Linguistic Architecture
Modeling Software Knowledge

SoftLang Team, University of Koblenz-Landau

Prof. Dr. Ralf Lämmel
Msc. Johannes Härtel
Msc. Marcel Heinz

(C) 2018, SoftLang Team, University of Koblenz-Landau
Outline

- Motivating Software Documentation
- Classic Software Documentation
- Megamodel-based Documentation
- Introducing MegaL
  - JAXB Sketch
  - Django Megamodel
Motivation

Brainstorming

Joe, the programmer

Why do I need to document?
Motivation

Problems with a lack of documentation.

- Missing Expertise.
- High costs for introducing new technologies.
- Exhaustion.
- Job Security?
- Vendor lock in (dependency on a software vendor).

Joe, the programmer

Why do I need to document?
Motivation

How should I document software?

Joe, the programmer
The notion of viewpoints:

- A viewpoint describes the types of entities and relationships that are used to model a system.
- Different viewpoints are necessary for distinct stakeholders.
- A viewpoint is described in a metamodel.
Modeling Software – Viewpoints

Architecture Descriptions contain different views.

https://commons.wikimedia.org/wiki/File%3A4%2B1_Architectural_View_Model.svg
By mpan [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons
Modeling Software – Viewpoints

- Structure
- Usage
- Process

Features
- „total“: Gehälter summieren
- „cut“: Gehälter halbieren
- Nutzerinterface
- Persistenz in der Datenbank
- ...

(C) 2018, SoftLang Team, University of Koblenz-Landau
Modeling Software - Viewpoints

- Structure
- Usage
- Process


(C) 2018, SoftLang Team, University of Koblenz-Landau
Modeling Software – Viewpoints

- Structure
- Usage
- Process


By Stkl [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

(C) 2018, SoftLang Team, University of Koblenz-Landau
Software Documentation

- Elucidative Programming
- Literate Programming
- Tutorials

https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html

(C) 2018, SoftLang Team, University of Koblenz-Landau
Software Documentation

- Elucidative Programming
- Literate Programming
- Tutorials


```latex
\section{Hello world}

Today I awoke and decided to write some code, so I started to write Hello World in \texttt{C}.

