Lecture 2: 
Domain-Specific Logic Languages (DSLL) 
or 
Logic-based Domain-Specific Languages (LDSL) 

Prof. Uwe Aßmann 
Christian Wende 
Technische Universität Dresden 
Software Engineering Group 
http://st.inf.tu-dresden.de 
Version 0.2 
Dresden, Aug. 27, 2013
Literature


OWLText URL http://www.emftext.org/index.php/OWLText
• 2.1 Why do we need DSL?
• 2.2 The historical development of DSL environments
• 2.3 We need semantics... But there is the problem of technical spaces
• 2.4 OWLText: a tool for textual LDSL
• 2.5 A graphical LDSL: Feature model with ontology semantics
• 2.6 Families of LDSD
• 2.7. Outlook: LDSL as Language Families
2.1 Why do we need Domain-Specific Logic Languages = Logic-Based Domain-Specific Languages?
In Dresden, we work on the eternal problems...

Coffee admins
Recently, we got a NAO Robot „Chuck‟

Chuck could fetch coffee!
Well, I like to teach
But, how do I teach Chuck to bring me a coffee?
Why Domain-Specific Languages (DSL)?

**Pros:**
+ Use the concepts and idioms of a domain
  + DSL embody domain knowledge (concepts and constraints)
  + Make domain modeling easy
+ Grandmothers (.. Domain experts ..) can understand and write DSL programs
  + Higher level of abstraction
  + Concise and self-documenting
  + Can enhance productivity, reliability, maintainability and portability

**But:**
- Costs of design, implementation and maintenance
- Costs of education for users
- Limited availability of DSLs

• From: http://homepages.cwi.nl/~arie/papers/dslbib/
Network Device Specification (DSL for Comarch)

K. Miksa, COMARCH
Graphic DSLs in Device Modeling

K. Miksa, COMARCH

Logic-Based DSL
Ex.: Graphic DSL for Process Guidance Ontologies in MOST Project

Logic-Based DSL

K. Miksa, COMARCH
Ex.: Guidance Task List During BPMN Process Refinement (SAP)

The activity "Hire Applicant (Hire Applicant)" has to be refined by some other activity.
2.2 The Historical Development of DSL
DSLs and Logic-Based DSLs

Logic-Based DSL

Hand implementation of DSL

Parsing environments for DSL

Egonomorphic Grammar based Compiler environments

Product families

Logic-Based DSLs

OWLText

Family LDSL

EMFText, XText

Lex, yacc

Eclipse based toolkits

LDSL Families

Attribute-Grammar based Compiler environments

ELI, Cocktail, JastAdd
Example: Building textual DSLs using EMFText

- EMFText is based on Eclipse Modeling Framework (EMF)
- Generates editors, parsers, printers for a EMOF metamodel
- [www.emftext.org](http://www.emftext.org)

- (Florian Heidenreich, Jendrik Johannes, Sven Karol, Mirko Seifert, Christian Wende)
What’s in a DSL?

Fakultät Informatik Institut Software- und Multimediatechnik, Lehrstuhl Softwaretechnologie

Logic-Based DSL

Concrete Syntax: Symbols, shapes

Meta Model (Structure): Domain concepts, Language concepts, relations, attributes

Static Semantics: Typing, use-definition relationships

Dynamic Semantics: Behavior, meaning
How to build a DSL with EMFText

Fakultät Informatik Institut Software- und Multimediatechnik, Lehrstuhl Softwaretechnologie

- Ecore Metamodel
- Concrete Syntax Specification
- EMFText
- Static Semantic Resolvers
- EMFText Resource
- Printer
- Grammar
- Parser Generator (ANTLR)
- Editor
- Input for
- Generates
- Derived from

Logic-Based DSL
How to build a DSL - Overview

- Ecore Metamodel
- CS Specification
- EMFText
- Grammar
- Parser Generator (ANTLR)
- Static Semantic Resolvers
- EMFText Resource
- Printer
- Parser

input for

generates

derived from

Logic-Based DSL
Creating a new meta model:
Define concepts, relations and properties in an EMOF/Ecore model

