Contenidos

- Introducción
  - Conceptos básicos: Definición de DSL y tipos de DSL
  - DSL y MDE
  - Técnicas de implementación
  - Creación de DSLs gráficos con herramientas basadas en metamodelado
  - Algunos estudios comparativos de herramientas

Suposición ¿razonable?

Todos los asistentes tienen conocimientos básicos de **modelado software** y **MDE**

¿Voluntarios para tutorial … ?

[JISBD] Se busca ponente de tutorial para JISBD-2017 Recibidos x

FRANCISCO RUIZ GONZALEZ <Francisco.RuizG@uclm.es> 18 e

para distjisbd

Algunas personas me han preguntado sobre la posibilidad de impartir un tutorial sobre "Herramientas para creación de lenguajes de modelado y/o DSLs en ECLIPSE", SIRIUS.
Es por ello que lanzo este mensaje para buscar voluntarios/as.
Si estas interesado en dar dicho tutorial te pido me envíes la propuesta urgente antes de 6 días.

Abajo pongo el llamado a tutoriales con detalles.
Saludos (y gracias a tope a los valientes que se ofrezcan)

****

XXII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2017)
San Cristóbal de La Laguna (Tenerife), del 19 al 21 de julio de 2017
http://fg.ull.es/sistedes2017/
****

SOLICITUD DE PROPUESTAS DE TUTORIALES
==========

En el marco de las JISBD 2017, se solicitan propuestas de tutoriales sobre temas relativos a la Ingeniería del Software y las Bases de Datos. El objetivo es brindar a los oportunidad de adquirir nuevos conocimientos y habilidades en áreas específicas de estas disciplinas. Por ello, es imprescindible que los tutoriales ofrecidos en las jorn proporcionen una utilidad clara a los participantes en temas de actualidad, ya sea en aspectos relacionados con la investigación o con la práctica profesional. La durac tutoriales podrá ser de 1'5 horas o de 3'0 horas, y se impartirán en paralelo a las demás sesiones de las Jornadas.
Las propuestas deberán enviarse al presidente del Comité de Programa (Francisco Ruiz, francisco.ruizg@uclm.es) por correo electrónico ~~antes de las 24h del 15 de er~~ indicando los siguientes contenidos:

- Título:
- Responsable: Nombre, filiación, email.
- Instructores: Nombre y filiación del resto de personas que lo llevaran a cabo (no pasar de 1 en tutoriales de 1'5 horas y de 2 en tutoriales de 3 horas).
- Objetivos: Indicar su motivación, relevancia, actualidad y destinatarios potenciales.
- Estimación de inscritos: Número de inscritos previstos (cifra realista).
- Duración: 1'5 o 3 horas (en caso de 3 justificar la necesidad).
- Contenidos: lista de temas tratados.
- Infraestructura: instalaciones (audiovisuales, máquinas, etc) necesarias para su realización.
- Material: material que será entregado a los participantes.

PROGRAM

This one-day free event will propose two tracks for both beginners and advanced Sirius users who will share best practices, insights, case studies, and innovations. By joining this community event, you will definitely learn a lot about Sirius!

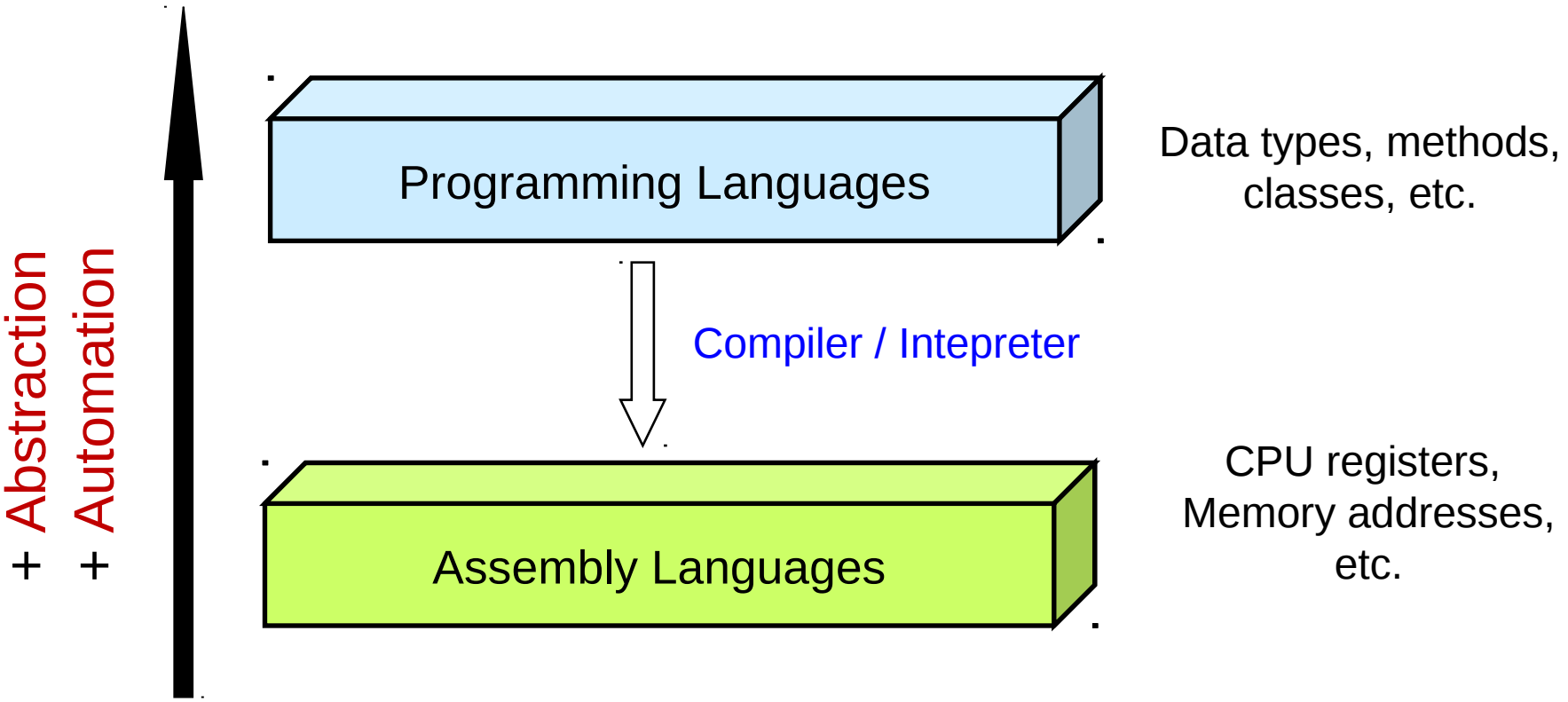
	ROOM 1 - CHAMPS-ÉLYSÉES/CONCORDE	ROOM 2 - OPÉRA
9h00 - 9h30	**Welcome Reception**	
9h30 - 10h00	**Welcome Speech** STÉPHANE LACRAMPE, Obeo	
10h05 - 10h35	**Collaborative Modeling** STÉPHANE BONNET, Thales	**Integrating Textual and Graphical Editing in the POOSL IDE** DR. IR. ARJAN MOOIJ, Embedded Systems Innovation by TNO
10h40 - 11h10	**ASML's MDE Going Sirius** WILBERT ALBERTS, ASML	**Visualization of Inferred Versioned Schemas from NoSQL Databases** ALBERTO HERNANDEZ, University of Murcia
11h10 - 11h40	**Break**	
11h40 - 12h10	**A Modelling Platform to Support Power Plant Life Cycle Management (PLM) for Nuclear Engineering** LUDOVIC LOUIS-SIDNEY, EDF JULIEN CHAMPEAU, EDF	**Extensible Sirius Editors for the Palladio Component Model** MISHA STRITTMATTER, Karlsruhe Institute of Technology
12h15 - 12h45	**Modelling Spacecraft On-board Software with Sirius** ANDREAS JUNG, European Space Agency	**V for Visualization in Viatra: The Good-old Integration of Graphical and Textual Representations** AKOS HORVATH, Inc Query Labs

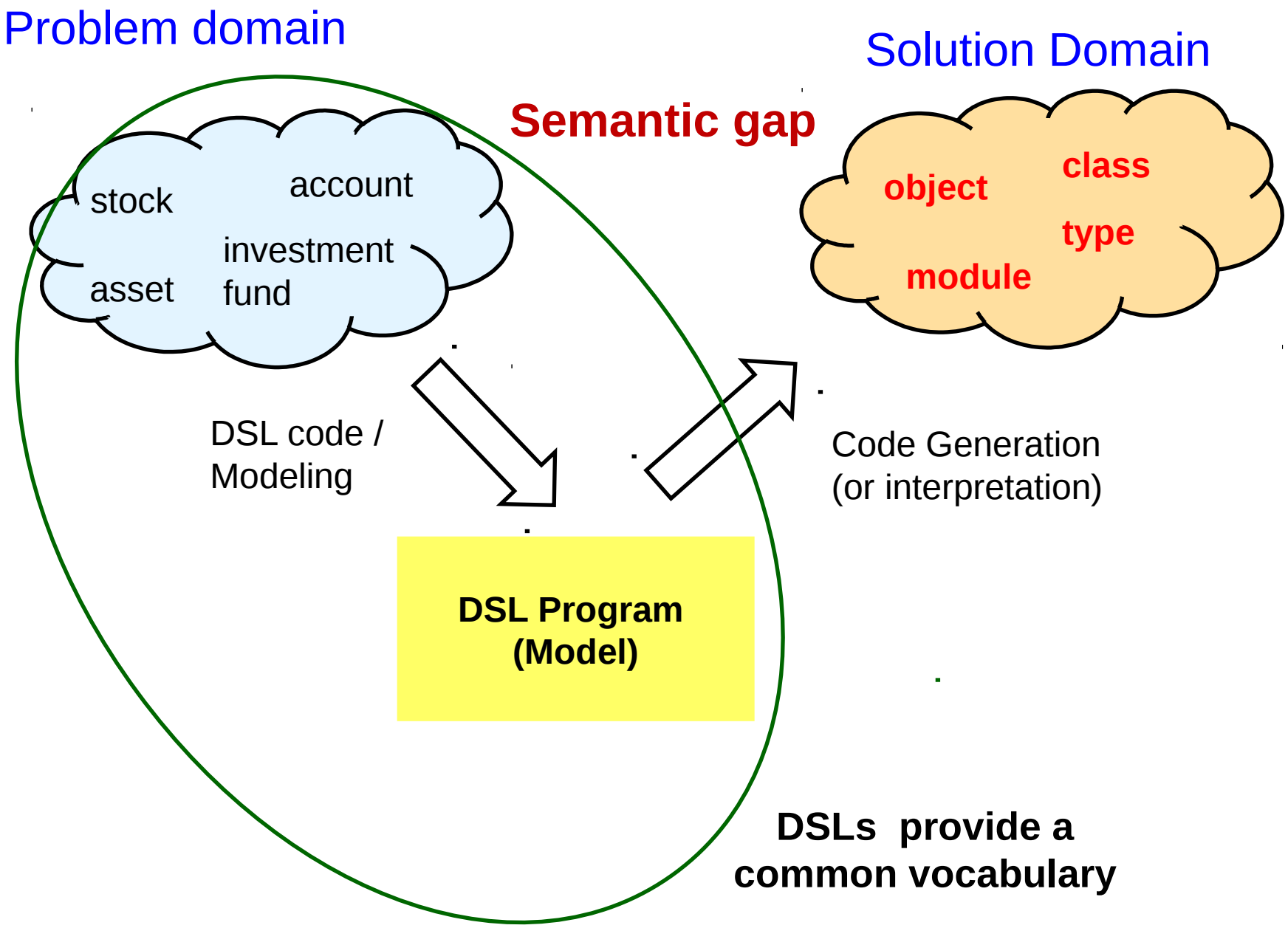
SIRIUS CLINIC

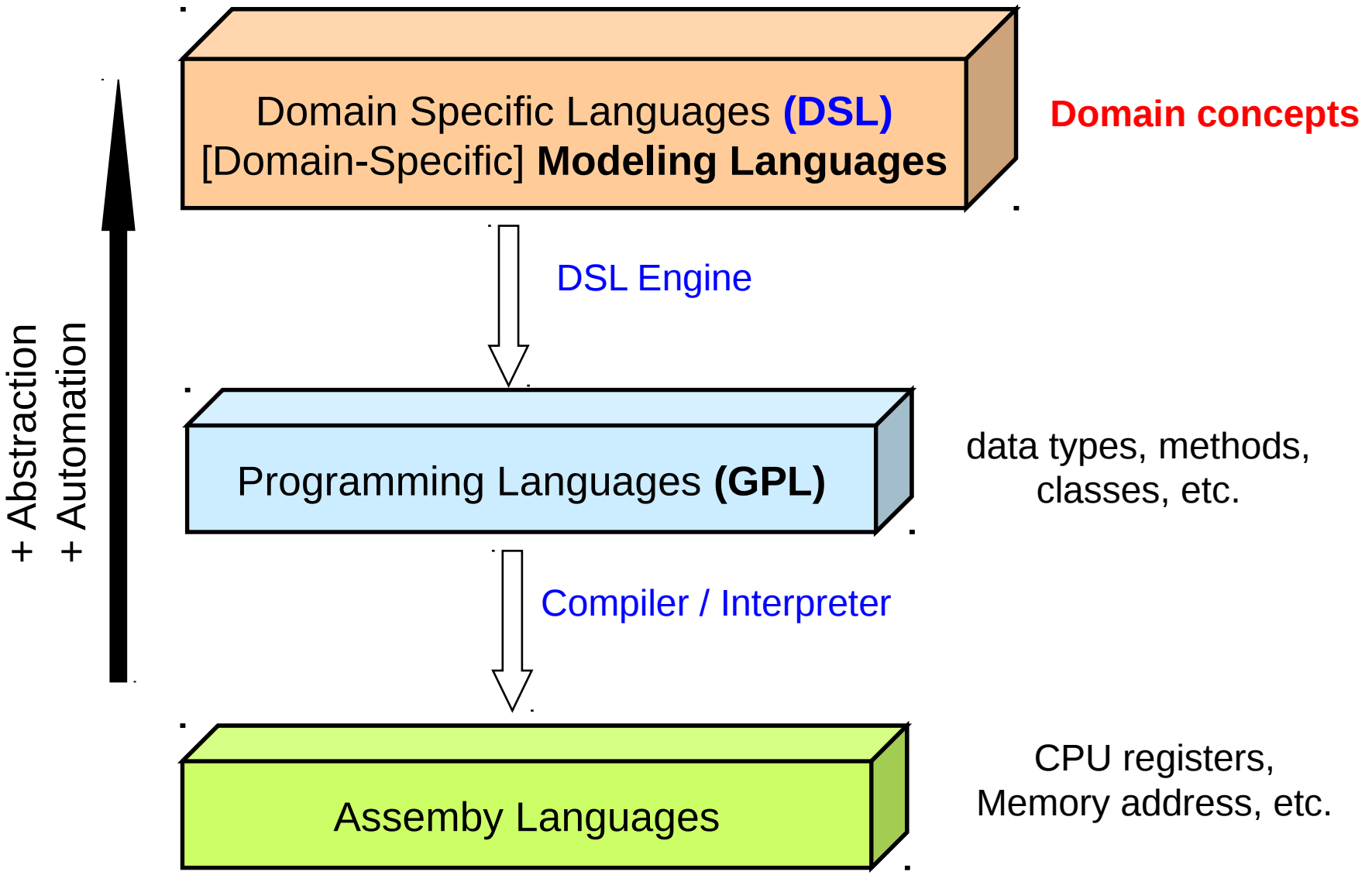
Petición de tutorial

“Herramientas para creación de lenguajes de modelado **y/o** DSLs en ECLIPSE, tipo EUGENIA y SIRIUS.”