```c
<<hello.c>>=

/*
  <<license>>
*/
#include <stdio.h>

int main(int argc, char *argv[]) {
  printf("Hello World!\n");
  return 0;
}
```

@\texttt{noindent \ldots} then I did the same in PHP.
Django's official Tutorials -
https://docs.djangoproject.com/en/1.11/intro/tutorial01/

Creating a project

If this is your first time using Django, you'll have to take care of some initial setup. Namely, you establish a Django project — a collection of settings for an instance of Django, including default and application-specific settings.

From the command line, cd into a directory where you’d like to store your code, then run the

$ django-admin startproject mysite

This will create a mysite directory in your current directory. If it didn’t work, see Problems running

Note

You’ll need to avoid naming projects after built-in Python or Django components, using names like django (which will conflict with Django itself) or test (which...
Modeling Software Knowledge - Problem

- What is <Technology> used for?
- How do I use <Technology> in the intended way?
- How does <Technology> work internally?
- What technologies are used by <Project> and how?
Modeling Software Knowledge – Problem

We have a problem!
Too many technologies.
Too little time.

Acknowledgement: Jean-Marie Favre designed this (such a) slide.

(C) 2018, SoftLang Team, University of Koblenz-Landau
Prescriptive vs. Descriptive

- Prescriptive models describe the properties holding for software that exists in the future.
  - Specification purpose
  - Target of code generation
- Descriptive models describe properties of existing software.
  - Documentation purpose
Modeling Software Knowledge — Big Picture @Softlang

(C) 2018, SoftLang Team, University of Koblenz-Landau
Definitions:
- Megamodels describe models and their relationships.
- Linguistic Architecture describes software from a conceptual perspective based on megamodels.
- MegaL is short for Megamodelling Language and is used to code such conceptual knowledge.

Implementations:
- **MegaL-Text** aims at tool support for modeling linguistic architecture of technologies and their usage in systems while emphasizing simplicity, modularity and well-formedness checking.
- Other:
  - **MegaL/Xtext** aims at validating facts in an actual system through custom plugins.
  - **QegaL** aims at processing and inferring facts from actual systems.
MegaL - A Model for JAXB
Brainstorming Task:
Let us assume that we are supposed to write a web application using Django as a framework.

What do we have to know to get started (Workflows, Facts)?
1. Conceptual Overview
   a. What are general concepts for web applications?
   b. Is there some usable documentation resource, where we can learn facts about Django?
   c. Based on which concepts does Django facilitate such principles?

2. Project Overview
   a. What are the roles of artifacts in a project?
   b. What is the manifestation type of artifacts in a project?
   c. What languages are used?

3. Scenarios
   a. Initialization
   b. Request to Response
Principle Architecture of Web Applications

- It is typically a Client-server architecture.
- End users access a client (in a Web browser).
- Services are implemented on a (Web) server.
- Services are materialized as documents.
- Client & server communicate via protocols over the internet.
Django’s Documentation
https://docs.djangoproject.com/en/2.0/

- Tutorials
  - Your first steps with Django

- Topic Guides
  - Key concepts and topics

- Reference Guides
  - Technical references for APIs, etc.

- How to Guides
  - Recipes on key problems and use cases.

(C) 2018, SoftLang Team, University of Koblenz-Landau
Django - Concepts Facilitated

Rapid Development

Security

Scalability

Model-View-Controller

Object-Relational-Mapping

Routing

Testing

Template Processing

(C) 2018, SoftLang Team, University of Koblenz-Landau
Django - Concepts Facilitated

A WebApp

- **Django.urls** facilitates **Routing**
- **Django.template** facilitates **Template Processing**
- **Django.test** facilitates **Testing**
- **Django.db** facilitates **Database Access**

(C) 2018, SoftLang Team, University of Koblenz-Landau
Agenda

1. Conceptual Overview
2. Project Overview
   a. What are the roles of artifacts in a project?
   b. What is the manifestation type of artifacts in a project?
   c. What languages are used?
3. Scenarios
   a. Initialization
   b. Request to Response
We need a Demo!

What is the best programming language?

<table>
<thead>
<tr>
<th>choice_text</th>
<th>votes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>15</td>
</tr>
<tr>
<td>Java</td>
<td>11</td>
</tr>
<tr>
<td>Cobol</td>
<td>4</td>
</tr>
</tbody>
</table>

Vote again?
> pip install Django

> python

>> import django

>> print (django.get_version())

2.0
Django

- Initialization

- Install
- Initialize Project
- Initialize App
- Run Server

Project vs. App: The project holds configuration while the app is the actual program executed on the web. One project can have multiple apps.

```bash
> django-admin startproject poll_analysis
```

Below current folder.

Below poll_analysis.
Django
- Initialization

- Install
- Initialize Project
- Initialize App
- Run Server

Project vs. App: The project holds configuration while the app is the actual program executed on the web. One project can have multiple apps.

```
> python manage.py startapp poll_analysis_app
```

Below poll_analysis_app:
Django - Initialization

- Install
- Initialize Project
- Initialize App
- Run Server

Project and App can be initialized at once using IDEs like PyCharm.
**Django**

- Initialization

- Install
- Initialize Project
- Initialize App
- Run Server