Meta model elements:
- Classes, Data Types
- Enumerations
- Attributes
- References (Containment, Non-containment)
- Cardinalities
- Inheritance

Very similar to the T-Box of a restricted RDFS
How to build a DSL – Meta model

- Meta model elements:
  - Classes
  - Data Types
  - Enumerations
  - Attributes
  - References (Containment, Non-containment)
  - Cardinalities
  - Inheritance

- Very similar to the T-Box of a restricted RDFS
Derive initial syntax of the DSL in HUTN (Human Usable Text Notation)

**Logic-Based DSL**

```
org.emftext.language.office
├── settings
├── bin
├── META-INF
└── metamodel
    ├── office.ecore 4915
    │    ├── office.ecorediag 4915
    ├── office.genmodel
    │    └── office.GMF.facad 4915
    ├── src-gen
    │    ├── .classpath 6584 07
    │    ├── .project 4915 16.0
    │    └── build.properties 4915
    │            └── plugin.properties 4915
    │                   └── plugin.xml 4915 16.0

```

Generate HUTN Syntax

Reload...
Generated Grammar of Initial HUTN syntax

f) Logic-Based DSL

```
SYNTAXDEF office
FOR <http://emtext.org/office>
START OfficeModel

TOKENS{
    DEFINE COMMENTS"//.*\n\n\r\n\t|\ufffe\ufffe\ufffe\uffff\ufffe\ufffe\uffff">;  
    DEFINE DIGITSS"0'...9'\"; 
    DEFINE FLOATS"-\"?\"\"0'...9'\"\"0'...9'\"\"0'...9'\"+\"; 
}

TOKENSTYLES{
    "OfficeModel" COLOR #7F0055, BOLD; 
    "name" COLOR #7F0055, BOLD; 
    "elements" COLOR #7F0055, BOLD; 
    "Employee" COLOR #7F0055, BOLD; 
    "worksIn" COLOR #7F0055, BOLD; 
    "worksWith" COLOR #7F0055, BOLD; 
    "Office" COLOR #7F0055, BOLD; 
}

RULES{
    OfficeModel::= "OfficeModel" "{" ("name" :: "name["",""] | "elements" :: elements )* "}" ; 
    Employee::= "Employee" "{" ("name" :: "name["",""] | "worksIn" :: worksIn[] | "worksWith" :: worksWith[] )* "}" ; 
    Office::= "Office" "{" ("name" :: "name["",""] )* "}" ;
}
```
Initial HUTN syntax – Example Document

Logic-Based DSL
Structure of a .cs file based on EBNF:

- Header
  - File extension
  - Meta model namespace URI, location
  - Start element(s)
  - Imports (meta models, other syntax definitions)
- Options
- Token Definitions
- Syntax Rules
Syntax refinement by changing the ENBF Syntax rules

- One per meta class
  - Syntax: MetaClassName ::= Syntax Definition ;

- Definition elements:
  - Static strings (keywords) "public"
  - Choices a|b
  - Multiplicities +,*
  - Compounds (ab)
  - Terminals a[
  - Non-terminals a
Syntax rules - Examples

OfficeModel ::= "officemodel" name[]
    "{" elements* "}"

officemodel SoftwareTechnology {
    ...
}
Syntax rules - Examples

OfficeModel ::= "officemodel" name[]
"{" elements* "}" ;

Employee ::= "employee" name[]
"works" "in" worksIn[]
"works" "with" worksWith[]
""," worksWith[]\)\)* ;

Office ::= "office" name[];

officemodel SoftwareTechnology {  
   office INF2080  
   employee Florian  
   works in INF2080  
}
Generic Syntax vs. Custom Syntax

b) Logic-Based DSL
• Import meta models optionally with syntax
• Extend and combine existing DSLs
• Create embedded DSLs (e.g., for Java)
• Create a template language from your DSL
• ...
The EMFText Syntax Zoo (~100 residents)

- Ecore, KM3 (Kernel Meta Meta Model)
- Quick UML, UML Statemachines
- Java 5 (complete), C# (in progress)
- Feature Models
- Regular Expressions
- natSpec controlled natural language
- DOT (Graphviz language)
...and lots of example DSLs

Ontology-related languages:
- OWL2 Manchester Syntax*
- OWL2 Functional-Style Syntax
- OWLCL*
- SPARQL*
- SPARQL-DL Abstract Syntax
- SWRL

http://www.emftext.org/index.php/EMFText_Concrete_Syntax_Zoo#Ontology_Languages
I have DCGs, so why should I want EMFText?