Language Evolution



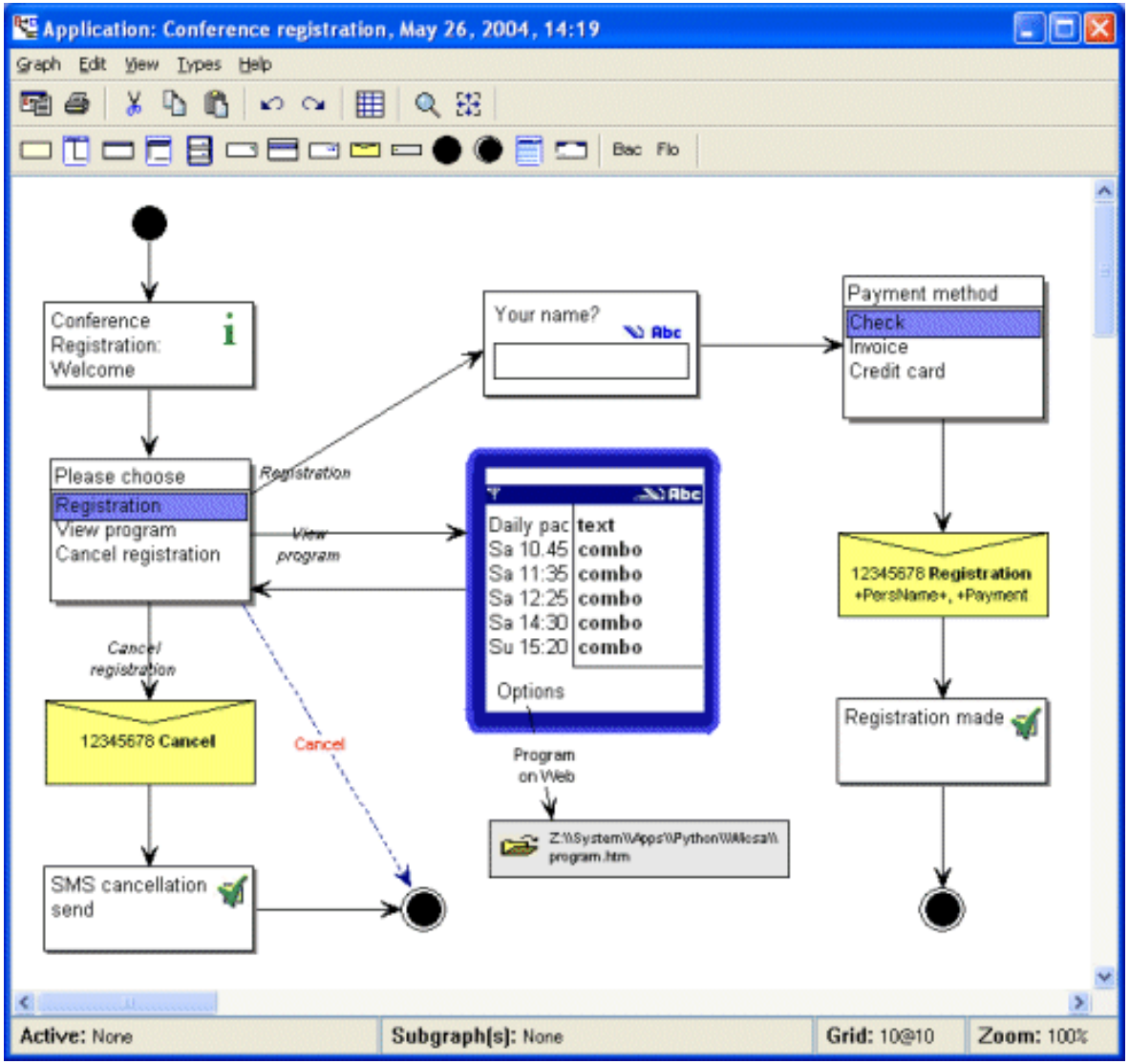

Problem domain
Solution Domain
Semantic gap
stock
account
investment fund
asset
object
class
type
module
DSL code / Modeling
Code Generation (or interpretation)
DSL Program (Model)
DSLs provide a common vocabulary


Domain Specific Languages (DSL)
[Domain-Specific] Modeling Languages
Domain concepts
DSL Engine
Programming Languages (GPL)
data types, methods,
classes, etc.
Compiler / Interpreter
Assemby Languages
CPU registers,
Memory address, etc.
+ Abstraction
+ Automation

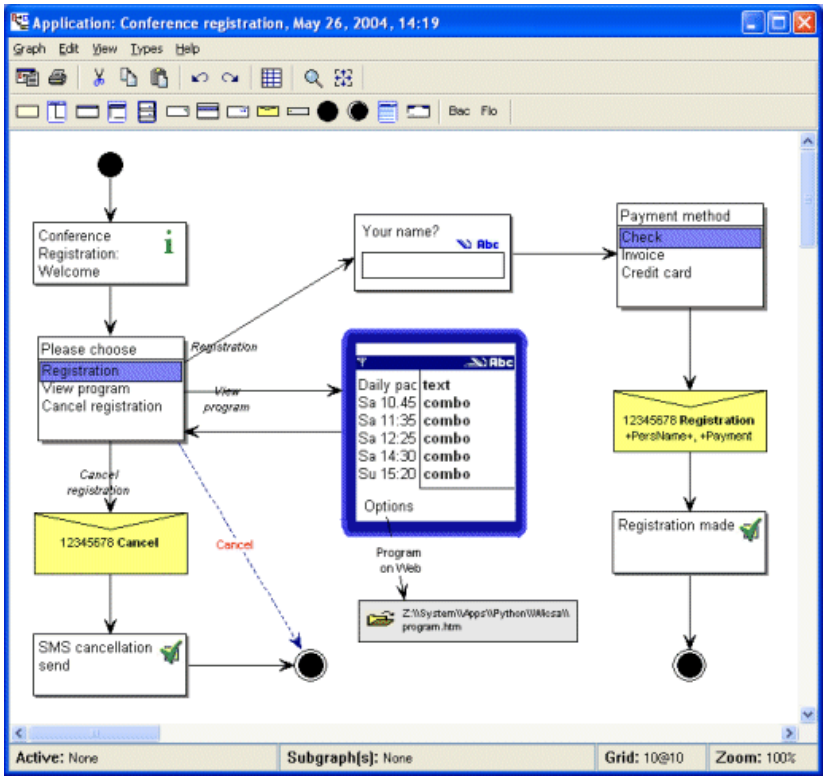
DSL to generate apps for smartphones

Nokia S60 (http://dsmforum.org/phone.html)

MetaEdit+



DSL Programs (Models)




```
Entity Cargo {
    - TrackingId trackingId key
    - Location origin required
    - Location destination required
    - Itinerary itinerary nullable opposite cargo
    - Set<HandlingEvent> events

    Repository CargoRepository {
        findByKey;
        findAll;
        save;
        @TrackingId nextTrackingId;
        protected findById;
    }
}

BasicType TrackingId {
    String identifier key
}

ValueObject Itinerary {
    belongsTo Cargo
    - Cargo cargo nullable opposite itinerary
    - List<Leg> legs inverse
}
```

DSL Definition

- *A Domain-Specific Language (DSL) offers constructs and notation which is appropriate for solving problems in a particular application domain.*

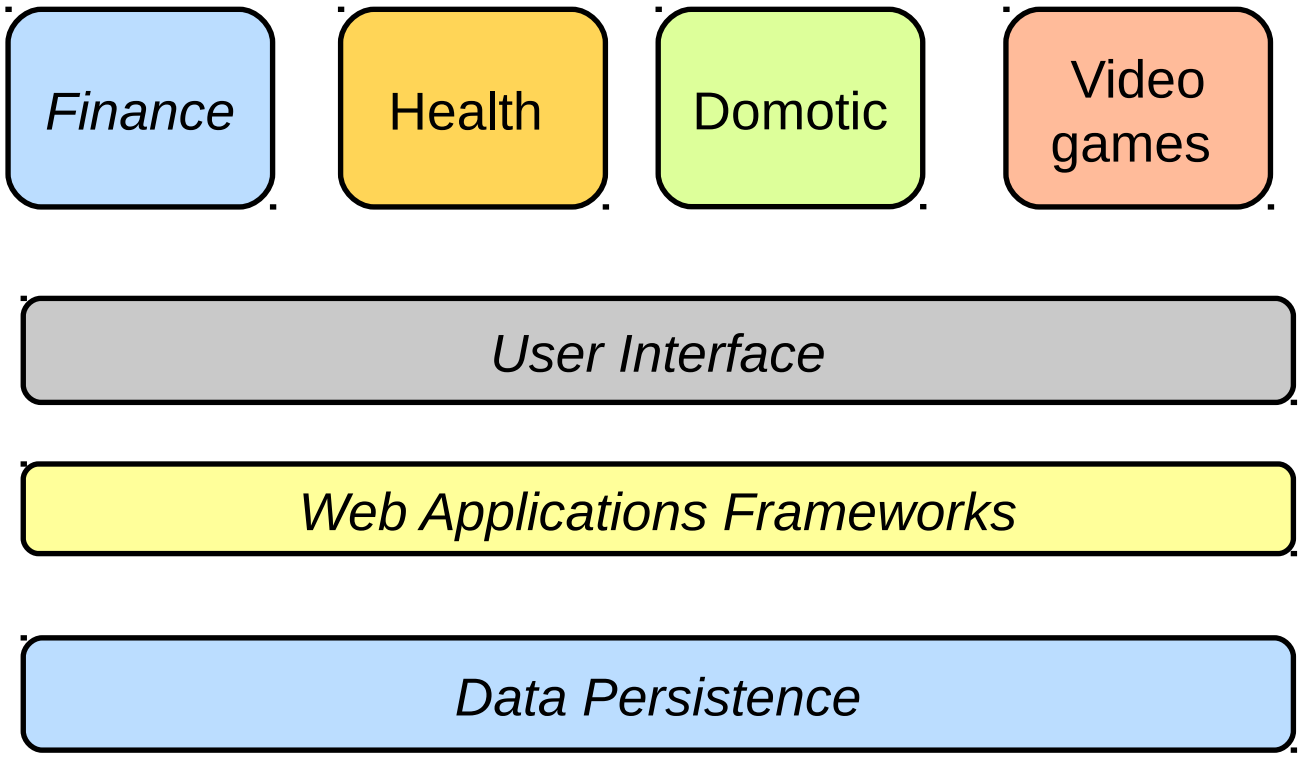
GPL vs. DSL ("DSL Engineering", Markus Voelter, 2012)

	GPLs	DSLs
Domain	large and complex	smaller and well-defined
Language size	large	small
Turing completeness	always	often not
User-defined abstractions	sophisticated	limited
Execution	via intermediate GPL	native
Lifespan	years to decades	months to years (driven by context)
Designed by	guru or committee	a few engineers and domain experts
User community	large, anonymous and widespread	small, accessible and local
Evolution	slow, often standardized	fast-paced
Deprecation/incompatible changes	almost impossible	feasible

DSL from early days of programming

- SQL
- GUI builder
- HTML
- make
- YACC
- Matlab
- Mathematica
- Excel
- OCL
- DOT/ GraphViz
- Csound
- WebUML
- etc.

Activity area: *Business, Domotic, Banking, Financial,..*
Knowledge area: *Graphs, Biomedical, Statistics…*
Software area: *Web, Mobile, Editors, Parsers,..*



Naturaleza de los DSL

Más extendidos y fáciles de crear

Diseño e implementación más costosos

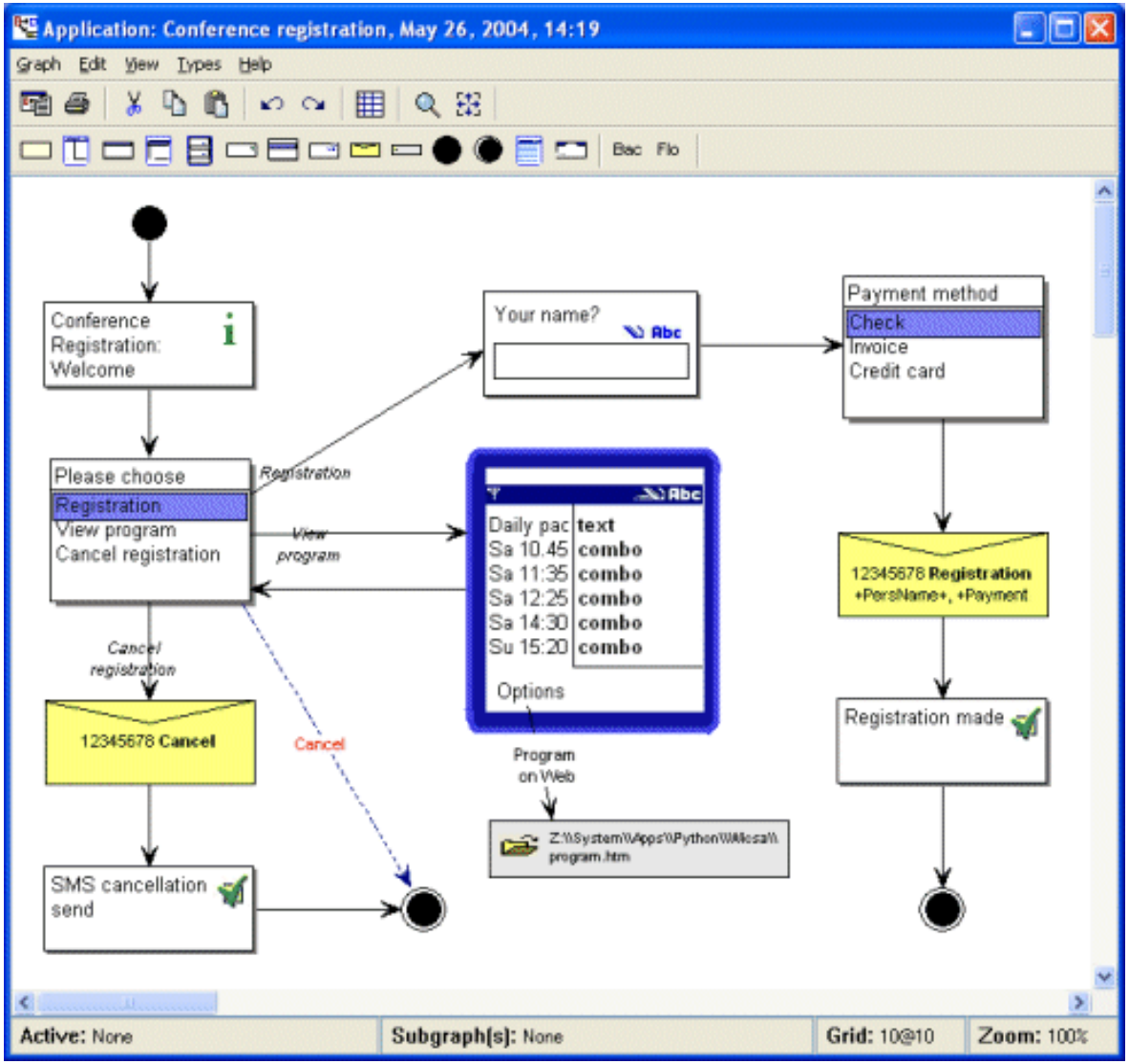
Son muy escasos

Excel es un ejemplo

DSL to generate apps for smartphones

Nokia S60 (http://dsmforum.org/phone.html)

MetaEdit+



sculptorgenerator.org

Sculptor

Home | Blog | Documentation | Forum | Contact

Enter search query

Sculptor is an open source productivity tool that applies the concepts from Domain-Driven Design and Domain Specific Languages for generating high quality Java code and configuration from a textual specification.

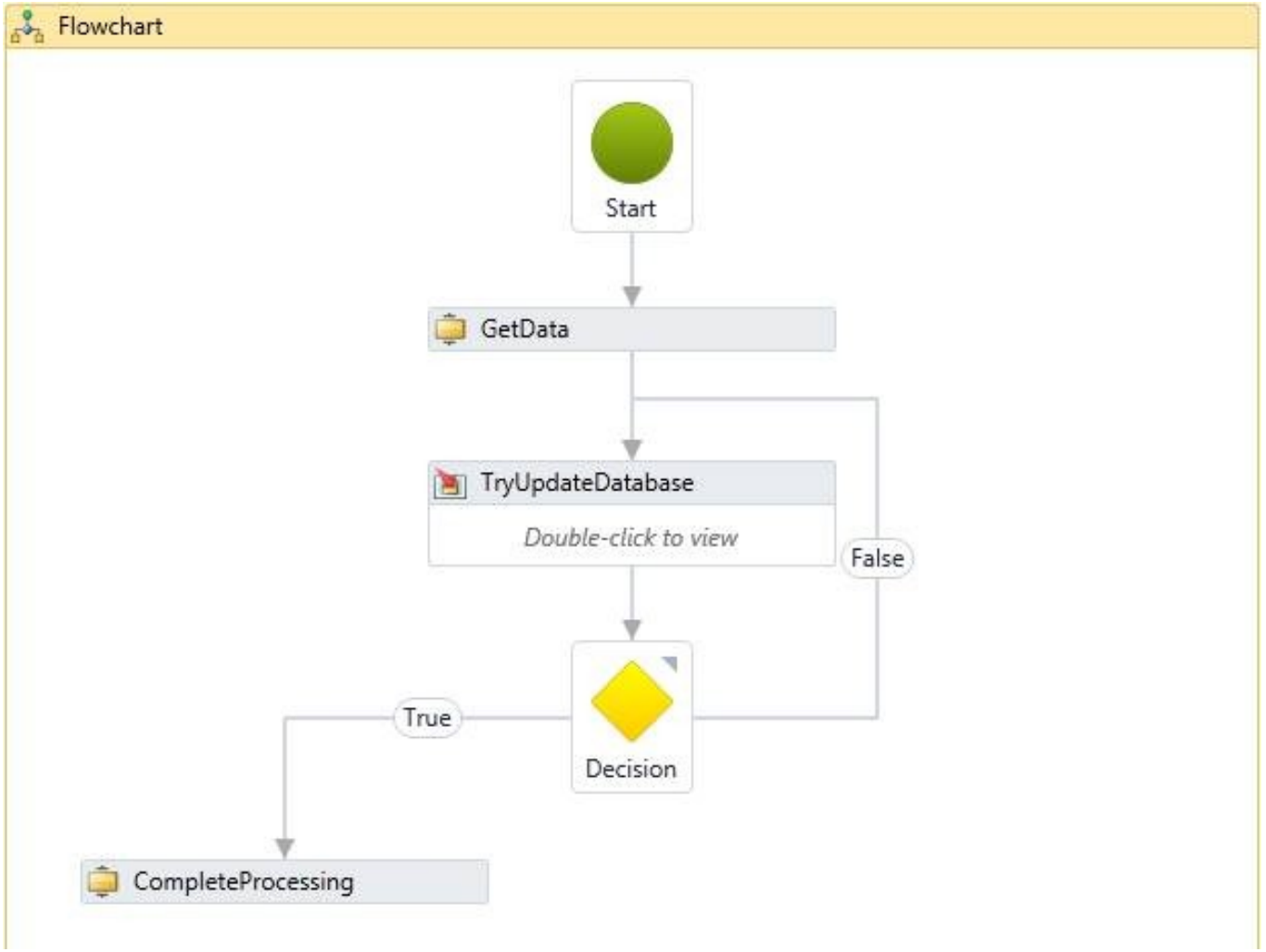
Installation | Documentation

GitHub project | Forum | Bugs & suggestions | Version 3.1.0

SCULPTOR (Generation of Java code)

