MegaModeling
Software Language Engineering Artefacts

ongoing research work

Jean-Marie Favre, OneTree Technologies, Luxembourg
Dragan Gasevic, Athabasca University, Canada
Ralf Lämmel, University of Koblenz-Landau, Germany

GPCE/SLE 2010 Tutorial

We are smart

Since Stone Age

when we have problems

we invent some technology

Today


We have a problem
XPATH
TXL
Sesame
JPA
Jena
Rose
JDBC
EMF.gen
XText
jDOM
JAXB
Jersey
RDF(S)
UTF8
Jena
ODM
MOF
XSD
JeanBeans
Teneo
UML
VLDB
Stratego
BNF
xerces
GWT
SLE2010
Json
OCL
sax
Ralf
RDFS
saxon
OWL
OWL
Ecore
Rest
MySQL
XMI
ORACLE
RDF
Jean
OMG
JMI
EMF
JMF
xalan
ODBC
SparQL
XMLSpy
Yacc
RDFa
XSD
ArgoUML
XSLT
JAXP
LALR
Prolog
Protegé
CFG
SBVR
DOM
Java
ATOM
ER
SQL DDL
Dragan
SQL
Antlr
QVT
TENEO
XLST
Awk
DTD
TCS
sed
Saxon
grep
ASCII
XSD
Hibernate
XQuery

Today's Issues

- **Silos of knowledge**
- **Combining technologies**
- **Complexity of technologies**
- **Teaching technologies?**
- **Entering a new space**

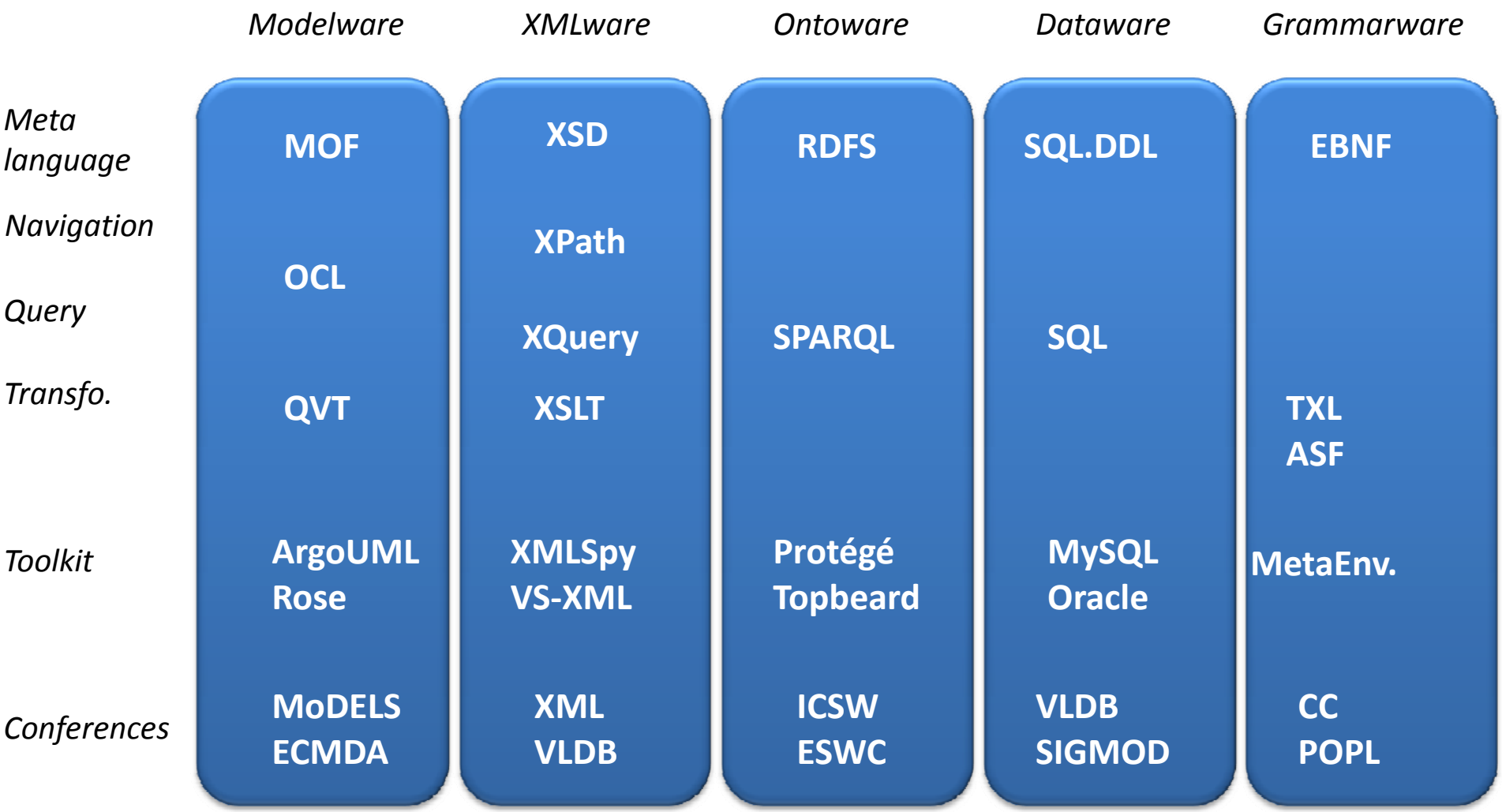
Today's Issues

- Silos of knowledge
- Combining technologies
- Complexity of technologies
- Teaching technologies?
- Entering a new space