- Metamodel-based, typed environment
  - Generation of tools
  - Statically typed
  - Integration with application software
  - Integration with many other tools
  - Modularity
  - Integration with software process

Software Process

Tools for management and guidance of the software development process

Methodology

Tools for supporting the software development methodology

Automation

Tools for automating repetitive validation and transformation tasks
2.3 We need Semantics – Why not try Ontologies?
What’s in a DSL?

Concrete Syntax
Symbols, shapes

Meta Model (Structure)
Domain concepts, relations, attributes

Static Semantics
Typing, use-definition relationships

Dynamic Semantics
Behavior, meaning

Logic-Based DSL
Language Semantics and Technical Spaces

Fakultät Informatik Institut Software- und Multimediatechnik, Lehrstuhl Softwaretechnologie

Domain world
- Ontologies
- Domain models
  - OWL

Static Semantic World
- Attributed Grammars
- Abstract Interpretation
- Model checking
- Ontologies
  - OWL
  - CTL

Structural (syntactic) Modeling World
- Structure Hierarchies
- Graphs
  - EMOF
  - UML-CD
  - MOF

Software Engineer

Dynamic Semantic World
- Interpretation
- State systems
- Simulation
- Petri nets
- SOS
- Natural Semantics

Static Semantic Expert

Domain Expert

Domain world

Dynamic Semantic Expert
Language Semantics and Technical Spaces

Fakultät Informatik Institut Software- und Multimediatechnik, Lehrstuhl Softwaretechnologie

Domain world
Ontologies
Domain models
OWL

Structural (syntactic) Modeling World
Structure
Hierarchies
Graphs
EMOF
UML-CD
MOF

Static Semantic World
Attributed Grammars
Abstract Interpretation
Model checking
Ontologies
OWL
CTL

Dynamic Semantic World
Interpretation
State systems
Simulation
Petri nets
SOS
Natural Semantics

Software Engineer
Dynamic Semantic Expert
Domain Expert
Logic-Based DSL
2.3.2 ... But there is the problem of technical spaces
Language Semantics and Technical Spaces

Fakultät Informatik Institut Software- und Multimediatechnik, Lehrstuhl Softwaretechnologie

Structural (syntactic) Modeling World
- Structure
- Hierarchies
- Graphs
- EMOF
- UML-CD
- MOF

Static Semantic World
- Attributed Grammars
- Abstract Interpretation
- Model checking
- Ontologies
- OWL
- CTL

Dynamic Semantic World
- Interpretation
- State systems
- Simulation
- Petri nets
- SOS
- Natural Semantics

Domain world
- Ontologies
- Domain models
- OWL

Domain Expert

Software Engineer

Dynamic Semantic Expert

Domain Expert

Logic-Based DSL
Bridging Ontologies and Software Technology

Ontology World (OntologyWare)

Software Modeler

Software Modeling World (ModelWare)

Ontology Expert

EMOF
UML-CD
MOF

OWL
RDFS

Logic-Based DSL
Bridging technologies

- **Transformation bridge** (physical transport of models to the other space)
- **Integration bridge** integrating metalanguages
Transformation Bridge

Fakultät Informatik Institute Software- und Multimediatechnik, Lehrstuhl Softwaretechnologie