```bash
> cd poll_analysis
> python manage.py runserver
```

See [http://127.0.0.1:8000/](http://127.0.0.1:8000/)

The install worked successfully! Congratulations!

You are seeing this page because `DEBUG=True` is in your settings file and you have not configured any URLs.
Concepts behind Python scripts

- mysite
  - __init__.py implements *Initialization*
- manage.py implements *Administration*
- polls
  - __init__.py implements *Initialization*
  - admin.py implements *View*
  - models.py implements *Model*
  - tests.py implements *Testing*
  - views.py implements *View*
- settings.py implements *Configuration*
- urls.py implements *Router (Routing)*
Concepts behind Python scripts

- mysite
  - __init__.py implements Initialization
  - manage.py implements Administration
- polls
  - __init__.py implements Initialization
  - admin.py implements View
  - models.py implements Model
  - tests.py implements Testing
  - views.py implements View
- settings.py implements Configuration
- urls.py implements Router (Routing)
Concepts behind Python scripts

- mysite
  - __init__.py implements **Initialization**
  - manage.py implements **Administration**
- polls
  - __init__.py implements **Initialization**
  - admin.py implements **View**
  - models.py implements **Model**
  - tests.py implements **Testing**
  - views.py implements **View**
  - settings.py implements **Configuration**
  - urls.py implements **Router (Routing)**
Django

Manifestation Types

- db.sqlite3: File
- mysite: Folder
  - __init__.py: File
  - manage.py: File
  - media: Folder
    - __init__.py: File
    - admin.py: File
    - models.py: File
    - tests.py: File
    - views.py: File
    - settings.py: File
    - templates: Folder
      - admin: Folder
        - polls: Folder
          - detail.html: File
          - index.html: File
          - results.html: File
    - urls.py: File
Django

Languages

- db_sqlite3: File ∈ SL3IMG (a language we made up)
- mysite: Folder
  - __init__.py: File ∈ Python
  - manage.py: File ∈ Python
  - media: Folder
  - polls: Folder
    - __init__.py: File ∈ Python
    - admin.py: File ∈ Python
    - models.py: File ∈ Python
    - tests.py: File ∈ Python
    - views.py: File ∈ Python
  - settings.py: File ∈ Python
  - templates: Folder
    - admin: Folder
    - polls: Folder
      - detail.html: File ∈ HTML
      - index.html: File ∈ HTML
      - results.html: File ∈ HTML
    - urls.py: File ∈ Python
?websiteFolder : Folder;
?manage : File;
  elementOf Python;
  hasRole Administration;
  partOf ?websiteFolder.
?settings : File;
  elementOf Python;
  hasRole Configuration;
  partOf ?websiteFolder.
?urls : File;
  elementOf Python;
  hasRole Django.URLDispatcher;
  partOf ?websiteFolder.
Django Megamodel

Agenda

1. Conceptual Overview
2. Project Overview
3. Scenarios
   a. Initialization
   b. Request to Response
?websiteParts : Artifact+;
   elementType : Python;
   ^partOf : ?manage;
   ^partOf : ?settings;
   ^partOf : ?urls.
?name : Transient;
   elementType : PythonString.
startProject : PythonString -> Python .
startProject (?name) |-> (?websiteParts) .
Initialization

```plaintext
?appParts : Artifact+
  elementOf Python;
  ^partOf ?admin;
  ^partOf ?models;
  ^partOf ?tests;
  ^partOf ?views.
?appname : Transient;
  elementOf PythonString.
startApp : PythonString -> Python .
startApp(?appname) |-> ?appParts.
```
Django
- The Polls App

- View I.
- Routing I.
- Model
- Setup
- Model API
- Admin
- View II.
- Routing II.

```python
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello world!")
```

Page not found (404)

Request Method: GET
Request URL: http://127.0.0.1:8000/index

Using the URLconf defined in poll_analysis.urls, Django tried these URL patterns, in this order:

1. admin/

The current path, index, didn’t match any of these.

You’re seeing this error because you have DEBUG = True in your Django settings file. Change that to False.
Design the URL fragments at poll_analysis/urls.py.

Dispatch the route to the app’s routing.
Create poll_analysis_app/urls.py.

```python
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
]
```

Hello world!
Django
-The Polls App

- View I.
- Routing I.
- Model
- Setup
- Model API
- Admin
- View II.
- Routing II.

from django.db import models

class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date_published')

class Choice(models.Model):
    question = models.ForeignKey(Question,
        on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)

Edit poll_analysis_app/model.py.
Django
-The Polls App

- View I.
- Routing I.
- Model
- Setup
- Model API
- Admin
- View II.
- Routing II.

- Project Setting Possibilities in <project>/settings.py
  a. ENGINE:
     i. default: django.db.backends.sqlite3
     ii. django.db.backends.postgresql
     iii. django.db.backends.mysql
     iv. django.db.backends.oracle
     v. etc.
  b. NAME:
     i. default: NAME = os.path.join(BASE_DIR,'db.sqlite3')
  c. INSTALLED_APPS:
     i. django.contrib.admin
     ii. django.contrib.auth
     iii. django.contrib.contenttypes
     iv. django.contrib.sessions
     v. django.contrib.messages
     vi. django.contrib.staticfiles

(C) 2018, SoftLang Team, University of Koblenz-Landau
Django
-The Polls App

- View I.
- Routing I.
- Model
- Setup
- Model API
- Admin
- View II.
- Routing II.

Edit poll_analysis/settings.py

```
INSTALLED_APPS = [
    'poll_analysis_app.apps.PollAnalysisAppConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

(C) 2018, SoftLang Team, University of Koblenz-Landau
Django
-The Polls App

- View I.
- Routing I.
- Model
- Setup
- Model API
- Admin
- View II.
- Routing II.