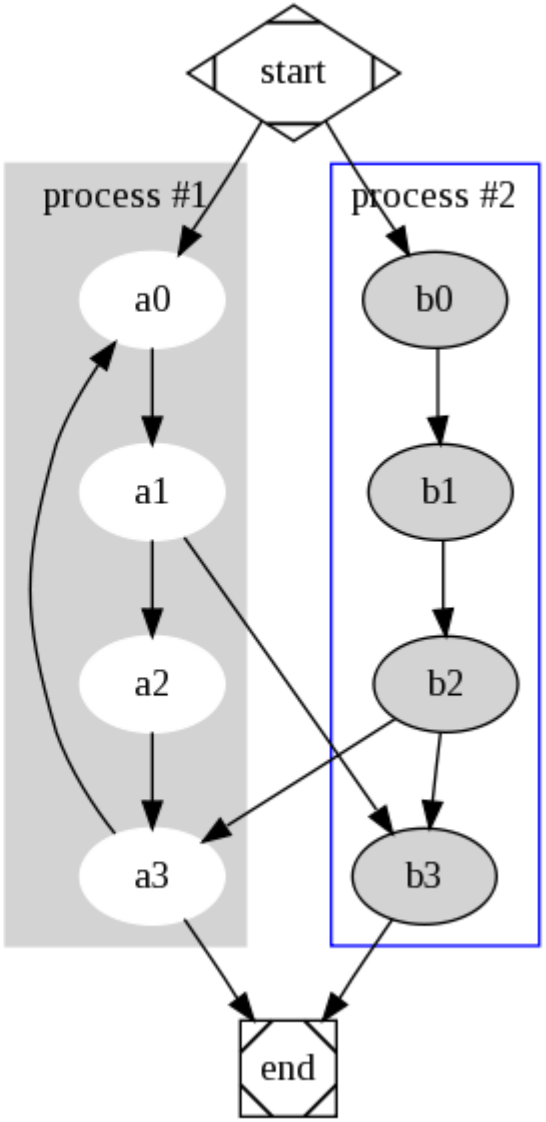

```
Entity Cargo {
    - TrackingId trackingId key
    - Location origin required
    - Location destination required
    - Itinerary itinerary nullable opposite cargo
    - Set<HandlingEvent> events

    Repository CargoRepository {
        findByKey;
        findAll;
        save;
        @TrackingId nextTrackingId;
        protected findById;
    }
}

BasicType TrackingId {
    String identifier key
}

ValueObject Itinerary {
    belongsTo Cargo
    - Cargo cargo nullable opposite itinerary
    - List<Leg> legs inverse
}
```

Windows Workflows Foundation (WF)



DOT/Graphviz (graph visualization)

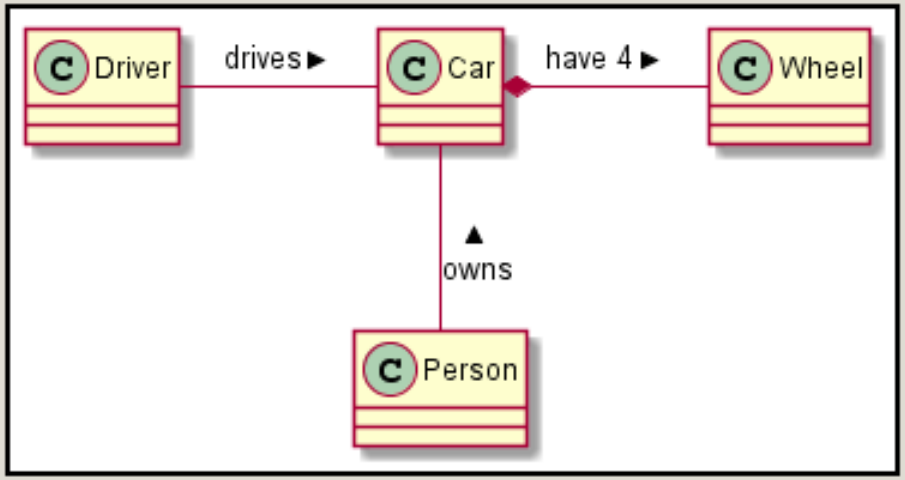
```
digraph G {

        subgraph cluster_0 {
                style=filled;
                color=lightgrey;
                node [style=filled,color=white];
                a0 -> a1 -> a2 -> a3;
                label = "process #1";
        }

        subgraph cluster_1 {
                node [style=filled];
                b0 -> b1 -> b2 -> b3;
                label = "process #2";
                color=blue
        }
        start -> a0;
        start -> b0;
        a1 -> b3;
        b2 -> a3;
        a3 -> a0;
        a3 -> end;
        b3 -> end;

        start [shape=Mdiamond];
        end [shape=Msquare];
}
```



http://www.graphviz.org/Gallery.php

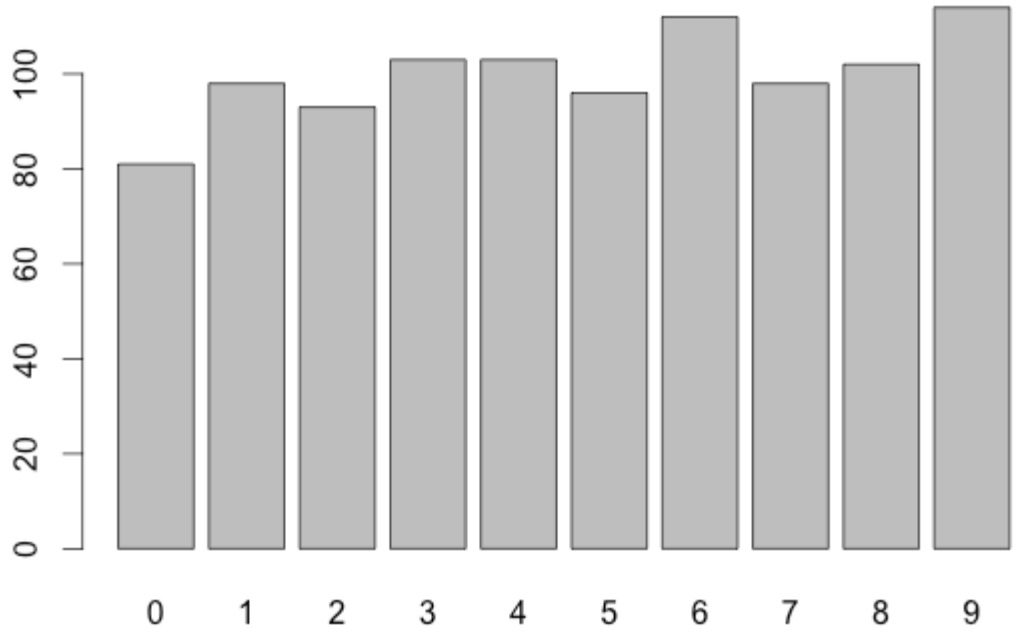
PlantUML (generation of Graphviz code for representing UML diagrams)

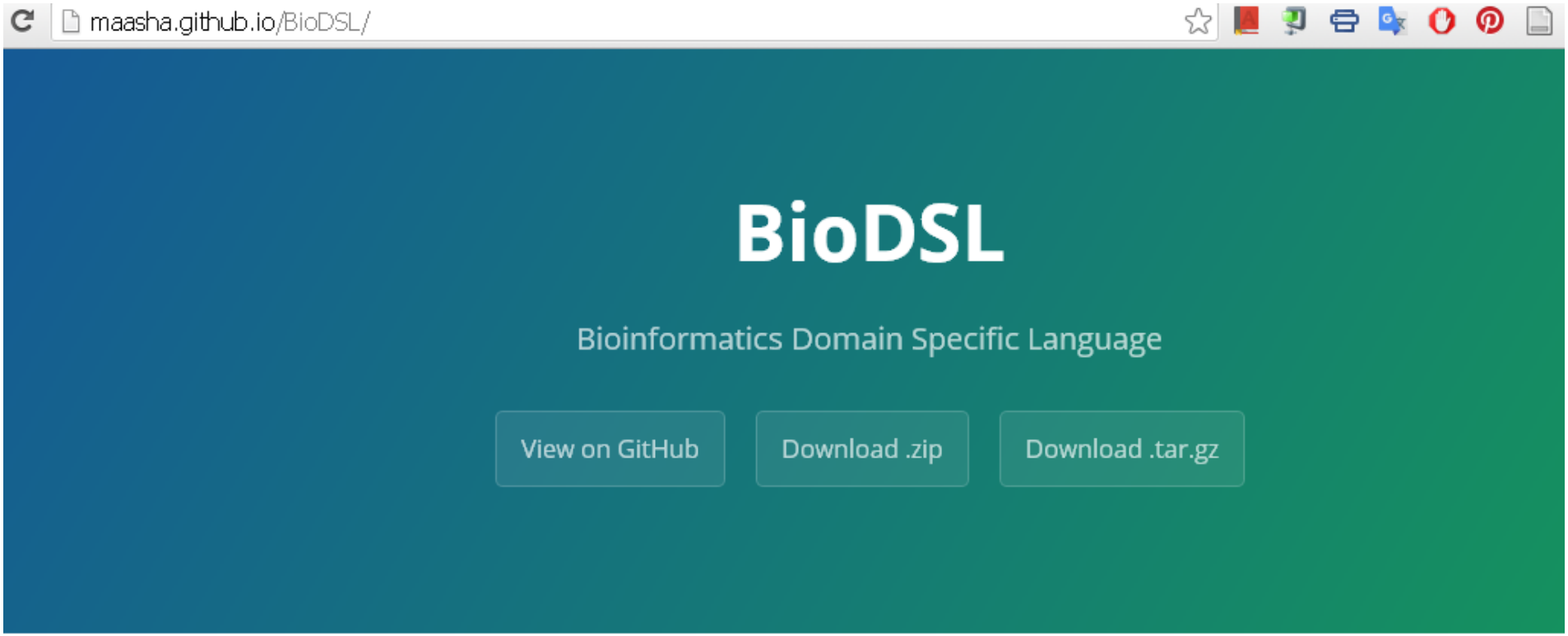
```
@startuml
class Car

Driver - Car : drives >
Car *- Wheel : have 4 >
Car -- Person : < owns

@enduml
```



Lenguaje R (Statistical computing)

```
n <- floor(runif(1000)*10)
t <- table(n)
barplot(t)
```





BioDSL (pronounced Biodiesel) is a Domain Specific Language for creating bioinformatic analysis workflows. A workflow may consist of several pipelines and each pipeline consists of a series of steps such as reading in data from a file, processing the data in some way, and writing data to a new file.

Getting started

BioDSL is implemented in Ruby making use of Ruby's powerful metaprogramming facilities. Thus, a workflow is basically a Ruby script containing one or more pipelines.

Here is a test script with a single pipeline that reads all FASTA entries from the file `input.fna`, selects all records with a sequence ending in `ATC`, and writing those records as FASTA entries to the file `output.fna`:

```
#!/usr/bin/env ruby

require 'BioDSL'

BD.new.
read_fasta(input: "input.fna").
grab(select: "ATC$", keys: :SEQ).
write_fasta(output: "output.fna").
run
```

DSL embedded into Ruby

This is the general reference for all Cucumber implementations. Please refer to the documentation ove
links to platform-specific documentation.

Gherkin

Cucumber executes your `.feature` files, and those files contain executable specifications written in a language called Gherkin.

Gherkin is plain-text English (or one of 60+ other languages) with a little extra structure. Gherkin is designed to be easy to learn by non-programmers, yet structured enough to allow concise description of examples to illustrate business rules in most real-world domains.

Here is a sample Gherkin document:

```
Feature: Refund item

  Scenario: Jeff returns a faulty microwave
    Given Jeff has bought a microwave for $100
    And he has a receipt
    When he returns the microwave
    Then Jeff should be refunded $100
```

In Gherkin, each line that isn't blank has to start with a Gherkin *keyword*, followed by any text you like. The main keywords are:

Cucumber (acceptance tests, https://cucumber.io/)

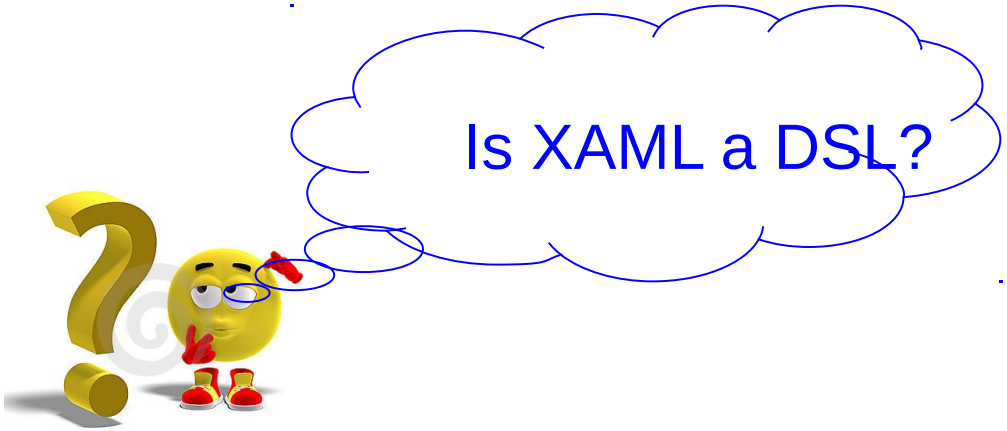
```
# feature/hello_cucumber.feature
Feature: Hello Cucumber
As a product manager
I want our users to be greeted when they visit our site
So that they have a better experience

Scenario: User sees the welcome message
When I go to the homepage
Then I should see the welcome message
```

DSL embedded into Ruby

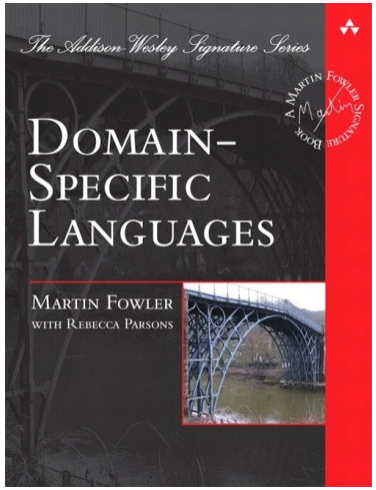
XAML (Extensible Application Markup Language)

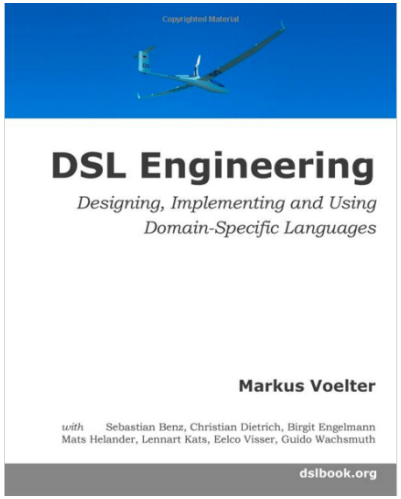
XAML

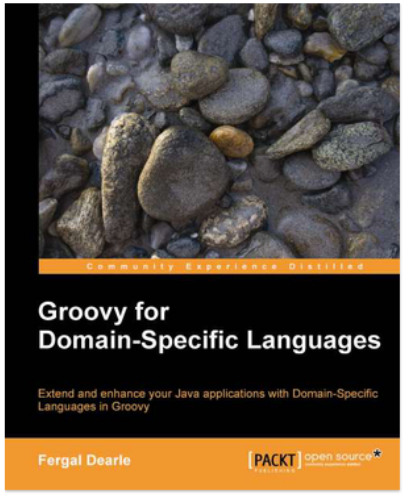
```
<Grid x:Name="ContentPanel" Background="Red" Grid.Row="1" Margin="12,0,12,0">
   <StackPanel Margin="20" Background="Blue" >
      <TextBlock Name="firstTextBlock" FontSize="30">First TextBlock</TextBlock>
      <TextBlock Name="secondTextBlock" FontSize="30">Second TextBlock</TextBlock>
      <TextBlock Name="thirdTextBlock" FontSize="30">Third TextBlock</TextBlock>
   </StackPanel>
</Grid>
```

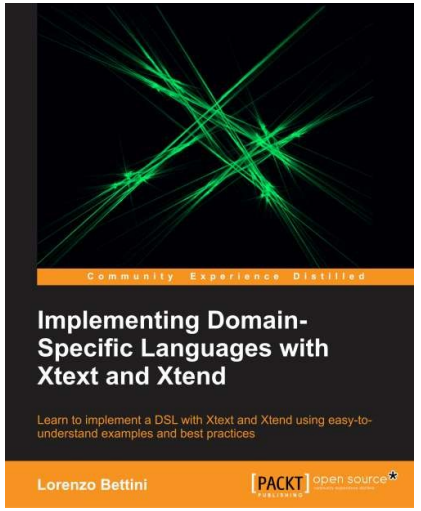


But academic and industrial interest in DSL
has increased significantly over the last 10 years