Ontoware TS
- OWL2 Metamodel
- Ontology TBox
- ABox

Modelware TS
- Ecore Metametamodel
- DSL Metamodel
- Model

Bridging Technology - Transformation -

Tools
- Transformation Engine
- Logic-Based DSL

Querying Reasoning Technology

Target model
- conformsTo
- source model

Validation Constraint Technology

M1
- conformsTo

M2
- conformsTo

M3
• Language mapping generates model bridges (same L)
• Language mapping can result from composition, integration, and extension of L+E1 and L+E2
• Language mapping can result from extension by constraints
• Models have *more* semantics in semantic space
2.3.2 Language Integration with MM-Integration Bridges

Logic-Based DSL
Ex.: OntoDSL: Integration Bridge

- [Tobias Walter, Steffen Staab, Uni Koblenz]
2.4 OWLText –
A tool for textual LDSL,
automatically bridging technical spaces

Courtesy:
Christian Wende, TU Dresden
Tobias Walter, U Koblenz
• From textual DSL to OWL constraint checking
• From textual DSL to OWL constraint checking
• **Technology Space Bridging**

• **Metamodelling Space:**
  • Comprehensible infrastructure with tools for metamodelling, modelling, persistence, transformation, code generation
  • Generated API for model manipulation
  • Plethora of integrated tools that share repository

• => Metamodelling is a syntactic island for technology space bridges

• **Interesting Bridges:**
  • Other Syntactical Spaces – EMFText, XText, GMF
  • Grammarware, Parsing and Printing
  • Semantic technical spaces (static and dynamic)
    • Ontologies
    • Attribute Grammars (JASTEMF)
**OWLText technology space bridging for LDSL**

- **Grammar Space**
  - EBNF
  - ANTLR
  - Textual Syntax

- **Metamodelling Space**
  - EMOF
  - EMF
  - Syntactic Repository

- **Ontology Space**
  - OWL
  - Pellet
  - Semantic Classification and Validation
OWLText technology space bridging for LDSL
OWLText technology space bridging for LDSL
OWLTText technology space bridging for LDSL
Generalised Architecture of technical space bridges

Service Integration

Metamodell Integration (Integration Bridges)

Modell Integration (Transformation Bridges)
Example: Office LDSL

Office DSLL

- Tutorial language to model offices, employees, and their interaction
- Employees work in offices, there are co-working relationships
Example: Office LDSL

Textual Syntax

office 2080

employee Prof. Assmann
  works in 2087

office 2087

Model Level

Metamodel Level

Metamodelling Space

OfficeModel :=
  (employees:Employee*,
   offices:Office*)

Office := (name: String);
Employee := (name: String,
            title: String, worksIn: Office,
            worksWith: Employee*);
Example: Office LDSL

Textual Syntax

office 2080
employee Prof. Assmann
   works in 2087
office 2087

Model Level
Metamodel Level
Grammar Space

OfficeModel :=
   (employees|offices)*;
Office := "office" name;
Employee := "employee" title name
   "works in" worksIn
   "works with" worksWith*;

Metamodelling Space

OfficeModel :=
   (employees:Employee*,
    offices:Office*);
Office := (name: String);
Employee: (name: String,
   title: String, worksIn: Office,
   worksWith: Employee*);

Logic-Based DSL
Example: Office

Textual Syntax

OfficeModel := (employees|offices)*;
Office := “office“ name;
Employee := “employee“ title name
“works in“ worksIn
“works with“ worksWith*;

Office := (name: String);
Employee: (name: String,
title: String, worksIn: Office,
worksWith: Employee*);

Model Level

Metamodel Level

Grammar Space

Metamodelling Space

Ontology Space

Constraints
ProfessorsRoom
ProfessorsAssistants
PhDCoworkers

Logic-Based DSL
Example: PetriNets LDSL

PetriNets DSLL

- Language to describe the dynamic semantics of distributed systems
- Places and transitions connected by arcs
- Consuming arcs take tokens from places
- Producing arcs put tokens to places
- Transition fires if all incoming arcs have tokens

Graphical Syntax

Textual Syntax

place P1; place P2; place P3; place P4;
transition T1; transition T2;

from P1 to T1 {...};
from T1 to P2 {...};
from T1 to P3 {...};
...
from T2 to P4 {...};
from T2 to P1 {...};
Example: PetriNets LDSL