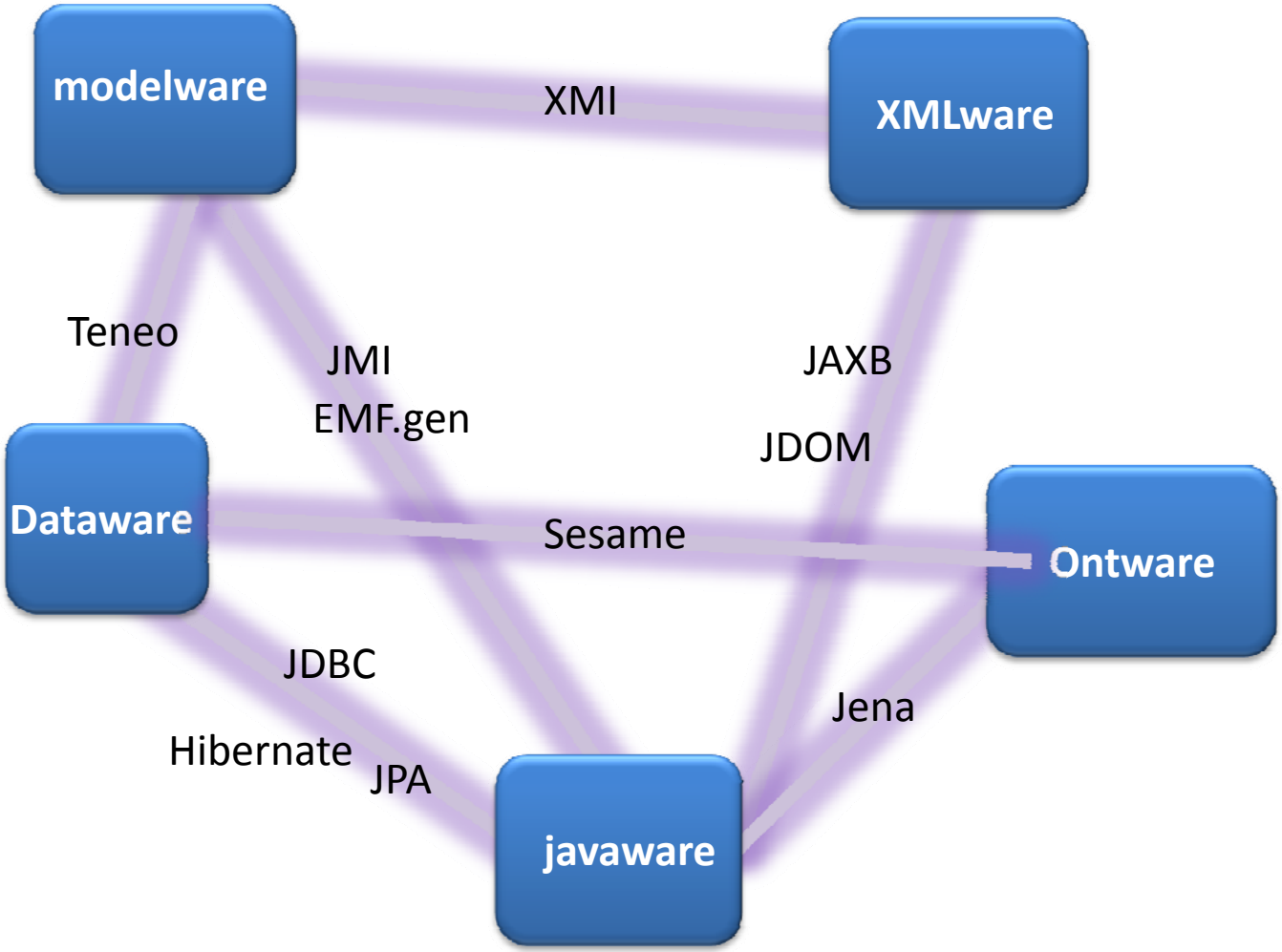
Our Approach

- Analogy
- Abstraction
- Common example

Working by analogy



From one space to another…

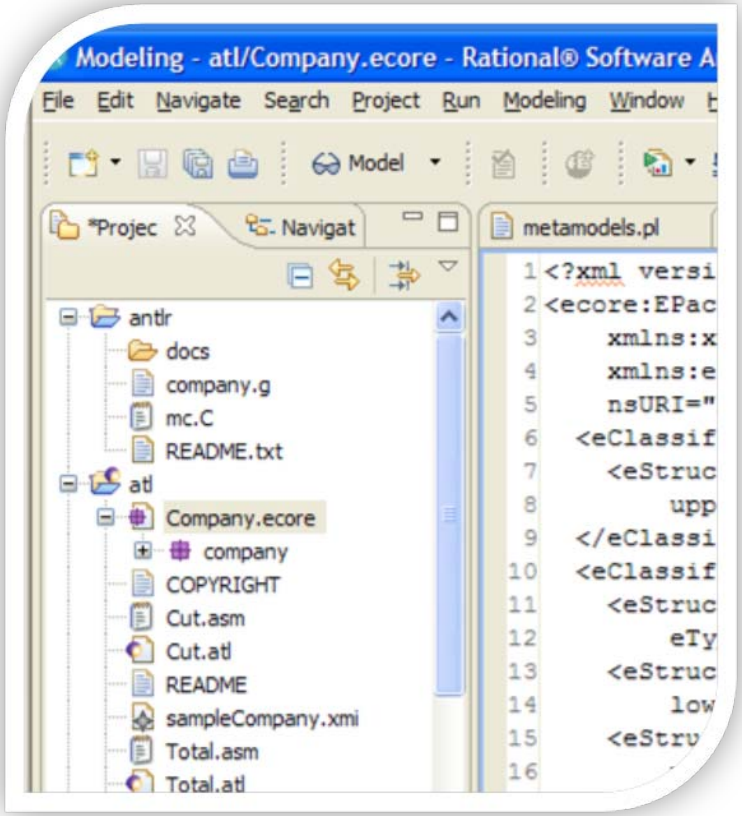


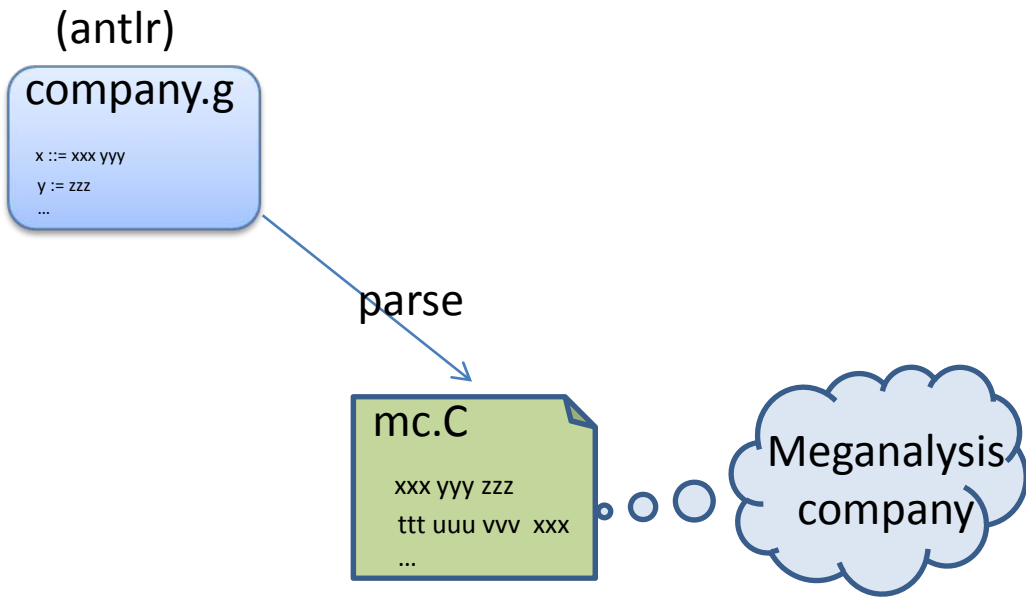
We need to be more precise

We need
a « theory »