> python manage.py migrate

Initialize the database according to poll_analysis/settings.py
Django
-The Polls App

- View I.
- Routing I.
- Model
- Setup
- Model API
- Admin
- View II.
- Routing II.

Migrations are committed to the version control system and shipped with applications to aid other developers and in production.

```
> python manage.py makemigrations poll_analysis_app
Create database migration for changes to the model.

> python manage.py migrate
Execute database migration.

> python manage.py check
Check for problems without actually modifying your database.

> python manage.py sqlmigrate poll_analysis_app 0001
Print SQL Statements:
- Which tables are created?
- Which fields are added to tables?
- etc.
```

(C) 2018, SoftLang Team, University of Koblenz-Landau
Model and Database – Correspondence

```
schemaResp:
CREATE TABLE "polls_poll" ( ... )
;
CREATE TABLE "polls_choice" ( ... )

mySite/polls/models.py:
class Poll(models.Model):
    ...

class Choice(models.Model):
    ...
```
DB schema – A Transient

> python manage.py sql polls

Response by Django

```
BEGIN;
CREATE TABLE "polls_poll" (  
"id" integer NOT NULL PRIMARY KEY,  
"question" varchar(200) NOT NULL,  
"pub_date" datetime NOT NULL
);
;
CREATE TABLE "polls_choice" (  
"id" integer NOT NULL PRIMARY KEY,  
"poll_id" integer NOT NULL REFERENCES "polls_poll" ("id"),  
"choice" varchar(200) NOT NULL,  
"votes" integer NOT NULL
);
;
COMMIT;
```
Django
-The Polls App

- View I.
- Routing I.
- Model
- Setup
- Model API
- Admin
- View II.
- Routing II.

> python manage.py shell

```python
>>> from poll_analysis_app.models import Question, Choice
>>> Question.objects.all()
<QuerySet [<Question: Question object (1)>]>
>>> from django.utils import timezone
>>> q = Question(question_text="What's the best programming language?", pub_date=timezone.now())
>>> q.save()
>>> Question.objects.all()
<QuerySet [<Question: Question object (1)>, <Question: Question object (2)>]>
>>> q.question_text = "What is the best language?"
>>> q.save()
>>> quit()
```

Interactive Model API

(C) 2018, SoftLang Team, University of Koblenz-Landau
Add a superuser after entering name, email and password.

Run the server to inspect administration at http://127.0.0.1:8000/admin/

> python manage.py createsuperuser

> python manage.py runserver
Django
-The Polls App

- View I.
- Routing I.
- Model
- Setup
- Model API
- Admin
- View II.
- Routing II.

(C) 2018, SoftLang Team, University of Koblenz-Landau
Add a generic list view for displaying questions on the index page by adding this snippet to poll_analysis_app/views.py.

```python
class IndexView(generic.ListView):
    template_name = 'poll_analysis/index.html'
    context_object_name = 'latest_question_list'

    def get_queryset(self):
        """
        Return the last five published questions (not including those set to be published in the future).
        """
        return Question.objects.filter(pub_date__lte=timezone.now()).order_by(' -pub_date')[0:5]
```

Constrain the displayed list of questions.
HTML template 'templates/poll_analysis/index.html' for the list view at the index page.

```html
{% if latest_question_list %}
    <ul>
    {% for question in latest_question_list %}
        <li><a href=" {% url 'detail' question.id % } ">
            {{ question.question_text }}
        </a></li>
    {% endfor %}
    </ul>
{% else %}
    <p>No polls are available.</p>
{% endif %}
```
Django
-The Polls App

- View I.
- Routing I.
- Model
- Setup
- Model API
- Admin
- View II.
- Routing II.

Python-like statements embedded into HTML.

(C) 2018, SoftLang Team, University of Koblenz-Landau
Embedding String expressions that are evaluated at runtime.
Django
-The Polls App

- View I.
- Routing I.
- Model
- Setup
- Model API
- Admin
- View II.
- Routing II.