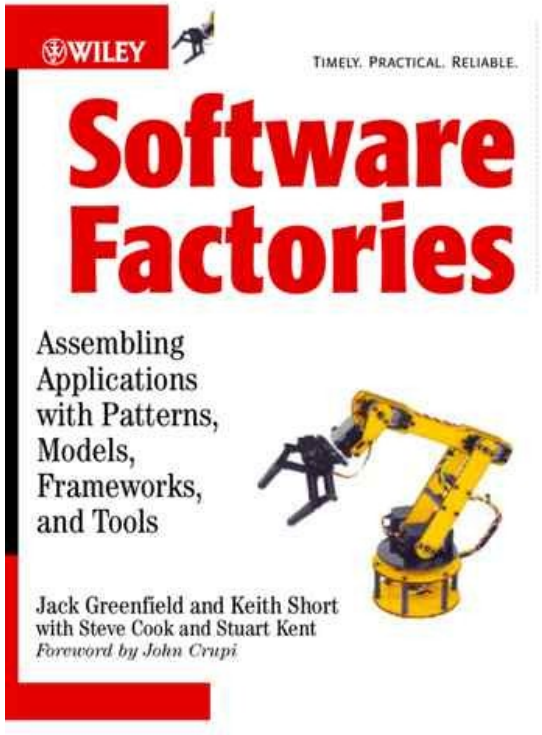




MetaEdit+ Sirius Xtext MPS Meta Programming System

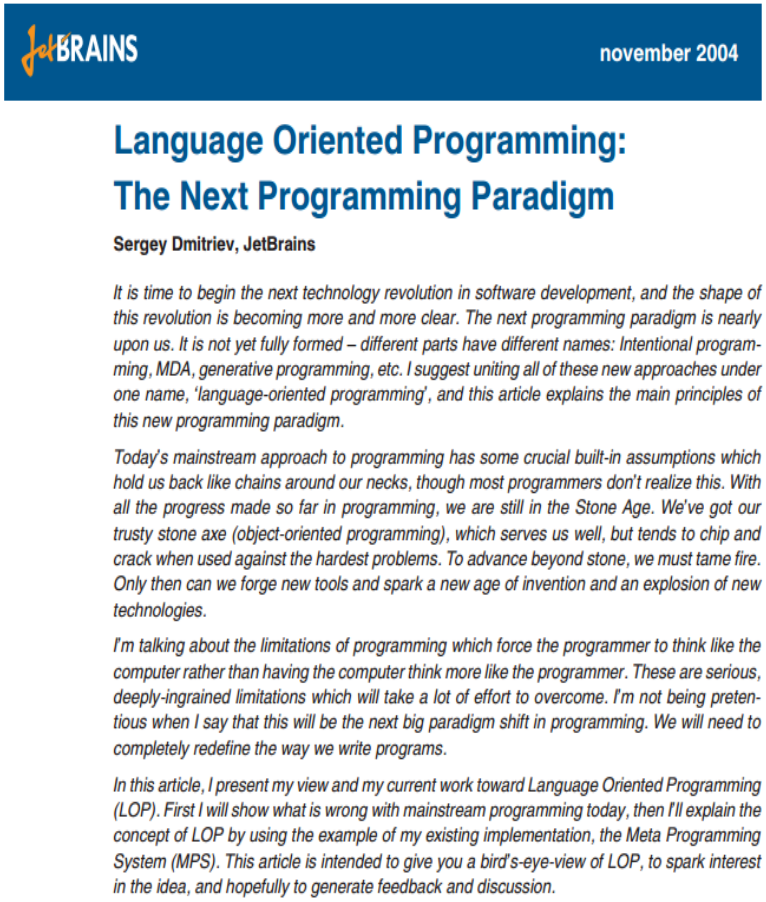
+ books + tools + research articles + DSL created

Jack Greenfield (2004)



- "We think that the latitude afforded by GPLs is needed only for a small part of the typical business application. Most of the work can be made faster, cheaper, and less risky by using constrained configurarion mechanisms like setting properties, wizards, features models, MDE, and patterns"
- "Just as moving from assemby language to GPLs yields significant benefits for … there is much more benefit to be gained by moving to even higher levels of abstractions (DSLs)"

Sergey Dmitriev (2004)



JetBRAINS november 2004

Language Oriented Programming: The Next Programming Paradigm

Sergey Dmitriev, JetBrains

It is time to begin the next technology revolution in software development, and the shape of this revolution is becoming more and more clear. The next programming paradigm is nearly upon us. It is not yet fully formed – different parts have different names: Intentional programming, MDA, generative programming, etc. I suggest uniting all of these new approaches under one name, 'language-oriented programming', and this article explains the main principles of this new programming paradigm.

Today's mainstream approach to programming has some crucial built-in assumptions which hold us back like chains around our necks, though most programmers don't realize this. With all the progress made so far in programming, we are still in the Stone Age. We've got our trusty stone axe (object-oriented programming), which serves us well, but tends to chip and crack when used against the hardest problems. To advance beyond stone, we must tame fire. Only then can we forge new tools and spark a new age of invention and an explosion of new technologies.

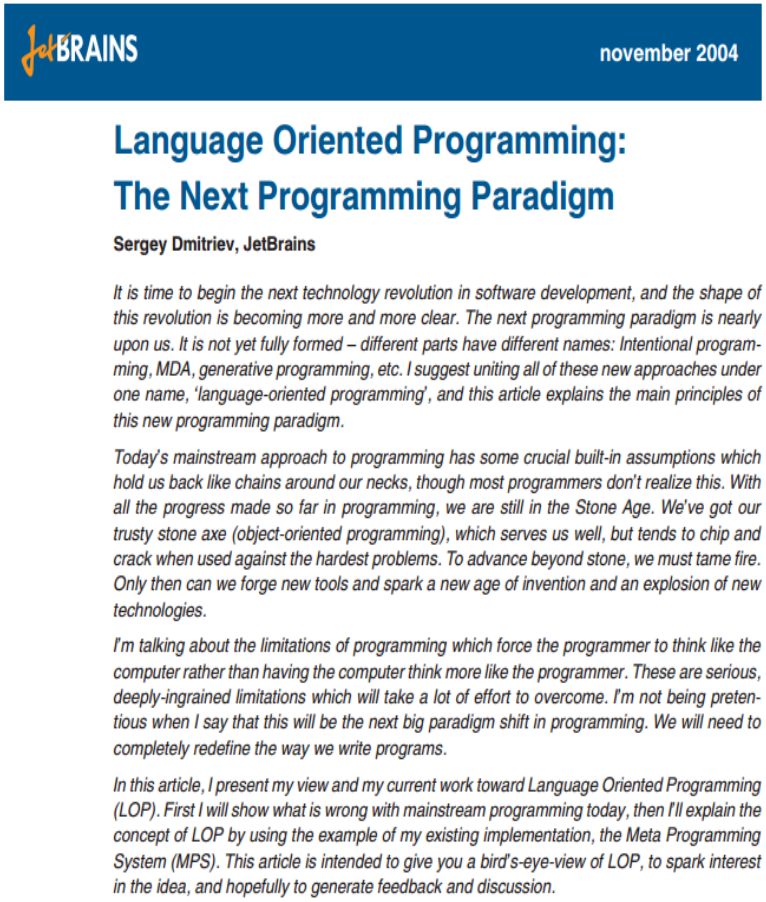
I'm talking about the limitations of programming which force the programmer to think like the computer rather than having the computer think more like the programmer. These are serious, deeply-ingrained limitations which will take a lot of effort to overcome. I'm not being pretentious when I say that this will be the next big paradigm shift in programming. We will need to completely redefine the way we write programs.

In this article, I present my view and my current work toward Language Oriented Programming (LOP). First I will show what is wrong with mainstream programming today, then I'll explain the concept of LOP by using the example of my existing implementation, the Meta Programming System (MPS). This article is intended to give you a bird's-eye-view of LOP, to spark interest in the idea, and hopefully to generate feedback and discussion.

“Any general-purpose language, like Java or C++, gives us the ability to do anything we want with a computer. This is true, at least in theory anyway, but GPLs tend to be unproductive as I will explore later.

Alternatively, we could use DSL, which are tailored to be highly productive in a specific problem domain, such as SQL for writing database queries. "

Sergey Dmitriev (2004)



JetBRAINS november 2004

Language Oriented Programming: The Next Programming Paradigm

Sergey Dmitriev, JetBrains

It is time to begin the next technology revolution in software development, and the shape of this revolution is becoming more and more clear. The next programming paradigm is nearly upon us. It is not yet fully formed – different parts have different names: Intentional programming, MDA, generative programming, etc. I suggest uniting all of these new approaches under one name, 'language-oriented programming', and this article explains the main principles of this new programming paradigm.

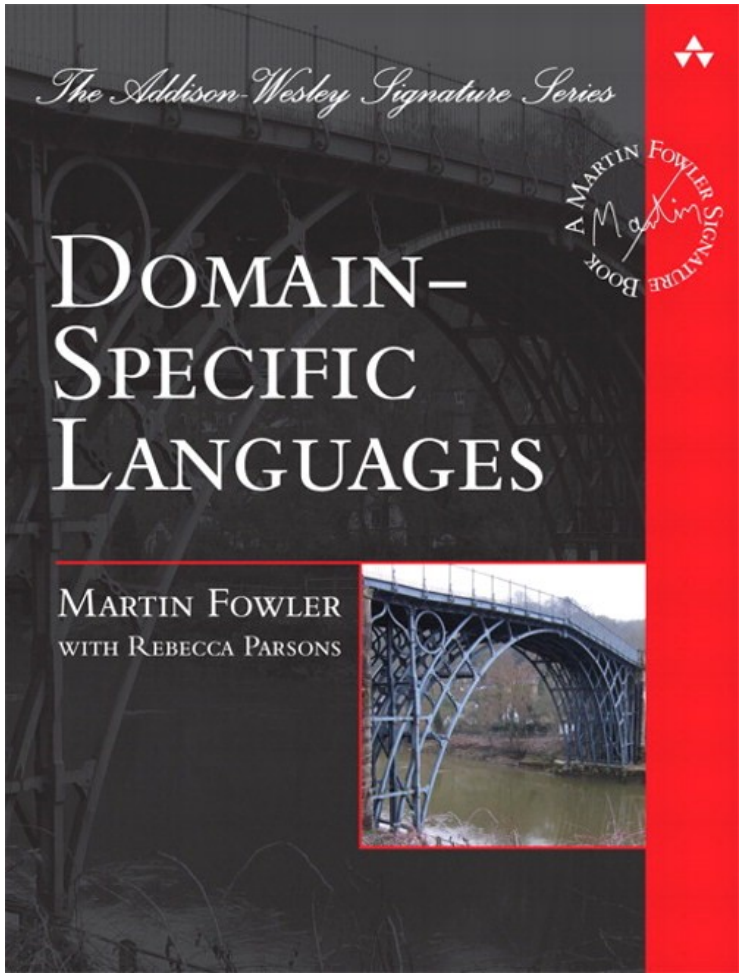
Today's mainstream approach to programming has some crucial built-in assumptions which hold us back like chains around our necks, though most programmers don't realize this. With all the progress made so far in programming, we are still in the Stone Age. We've got our trusty stone axe (object-oriented programming), which serves us well, but tends to chip and crack when used against the hardest problems. To advance beyond stone, we must tame fire. Only then can we forge new tools and spark a new age of invention and an explosion of new technologies.

I'm talking about the limitations of programming which force the programmer to think like the computer rather than having the computer think more like the programmer. These are serious, deeply-ingrained limitations which will take a lot of effort to overcome. I'm not being pretentious when I say that this will be the next big paradigm shift in programming. We will need to completely redefine the way we write programs.

In this article, I present my view and my current work toward Language Oriented Programming (LOP). First I will show what is wrong with mainstream programming today, then I'll explain the concept of LOP by using the example of my existing implementation, the Meta Programming System (MPS). This article is intended to give you a bird's-eye-view of LOP, to spark interest in the idea, and hopefully to generate feedback and discussion.

"It's not a question of general-purpose versus domain-specific. I want all freedoms. I want to be able to do anything, and also be highly productive at the same time. There aren't any good ways to do this yet. Ideally, I would be able to use different languages for each specialized part of the program, all working together coherently. And the environment would fully support these languages with refactoring, code completion, navigation, and all the other productivity tools that are available for mainstream languages."

Martin Fowler (2010)



- The interest is increasing due to the framework and XML-intensive software development.
- Metamodeling-based DSL definition tools will play an important role in the near future.

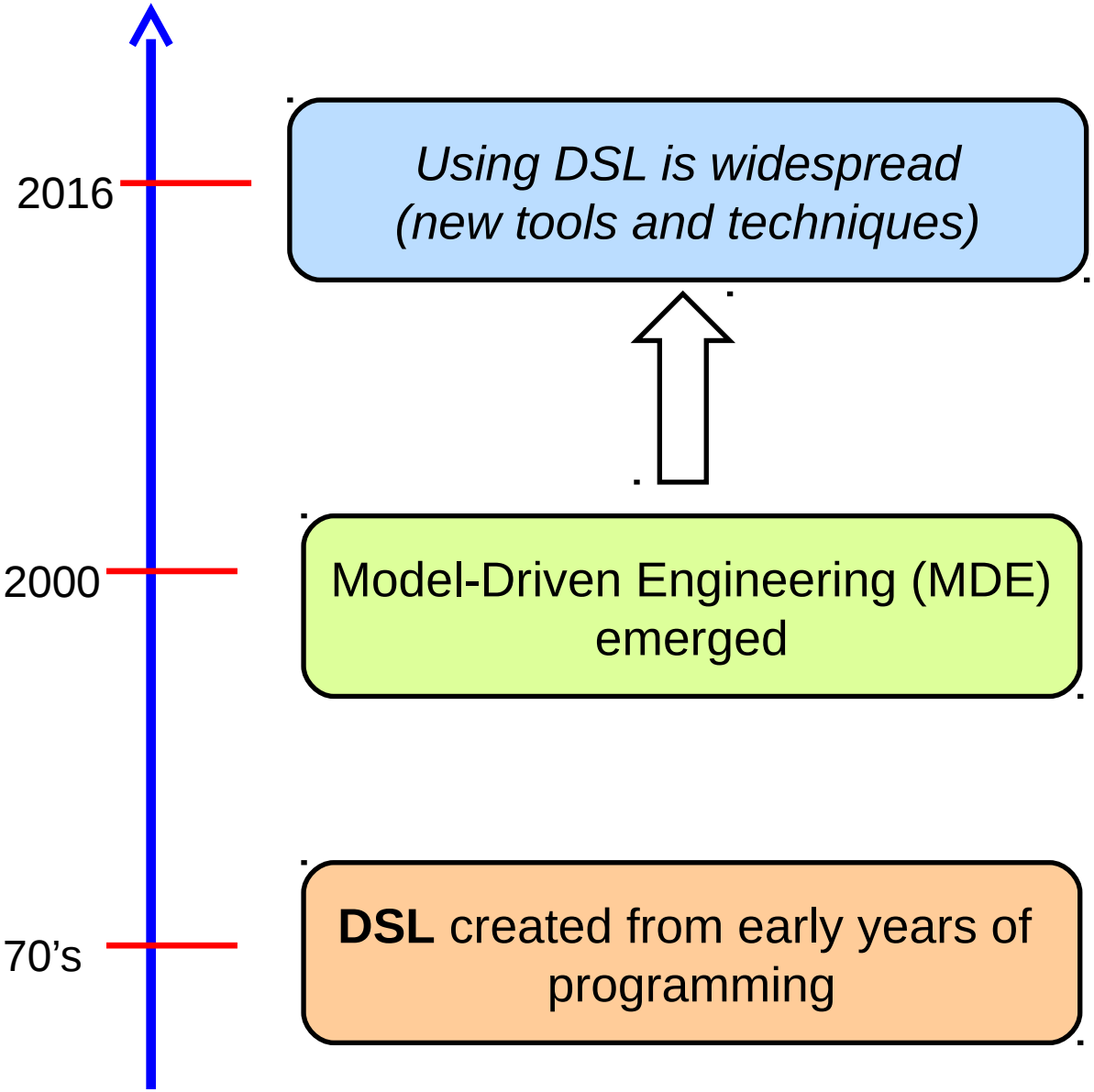
Martin Fowler

- “DSLs have been a part of the computing landscape since before I got into programming”

- “I saw people using XML where a custom syntax would be more readable and not harder to do. I saw people bending Ruby into complicated contortions when a custom syntax would be easier. I saw people playing around with parsers when a fluent interface in their regular interface would be a lot less work.

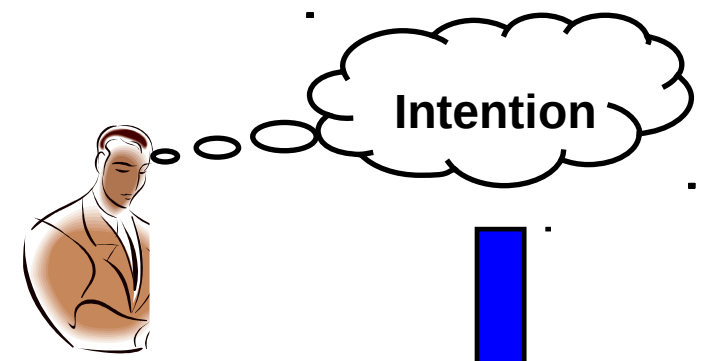
  My hypothesis is that these things are happening because of a knowledge gap. Skilled programmers don't know enough about DSL techniques to make an informed decision about which ones to use. That’s the kind of gap I enjoy trying to fill.”

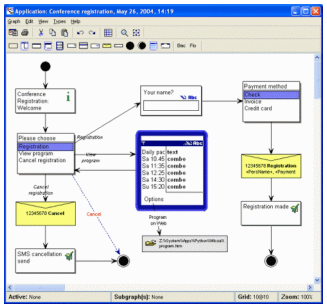
(Martin Fowler, “Domain-Specific Languages, Addison-Wesley, 2010)


2016
Using DSL is widespread
(new tools and techniques)
2000
Model-Driven Engineering (MDE)
emerged
70’s
DSL created from early years of
programming

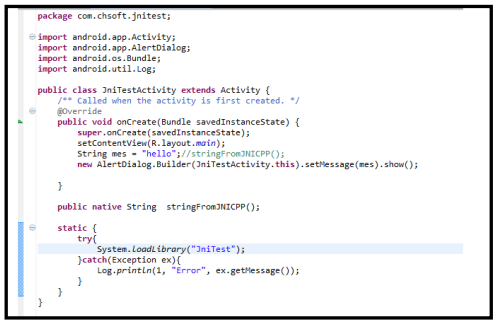
Model-Driven Software Engineering (MDE)

Systematic using of models in the stages of the software development life cycle in order to increase the levels of abstraction and automation.



