14589346/is-c-context-free-or-context-sensitive

3. Natural language

- Natural Language Toolkit (NLTK): www.nltk.org/howto/grammar.html

4. OSL

Example 4: OSL

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

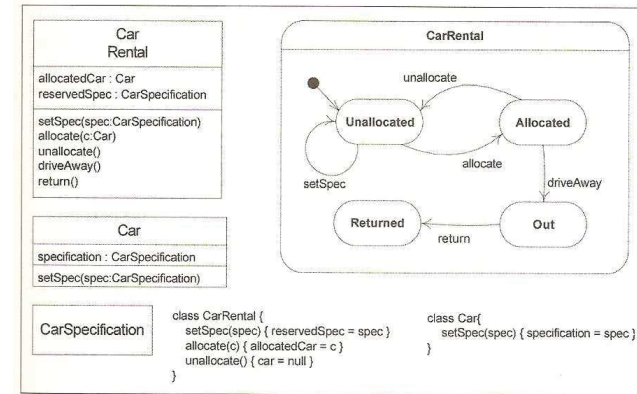


Figure 8.2 Example OSL model

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

DSLs

25

CISC844, Winter 2026

DSLs

26

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

```

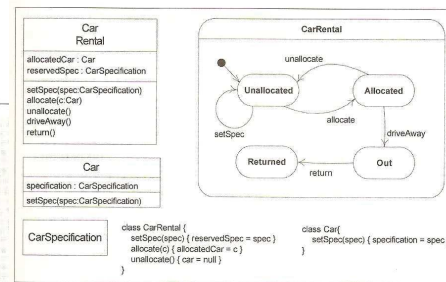


Figure 8.2 Example OSL model

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

An OSL model as AST

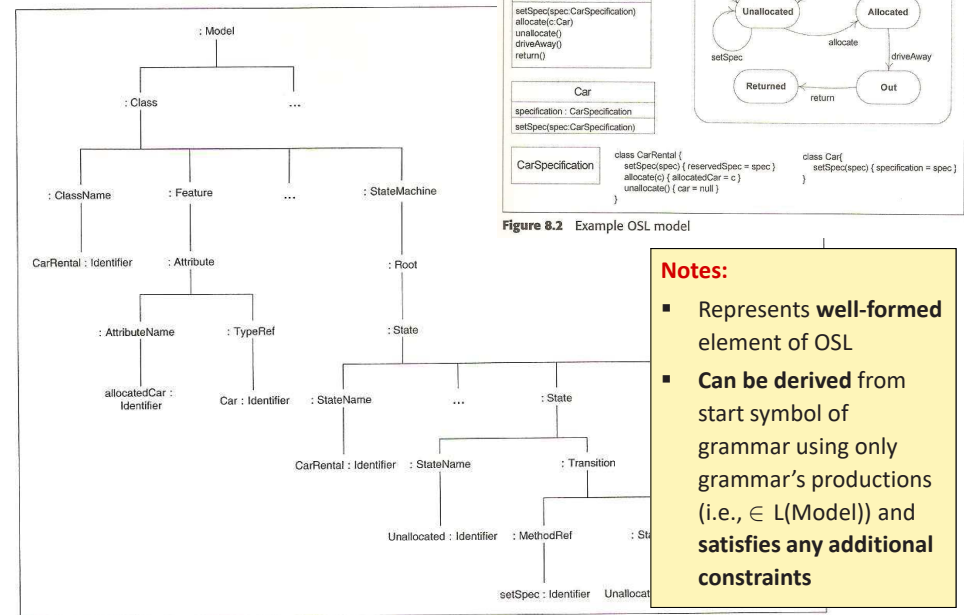


Figure 8.2 Example OSL model

Notes:

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

Figure 8.3 BNF abstract syntax

[GS04] J. Greenfield, K. Short. Software Factories. Wiley. 2004
CISC844, Winter 2026 DSLs

Figure 8.4 AST for car rental model [GS04]

28

Examples of abstract syntax defined using meta models (class diagrams)

1. OSL
2. General purpose programming languages (GPLs)
 - Java, C, C#
3. UML
 - State machines, sequence diagrams
4. DSLs
 - R4: Model-Driven Engineering for Augmented Reality
 - https://www.iot.fm/issues/issue_2023_02/article7.pdf
 - R9: A DSL for Testing LLMs for Fairness and Bias
 - <https://dl.acm.org/doi/10.1145/3640310.3674093>

Example 1: OSL

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

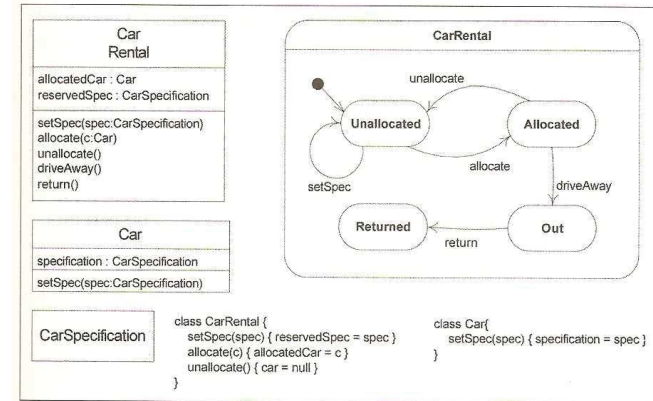


Figure 8.2 Example OSL model

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

A meta model for OSL

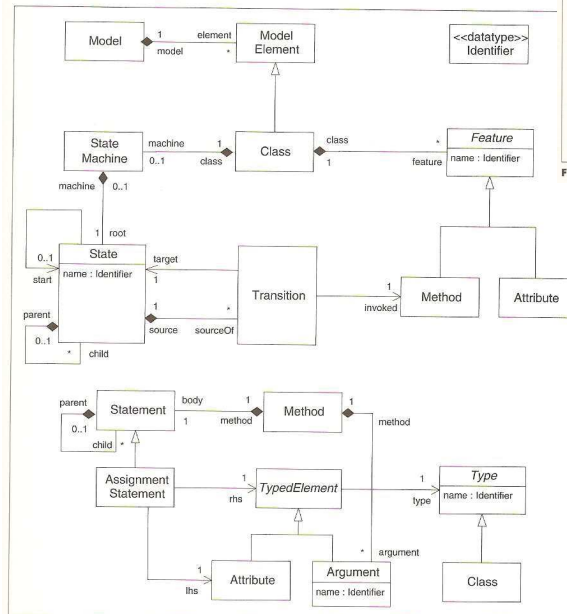


Figure 8.5 Metamodel abstract syntax [GS04]

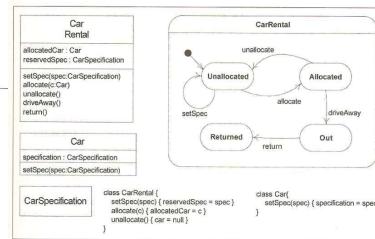


Figure 8.2 Example OSL model

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”

Transition

target.root().machine = source.root().machine

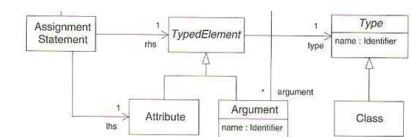
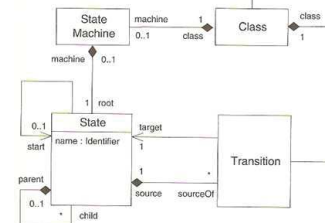
where root() is

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

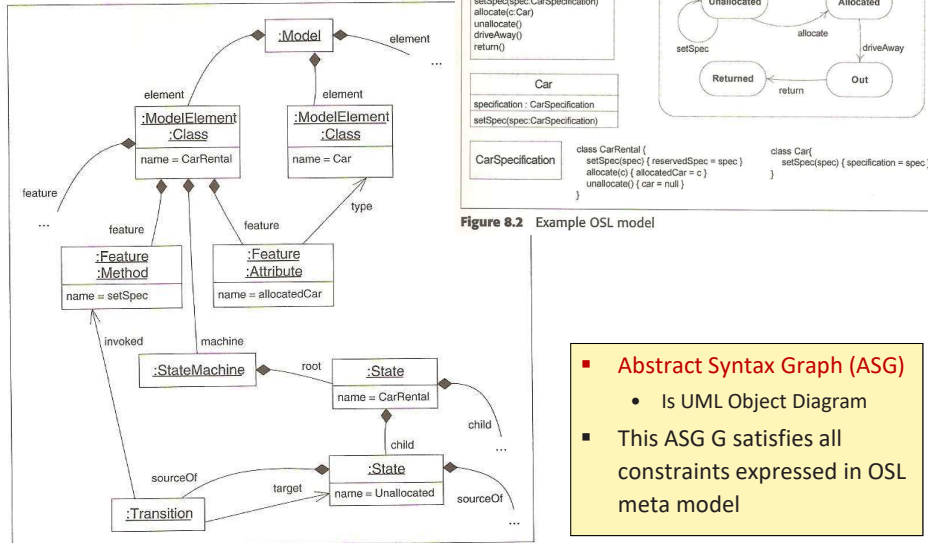
- “The left-hand side and the right-hand side of an assignment have the same type”

AssignmentStatement

lhs.type = rhs.type



An OSL model as ASG



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

CISC844, Winter 2026

DSLs

33

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

CFGs vs Meta models

CFGs

- +/- textual
- +/- supports concrete & abstract syntax
- + 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

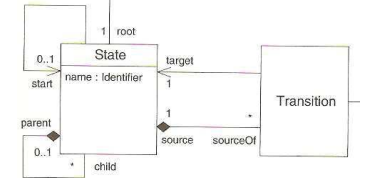
- +/- graphical
- +/- abstract syntax only
- + attributes aid readability
- + elements can be referred to directly
- + classes define a namespace
- + OCL can be used for additional constraints
- relatively novel
- lack of tool support
- harder to define semantic mappings