We need
megamodeling
XPATH
TXL
Sesame
JPA
Jena
Rose
JDBC
EMF.gen
XText
jDOM
JAXB
Jersey
RDF(S)
UTF8
Jena
ODM
MOF
XSD
JeanBeans
Teneo
UML
VLDB
BNF
Stratego
xerces
GWT
SLE2010
Json
OCL
sax
Ralf
RDFS
saxon
OWL
Rest
MySQL
XMI
ORACLE
RDF
Jean
OMG
JMI
EMF
JMF
xalan
ArgoUML
XSLT
CFG
Java
ATOM
ER
SQL DDL
Dragan
SQL
Antlr
QVT
TENEO
XLST
Awk
DTD
TCS
sed
Saxon
grep
ASCII
XSD
Hibernate
XQuery

Empirical megamodeling





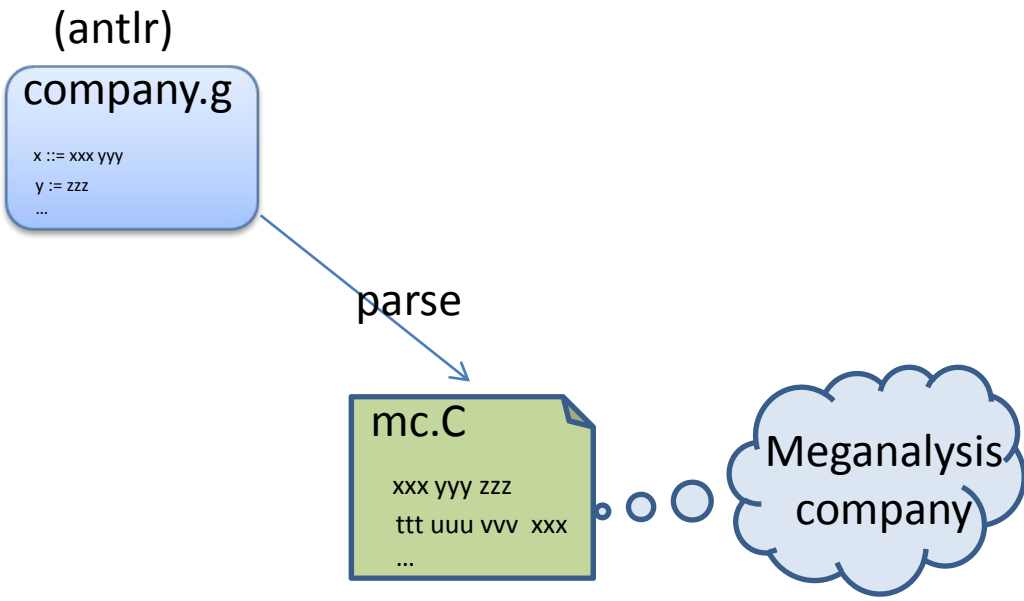
A megamodel

Specific megamodeling

$w \in L(G_c)$

$G_c \in A$

...