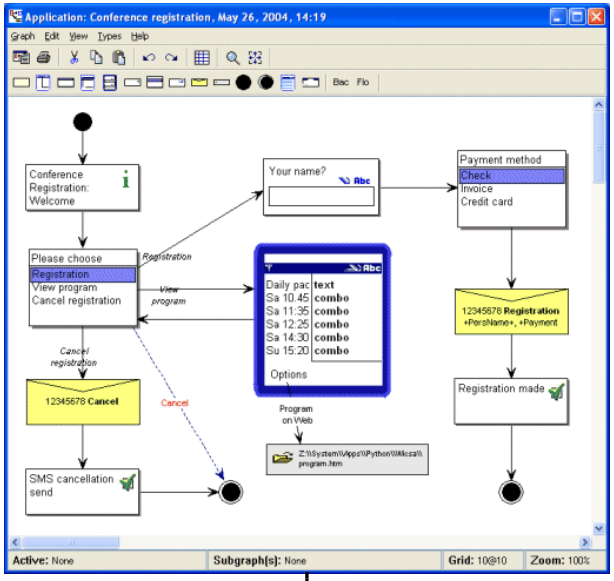


Model



Generation Tools

Code



<<conforms>>

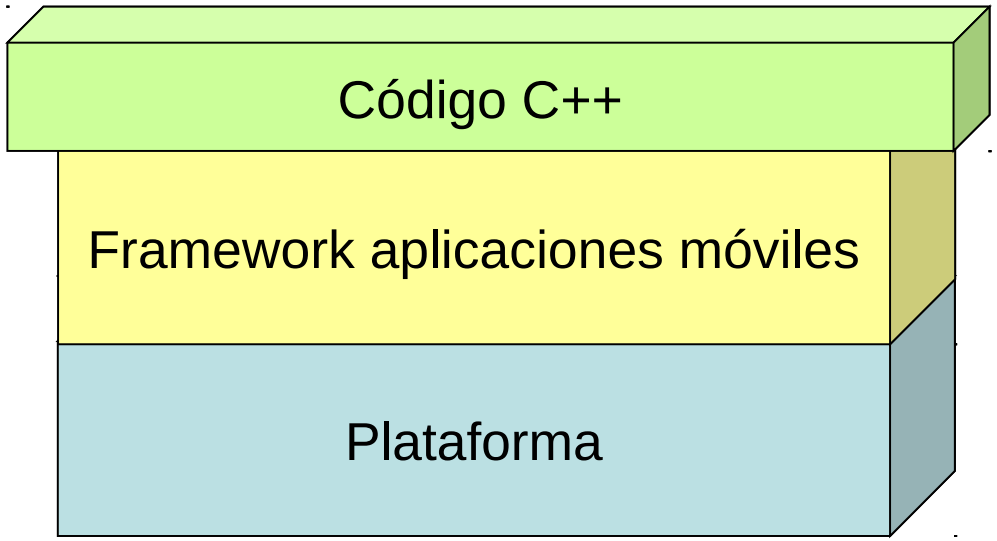
Metamodel DSL Nokia

DSL Nokia

- Effort
+ Legibility
+ Communication
- Errors
- Maintenance

One or more transformations

Automatic code generation



MDE Solution

The State of Practice in Model-Driven Engineering

Jon Whittle, John Hutchinson, and Mark Rouncefield,
Lancaster University

// Despite lively debate over the past decade on the benefits and drawbacks of model-driven engineering (MDE), there have been few industry-wide studies of MDE in practice. A new study that surveyed 450 MDE practitioners and performed in-depth interviews with 22 more suggests that although MDE might be more widespread than commonly believed, developers rarely use it to generate whole systems. Rather, they apply MDE to develop key parts of a system. //

of course, but it did lead to a resurgence of activity in the area as well as hotly contested debates between proponents and detractors of model-driven approaches.[1]

Many years later, there remains a lack of clarity on whether model-driven engineering (MDE) is a good way to develop software (see the "What Is MDE, Anyway?" sidebar). Some companies have reported great success with it, whereas others have failed horribly. What's missing is an industry-wide, independent study of MDE in practice, highlighting the factors that lead to success or failure. Although there have been a few prior surveys of modeling in industry, they've focused on only one aspect of modeling, such as the use of UML[2] or formal models.[3]

In this article, we report on a new study of MDE practice that covers a broad range of experiences. In particular, we focus on identifying MDE's success and failure factors. We surveyed 450 MDE practitioners and interviewed 22 more from 17 different companies representing 9 different industrial sectors (see the "Methods" sidebar for more information on the particulars). The study reflects a wide range of maturity levels with MDE: questionnaire respondents were equally split among those in early exploration phases, those carrying out their first MDE project, and those with many years' experience with MDE. Interviewees were typically very experienced with MDE.

Perhaps surprisingly, the majority of MDE examples in our study followed domain-specific modeling paradigms: the companies who successfully applied MDE largely did so by creating or using languages specifically developed for their domain, rather than using UML.

It's common to develop small domain-specific languages (DSLs) for narrow, well-understood domains. In contrast to perceived wisdom practical application of domain modeling is "quick and dirty," where DSLs (and accompanying generators) can be developed sometimes in as little as two weeks. There's also widespread use of mini-DSLs, even within a single project.

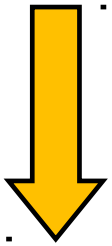
A clear challenge, then, is how to integrate multiple DSLs.

Our participants tended to use them in combination with UML—in some cases, the DSL was a UML profile. Whatever the context, however, modeling languages requires significant customization before the languages can be applied in practice.

MDE

MDA = MDE

Profile mechanism
to create graphical DSLs

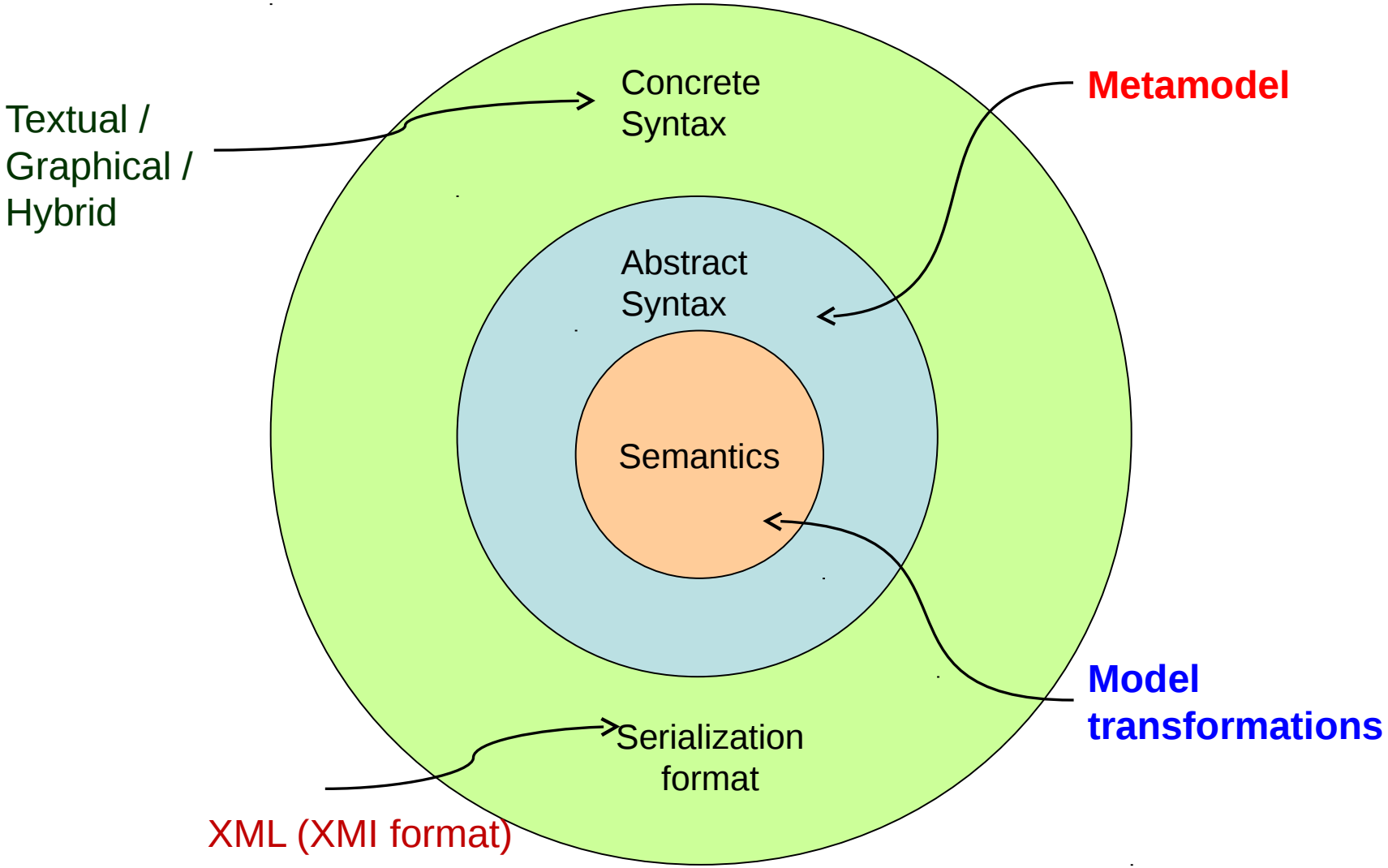


MDE

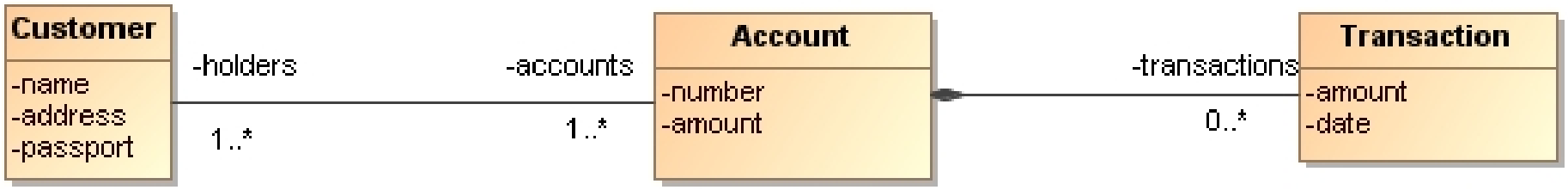
UML profiles are not
recommended

How can DSLs be created?

DSL elements



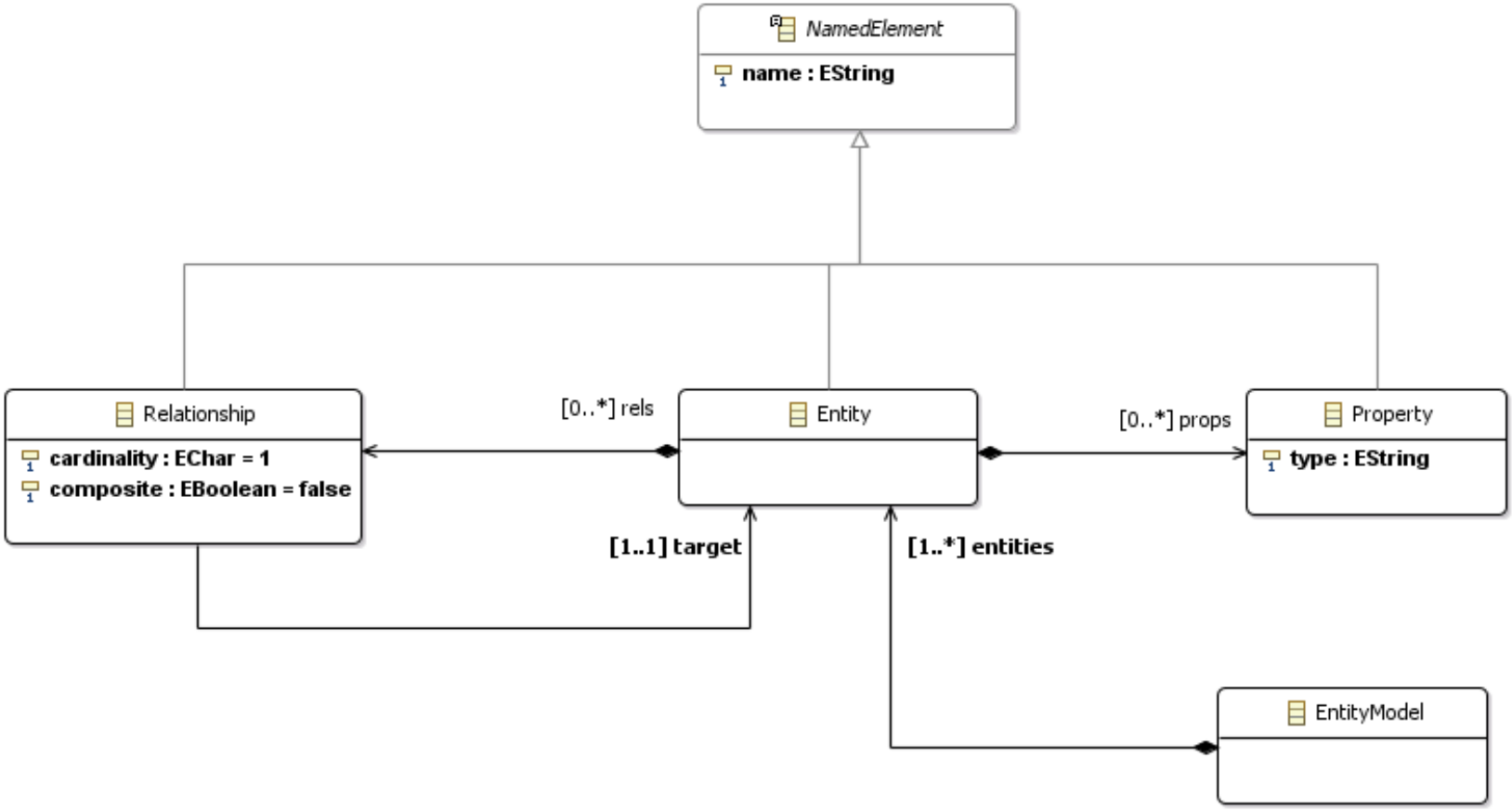
Running example: Entity models

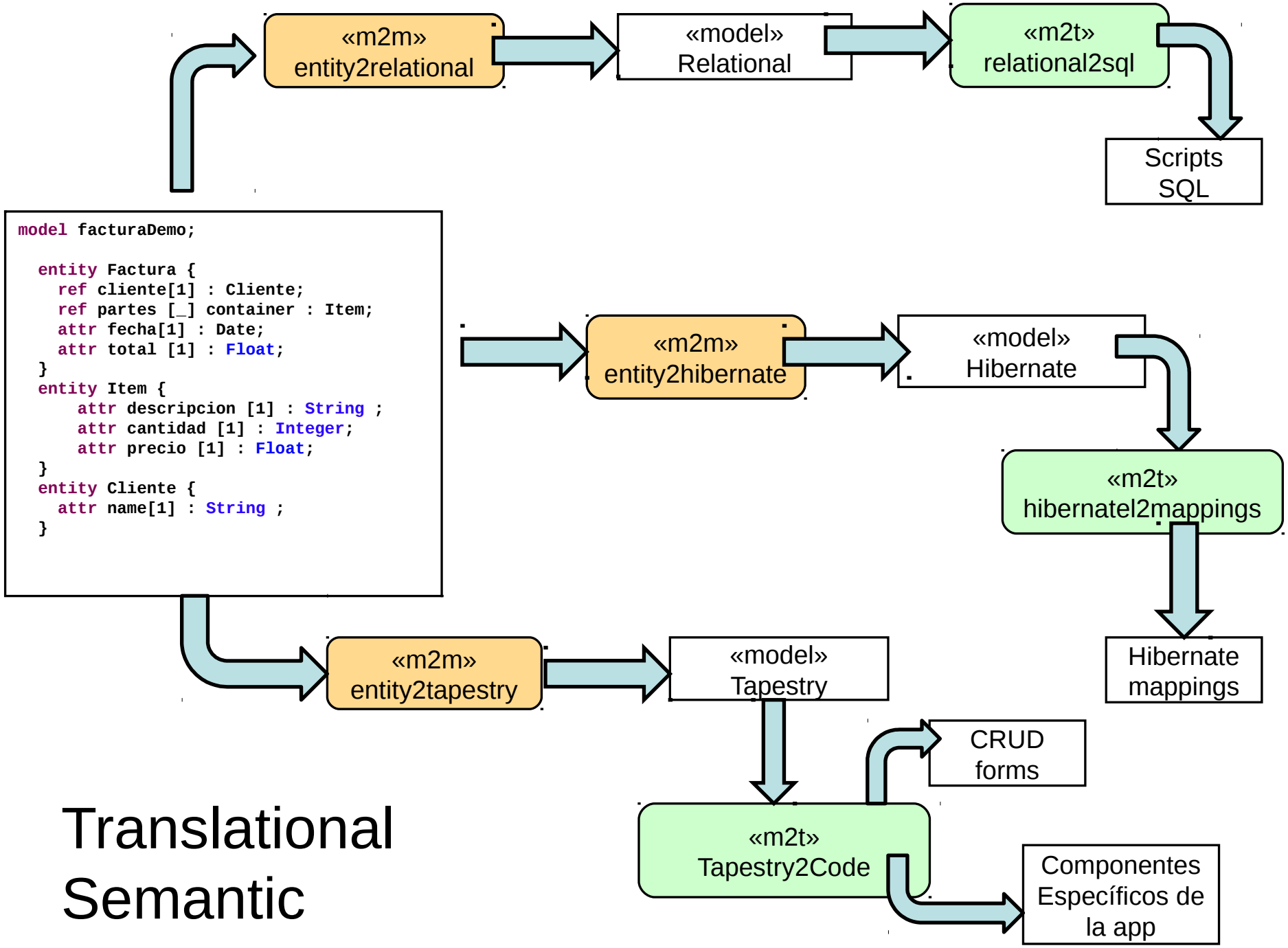


Running example: Entity models

```
model 'Bank' {

  entity 'Account' {
      attr String number;
      attr Integer amount;
      ref Customer[+] holders;
      val Transaction[*] transactions;
  }

  entity 'Customer' {
      attr String name;
      attr Integer amount;
      ref Account[+] accounts;
  }

  entity 'Transaction' {
      attr Date date;
      attr Integer amount;
  }
```

Running example: Entity metamodel




«m2m»
entity2relational
«model»
Relational
«m2t»
relational2sql
Scripts
SQL
model facturaDemo;
entity Factura {
ref cliente[1] : Cliente;
ref partes [_] container : Item;
attr fecha[1] : Date;
attr total [1] : Float;
}
entity Item {
attr descripcion [1] : String ;
attr cantidad [1] : Integer;
attr precio [1] : Float;
}
entity Cliente {
attr name[1] : String ;
}
«m2m»
entity2hibernate
«model»
Hibernate
«m2t»
hibernatel2mappings
Hibernate
mappings
«m2m»
entity2tapestry
«model»
Tapestry
CRUD
forms
«m2t»
Tapestry2Code
Componentes
Específicos de
la app
Translational
Semantic

DSL implementation techniques

External

- Created from scratch

Internal
(Embedded and
Fluent-API)

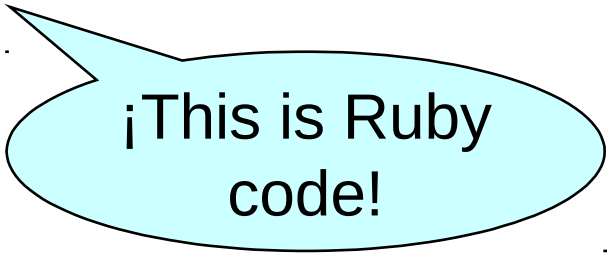
- Embedded into a dynamically typed language.
- Definition of an API to facilitate coding and legibility.