```

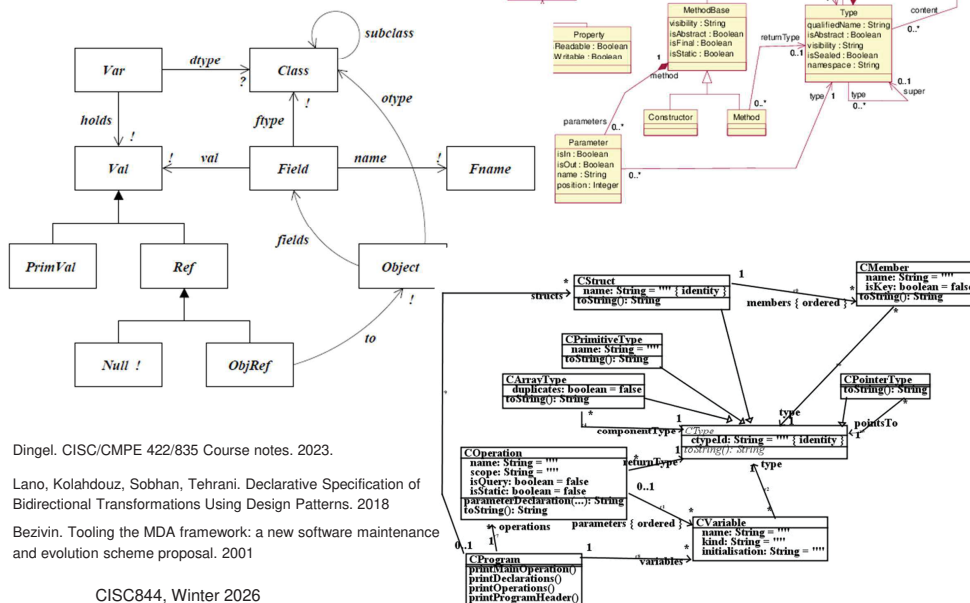
19. StateMachine ::= State
20. State ::= StateName (StartState)? (State)* (Transition)*
21. StateName ::= Identifier
22. StartState ::= StateRef
23. Transition ::= MethodRef StateRef
24. MethodRef ::= Identifier
25. StateRef ::= Identifier
    
```

CISC844, Winter 2026

DSLs



Example 2: General Purpose Languages (Java, C, C#)



Dingel. CISC/CMPE 422/835 Course notes. 2023.

Lano, Kolahdouz, Sobhan, Tehrani. Declarative Specification of Bidirectional Transformations Using Design Patterns. 2018

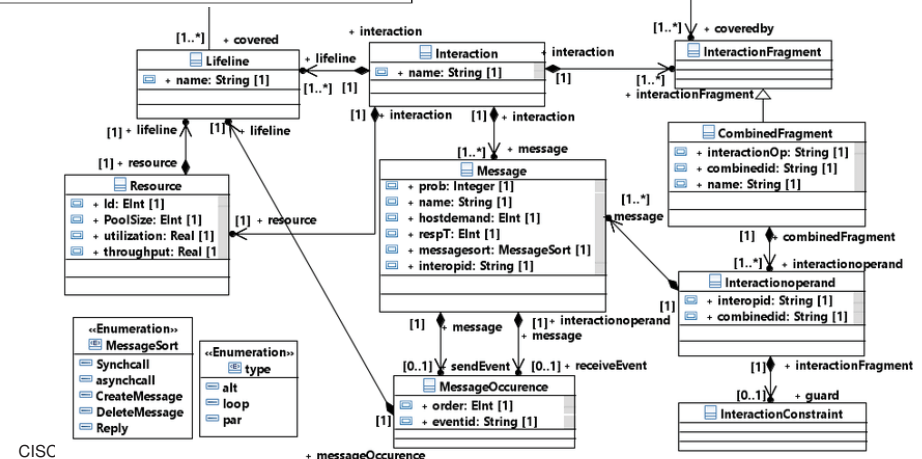
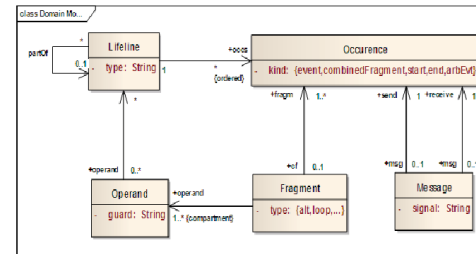
Bezivin. Tooling the MDA framework: a new software maintenance and evolution scheme proposal. 2001

CISC844, Winter 2026

Example 3: UML Sequence Diagrams

Rahmouni, Mbarki. Model-Driven Generation: From Models to MVC2 Web Applications. 2014

Shailesh, Nayak, Prasad. An UML Based Performance Evaluation of Real-Time Systems Using Timed Petri Net. 2020

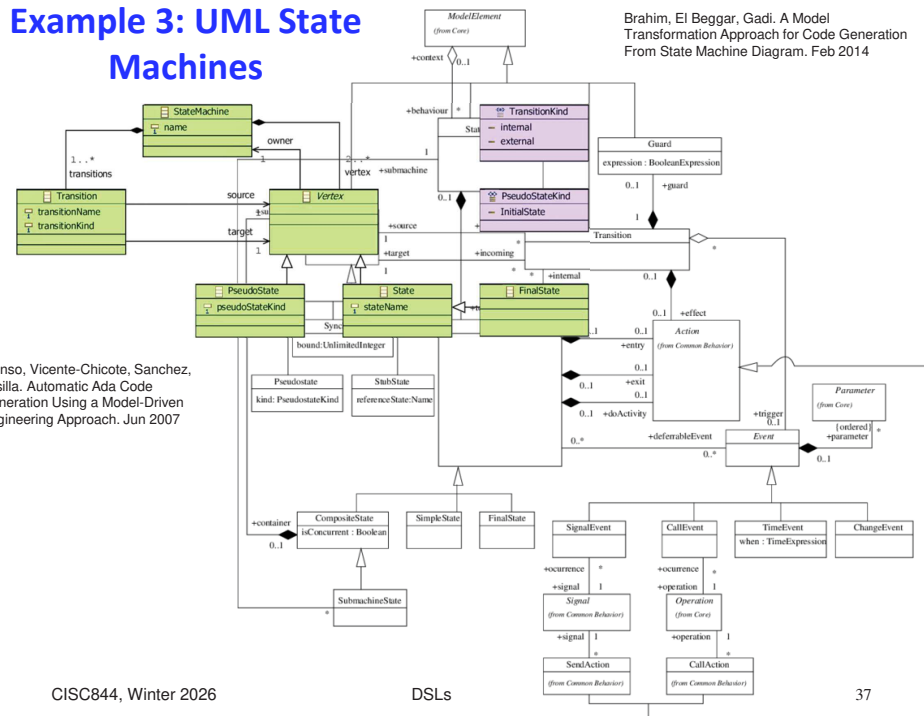


CISC

Example 3: UML State Machines

Alonso, Vicente-Chicote, Sanchez, Losilla. Automatic Ada Code Generation Using a Model-Driven Engineering Approach. Jun 2007

Brahim, El Beggar, Gadi. A Model Transformation Approach for Code Generation From State Machine Diagram. Feb 2014

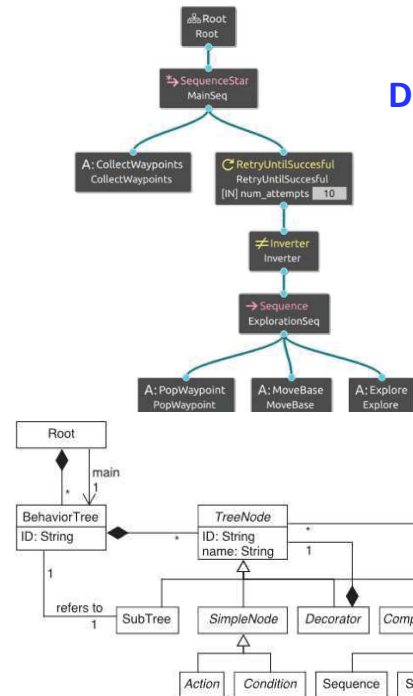


CISC844, Winter 2026

DSLs

37

Example 4: Domain Specific Languages

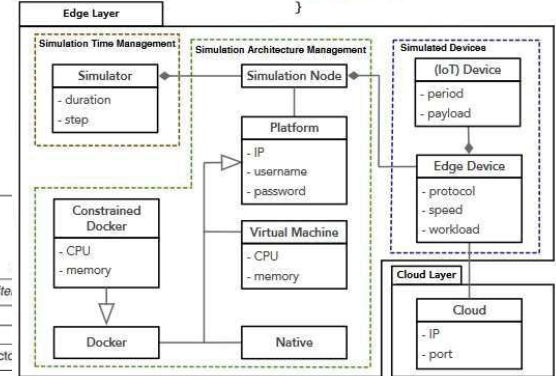


Ghazouli, Berges, Li, Nejati, Sabetzadeh, McCallen. A Domain-Specific Language for Simulation-Based Testing of IoT Edge-to-Cloud Solutions. MODELS'22. 2023

CISC844, Winter 2026

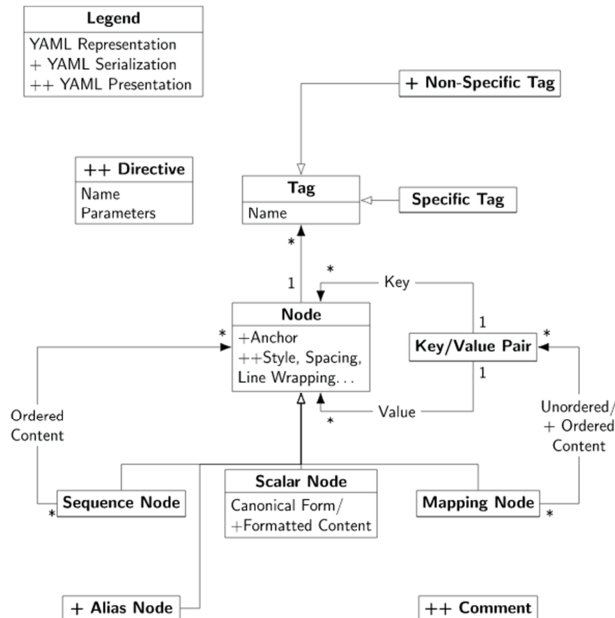
DSLs Li, Nejati, Sabetzadeh, McCallen. A Domain-Specific Language for Simulation-Based Testing of IoT Edge-to-Cloud Solutions. MODELS'22. 2023