Graphical Syntax

Metamodelling Space

Textual Syntax

place P1; place P2; place P3; place P4;
transition T1; transition T2;

from P1 to T1 {...};
from T1 to P2 {...};
from T1 to P3 {...};
...
from T2 to P4 {...};
from T2 to P1 {...};
**Example: PetriNets LDSL**

**Graphical Syntax**

```
place P1; place P2; place P3; place P4;
transition T1; transition T2;
```

**Textual Syntax**

```
from P1 to T1 {...};
from T1 to P2 {...};
from T1 to P3 {...};
...
from T2 to P4 {...};
from T2 to P1 {...};
```

**Model Level**

**Metamodel Level**

**Grammar Space**

```
Arc := from "->" to
Transition := "transition" name;
Place := "place" place
":""type;
```

**Metamodelling Space**

```
Nameables := (name: String);
Components : Nameable :=
  (in: Arc*, out: Arc*);
Transition : Component;
Place : Component (type: EClass);
Arcs :=
  (from:Component, to:Component)
```

**Logic-Based DSL**
Example: PetriNets LDSL

**Graphical Syntax**

- **Model Level**
  - PetriNet diagram with places P1, P2, P3, P4 and transitions T1, T2.

**Textual Syntax**

- **Textual Syntax**
  - ```
  place P1; place P2; place P3; place P4;
  transition T1; transition T2;
  from P1 to T1 {...};
  from T1 to P2 {...};
  from T1 to P3 {...};
  ...
  from T2 to P4 {...};
  from T2 to P1 {...};
  ```

**Metamodel Level**

**Grammar Space**

- Arc := from "->" to Transition := "transition" name;
- Place := "place" place "":"type;

**Metamodelling Space**

- Nameables := (name: String);
- Components : Nameable := (in: Arc*, out: Arc*);
- Transition : Component;
- Place : Component (type: EClass);
- Arcs := (from:Component, to:Component)

**Ontology Space**

- Global Constraints
- Invalid Arcs
- Consuming Arcs
- Producing Arcs
- Domain Specific Constraints
- Start Places
- Disallow Parallelism
2.5 Example of a Graphical LDSL:

Feature Models for Tool Families

Christian Wende, Uwe Aßmann
Challenges for Product Families

- Build many tools with common platform
- Strengthen reuse by sharing commonalities

- Control by configuration ontology

MOST TOPF (Tool Product Family)

-> Ontology-controlled tool configuration
Development Process in MOST TOPF Tool Family

(S1) Variability Specification

Feature model for Most TOPF

MOST TOPF Engineering

MOST TOPF Instantiation

Variant model for concrete product

(S2) Feature Realisation

MOST TOPF architecture

Feature-driven Derivation

Concrete MOST TOPF product

(S3) Feature Mapping

MOST TOPF mapping model

(S4) Variant Specification

(S5) Variant Derivation

Input for

Relates to
**MOST TOPF Graphic Feature Model**

- Enables systematic classification and documentation of commonalities and variability
- Enables explicit management of feature dependencies
- Enables easy and systematic configuration of generic tools
- Propositional logic, translates to ontology

**Ontology Technology**

- Pellet
- Fact+
- TrOWL
- REL

**Ontology Language**

- OWL Full
- OWL DL
- OWL Lite
- OWL FA

**Development Method**

- PDIDSL
- BEDSL
- MEDSL
- DBDSL
- BPMN

**Software Process Guidance**

- Comarch Network Configuration
- Comarch OSS Development
- Business Process Development

**Automation**

- Specification Validation
  - Specification Transformation

**Modelling Language**

- PDDSL
- PDDSL
- BEDSL

**BPMN**

- (Specification Validation and OWL DL)
  - or not (Specification Validation)
MOST TOPF Architecture
MOST TOPF Architecture

- Component-based architecture modelled using UML
- Defines blue-print for layered ODSD tool architecture
- Platform-independent – can be implemented using various tool platforms (e.g. AdoXX, Eclipse)