{% if latest_question_list %}
  <ul>
    {% for question in latest_question_list %}
    <li><a href="{% url 'detail' question.id %}"
      >{{ question.question_text }}\</a</li>
    {% endfor %}
  </ul>
{% else %}
  <p>No polls are available.</p>
{% endif %}

Linking to defined routes, e.g., to the details view.
Add a generic detail view for poll results of a single question at poll_analysis_app/views.py

```python
class ResultsView(generic.DetailView):
    model = Question
    template_name = 'poll_analysis/results.html'
```

Add a template at templates/poll_analysis/results.html

```html
<h1>{{ question.question_text }}</h1>
<ul>
{% for choice in question.choice_set.all %}
    <li>{{ choice.choice_text }} -- {{ choice.votes }} vote{{ choice.votes|pluralize }}</li>
{% endfor %}
</ul>
<a href="% url 'statistics' question.id ">% Visual?
</a>
<a href="% url 'detail' question.id ">% Vote again?
</a>
```

(C) 2018, SoftLang Team, University of Koblenz-Landau
Django
-The Polls App

- View I.
- Routing I.
- Model
- Setup
- Model API
- Admin
- View II.
- Routing II.

A question object is the input.

```
33 class ResultsView(generic.DetailView):
34     model = Question
35     template_name = 'poll_analysis/results.html'

1 <h1>{{ question.question_text }}</h1>
2 <ul>
3  {% for choice in question.choice_set.all %}
4     <li>{{ choice.choice_text }} -- {{ choice.votes }} votes{{ choice.votes|pluralize }}</li>
3  {% endfor %}
4 </ul>
5 <a href="{% url 'statistics' question.id % }">Visual?</a>
6 <a href="{% url 'detail' question.id % }">Vote again?</a>
```
Add a form template for voting to templates/poll_analysis/detail.html

```html
<form action="{% url 'vote' question.id %}" method="post">
  {% csrf_token %}
  {% for choice in question.choice_set.all %}
    <input type="radio" name="choice" id="choice{{ forloop.counter }}" value="{{ choice.id }}" />
    <label for="choice{{ forloop.counter }}">{{ choice.choice_text }}</label>
  {% endfor %}
  <input type="submit" value="Vote" />
</form>
```
Add a custom function for voting to poll_analysis_app/views.py

```python
44 def vote(request, question_id):
45     question = get_object_or_404(Question, pk=question_id)
46     try:
47         selected_choice = question.choice_set.get(pk=request.POST['choice'])
48     except (KeyError, Choice.DoesNotExist):
49         # Redisplay the question voting form.
50         return render(request, 'poll_analysis/detail.html', {
51             'question': question,
52             'error_message': 'You didn't select a choice.'},
53         )
54     else:
55         selected_choice.votes += 1
56         selected_choice.save()
57         # Always return an HttpResponseRedirect after successfully dealing
58         # with POST data. This prevents data from being posted twice if a
59         # user hits the Back button.
60         return HttpResponseRedirect(reverse('results', args=(question.id,)))
```
A post request with the vote and the question's ID are input.

```python
def vote(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    try:
        selected_choice = question.choice_set.get(pk=request.POST['choice'])
    except (KeyError, Choice.DoesNotExist):
        # Redisplay the question voting form.
        return render(request,
                      'poll_analysis/detail.html', {
                        'question': question,
                        'error_message': "You didn't select a choice."
                      })
    else:
        selected_choice.votes += 1
        selected_choice.save()
        # Always return an HttpResponseRedirect after successfully dealing
        # with POST data. This prevents data from being posted twice if a
        # user hits the Back button.
        return HttpResponseRedirect(reverse('results', args=(question.id,)))
```

(C) 2018, SoftLang Team, University of Koblenz-Landau
Django provides shortcuts for error handling.

```python
44 def vote(request, question_id):
45     question = get_object_or_404(Question, pk=question_id)
46     try:
47         selected_choice = question.choice_set.get(pk=request.POST['choice'])
48     except (KeyError, Choice.DoesNotExist):
49         # Display error message if no choice is selected.
50         return render(request, 'poll_analysis/detail.html', {  
51             'question': question,  
52             'error_message': 'You didn’t select a choice.',  
53         })
54     else:
55         selected_choice.votes += 1
56         selected_choice.save()
57         # Always return an HttpResponseRedirect after successfully dealing
58         # with POST data. This prevents data from being posted twice if a
59         # user hits the Back button.
60         return HttpResponseRedirect(reverse('results', args=(question.id,)))
```
```python
def vote(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    try:
        selected_choice = question.choice_set.get(pk=request.POST['choice'])
    except (KeyError, Choice.DoesNotExist):
        # Redisplay the question voting form.
        return render(request,
                       'poll_analysis/detail.html', {
                       'question': question,
                       'error_message': "You didn't select a choice."
                       })
    else:
        selected_choice.votes += 1
        selected_choice.save()
        # Always return an HttpResponseRedirect after successfully dealing
        # with POST data. This prevents data from being posted twice if a
        # user hits the Back button.
        return HttpResponseRedirect(reverse('results', args=(question.id,)))
```
Django
-The Polls App

- View I.
- Routing I.
- Model
- Setup
- Model API
- Admin
- View II.
- Routing II.