```
SimulationNode: SN1 {
  platform: P1
  EdgeDevices: {E1[7], E2[3]}
}
SimulationNode: SN2 {
  platform: P2
  EdgeDevices: {E1[30]}
}
Platform: P1 {
  type: Native
}
Platform: P2 {
  type: Docker
  IP: 192.168.0.4
  username: user2
  password: password2
  CPU: 4
  memory: 2G
}
```



Example 5: YAML

- Representation model
- Abstract syntax
- Serialization model
- Serialization syntax
- Presentation model
- Concrete syntax



yaml.org/spec/1.2.2/#chapter-3-processes-and-models

CISC844, Winter 2026

DSLs

39



CISC844, Winter 2026

DSLs

40

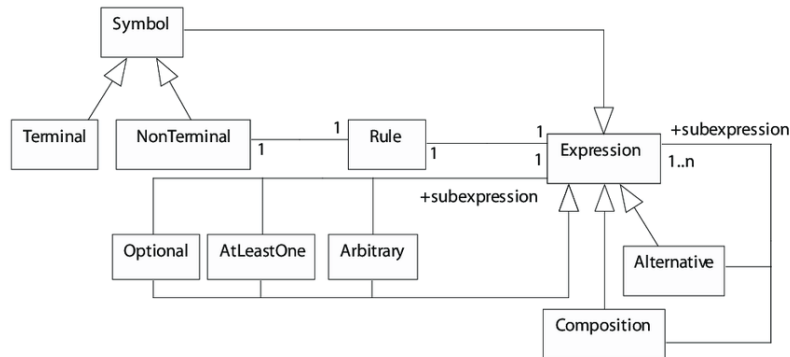
Going meta/self-referential/crazy:

How to describe a language for the description of languages?



1. How to describe BNF?

a) Using class diagrams



[Joachim Fischer, Michael Piefel Markus Scheidgen. A metamodel for SDL-2000 in the context of metamodeling UML. 2005]

CISC844, Winter 2026

DSLs

41

Going meta/self-referential/crazy:

How to describe a language for the description of languages?



1. How to describe BNF?

b) Using BNF!

```
Production = production_name "=" Expression "." .
Expression = Alternative { "|" Alternative } .
Alternative = Term { Term } .
Term = Reference | literal [ "..." literal ] |
      Exclusion | Group | Option | Repetition .
Reference = production_name [ "-" Subtraction ] .
Subtraction = Atom | "(" Atom { "|" Atom } ")" .
Atom = production_name | literal .
Exclusion = "!" literal | "!" "(" literal { "|" literal } ")" .
Group = "(" Expression ")" .
Option = "[" Expression "]" .
Repetition = "{" Expression "}" .
```



[Google Protobuf Guide, <https://protobuf.com/docs/language-spec>]

CISC844, Winter 2026

DSLs

42

Going meta/self-referential/crazy:

How to describe a language for the description of languages?

```
grammar bnf;
rulelist
: rule_* EOF
;
rule_
: lhs ASSIGN rhs
;
lhs
: id_
;
rhs
: alternatives
;
alternatives
: alternative ('|' alternative)*
;
alternative
: element*
;
element
: optional_
| zeroormore
| oneormore
| text_
| id_
;
optional_
: '[' alternatives ']'
;
zeroormore
: '{' alternatives '}'
;
oneormore
: '(' alternatives ')'
;
text_
: ID
;
```

lab.antlr.org

github.com/antlr/grammars-v4/blob/master/antlr/antlr4/ANTLRv4Parser.g4

CISC844, Winter 2026

DSLs

Going meta/self-referential/crazy:

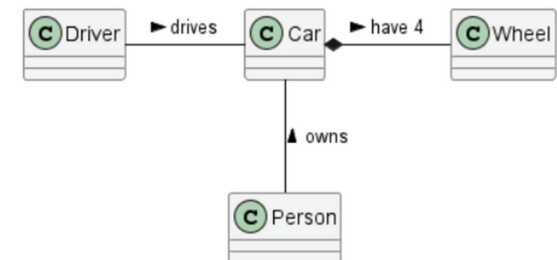
How to describe a language for the description of languages?



2. How to describe class diagrams?

a) Using BNF: E.g., PlantUML.com

```
@startuml
class Car
Driver - Car : drives >
Car *- Wheel : have 4 >
Car -- Person : < owns
@enduml
```



plantuml.com/class-diagram

CISC844, Winter 2026

DSLs

44

Going meta/self-referential/crazy: How to describe a language for the description of languages?



2. How to describe **class diagrams**?

a) Using BNF: E.g., TextUML

```
class Cart
end;

class CartItem
end;

association CartItemProduct
    role item : CartItem[*];
    navigable role product : Product[1];
end;

aggregation CartHasItems
    navigable role item : CartItem[*];
    navigable role cart : Cart[1];
end;
```

<http://abstratt.github.io/textuml/docs/tutorial.html>

<https://github.com/abstratt/textuml/blob/master/plugins/com.abstratt.mdd.frontend.textuml.grammar/textuml.scc>

CISC844, Winter 2026

DSLs

45

Going meta/self-referential/crazy: How to describe a language for the description of languages?



2. How to describe **class diagrams**?

a) Using BNF: E.g., Umlpe, umple.org