a megamodel

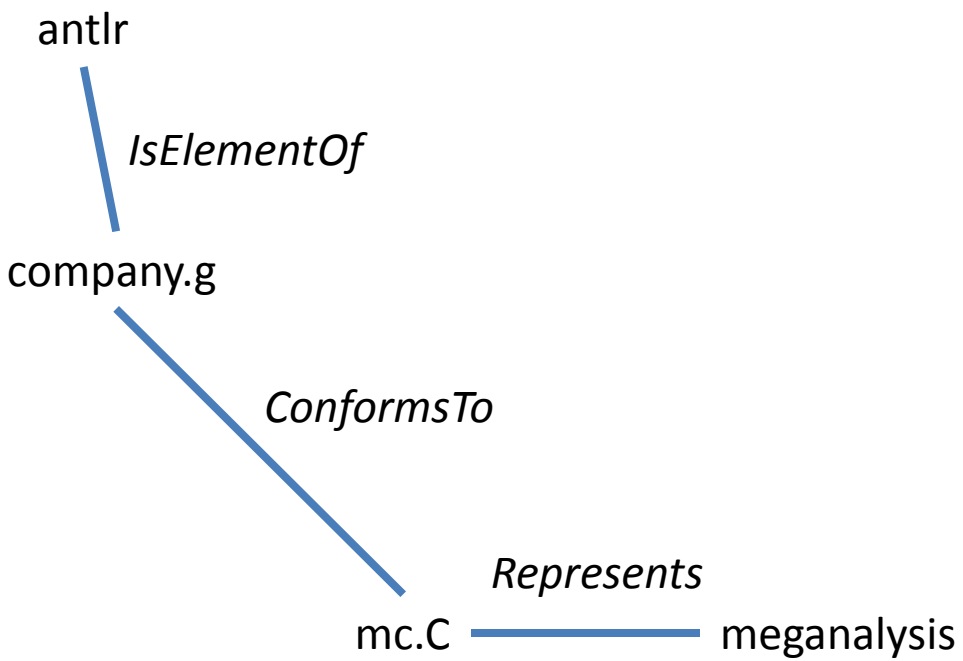
a megamodel

Precise megamodeling

Represents

ConformsTo

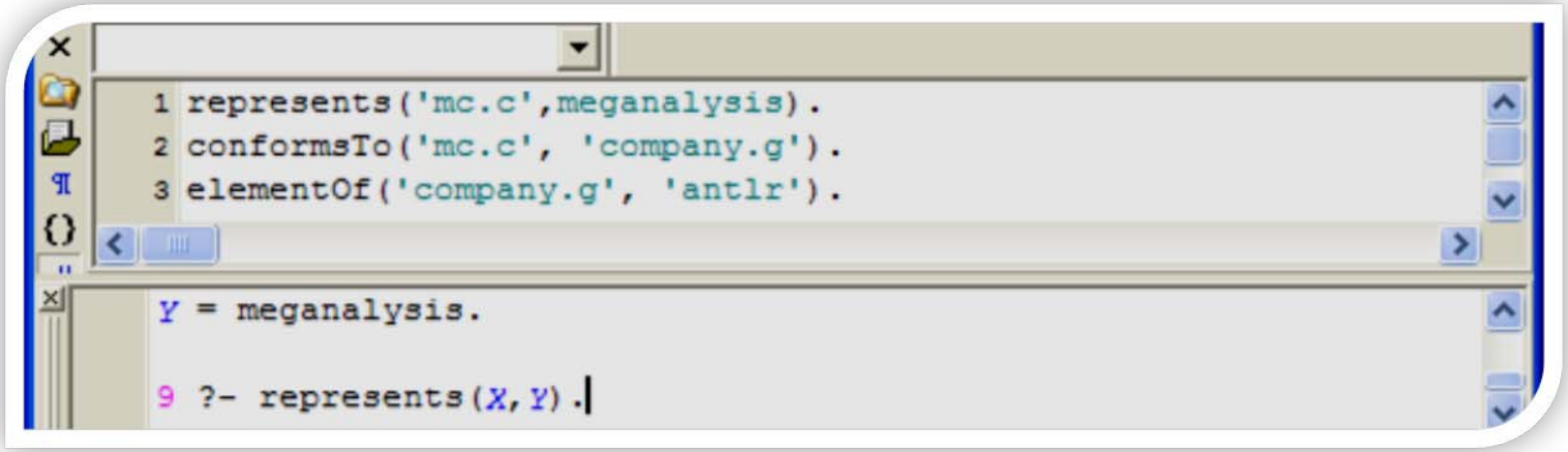
IsElementOf

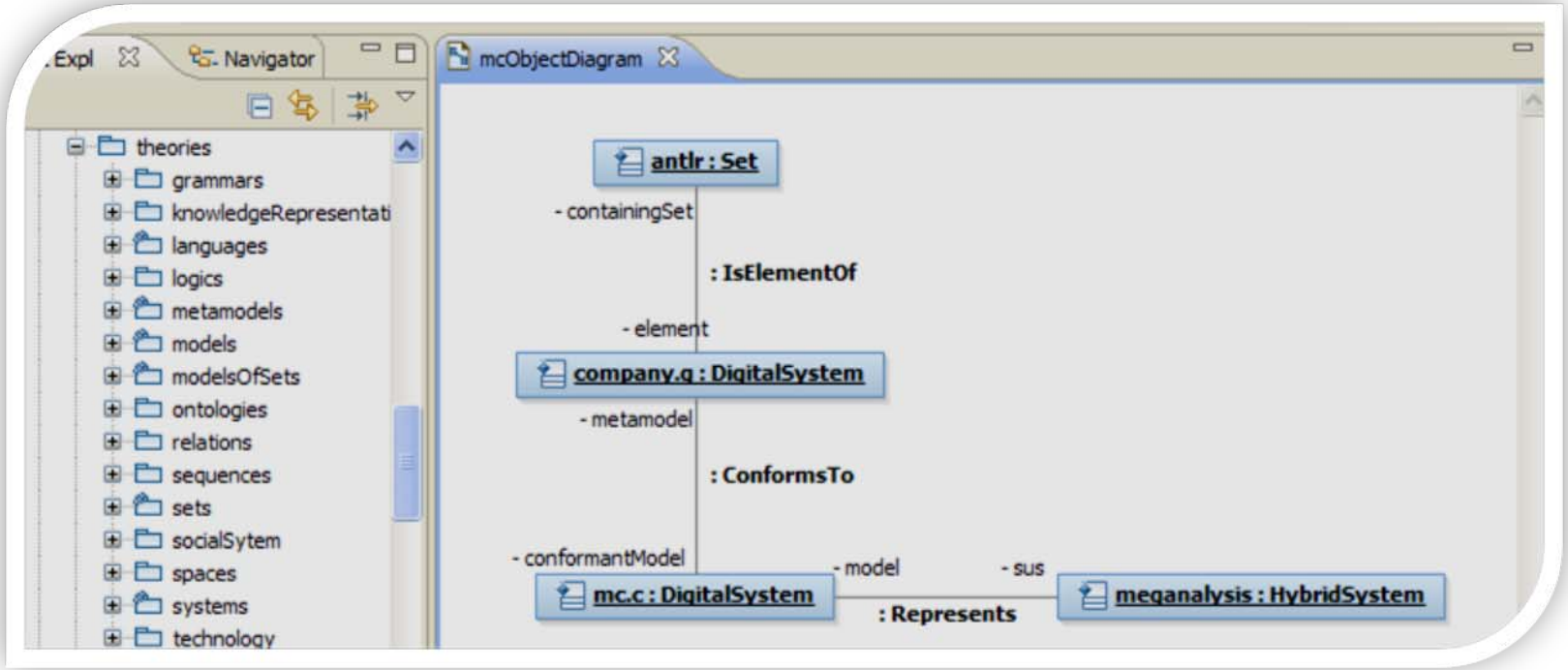


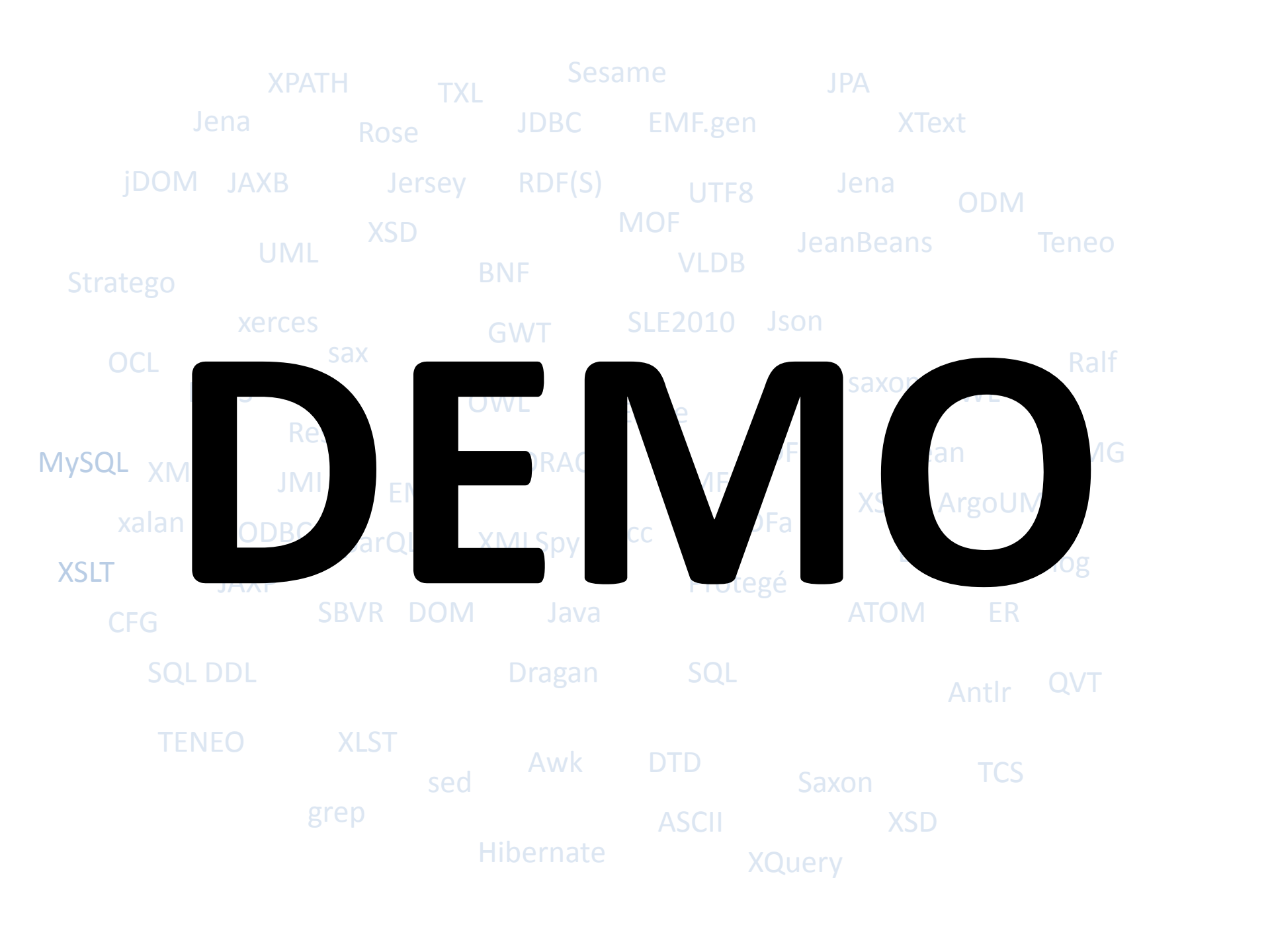