(S2) Feature Realisation

Ontology-Aware Workbench

Model and Ontology Editors (Graphical/Textual)

Access Layer

Model-aware and Ontology-aware Mechanisms

TGraphs Modelling Infrastructure

ADOxx Modelling Infrastructure

Ontology Layer
**Variant Model for Concrete Demonstrator**

- Variant Editor guides tool customisation
- Interactively checks and annotates consistency w.r.t. feature model and constraints
- Provides repair suggestions
(S3) Feature Mapping

MOST TOPF mapping model
- Interactive, graphical mapping approach
- Mapping of boolean feature terms to elements of Ecore-based models
- Mapping stored in a dedicated, external mapping model
(S3) Feature Mapping
Feature-Driven Demonstrator Derivation

- Variant model is used together with feature mapping and component diagram to derive variant specific component diagram.
Lessons Learned

Feature models form a graphical, logic-based DSL (LDSL) for specification of software product lines

- **Conciseness**: Concise feature model (DSL, boolean expressions)
- **Automatic checking**
- **Communicative**: Unified and clarified terminology used in communication
- **Interactivity**: Interactive and graphical tooling helped visualising the mapping without interfering with UML syntax

Logic-based DSL provide simple configurability

- **Uniformity**: Extension of feature model were quite similar for each demonstrator
- **Harmlessness**: Component-based infrastructure reduced threat of feature interaction
- **Evolvability**: External mapping model helped extending the architecture without breaking the mapping
- **Stability**: Unique identifier for solution space elements required to provide stable mapping
- **Complexity**: Complex mappings with terms hard to visualise
2.6 A Feature-Modeled Family of Ontology Languages

Christian Wende and Florian Heidenreich, TU Dresden

(MDPLE Workshop, Twente, 2009)
Causes of the Scalability Problem

- Computational Complexity
  - language expressiveness
  - language features

- Combined Data Complexity
  - facts size
  - axioms size
  - queries complexity

- Ontology

- Language

- Knowledge Base

- Queries

Use Case implies Language Customisation

Logic-Based DSL
Existing Approaches to Achieve Scalability

Language Design
- SHOQ(D)
- OWL-Full
- OWL2
- OWL-Lite
- EL++
- OWL-FA
- OWL-DL
- DL-Lite
- RDF(S)

Reasoning Infrastructure
- Pellet
- RACER
- FaCT++
- Minerva
- Sesame
- TrOWL
- Syntactic Approximation
- Semantic Approximation

Logic-Based DSL
Current Status: An Ontology Language Hierarchy

- Subset-of relationships w.r.t syntax and semantics
- Trade-off between expressiveness and reasoning complexity
- Implemented mostly independent

Goal: An Ontology Language Family

Problem Space
- Systematic organisation of family members
- Specific expressiveness and reasoning requirements can be addressed by recombining existing language features

Solution Space
- Systematic reuse of common language features
- Generating language tooling (e.g., dedicated parsers, printers, editors, and reasoners)
- Language evolution by contributing new features
Realising an Ontology Language Family

Language Customisation

Feature-based Family Specification  Mapping  Model-Based Language Specification

Language Family Engineering

Language Instantiation

Use Case Requirements  Feature-based Variant Selection  Transformation of Language Specification  Requirements Specific Language

Logic-Based DSL
Feature Model to specify Ontology Language Families

- Ontology languages are classified by logical constructors (language features) they provide
- Feature Model restricts combinatorial variability to valid variation space (language family)
- Expressiveness and Computational Complexity (Scalability) can be derived from used constructors

![Diagram of Description Logics and their constructors]

\[ \text{Problem Space of Ontology Language Family} \]

\[ \text{SHOIN(D+)} \equiv \text{OWL DL} \]

\[ \text{ALCR+} \equiv S \]

- Reduced feature models describe specific ontology language family member
Dimensions of a Language Specification

- **Abstract Syntax** – domain concepts relations and properties (OWL metamodel)
- **Concrete Syntax** (OWL parser)
- **Language Semantics** (OWL reasoner)