```
view plain
01. class Student {}
02.
03. class CourseSection {}
04.
05. class Registration
06. {
07.     String grade;
08.     * -- 1 Student;
09.     * -- 1 CourseSection;
10. }
11.
```

The class diagram to reflect the Umlpe code

github.com/umple/umple/blob/master/cruise.umple/src/class/umple_classes.grammar

CISC844, Winter 2026

DSLs

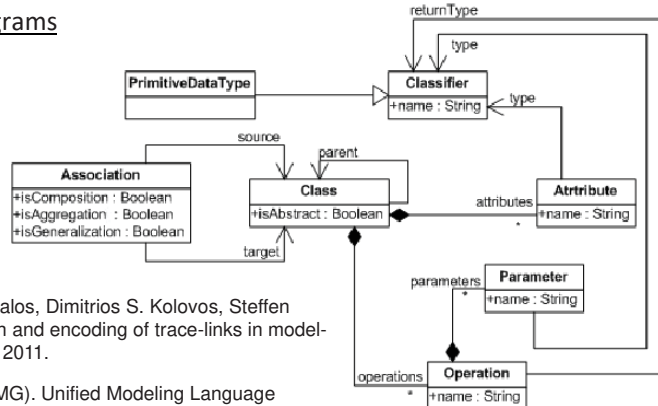
46

Going meta/self-referential/crazy: How to describe a language for the description of languages?



2. How to describe **class diagrams**?

b) Using class diagrams



Richard F. Paige, Nikolaos Drivalos, Dimitrios S. Kolovos, Steffen Zschaler. Rigorous identification and encoding of trace-links in model-driven engineering. SoSyM 10. 2011.

Object Management Group (OMG). Unified Modeling Language Specification v2.5.1. <https://www.omg.org/spec/UML/2.5>

CISC844, Winter 2026

DSLs

47

described by

Meta-meta model,
i.e., meta modeling language,
i.e., language to express meta models

Meta model of class diagrams

a class diagram

The 4-layer meta modeling hierarchy

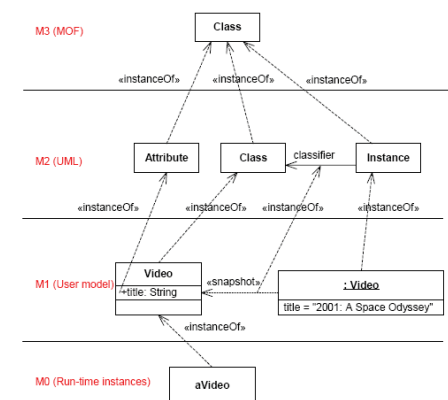


Figure 7.8 - An example of the four-layer metamodel hierarchy

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

CISC844, Winter 2026

DSLs

48

Eclipse Modeling Framework (EMF)

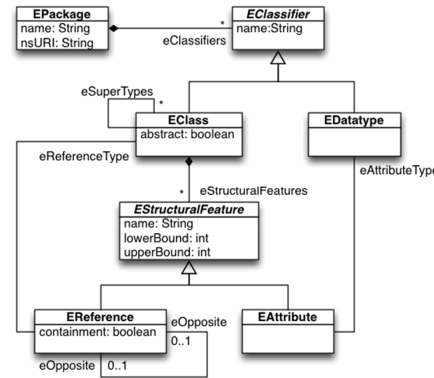
- modeling framework and code generation facility for building tools and other applications based on a structured data model
- <http://eclipse.org/modeling/emf/>

Meta modeling languages



Ecore

- Metamodeling language of EMF, i.e., language to express meta models, i.e., language to describe (abstract syntax of) languages
- Runtime support
 - change notification
 - persistence w/ XMI serialization
 - API for manipulation



https://eclipse.dev/Xtext/documentation/308_emf_integration.html

Meta Object Facility (MOF)

- OMG's version of ECore

CISC844, Winter 2026

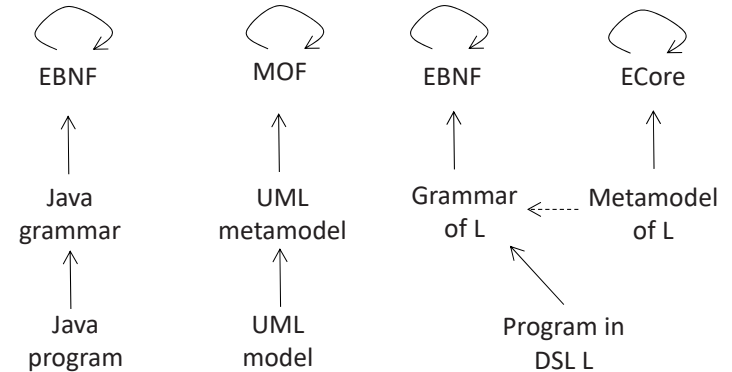
DSLs

49



Concrete/abstract syntax specification approaches

↑ described in, conforms to
← generated from



GPLs

UML

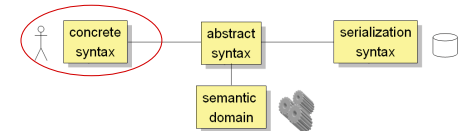
DSLs w/ Xtext

CISC844, Winter 2026

DSLs

50

Concrete Syntax

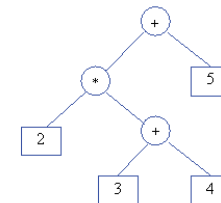


- Need to decide how AST or ASG is displayed to and input by the user
- The abstract-to-concrete mapping assigns elements of abstract syntax to some concrete syntax

Examples:

- Linear concrete syntax

Abstract Syntax



Concrete Syntax (examples)

“unparse”

Which of these is bad?

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

Ways to define concrete syntax

Regular expressions

- <https://regex101.com/>

Grammars

- E.g., Java: github.com/antlr/grammars-v4/tree/master/java/java
 - Lexer (201 LoC): from characters to tokens

Annotations

Example 2: Graphical concrete syntax

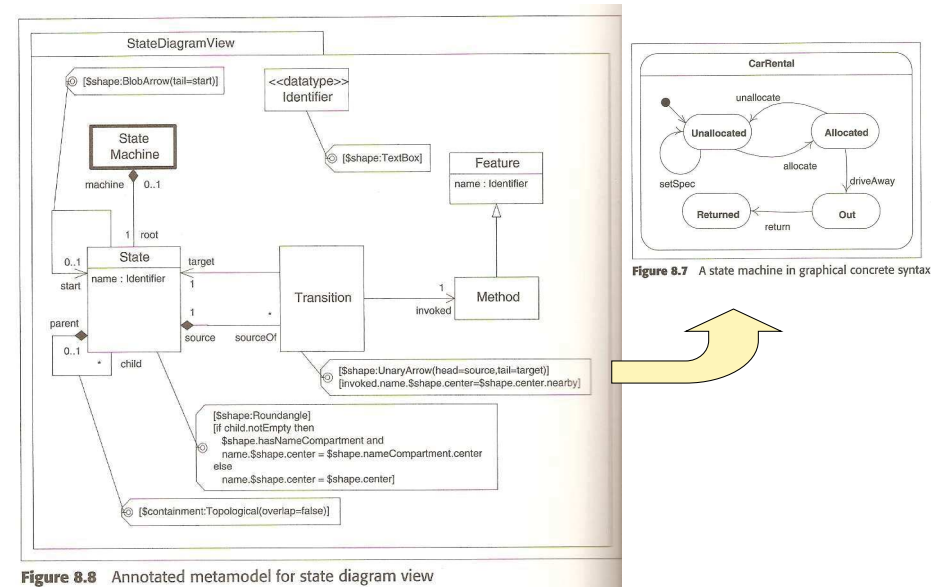


Figure 8.8 Annotated metamodel for state diagram view

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

Example 2: Graphical concrete syntax (Cont'd)

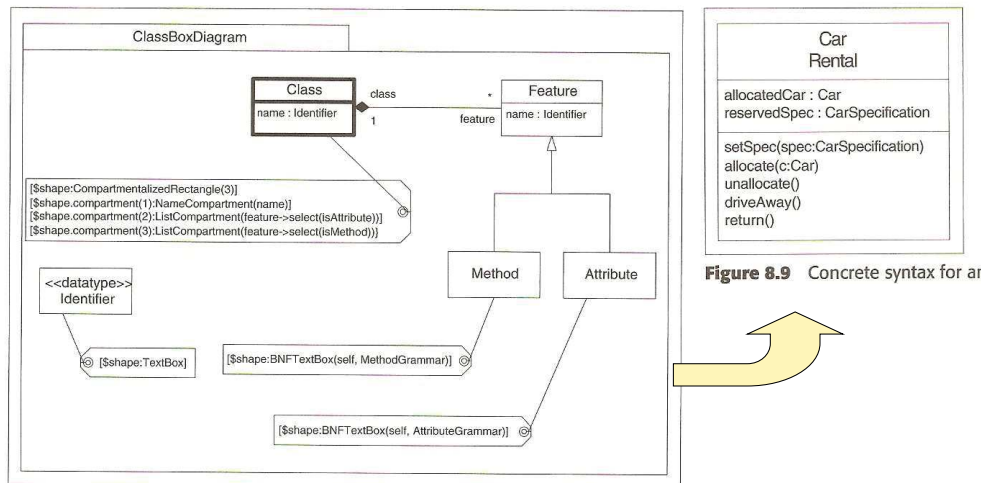
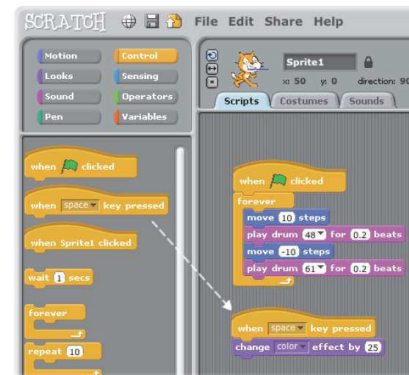


Figure 8.9 Concrete syntax for ar

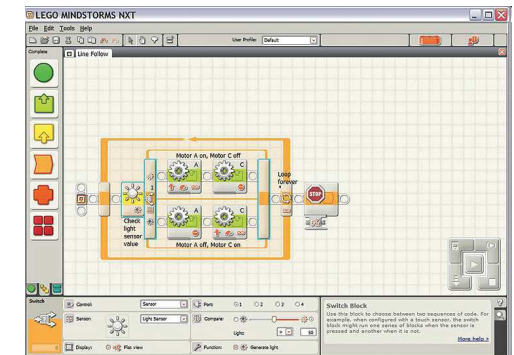
Figure 8.10 Annotated metamodel for class notation

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

Other examples: Graphical concrete syntax



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



Lego Mindstorms' NXT-G language

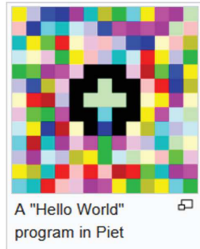
How about another dimension?

- UML state machines in Second Life: https://www.youtube.com/watch?v=mkiXRZz_mj0
- X3D-UML [MHS08]: <https://www.youtube.com/watch?v=gcgQaiTXVrA>

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

Esoteric programming languages

- Intentionally unusual syntax (concrete and/or abstract) and/or semantics
- en.wikipedia.org/wiki/Esoteric_programming_language
- E.g., 'Hello World' in
 - Shakespeare
 - Piet
 - Malbolge



This program displays "Hello, World".^[9]