Metamodel-based tools
(Language Workbench)

- IDEs for the DSL definition and the generation of tooling needed to use them.

DSL embedded into Ruby

```
model 'Bank' {

  entity 'Account' {
      attr String number;
      attr Integer amount;
     ref Customer[+] holders;
     val Transaction[*] transactions;
  }

  entity 'Customer' {
      attr String name;
      attr Integer amount;
     ref Account[+] accounts;
  }

  entity 'Transaction' {
      attr Date date;
      attr Integer amount;
  }
```



Fluent-API

Method chaining

```
public static void main(String[] args) {
        EntityModel entidad = EntityModelBuilder
                .entityModel()
                        .entity()
                            .name("Account")
                            .property()
                               .name("number")
                               .type("String")
                             .propertyEnd()
                             .property()
                                .name("amount")
                                .type("double")
                            .propertyEnd()
                            .relationship()
                                 .name("holders“)
                                 .cardinality('+')
                                 .composite(false)
                                 .target("Customer")
                            .relationshipEnd()
                      .entityEnd()
                .endModel(); }
```

Metamodel-based DSL workbench

- IDEs for developing DSLs
  - Parser, injector, generator and editor are automatically generated from a metamodel-based specification

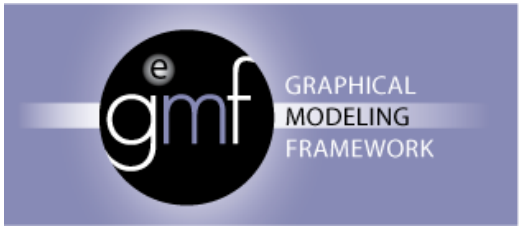
"I think that language workbenches have a remarkable potential. If they fulfill this they could entirely change the face of software development. Yet this potential, however profound, is still somewhat in the future.“

(M. Fowler)

- Workbenches for textual DSL
  - Xtext, MPS y EMFText
- Workbenches for graphical DSL
  - Sirius and GMF/Eugenia (Eclipse), MetaEdit+ (Metacase), DSL Tools (Microsoft)

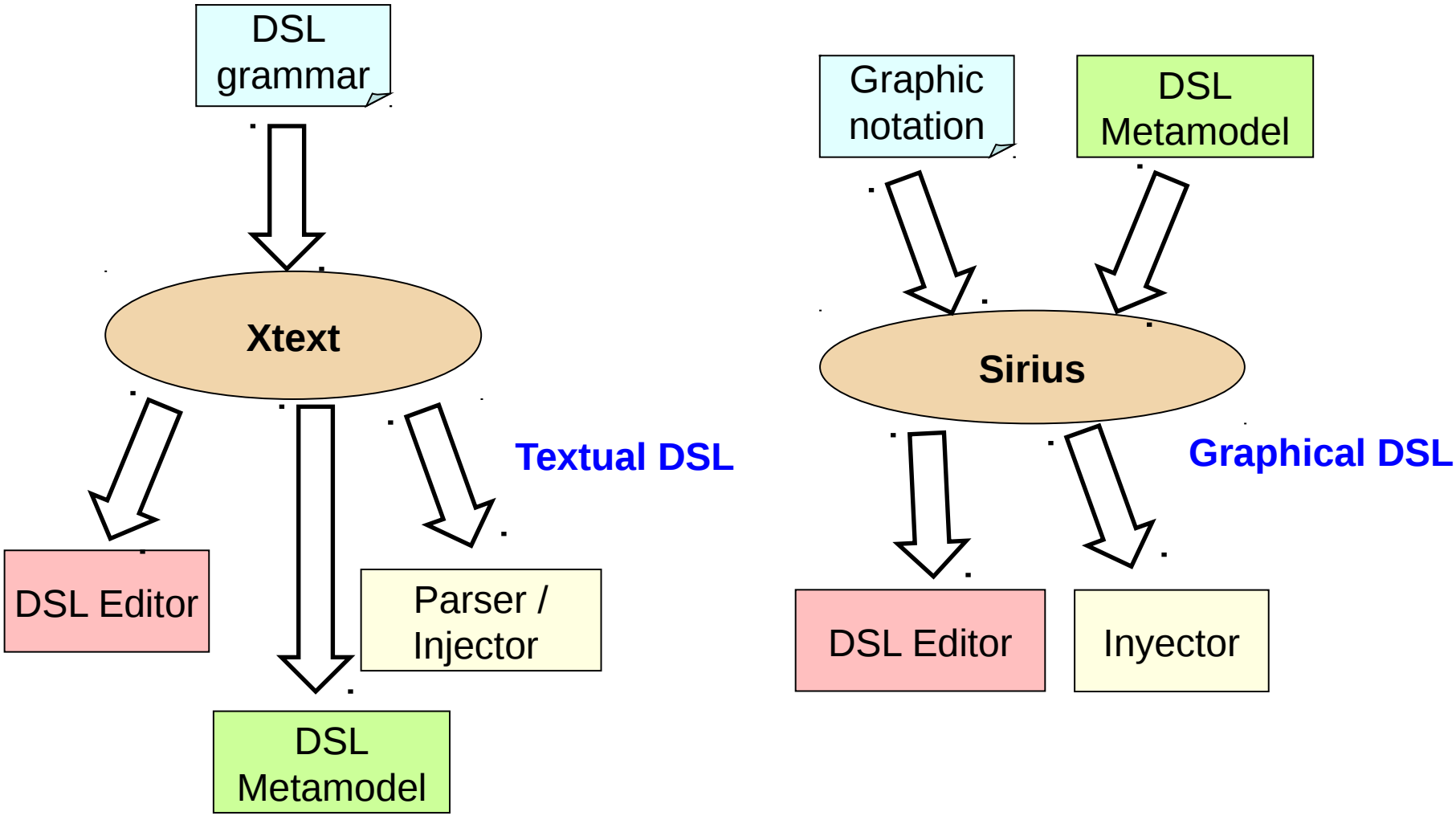
Textual Workbenches

Graphical Workbenches

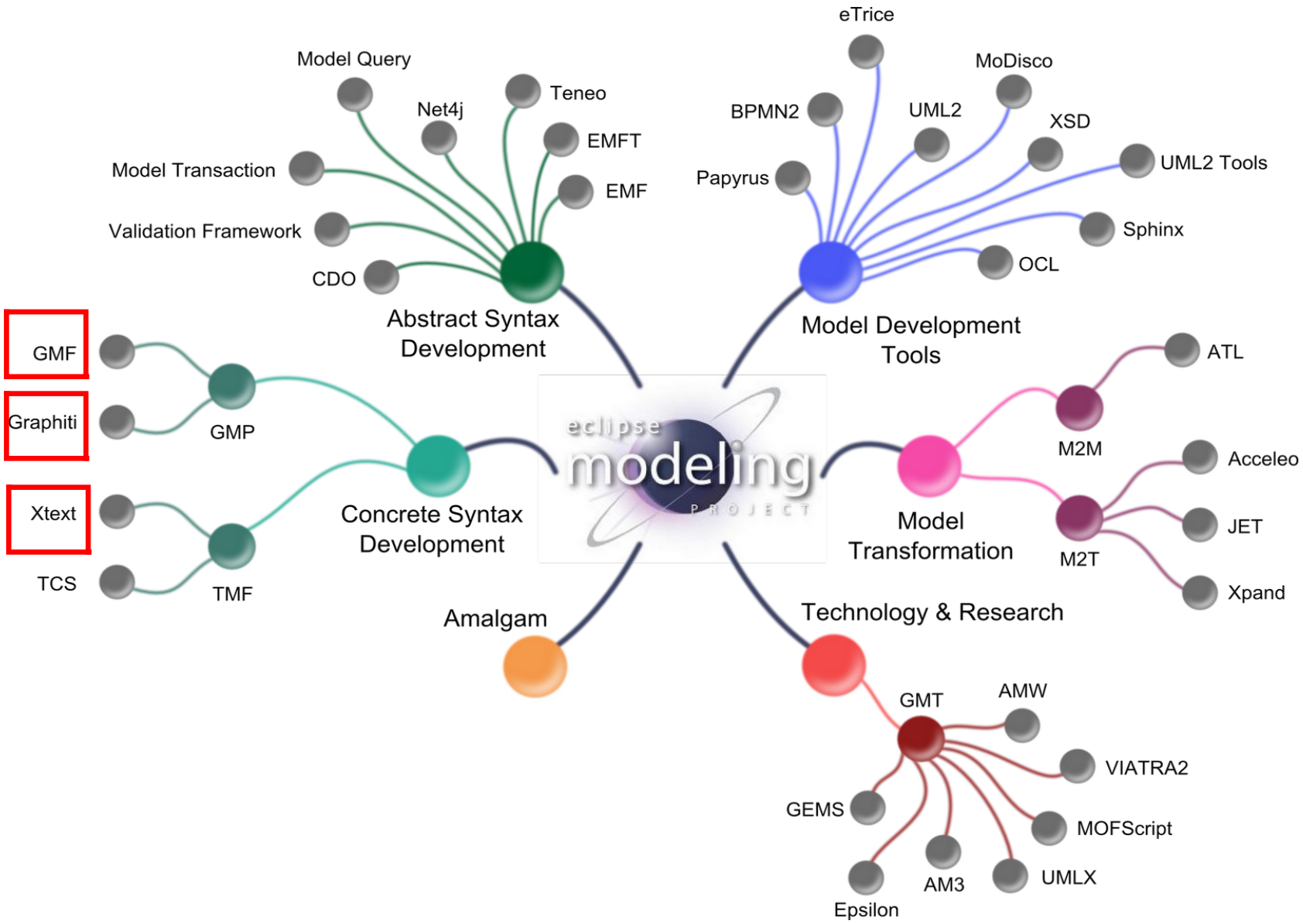


EuGENia

DSL Workbench



Eclipse Modeling Project



https://www.eclipse.org/Xtext/

Aplicaciones | Sitios sugeridos | Galería de Web Slice | Importado de Internet... | MouseAdapter (Java ... | Hotmail gratuito | Otr

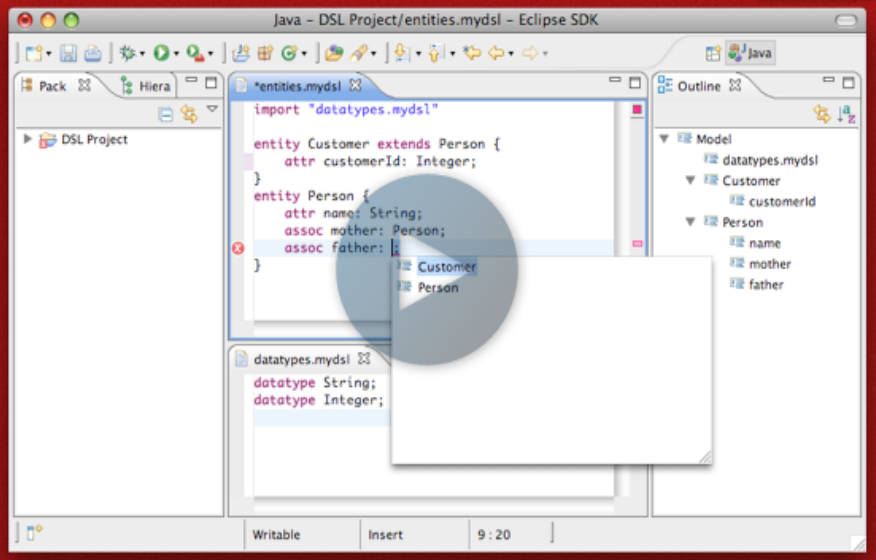
Xtext | News | Download | 7 Languages | Documentation | Community | Xtend | Eclipse.org | 8+1 467

Come to XtextCON

LANGUAGE DEVELOPMENT MADE EASY!

Building your own domain-specific languages has never been so easy. Just put your grammar in place and you not only get the working parser and linker but also first class Eclipse support.

Download



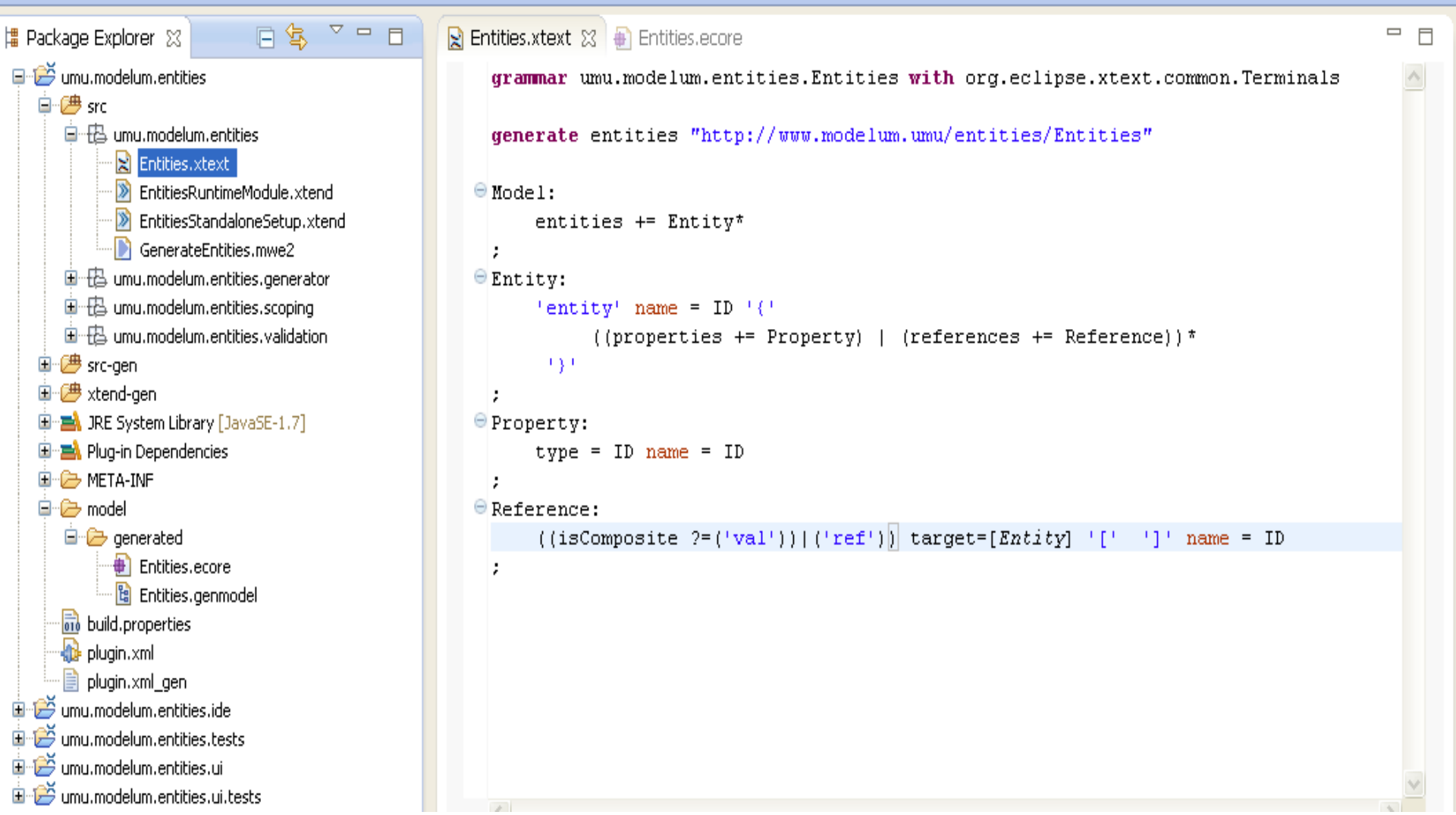
WHAT IS XTEXT?

Xtext is a framework for development of programming languages and domain specific languages.