THE megamodel
metamodel

a megamodel

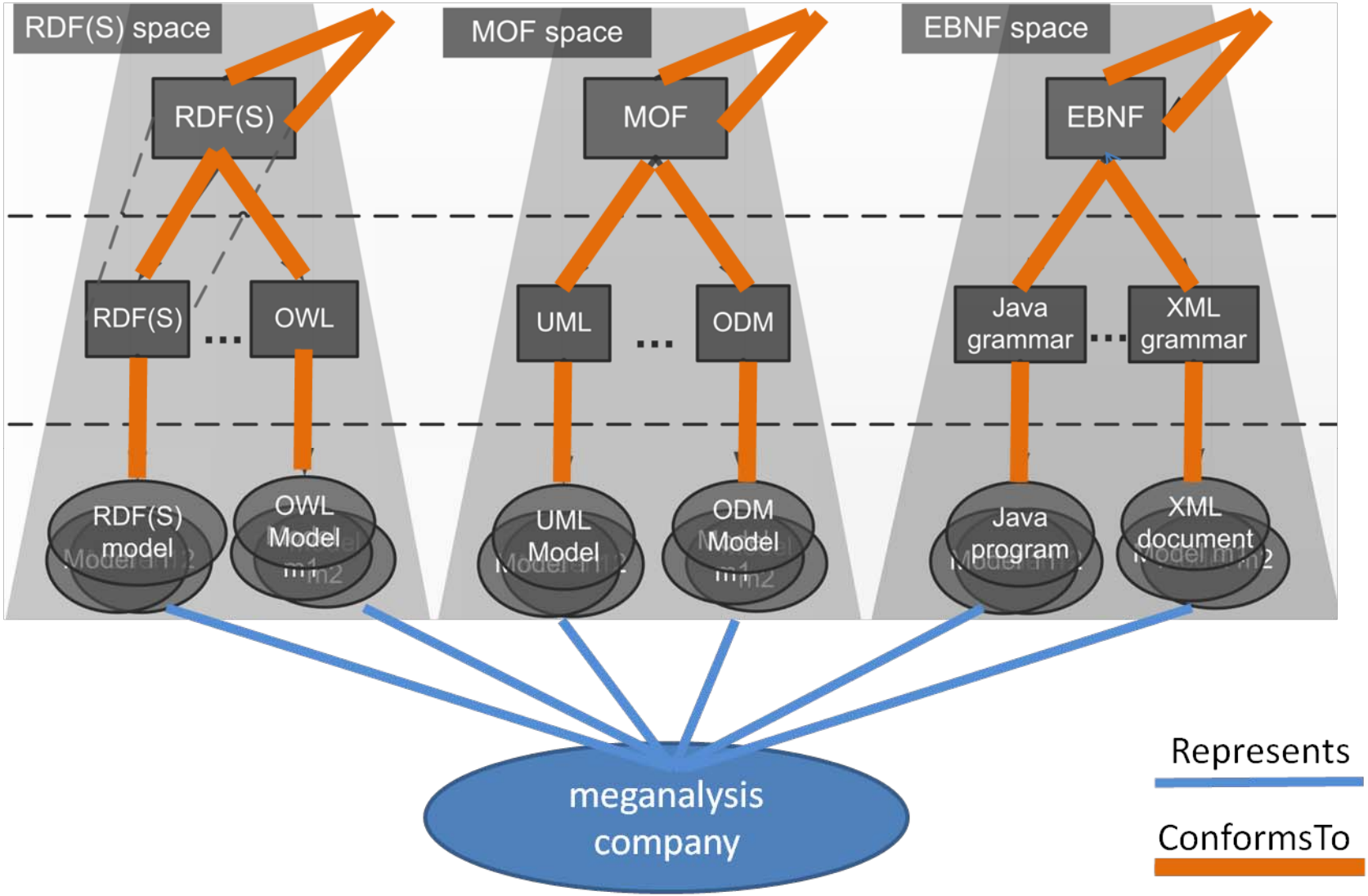
Multi-language megamodeling



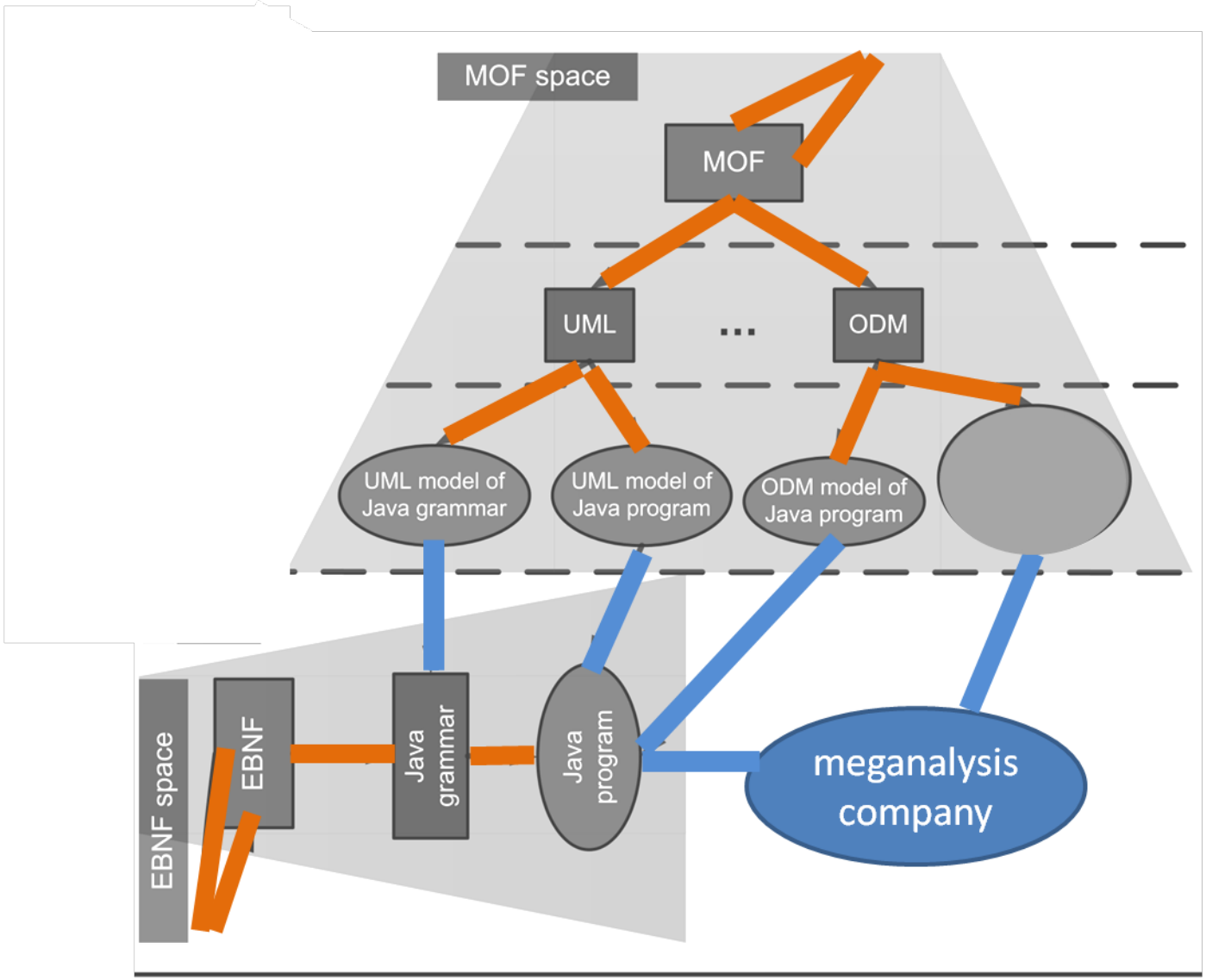



DEMO
XPATH
TXL
Sesame
JPA
Jena
Rose
JDBC
EMF.gen
XText
jDOM
JAXB
Jersey
RDF(S)
UTF8
Jena
ODM
MOF
XSD
JeanBeans
Teneo
UML
VLDB
Stratego
BNF
xerces
GWT
SLE2010
Json
OCL
sax
Ralf
OWL
MySQL
JMI
ArgoUM
xalan
ODBC
XMLSpy
XSLT
Protegé
CFG
SBVR
DOM
Java
ATOM
ER
SQL DDL
Dragan
SQL
Antlr
QVT
TENEO
XLST
Awk
DTD
TCS
sed
Saxon
grep
ASCII
XSD
Hibernate
XQuery