```

Do Not Know, a play in Two acts.

Hame, a young man with a remarkable patience.
Juliet, a virtuous young woman of remarkable grace.
Ophelia, a remarkable woman much in dispute with Hamlet.
Hamlet, the lastword of modern thinking etc.

ACT I: Hamlet's thoughts and feelings.

Scene I: The thinking of Hamlet.

[Enter Hamlet and Hame]

Hamlet:
You lying ethical fatherless big weally half-witted count!
You are as stupid as the difference between a handsome rich brown
hair and myself! Speak your mind!

You are as brave as the use of your fat little stuffed assed dusty
old rotten candle and a beautiful fair new powerful sunny summer's
day. You are as healthy as the difference between the use of the
sweetest roasted meat and my father and yourself! Speak your mind!

You are as cowardly as the use of yourself and the difference
between a big mighty proud kingdom and a horse. Speak your mind.

Speak your mind!

[Exit Hamlet]

Scene II: The thinking of Juliet.

[Enter Juliet]

Juliet:
You art as sweet as the use of the use of Hame and his horse and his
black cat! Speak thy mind!

[Exit Juliet]

Scene III: The thinking of Ophelia.

[Enter Ophelia]

Ophelia:
You art as beautiful as the difference between Hame and the square
of a huge green peaceful tree. Speak thy mind!

You art as lovely as the product of a large rural town and my assed
bedroom's administrative parts. Speak thy mind!

You art as loving as the product of the bluest clearest sweetest city
and the use of a haploid and a white horse. You art as beautiful as
the difference between Juliet and myself. Speak thy mind!

[Exit Ophelia and Hamlet]

ACT II: Refine Hamlet's luck.

Scene I: Hame and Juliet's conversation.

[Enter Hame and Juliet]

Hame:
Speak more mind! You are as bad as Hamlet! You are as small as the
difference between the square of the difference between my little pony
and your big hairy hand and the cube of your sorry little
cousin. Speak your mind!

[Exit Hame]

Scene II: Juliet and Ophelia's conversation.

[Enter Ophelia]

Juliet:
You art as good as the quotient between Hame and the use of a small
furry animal and a lion. Speak your mind!

Ophelia:
You art as disgusting as the quotient between Hame and twice the
difference between a politician and an assed senator! Speak
your mind!

[Exit Ophelia]
    