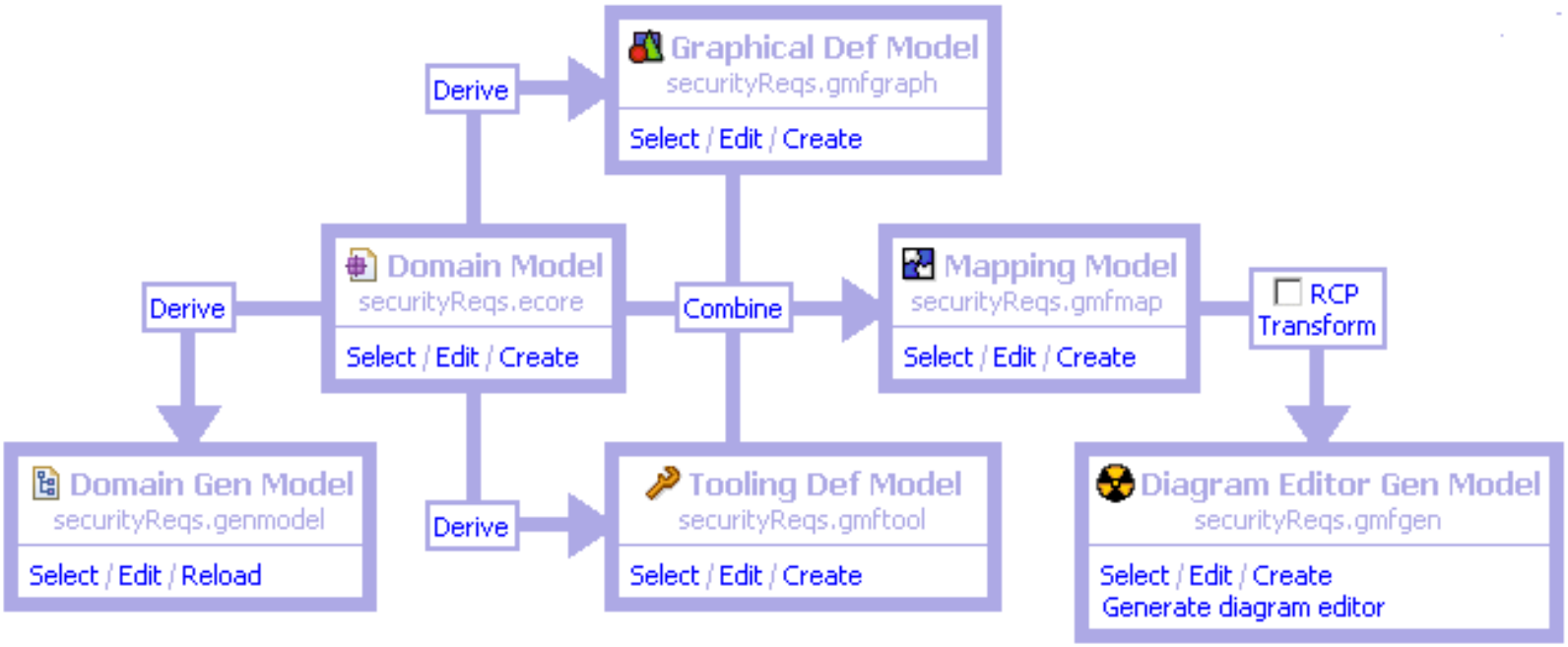
It covers all aspects of a complete language infrastructure, from parsers, over linker, compiler or interpreter to fully-blown top-notch Eclipse IDE integration. It comes with good defaults for all these aspects and at the same time every single aspect can be tailored to your needs.

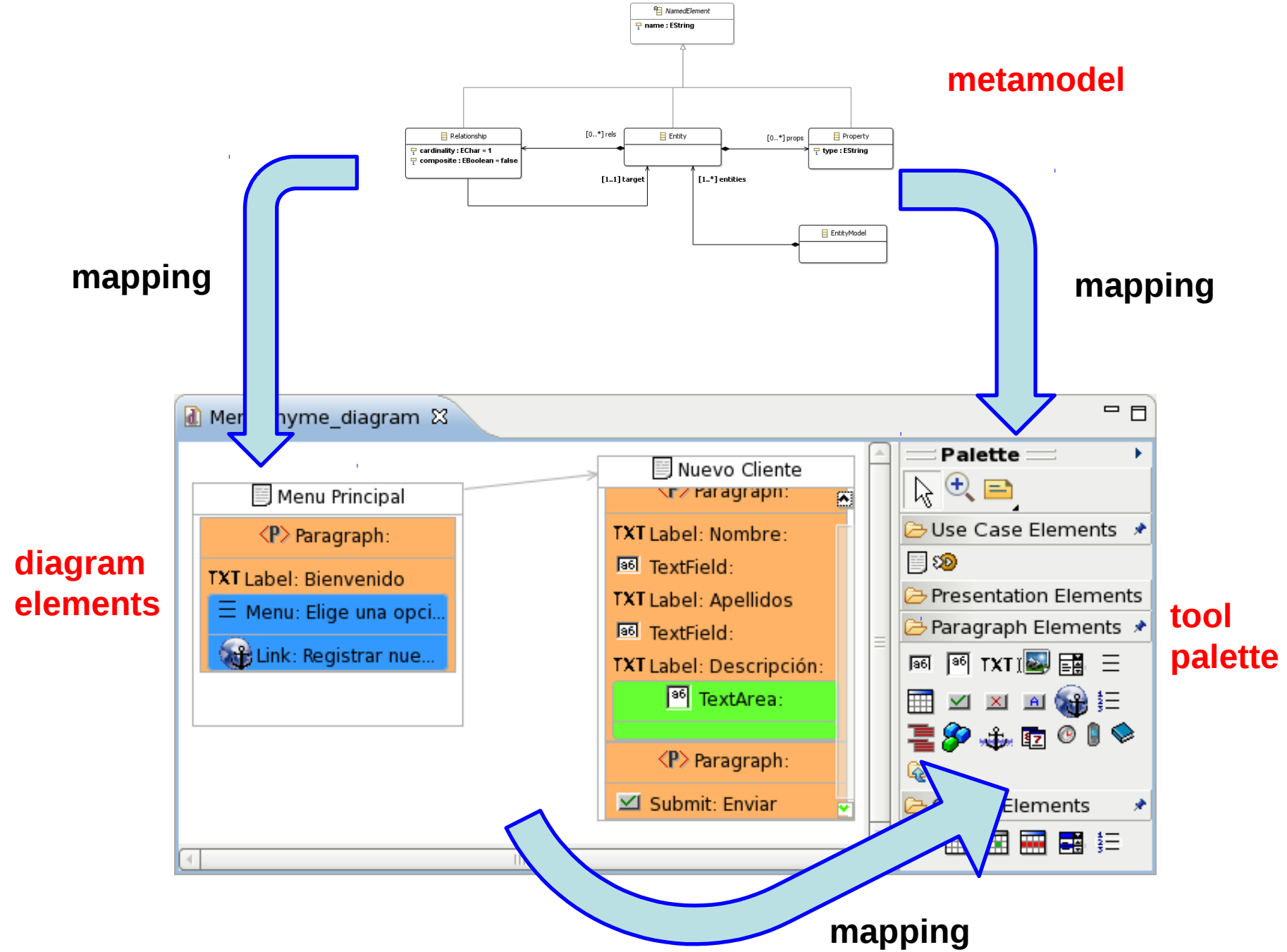
A Selection Of Supported Features

Step 1. Create Xtext grammar



GMF/Eclipse


Graphical Def Model
securityReqs.gmfgraph
Select / Edit / Create
Derive
Domain Model
securityReqs.ecore
Select / Edit / Create
Derive
Combine
Mapping Model
securityReqs.gmfmap
Select / Edit / Create
RCP
Transform
Domain Gen Model
securityReqs.genmodel
Select / Edit / Reload
Derive
Tooling Def Model
securityReqs.gmftool
Select / Edit / Create
Diagram Editor Gen Model
securityReqs.gmfgen
Select / Edit / Create
Generate diagram editor


NamedElement
name : EString
Relationship
cardinality : EChar = 1
composite : EBoolean = false
[0..*] rels
Entity
[0..*] props
Property
type : EString
[1..1] target
[1..*] entities
EntityModel
metamodel
mapping
mapping
diagram elements
Menu Principal
Paragraph:
TXT Label: Bienvenido
Menu: Elige una opci...
Link: Registrar nue...
Nuevo Cliente
TXT Label: Nombre:
TextField:
TXT Label: Apellidos
TextField:
TXT Label: Descripción:
TextArea:
Paragraph:
Submit: Enviar
Palette
Use Case Elements
Presentation Elements
Paragraph Elements
tool palette
mapping

Eugenia/GMF

```
@namespace(uri="http://www.modelum.um/entity", prefix="entity")
package entity;

class EntityModel {
  val Entity[+] entities;
}

class Entity extends NamedElement {
  val Property[*] props;
  val Relationship[*] rels;
}

class Property extends NamedElement {
  attr String[1] type;
}

class Relationship extends NamedElement {
  attr char[1] cardinality = '1';
  ref Entity[1] target;
  attr boolean[1] composite;
}

abstract class NamedElement {
  attr String[1] name;
}
```

@gmf

@gmf.diagram

@gmf.node(label="name")

@gmf.compartiment

@gmf.link(target.decoration="arrow",

MetaCase

Google Custom Search

HOME SOLUTION PRODUCTS STORE SUPPORT COMPANY Print | Search

MetaEdit+ Workbench MetaEdit+ Modeler Multi-user Integration System requirements Feature list

User-friendly modeling tools:

- Diagram Editor
- Matrix Editor
- Table Editor
- Browsers
- Changes and Versions

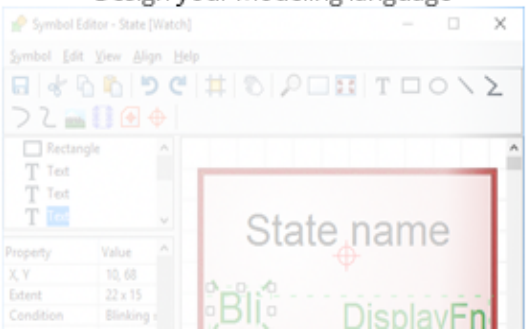
See Changes

VIEW VIDEO

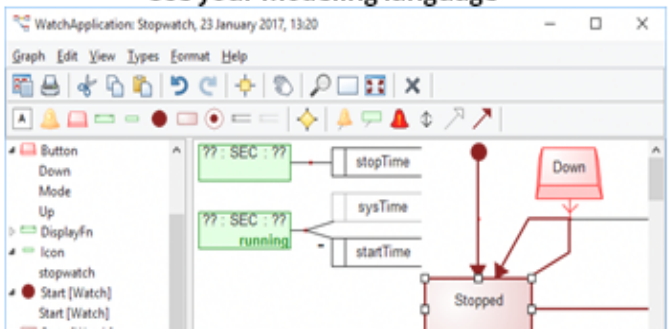
MetaEdit+ Modeler - Supports your modeling language

MetaEdit+ provides full modeling tool functionality for multiple users, multiple projects, running on all major platforms. It automatically follows the modeling language defined with MetaEdit+ Workbench by retrieving it from the repository.

Design your modeling language

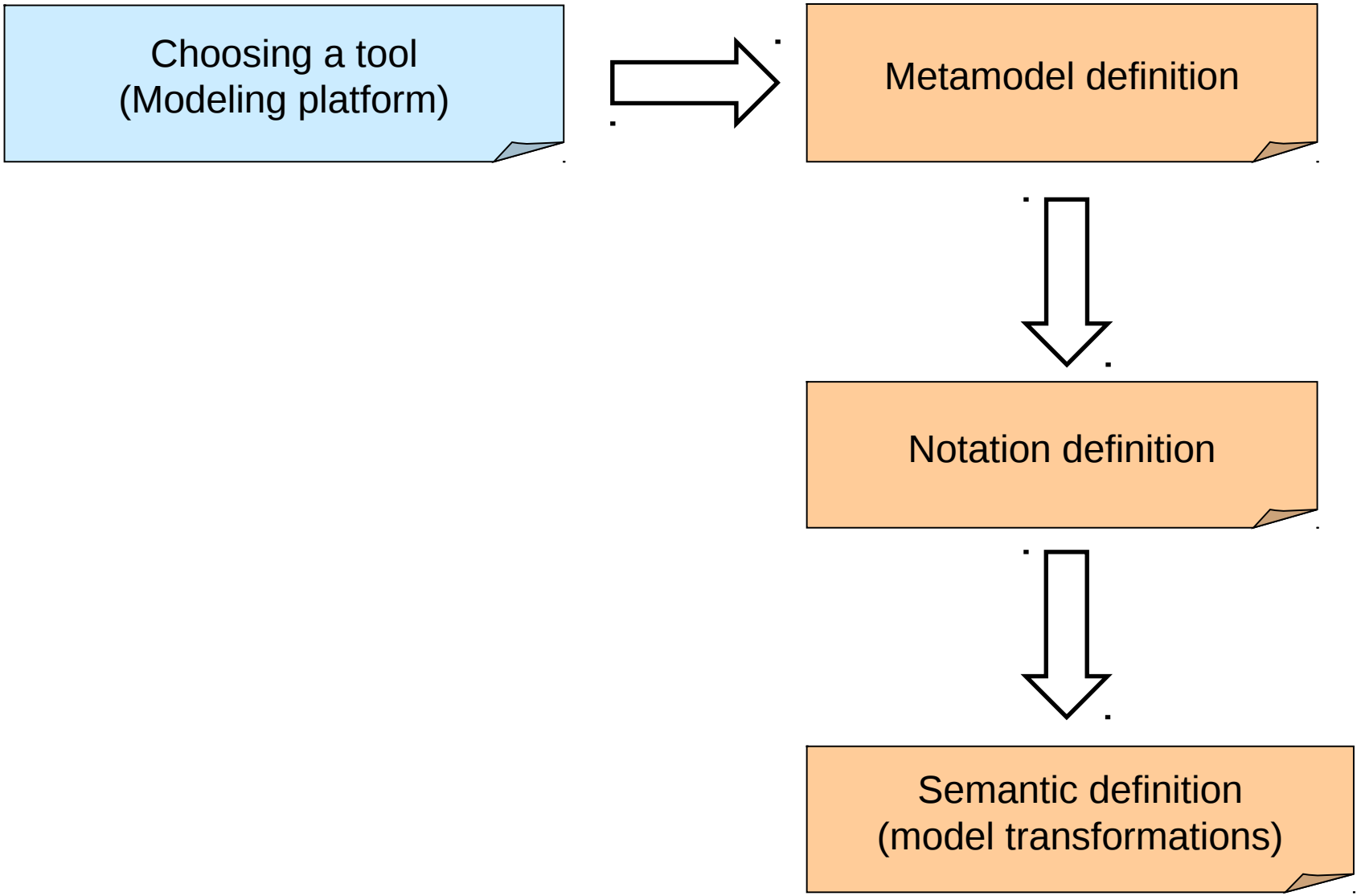


Use your modeling language

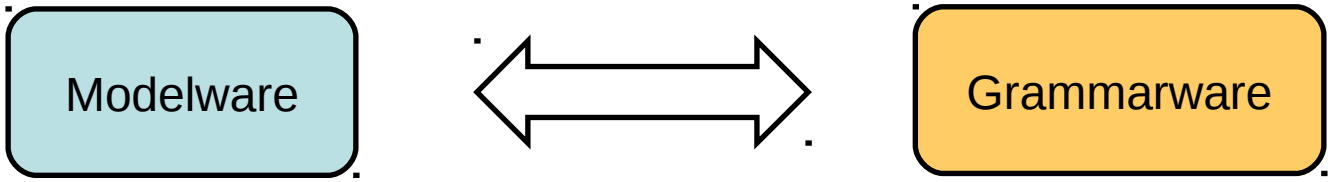


La herramienta más madura y robusta pero no está integrada en Eclipse y es de pago.