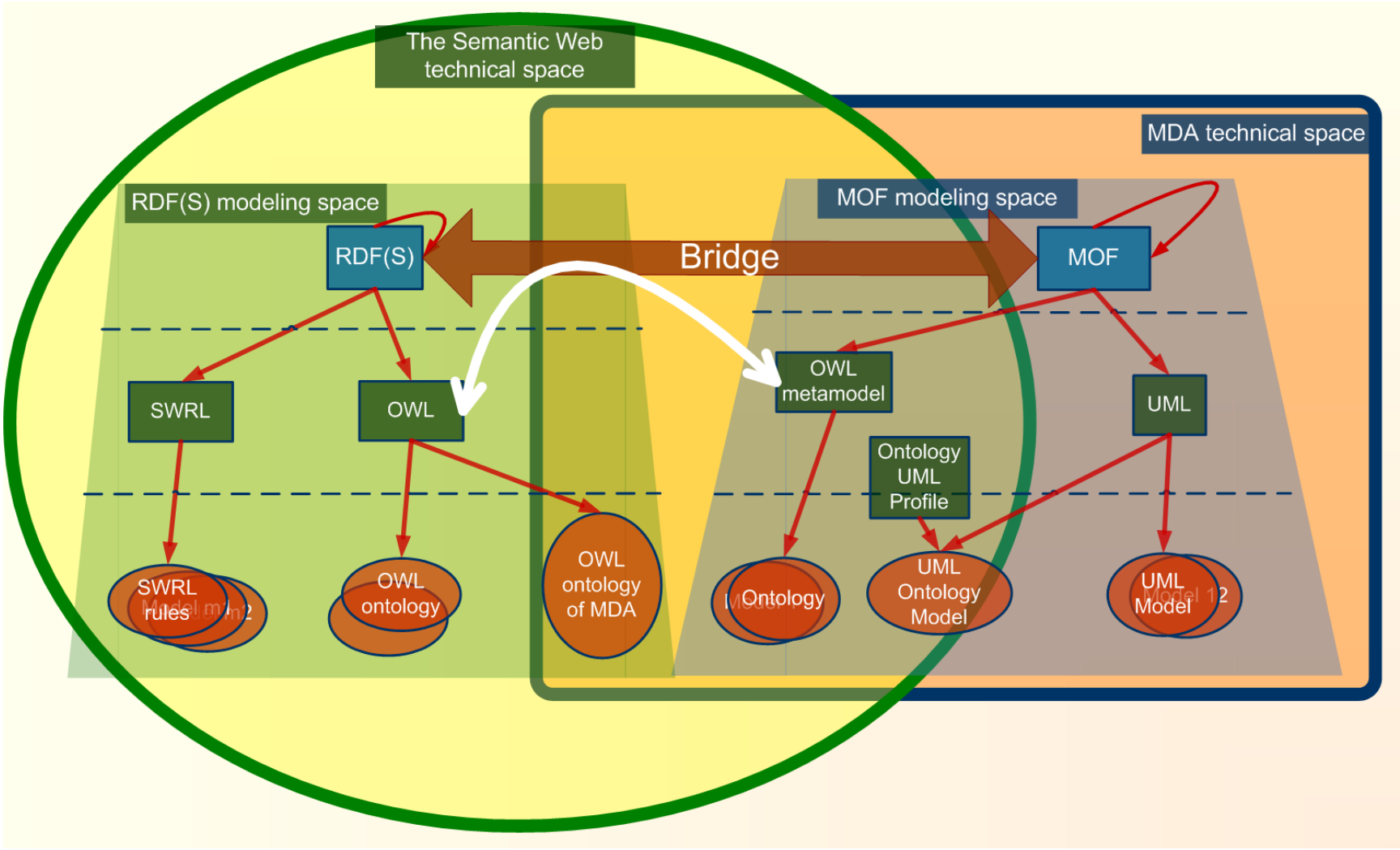
Parallel Spaces



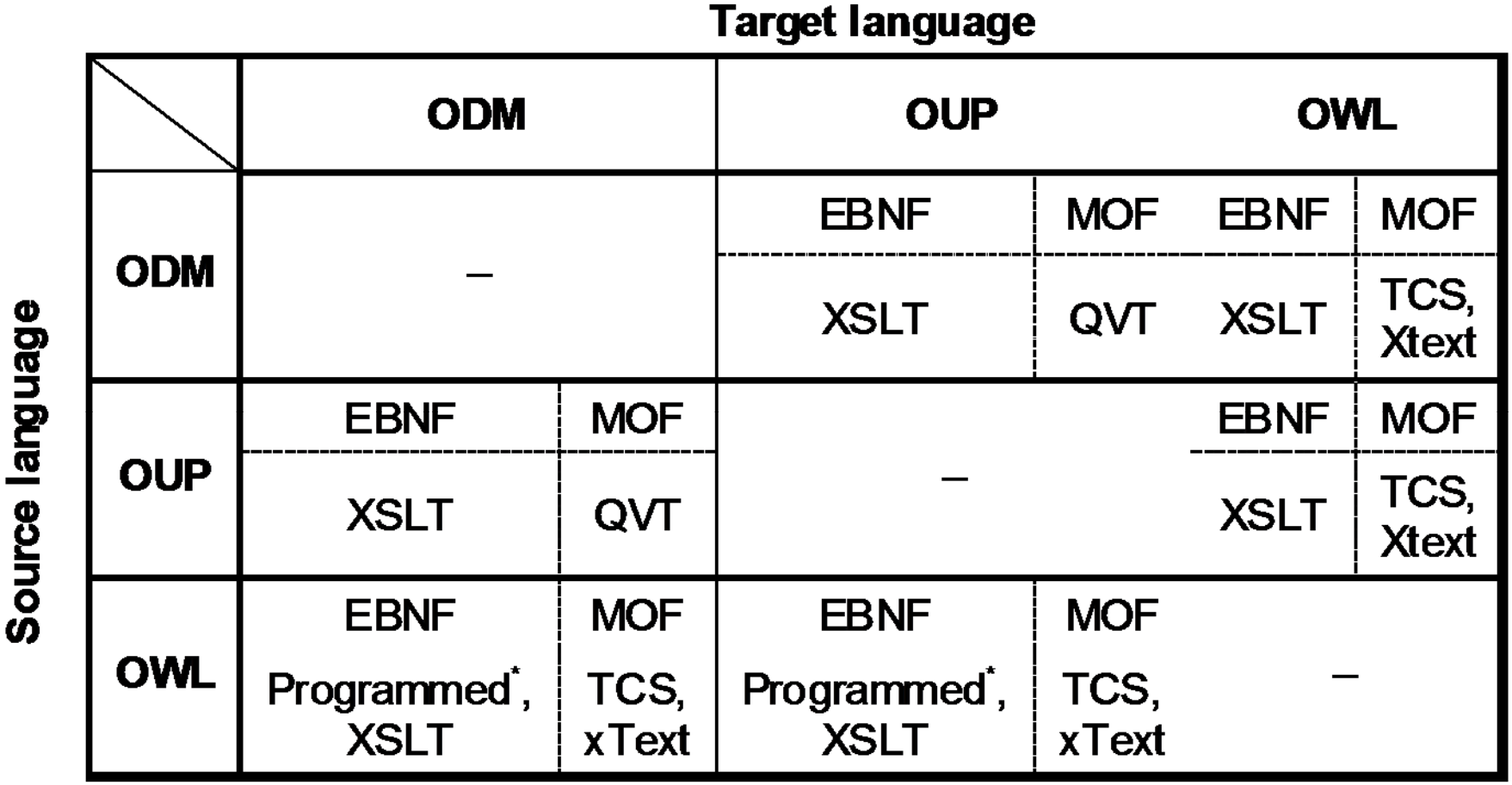
Orthogonal spaces



Bridges between spaces



Example of bridges between languages



We need
a catalog of
concrete examples

101companies

Summary

http://sourceforge.net/apps/mediawiki/developers/index.php?title=101companies

- *101companies* is a software corpus for company modeling and processing.
- Many different models and scenario implementations are exercised.
- The diversity feeds into a major megamodeling effort.