```

(=<`#9]~6ZY327Uv4-QsqPmN&+Ij"'E%e{Ab~w=_:]Kw%o44Uqp0/Q?xNvL:~H%c#DD2^)

en.wikipedia.org/wiki/Malbolge

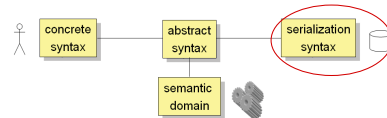
en.wikipedia.org/wiki/Shakespeare_Programming_Language

CISC844, Winter 2026

DSLs

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
- Often done using **Extensible Markup Language (XML)**
- Another relevant standard for MOF-based models**
 - 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

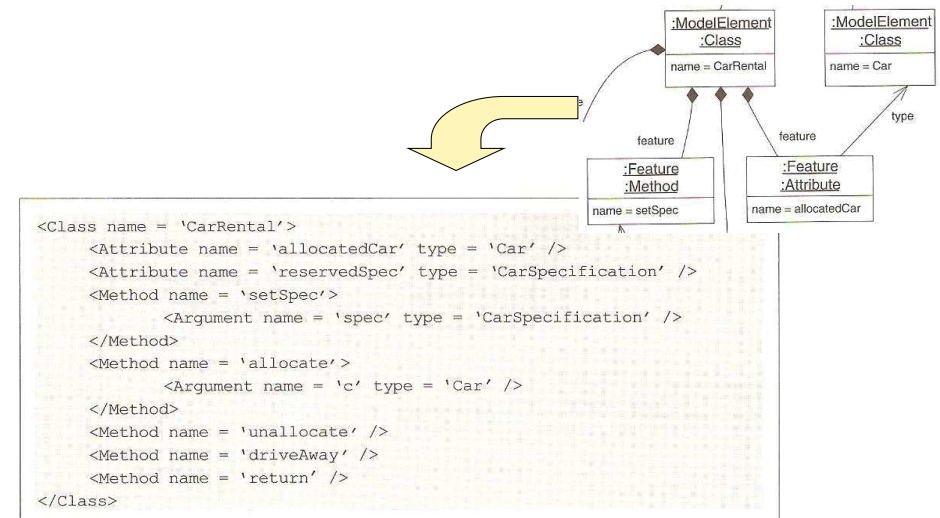
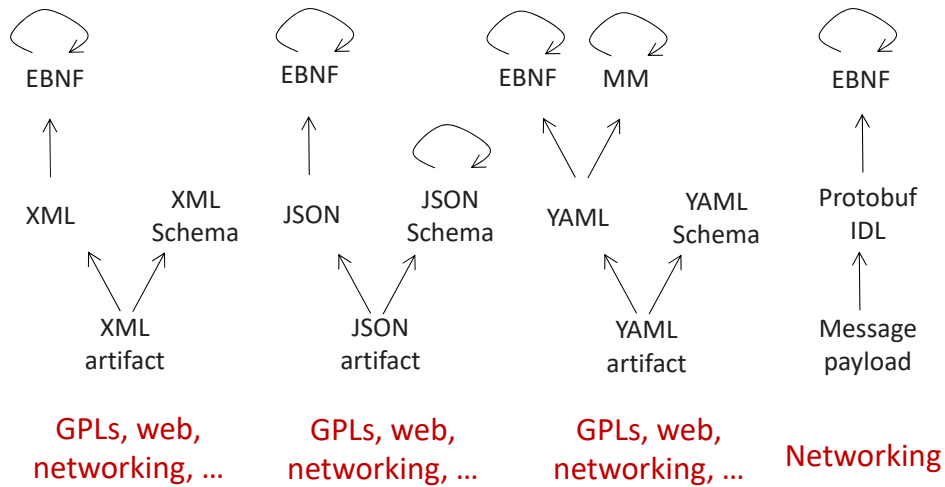


Figure 8.11 XML for ASG fragment of car rental model

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

Serialization syntax specification approaches

↑ described in,
conforms to



en.wikipedia.org/wiki/Comparison_of_data_serialization_formats

CISC844, Winter 2026

DSLs

61



CISC844, Winter 2026

DSLs

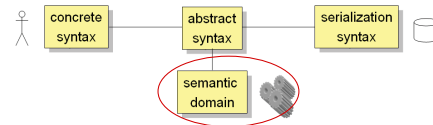
62

Expressing SW models: Overview (Part 2)

2. Domain-specific languages

1. Intro and examples (e.g., Risl, 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. DSL tools

- EMF, GMF, Graphiti, Xtext

CISC844, Winter 2026

DSLs

63

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

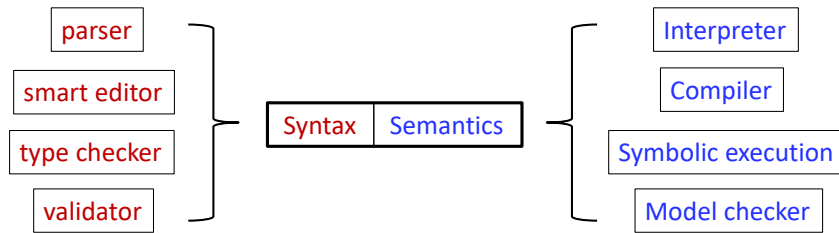
CISC844, Winter 2026

DSLs

64

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”)



CISC844, Winter 2026

DSLs

65

DSL tools

- Eclipse, EMF, GMF, Graphiti, Sirius
- Xtext [Assignment 4], Langium, textX
- 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://docs.microsoft.com/en-us/visualstudio/modeling/modeling-sdk-for-visual-studio-domain-specific-languages>

CISC844, Winter 2026

DSLs

67

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. DSL tools
 - EMF, GMF, Graphiti, Xtext

CISC844, Winter 2026

DSLs

66

Efforts related to DSLs

- Software Factories (Microsoft)
- Intensional Programming
- Language-oriented programming
- Language workbenches
 - Language Workbench Challenge 2016
 - <https://2016.splashcon.org/track/lwc2016>
- Language engineering (SLE conference series)

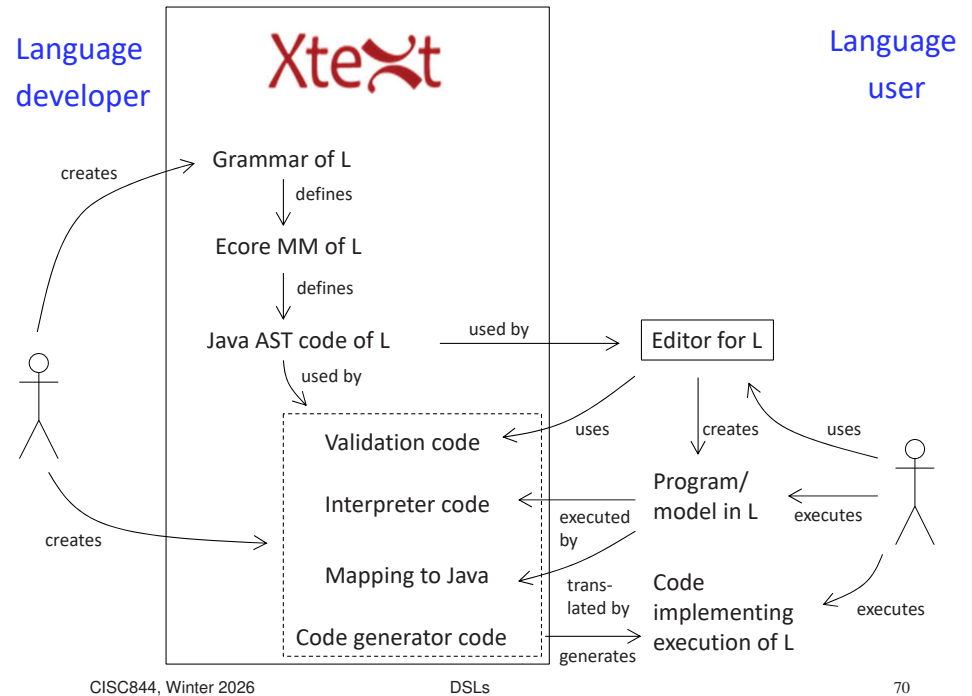
CISC844, Winter 2026

DSLs

68

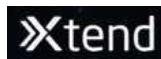
- Eclipse-based open-source framework for development of programming languages and domain-specific languages
- Offers
 - Parser generator
 - Editor plugin generator supporting
 - Syntax highlighting
 - Well-formedness checking (validation) w/ error markers and quick fixes
 - Background parsing
 - Auto-completion 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

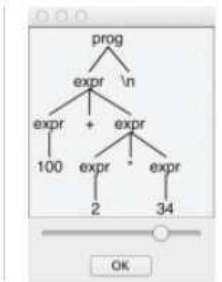


From www.antlr.org:

"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 '+' '-'
    | INT
    | '(' expr ')'
    ;
NEWLINE: [\r\n]+ ;
INT: [0-9]+ ;
```

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



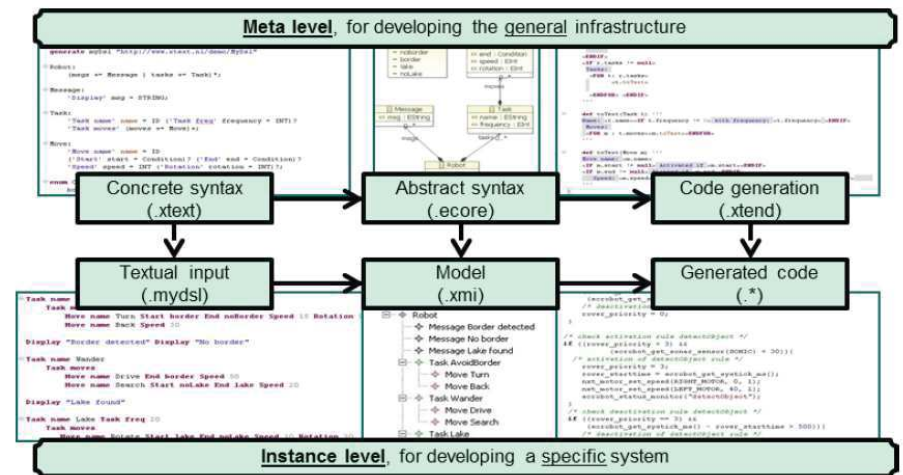
From eclipse.org/xtend/:

"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

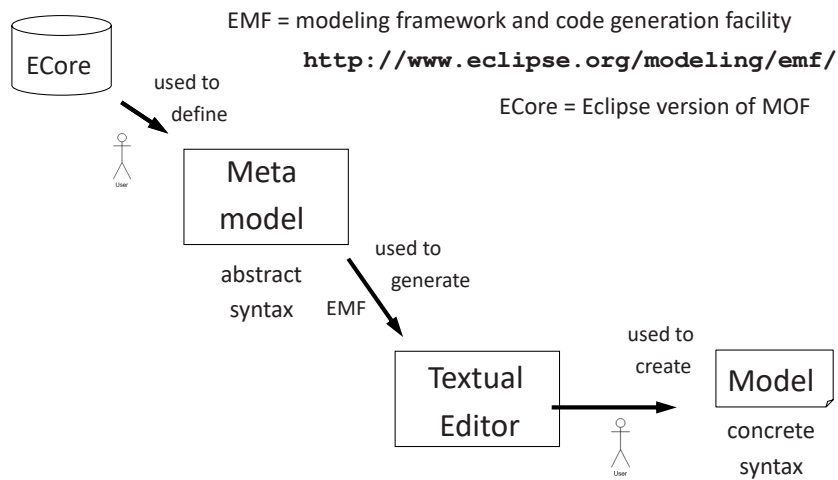
Overview of key Xtext artifacts



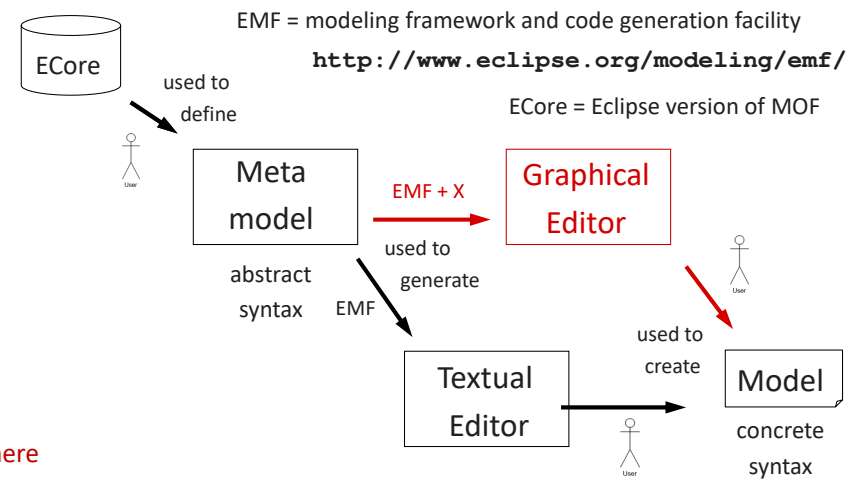
From: A. Mooij, J. Hooman. Creating a Domain Specific Language (DSL) with Xtext. Version 2.14.