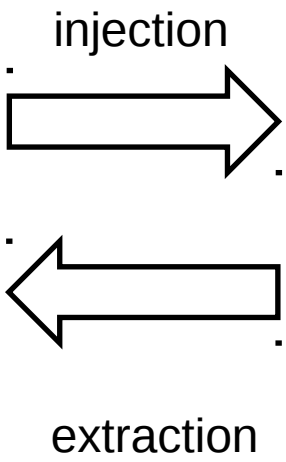
Steps in the creation process

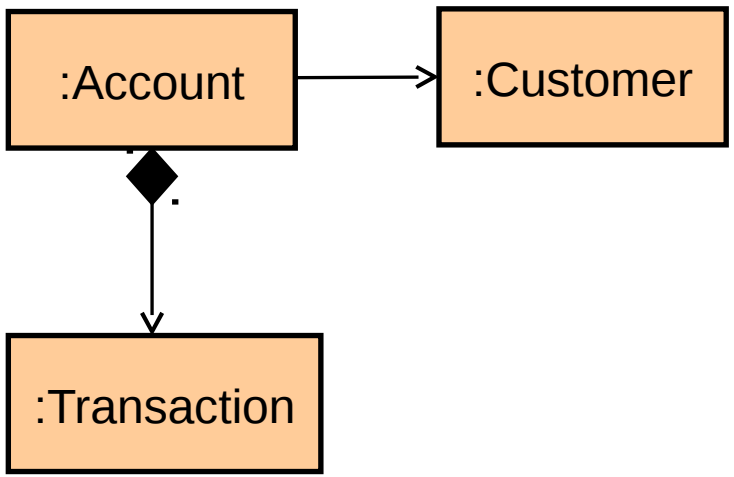


Textual concrete syntax



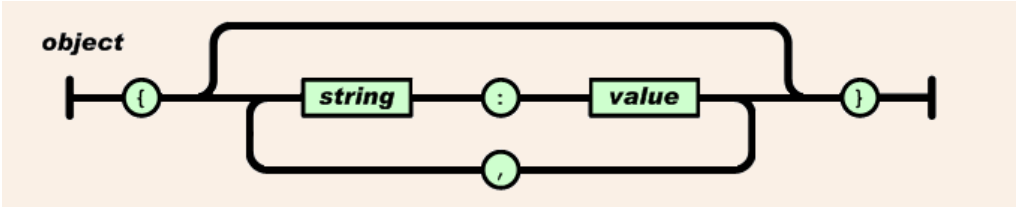

```
entity Account{
    ref Customer[+] holder;
    val Transaction[*] transactions;
    String number;
    Integer amount;
    String passport;
}
entity Customer{
     String name;
     String address;
     Account[+] accounts;
}
entity Transactions{
     Integer amount;
     Date date;
}
```

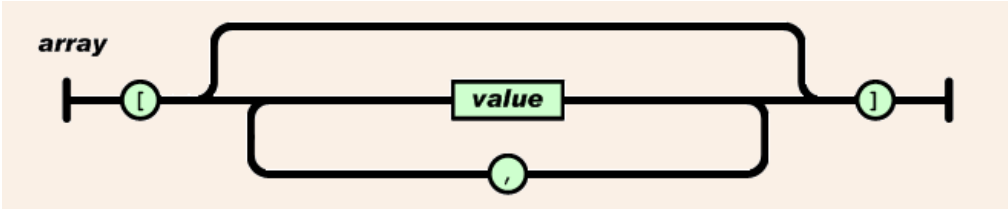




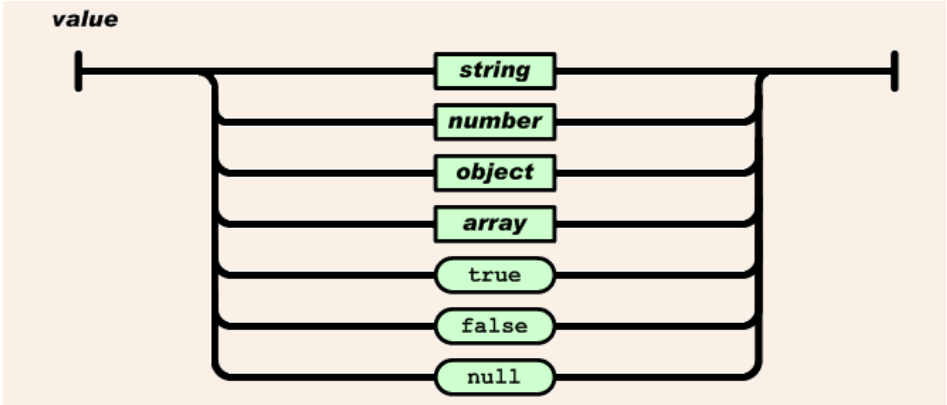
Injecting JSON code

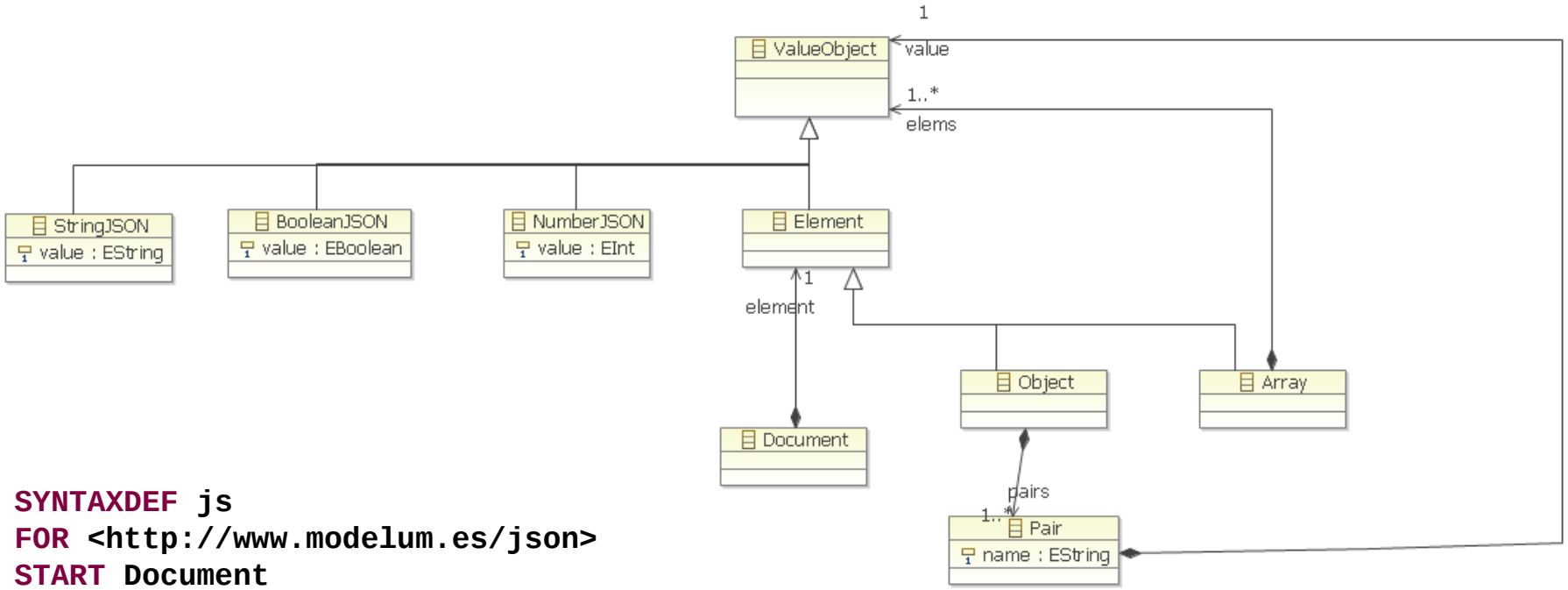
```
{"menu": {
    "id": "file",
    “value": "File",
    "popup": {
        "menuitem": [
            {"value": "New",  "onclick": "CreateNewDoc()"},
            {"value": "Open", "onclick": "OpenDoc()"},
            {"value": "Close", "onclick": "CloseDoc()"}
        ]
    }
} }
```





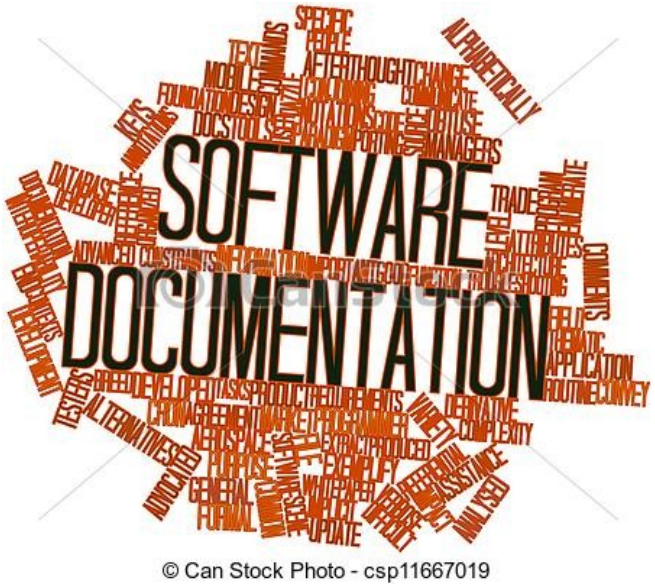

```
{ “nombre”:”CarlosGM”
  “edad”: 44
  “empresa”:”UMU”
  “ocio”: [“futbol”,”cine”, “musica”]
}
```



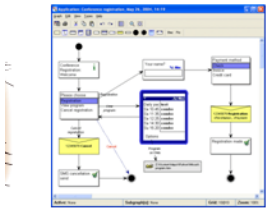


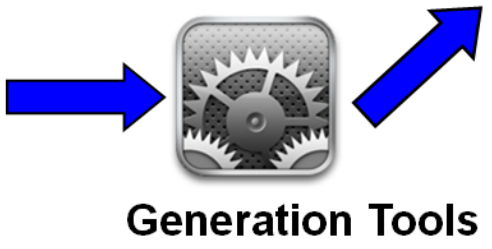

```
SYNTAXDEF js
FOR <http://www.modelum.es/json>
START Document
TOKENS {
  DEFINE BOOLEAN $'true'|'false'$;
  DEFINE COMMENT $'//'(~('\n'|'\r'|'\uffff'))*$;
  DEFINE INTEGER $('-')?('1'..'9')
('0'..'9')*|'0'$;
}

RULES {
   Document   ::= element ;
   Object ::= "{" pairs ( "," pairs)* "}";
   Pair ::= name['"','"'] ":" value;
   StringJSON ::= value['"','"'];
   BooleanJSON ::= value[BOOLEAN];
   NumberJSON ::= value[INTEGER];
   Array ::= "[" elems ("," elems)* "]";
}
```

Por qué son útiles los modelos?



© Can Stock Photo - csp11667019





Lenguajes de modelado vs. DSL

- Lenguajes de modelado
  - Sólo estamos interesados en documentación y/o visualización, comunicación y razonamiento.
  - Interesados en la notación
  - Se requiere un editor
  - UML ha sido normalmente utilizado de este modo
- DSL/DSML
  - También estamos interesados en la semántica
  - Generación de código (GPL, XML, HTML, SQL, …)
  - Lenguajes de modelado específicos del dominio (DSML) o lenguajes específicos del dominio (DSL)

Recomendaciones para diseñar DSLs (Markus Völter, “Best MD* Practices”, http://www.voelter.de/data/pub/DSLBestPractices-2011Update.pdf)

- Comienza con la creación de un DSL para un dominio técnico en vez de un DSL de un dominio del negocio.
- Expresividad: Debe permitir especificar el qué no el cómo.
- Notación adaptada al dominio que facilite la tarea de los usuarios
- Antes de construir un sofisticado editor gráfico, crear uno más simple para comprender bien los conceptos.
- Puede ser útil combinar un DSL textual con uno visual.
- DSLs con diferentes viewpoints
- DSLs deben permitir particionar los modelos o especificaciones

Recomendaciones para crear procesadores de DSLs

- Elegir implementación más adecuada
- Aplicar validación de modelos antes de procesarlos.
- ¡No modificar el código generado!
- Preocupación por el código generado: comentarios, seguir convenciones,..
- Crear plantillas de calidad: modulares y legibles.
- Aplicar transformaciones M2M intermedias para simplificar los generadores.
- ¡No olvidar el testing!

Recomendaciones para el proceso

- Aplicar un proceso iterativo
- Implementar el lenguaje conforme se realiza el análisis del dominio ayuda a comprenderlo.
- Documentación es importante: manual de referencia del DSL, manual de editores y generadores, cómo y dónde escribir código manual y cómo integrarlo.

Issues to be evaluated

- Cost of creation
  - Tools to be implemented: parsers, editors, ...
- ROI (return of investment)

DSL Creator

- Interoperability (GPLs, tools, ..)
- Ease to learn.
- Tooling
- Documentation

DSL User

Empirical Comparison of Language Workbenches

Steven Kelly

MetaCase
Ylistönmäentie 31
40500 Jyväskylä, Finland
+358 14 641000 ext. 21
stevek@metacase.com

DSM'13, October 27, 2013,
http://dx.doi.org/10.1145/2541935.2541935

Abstract

Production use of Domain-Specific Modeling languages has consistently shown productivity increase by a factor of 5–10. However, the spread of DSM has been slowed by projects stalling even before the language was built, often citing problems with the chosen tool. With a wide variety of language workbench tools for DSM, there is a need for objective empirical tool comparison – particularly as the little research so far shows a range of 50 times more effort between the most and least efficient tools. This article looks at existing empirical research and an experimental design for a future comparison. We aim at increasing objectivity and repeatability while keeping overall effort practical, and providing worthwhile returns for the participants.

This article aims to form a preliminary investigation of the area of empirical comparison of language workbenches. We will look at the particularly challenges of quantitative comparison in this area, the different factors that could be compared, and previous comparisons. Based on this we will offer a suggestion for an experimental design appropriate for a future comparison.

2. Challenges of Comparison

Conducting an empirical comparison of language workbenches is difficult, particularly given the wide range of effort required for the same results. In most cases, only an unrealistically small language could be built purely for an experiment: otherwise using the less efficient tools will take too long. The more realistic data

a) comparing language workbenches as different ways of producing a DSM tool for the same language;
b) comparing the effort to update the resulting DSM tool when the LWB, problem or solution domain evolves;
c) comparing the productivity of the resulting DSM tool and generation against hand-writing the same code;
d) comparing the productivity of different languages made for the same domain with different workbenches;
e) comparing the performance of the resulting DSM tool: how long the user has to wait for the tool to open a model, generate code, show model changes etc.

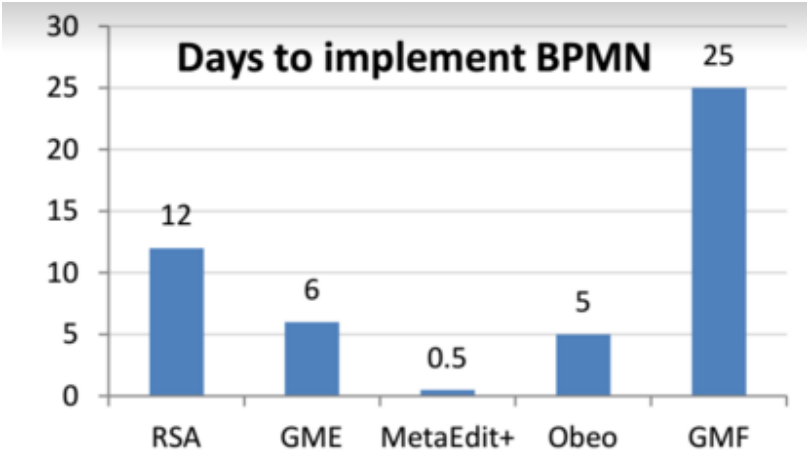


Figure 1: Days to implement BPMN [2]

Hanns-Alexander Dietrich, Dominic Breuker, Matthias Steinhorst,
Patrick Delfmann, Jörg Becker

Developing Graphical Model Editors for Meta-Modelling Tools

Requirements, Conceptualisation, and Implementation

Capability / Product [Source]	C1: Spec. at run-time	C2: Multiple languages	C3: Editor for rep.	C4: Cust. of symbols	C5: Cust. of edges	C6: Cust. of edge sym.	C7: Implicit rel.	C8: Shared repository
ADONIS (Junginger et al. 2000)	—	X	—	B	B	B	—	X
ARIS (Davis 2008; Scheer 2000)	(X)	X	X	B	B	B	—	X
AToM³ (Lara and Vangheluwe 2002, 2004)	—	X	—	B	B	B	—	X
DIAGEN (Minas 2002)	—	—	—	A	A	A	—	—
DIAMETA (Minas 2006)	—	—	—	A	A	A	—	—
DOME (Engstrom and Krueger 2000)	—	—	—	B	B	B	—	—
EuGENia (Kolovos et al. 2009)	—	—	—	A	A	A	X	—
GEMS (J. White et al. 2007)	—	—	(X)	C	B	B	—	—
GenEd (Haarslev and Wessel 1996)	—	X	—	B	B	B	X	X
GenGed (Bardohl 2002; Bardohl and Ehrig 2000)	—	X	X	B	B	B	—	X
GME (Ledeczi et al. 2001)	X	X	—	A	A	A	—	—
GMF (The Eclipse Graphical Modeling Framework (GMF) 2012)	—	—	—	A	A	A	X	—
GraMMI (Sapia et al. 2000)	X	X		A	A	A		X

Comparing tools to build graphical modeling editors

By **David Granada** 28/09/2016 | 10:00 Posted in article, DSLs, empirical studies 5

Tools	*Scope*	*Frame-work*	*Abs. syntax*	*Conc. syntax*	*Syntax distinct.*	*Editing*	*Models*	*Auto-mation*	*Usabi-lity*	*Meth. basis*
Diagen	OS (GPL)	Eclipse	Ecore/ UML	DiaMeta Design	No	✔✔	✔✔	✔✔	✔✔✔	-
Eugenia	OS (EPL)	Eclipse	Ecore	EOL	Yes	✔✔	✔✔✔	✔✔✔	✔✔✔	-
GMF	OS (EPL)	Eclipse	Ecore	Draw2D	Yes	✔✔✔	✔✔✔	✔✔	✔✔	-
Graphiti	OS (BSD)	Eclipse	Ecore/ Java	Draw2D	Yes	✔✔✔	✔✔✔	✔✔✔	✔✔	-
MetaEdit+	Com	Own	GOP-PRR	Internal API	Yes	✔✔✔	✔✔	✔✔✔	✔✔✔	✔
Obeo Designer	Com	Eclipse	Ecore	Odesign	Yes	✔✔✔	✔✔✔	✔✔✔	✔✔✔	✔
Sirius	OS	Eclipse	Ecore	Odesign	Yes	✔✔✔	✔✔✔	✔✔✔	✔✔✔	✔
Tiger	OS (GPL)	Eclipse	AGG	Shape-Fig	No	✔	✔✔	✔	✔✔✔	-
% of compliance of the qualitative criteria:						83.3%	87.5%	83.3%	91.6%	12.5%

Legend (for weightable fields): *None* (-), *Poor* (✔), *Good* (✔✔), *Excellent* (✔✔✔)

Summary of tools to build graphical modeling editors

Criterios de comparación

- Lenguaje de metamodelado y editores
- Notación y editores; cómo notación afecta al metamodelo
- Permite varios lenguajes o diagramas, asociar un diagrama a un elemento del modelo.
- Posibilidad de especificar los puntos de conexiones en relaciones
- Nivel de personalización del aspecto gráfico del DSL
- Soporte para restricciones sintácticas/semánticas
- Soporte para la generación de código: lenguajes m2m y m2t
- Interoperabilidad
- Usabilidad
- Documentación
- Precio, tipo de licencia, plataforma de despliegue,
- …