What's a company?

- A *company* is a nested structure of *departments* with *employees* as leafs.
- Employees are characterized by *name*, *salary*, and possibly other properties.
- Companies and departments have names, too.
- Each department has a *manager*.
- Employees may be associated with *mentees*.

Implementations may differ with regard to the level of detail.

A sample company

This company is named ***meganalysis***.

For what it matters, *meganalysis* is into megamodeling (as opposed to selling ice cream).

We only capture some basic structural facets of *meganalysis* below.

```
company "meganalysis" {
        department "Research" {
                manager "Craig" {
                        address "Redmond"
                        salary 123456
                }
                employee "Erik" {
                        address "Utrecht"
                        salary 12345
                }
                employee "Ralf" {
                        address "Koblenz"
                        salary 1234
                }
        }
        department "Development" {
                manager "Ray" {
                        address "Redmond"
                        salary 234567
                }
                department "Dev1" {
                        manager "Klaus" {
                                address "Boston"
                                salary 23456
                        }
                        department "Dev1.1" {
```

...

Company scenarios

- *total*: Total all salaries in a company.
- *cut*: Cut all salaries in half.
- *depth*: Determine depth of department nesting.
- *containment*: Check that tree topology holds for the instance.
- *precedence*: Check that salaries increase with rank in hierarchy.

Implementations do not need to cover all scenarios.

Implementations

All implementations are *labeled* for consistency of reference.

The implementations are listed in alphabetical order.

- *alpha*: a simple POJO object model for companies with methods for some of the scenarios.
- *antlr*: an ANTLR-based acceptor for a human-readable notation for companies.
- *antlr2*: a variation on *antlr* that actually constructs ASTs over some generic object model.
- *antlr3*: a variation on *antlr2* that constructs ASTs according to an object model for companies.
- *atl*: Ecore/ATL-based model transformations for some of the company scenarios.
- *atl2*: a variation on *atl* that uses a slightly different metamodel (with proper subtyping).
- *atl3*: a variation on *atl[2]* that uses KM3 instead of Ecore; this option is potentially obscure.
- *dom*: in-memory XML processing in Java with the DOM API.
- *emf*: EMF/Java-based model queries and transformations.
- *gwt*: C/S (Browser/Server) architecture for a WebApp based on GWT (Google Web Toolkit).
- *haskell*: model companies and implement company scenarios in Haskell 98 + SYB (using GHC).
- *hibernate*: maintain companies in RDBMS and access them through hibernate's O/R mapping.
- *hibernate2*: variation on *hibernate* to illustrate a different O/R mapping.
- *jaxb*: represent companies in XML and access them through JAXB's XML data binding.
- *jaxb2*: variation on ``jaxb *to illustrate a different X/O mapping*.
- *jdbc*: Relational database programming in Java with JDBC.
- *jdbc2*: A sophistication of *jdbc* to approach to O/R mapping in a homegrown manner.
- *jena*: in-memory RDF processing in Java with the Jena API (RDF part).
- *jena2*: a further use of *jena* which leverages Jena's query engine / ARQ implementation of SPARQL.
- *library*: a collection of third-party libraries that are leveraged by the implementations.
- *prolog*: model companies and implement company scenarios through logic programming in SWI-Prolog.
- *sax*: push-based XML processing in Java with the SAX API.
- *sql*: SQL DML-based implementation of some of the company scenarios.
- *swing*: a simple GUI for navigating companies and performing scenarios based on Swing/AWT.
- *xpath*: in-memory XML processing in Java with the XPath embedding into DOM.
- *xslt*: XSLT-based XML processing.
- *xquery*: XQuery-based XML processing.

Implementations

All implementations are *labeled* for consistency of reference.

The implementations are listed in alphabetical order.

- *alpha*: a simple POJO object model for companies with methods for some of the scenarios.
- *antlr*: an ANTLR-based acceptor for a human-readable notation for companies.
- *antlr2*: a variation on *antlr* that actually constructs ASTs over some generic object model.
- *antlr3*: a variation on *antlr2* that constructs ASTs according to an object model for companies.
- *atl*: Ecore/ATL-based model transformations for some of the company scenarios.
- *atl2*: a variation on *atl* that uses a slightly different metamodel (with proper subtyping).
- *atl3*: a variation on *atl[2]* that uses KM3 instead of Ecore; this option is potentially obscure.
- *dom*: in-memory XML processing in Java with the DOM API.
- *emf*: EMF/Java-based model queries and transformations.
- *gwt*: C/S (Browser/Server) architecture for a WebApp based on GWT (Google Web Toolkit).

Code snippets

The *total* scenario in XQuery:

```
<result>
        {sum(//salary)}
</result>
```

The *cut* scenario in SQL DML:

```
UPDATE employee SET salary = salary / 2;
```

The *total* scenario with Jena's RDF API:

```
        public static double total(CompanyModel c) {
                double total = 0;
                StmtIterator i =
                        c.getModel().listStatements(
                                new SimpleSelector(
                                        null, c.SALARY, (RDFNode) null));
                while (i.hasNext()) {
                        Statement s = i.next();
                        total += s.getDouble();
                }
                return total;
        }
```

Next steps

- Please contribute:
  - Let's add some implementations at GPCE/SLE 2010.
  - Get in touch with us.
- Please leverage:
  - There is class-room material related to 101companies.
  - More profound textbook-like material in the planning.
- To be cont'd:
  - GPCE 2010 Keynote.

CONCLUSION

- Many technologies and silos of knowlege
- Huge essential and accidental complexity

- **Abstraction** (=> **megamodel**)
- **Common example** (=> **meganalysis** company)
- **Analogy**