Available at <http://www.cs.kun.nl/J.Hooman/DSL/>

EMF + X



EMF + X



where

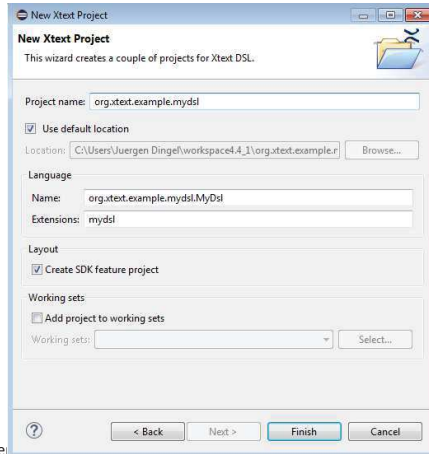
- X = Graphiti, <https://eclipse.org/graphiti/>, or
- X = GMF, <http://eclipse.org/modeling/gmf/>
- X = Sirius, <https://www.eclipse.org/sirius/>

Using Xtext

0. Installation instructions etc on Assignment 4 page

1. Create Xtext project

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



CISC844, Winter



77

Using Xtext (Cont'd)

2. Create grammar .xtext in folder

"src/<project name>"

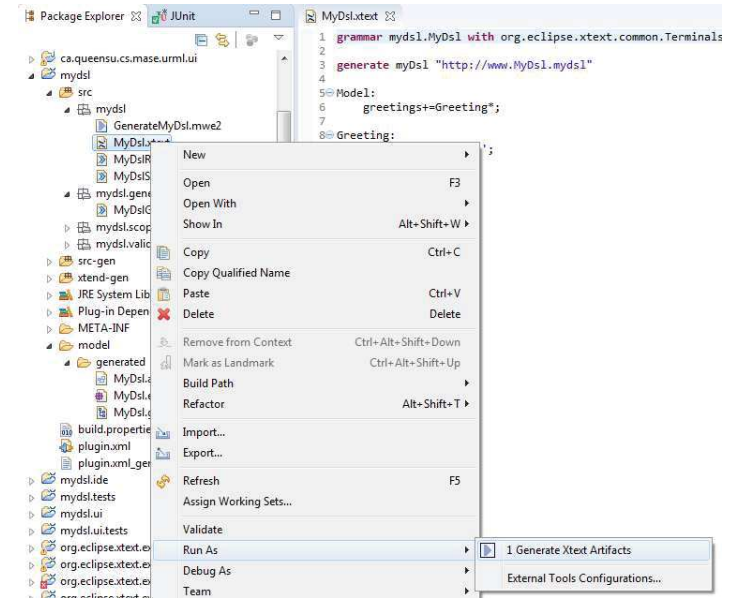
3. Generate Xtext artifacts

■ in "src-gen" folder:

.java

■ in "model/generated" folder:

.ecore, .genmodel

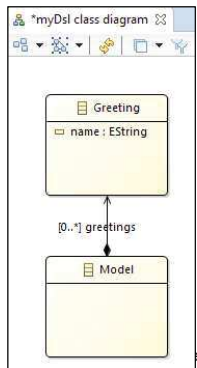
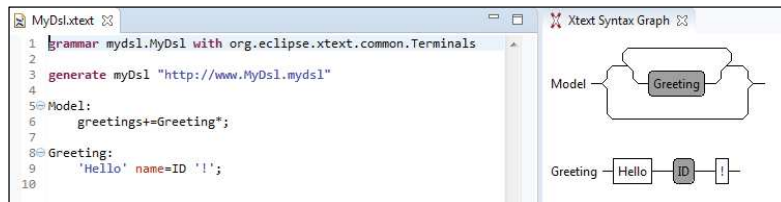


CISC844, Winter 2026

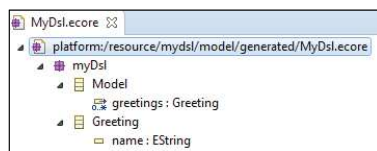
DSLs

78

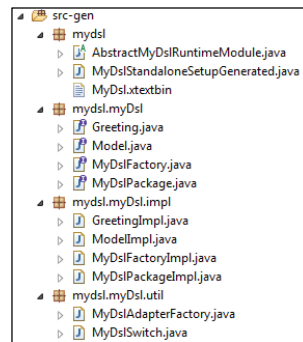
Using Xtext (Cont'd)



2026



DSLs



79

Using Xtext (Cont'd)

4. Start editor

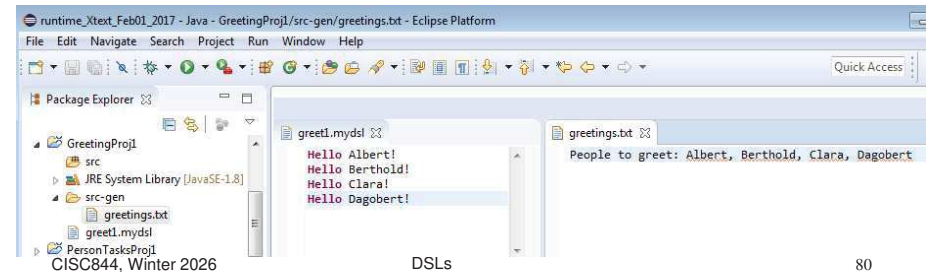
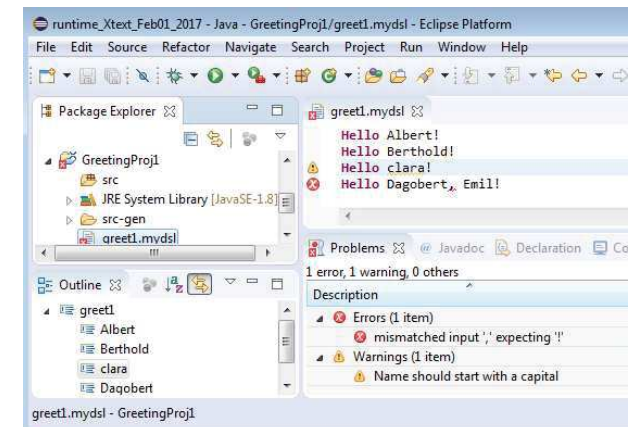
Right-click project, "Run As | Eclipse Application"

5. Create new Java project

6. Input text, validate, etc

7. Inspect generated output

8. Run generated code



CISC844, Winter 2026

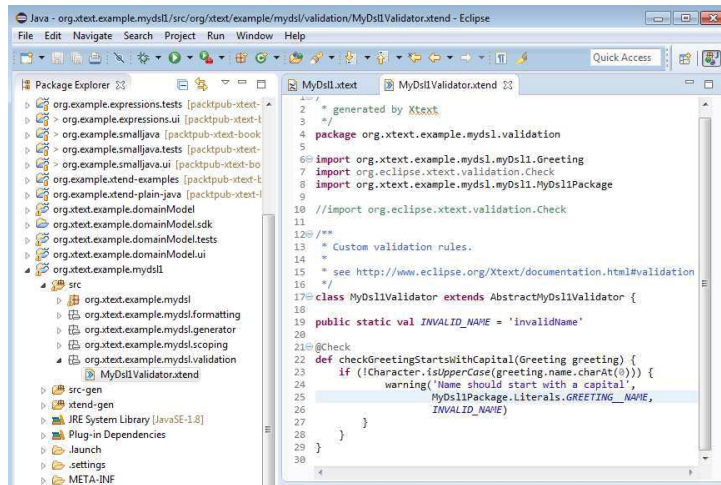
DSLs

80

Using Xtext (Cont'd)