Edit the routes for the app at poll_analysis_app/urls.py

```python
urlpatterns = [
    path('', views.IndexView.as_view(), name='index'),
    path('<int:pk>/', views.DetailView.as_view(), name='detail'),
    path('<int:pk>/results/', views.ResultsView.as_view(), name='results'),
    path('<int:pk>/statistics/', views.StatisticsView.as_view(), name='statistics'),
    path('<int:question_id>/vote/', views.vote, name='vote')
]
```
Django
-The Polls App

- View I.
- Routing I.
- Model
- Setup
- Model API
- Admin
- View II.
- Routing II.

```python
urlpatterns = [
    path('', views.IndexView.as_view(), name='index'),
    path('<int:pk>/detail/', views.DetailView.as_view(), name='detail'),
    path('<int:pk>/results/', views.ResultsView.as_view(), name='results'),
    path('<int:pk>/statistics/', views.StatisticsView.as_view(), name='statistics'),
    path('<int:question_id>/vote/', views.vote, name='vote')
]
```

Every route is given a URL fragment and a view operation.

(C) 2018, SoftLang Team, University of Koblenz-Landau
Django

- The Polls App

- View I.
- Routing I.
- Model
- Setup
- Model API
- Admin
- View II.
- Routing II.

Reference resolution to routes based on names.

```python
urlpatterns = [
    path('', views.IndexView.as_view(), name='index'),
    path('<int:pk>/', views.DetailView.as_view(), name='detail'),
    path('<int:pk>/results/', views.ResultsView.as_view(), name='results'),
    path('<int:pk>/statistics/', views.StatisticsView.as_view(), name='statistics'),
    path('<int:question_id>/vote/', views.vote, name='vote'),
]
```

Pretty references in templates.

```html
<form action="{% url 'vote' question.id %}">
```

Pretty references at views.

```python
return HttpResponseRedirect(reverse('results', args=(question.id,)))
```
Django – View Namespaces

- There may be more than one application per website.
- Add namespaces by adding, e.g. app_name = 'poll_analysis' to poll_analysis_app/urls.py
- Refer to namespaces as follows:

  <li><a href="{% url 'poll_analysis:detail' question.id %}">{{ question.question_text }}</a></li>
?request : Transient;
    elementoOf HTTPRequest.
?url : Transient;
    elementoOf URL;
    partOf ?request.
$response : Transient;
    elementoOf HTTPResponse.
$responseHTML : Transient;
    elementoOf HTML5;
    partOf ?response.
respond : HTTPRequest -> HTTPResponse.
?App implements respond.
DjangoViewLanguage : ValueLanguage;
   = "https://docs.djangoproject.com/en/2.1/intro/tutorial01"
   subsetOf PythonFunctionLanguage.
?CustomViewLanguage : ValueLanguage;
   subsetOf DjangoViewLanguage;
   ^defines ?views.
?customView : Transient;
   elementOf ?CustomViewLanguage;
   hasRole MvcView.
?viewFunction : Transient;
   elementOf DjangoViewLanguage.
getView : URL -> DjangoViewLanguage. //TODO: Dispatch variant
getView ^composedOf respond;
   ^defines ?urls.
getView(?url) |-> ?customView.
Django

MegaL

generateHTML : HttpRequest # DjangoTemplateLanguage -> HTML5 .
?customView defines generateHTML.
generateHTML(?request, ?template) |-> ?responseHTML .
● Integrate static web content.
● Prettify the content using CSS.
● Test model and view code.
● Customize Administration pages.
Actually writing megamodules is complex. The modularization assumes the following structure:
Summary

- MegaL can be used to share knowledge on technologies.
- The root entity types are Technologies, Languages, Artifacts and Concepts.
- Such knowledge on artifacts can be derived from projects.
  - Visualized megamodels can enhance IDEs.

- See [http://softlang.wikidot.com/mega](http://softlang.wikidot.com/mega) for papers and ongoing work.