Benefits of a Model-based Approach

- Provides means for transformation (Variant Refinement)
- Provides means for code generation (Tool Generation)
- Provides infrastructure based on metamodels (Tool Interoperability)

**Concrete Syntax**
- **Ontology**: http://ex.org//pizza.owl
- **Class**: Pizza
- **Spiciness**
- **ObjectProperty**: hasSpiciness
  - **Range**: Spiciness

**Abstract Syntax**
- OWL
- Ontology
- Frame
- Property
- OWLClass

**Semantics**
Mapping Features to Language Specifications

- During Family Engineering single Ontology features are mapped to artefacts in the language specification models
- This mapping is stored in a Mapping Model

Problem Space

OWL Featuremodel

- Concepts
- DescriptionLogics
- Roles \( \mathcal{R} \)
- Minimal
- Top
- Bottom
- Intersection
- ValueRestriction
- AtomicNegation

Solution Space

OWL Metamodel

- ObjectProperty
- ObjectPropertyReference
- InverseProperties
- inverse

OWL Syntax Specification

ObjectProperty ::= "ObjectProperty:" iri[IRI] !1 ((annotations !1) | ... | ("InverseOf:" inverseProperties ("," inverseProperties)* !1))

OWL Mapping Model

Logic-Based DSL

Christian Wende, TU Dresden
Mapping between Problem and Solution Space

FeatureMapper: Tool Support for Mapping

Logic-Based DSL

Christian Wende, TU Dresden
Transformation Process for Language Instantiation

OWL Feature Model

< variant of >

OWL Variant Model

< maps to >

OWL Metamodell

< refers to >

OWL Textual Syntax

< maps to >

FeatureMapper

Model Transformation

OWL Variant Metamodell

< refers to >

OWL Variant Concrete Syntax

< refers to >

EMFText

Code Generation

OWL Variant Parser

OWL Variant Printer

OWL Variant Editor
Feature-Based Customisation for Families of Ontology Languages

- Ontology Languages are used for different Use Cases
- Use Cases imply data complexity

- Use Cases should also guide computation complexity
- Feature-Based Customisation provides means for
  - Use Case driven language customisation

- Feature-Based Customisation is also supports
  - Ontology Language Extension
  - Ontology Language Evolution
  - Ontology Language Integration

- Research Questions
  - How can we compose reasoners w.r.t. language features?
  - How to get from Use Case to required features?
2.7 LDSL as Language Families
How do I teach Chuck to bring me a coffee?

with a family of Gesture-LDSD
Gesture DSL will be Domain-Specific and should be Logic Based

- Gestures in Cars
- Gestures in Public
- Gestures in Factories

- Context modeling
- User editable
- Semantics
Logic-Based DSLs

OWLGraphics

Attribute-Grammar based Compiler environments

Eclipse based toolkits

Parsing environments for DSL

Hand implementation of DSL

OWLText

OWL Gestures

Family LDSL

Logic-Based DSL

OWLGraphics

Attribute-Grammar based Compiler environments

Eclipse based toolkits

Parsing environments for DSL

Hand implementation of DSL

OWLText

OWL Gestures

Family LDSL
Logic-Based DSL Families

- Gestural Logic-Based DSL
- Graphical Logic-Based DSL
- Textual Logic-Based DSL
- Ontology-controlled Product families

Technical Space Bridging
LDSL need Bridging between Logics and Software Engineering

Fakultät Informatik Institut Software- und Multimediatechnik, Lehrstuhl Softwaretechnologie

Product family engineering

Logic-based Semantics

Adequate Logic languages

LDSL Families

DSL

Logic-Based DSL
• Many slides are courtesy to the MOST project
• www.most-project.eu
  – Tobias Walter (U Koblenz)
  – Srdjan Zivkovic, Harald Kühn (BOC Information Systems GmbH)
  – Krzysztof Miksa (Comarch)
  – And many others..
• Forthcoming Book with Springer
• “Ontology-Driven Software Development”
The End