6. Implement custom validation rules

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



CISC844, Winter 2026

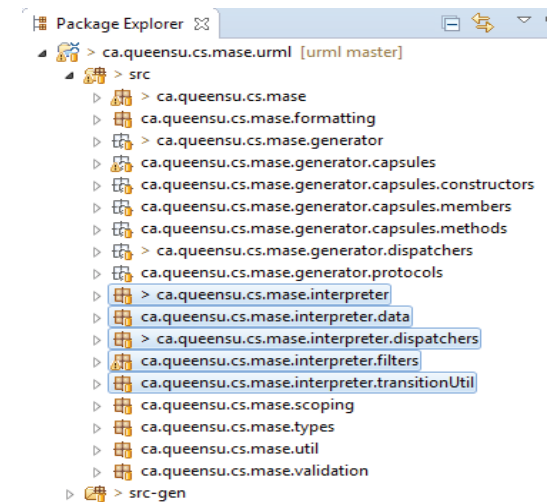
DSLs

81

Using Xtext (Cont'd)

7. Implement interpreter

- in “src/<project name>/interpreter”

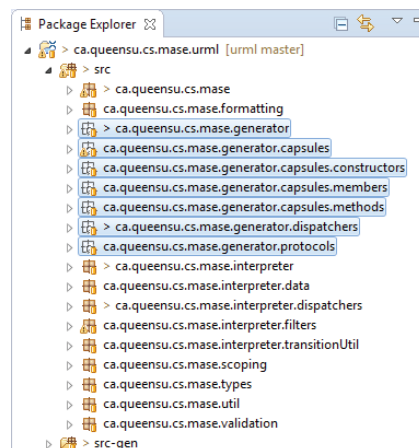


CISC844, Winter 2026

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



CISC844, Winter 2026

DSLs

83

A4: Urml

- Textual modeling language for reactive systems

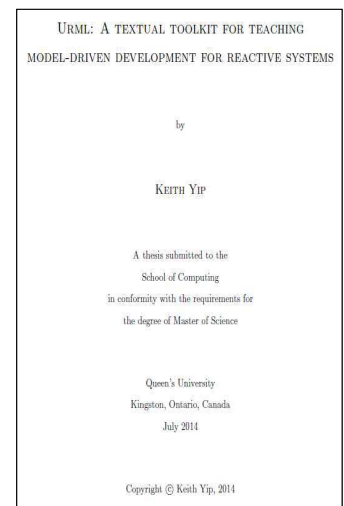
- Support for

- 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 MSc

<https://qspace.library.queensu.ca/handle/1974/12274>

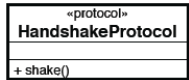
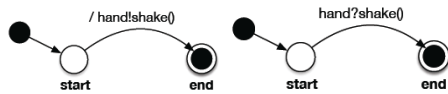
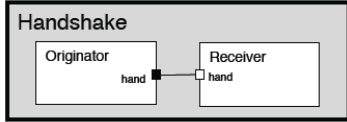


CISC844, Winter 2026

DSLs

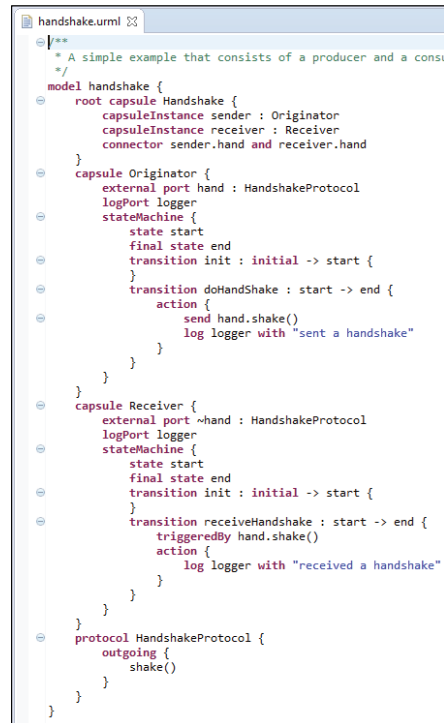
84

A4: Urml (Cont'd)

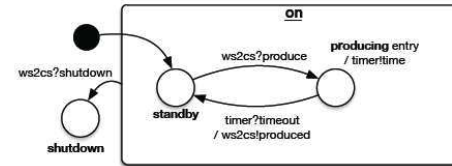
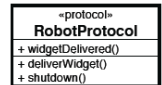
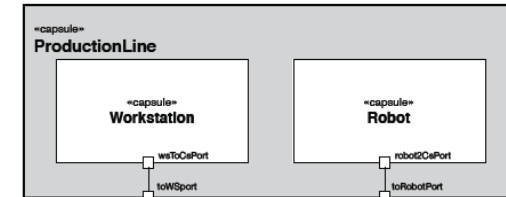
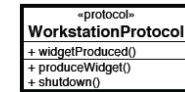
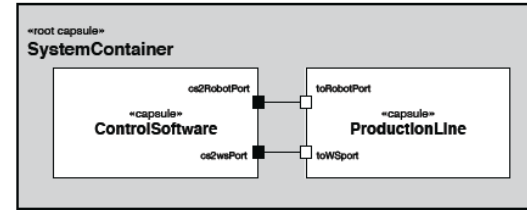


CISC844, Winter 2026

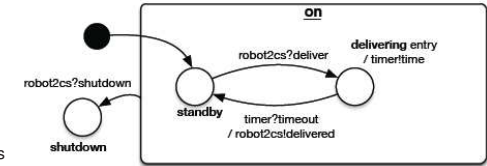
DSLs



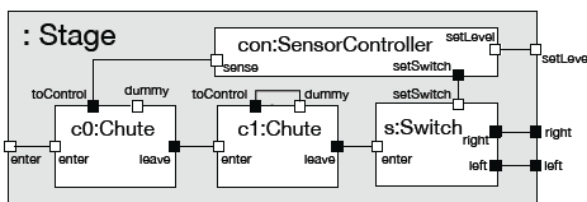
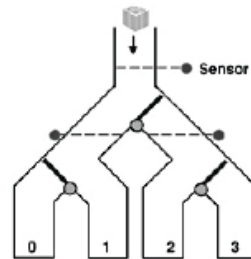
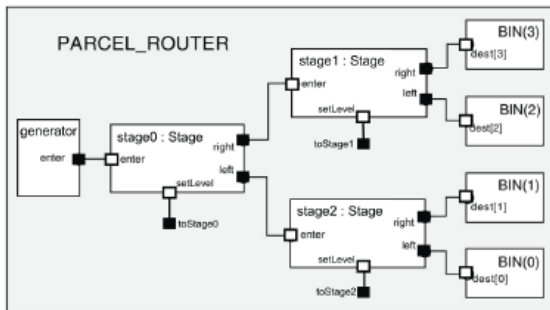
A4: Urml (Cont'd)



DSLs



A4: Urml (Cont'd)



CISC844, Winter 2026

DSLs

87

Leveraging language descriptions

- ANTLR (GPLs, DSLs)
 - Lexer/parser generation
- Xtext (DSLs)
 - Language infrastructure generation
- EMF/Ecore (MM)
 - AST API generation
- JSON, XML, and YAML (GPLs, web, ...)
 - Manipulation (conversion, serialization, ...)
 - With schemas: validation
- Protobuf IDL (Networking, distributed systems)
 - Serialization generation
- YANG (Networking)
 - API generation

CISC844, Winter 2026

DSLs

88

Model-driven network configuration

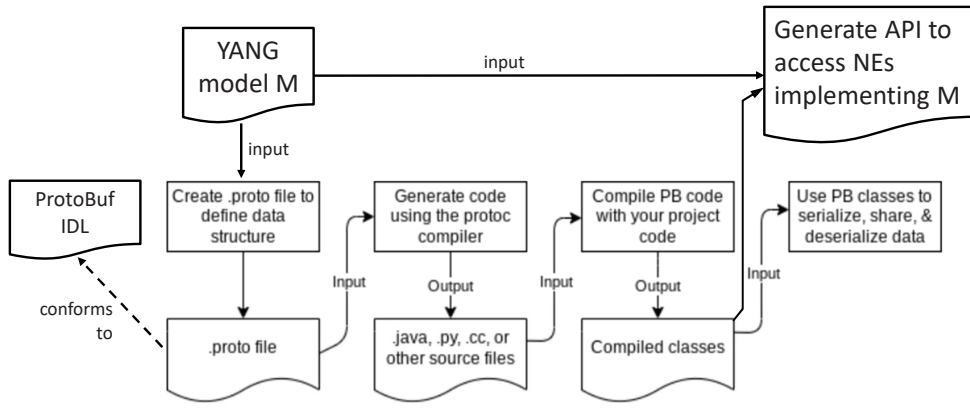


Figure 1. Protocol buffers workflow

[Protocol Buffers Documentation at protobuf.dev]

Course summary!?

