Technology modeling

Ralf Lämmel
Software Languages Team
University of Koblenz-Landau

Technologies are at the heart of software development. Let’s model them for understanding.

Acknowledgement: This is joint work with Andrei Varanovich (PhD student), Dr. Favre (collaboration). Thanks also to various students for their involvement with the topic.
What’s modeling again?
The notion of *model*

An abstract description of structure or behavior

**Structure** (and behavior) of an **information system**
Different kinds of models in software development

<table>
<thead>
<tr>
<th>Kind of model</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data models</td>
<td>an XML schema of an exchange format</td>
</tr>
<tr>
<td>Structure models</td>
<td>an object model of an application</td>
</tr>
<tr>
<td>Behavioral models</td>
<td>a state model for a certain class</td>
</tr>
<tr>
<td>Enterprise models</td>
<td>an org chart</td>
</tr>
<tr>
<td>Technology models</td>
<td>pattern of usage for a code generator</td>
</tr>
</tbody>
</table>
What's the coolest language?

- Cobol
- Python
- Haskell
- Java

Data model of a polls app such as Doodle

```
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
);
```
What's the coolest language?

- Cobol
- Python
- Haskell
- Java

Structure model of a polls app such as Doodle

<table>
<thead>
<tr>
<th>Poll</th>
<th>Choice</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ question: String</td>
<td>+ choice_text: String</td>
</tr>
<tr>
<td>+ pub_date: DateTime</td>
<td>+ votes: int</td>
</tr>
<tr>
<td>+ recently_published(): boolean</td>
<td></td>
</tr>
</tbody>
</table>

Definition of classes, attributes, and method signatures?
Behavioral model of a turnstile / revolving door

Definition of input, states, events, and transitions

Source: http://en.wikipedia.org/wiki/Turnstile
Unified Modeling Language

A general-purpose modeling language in software engineering

• Models of structure
  • Class diagrams
  • Component diagrams
  • Object diagrams
  • Package diagrams
  • …

• Models of behavior
  • Activity diagrams
  • State machine diagrams
  • Use case diagrams
  • Sequence diagrams
  • …
Domain-specific modeling is a software engineering methodology for designing and developing systems, such as computer software. It involves systematic use of a domain-specific language to represent the various facets of a system. [Wikipedia, 1 July 2014]

UML may be used or customized. Domain-specific modeling languages may be created instead.
Examples of domain-specific modeling

**Track control of railway systems:** Demonstrates a Domain-Specific Modeling language that uses the layout of the railway as the notation.

Source: [http://www.metacase.com/cases/dsm_examples.html](http://www.metacase.com/cases/dsm_examples.html)
Examples of domain-specific modeling

**Automotive infotainment system:** A DSM language for designing car infotainment systems.

Source: [http://www.metacase.com/cases/dsm_examples.html](http://www.metacase.com/cases/dsm_examples.html)
Examples of domain-specific modeling

**Industrial machine control:** A DSM solution for hydraulic machine control.

Source: [http://www.metacase.com/cases/dsm_examples.html](http://www.metacase.com/cases/dsm_examples.html)
Examples of domain-specific modeling

**Blood separation machines:** A DSM for developing blood separation machines, covering functional code generation and model-level debugging.

Source: [http://www.metacase.com/cases/dsm_examples.html](http://www.metacase.com/cases/dsm_examples.html)
It’s Ok to model software systems, but why to model technologies?
We have a problem!

Too much software technologies.
Too much software languages.
Too little time.

We need an ontology, standardized examples, and abstractions (models).
Example of „not quite“ a technology model

Java Architecture for XML Binding

From Wikipedia, the free encyclopedia

Java Architecture for XML Binding (JAXB) allows Java developers to map Java classes to XML representations. JAXB provides two main features: the ability to marshal Java objects into XML and the inverse, i.e. to unmarshal XML back into Java objects. In other words, JAXB allows storing and retrieving data in memory in any XML format, without the need to implement a specific set of XML loading and saving routines for the program's class structure. It is similar to xsd.exe and XmlSerializer in the .NET Framework.

JAXB is particularly useful when the specification is complex and changing. In such a case, regularly changing the XML Schema definitions to keep them synchronised with the Java definitions can be time consuming and error-prone.

JAXB is a part of the Java SE platform and one of the APIs in the Java EE platform, and is part of the Java Web Services Development Pack (JWSDP). It is also one of the foundations for WSIT. JAXB is part of SE version 1.6.

JAXB 1.0 was developed under the Java Community Process as JSR 31.[1] As of 2006, JAXB 2.0 is being developed under JSR 222.[2] Reference implementations for these specifications are available under the CDDL open source license at java.net.

Example of „not quite“ a technology model

Django (web framework)

From Wikipedia, the free encyclopedia

Django (/ˈdʒæŋɡoʊ/ JAHNG-goh)[3] is a free and open source web application framework, written in Python, which follows the model–view–controller architectural pattern.[4][5] It is maintained by the Django Software Foundation (DSF), an independent organization established as a 501(c)(3) non-profit.

Django's primary goal is to ease the creation of complex, database-driven websites. Django emphasizes reusability and "pluggability" of components, rapid development, and the principle of don't repeat yourself. Python is used throughout, even for settings, files, and data models. Django also provides an optional administrative create, read, update and delete interface that is generated dynamically through introspection and configured via admin models.


Example of „not quite“ a technology model

Example of „not quite“ a technology model

ANTLR

From Wikipedia, the free encyclopedia

In computer-based language recognition, ANTLR (pronounced Antler), or ANOther Tool for Language Recognition, is a parser generator that uses LL(*) parsing. ANTLR is the successor to the Purdue Compiler Construction Tool Set (PCCTS), first developed in 1989, and is under active development. Its maintainer is professor Terence Parr of the University of San Francisco.

ANTLR takes as input a grammar that specifies a language and generates as output source code for a recognizer for that language. While version 3 supported generating code in the programming languages Ada95, ActionScript, C, C#, Java, JavaScript, Objective-C, Perl, Python, Ruby, and Standard ML,[1] the current release at present only targets Java and C#. A language is specified using a context-free grammar which is expressed using Extended Backus–Naur Form (EBNF).

ANTLR allows generating lexers, parsers, tree parsers, and combined lexer-parsers. Parsers can automatically generate abstract syntax trees which can be further processed with tree parsers. ANTLR provides a single consistent notation for specifying lexers, parsers, and tree parsers. This is in contrast with other parser/lexer generators and adds greatly to the tool’s ease of use.

Source: http://en.wikipedia.org/wiki/ANTLR, 1 July 2014

© 2014, Ralf Lämmel, rlaemmel@acm.org
Introduction to technology modeling
A „technology model“ for bootstrapping a compiler

http://en.wikipedia.org/wiki/Tombstone_diagram
A „technology model“ for the mechanics of model transformation

The notion of technology model

- Technology models are „ER models“.
- Entities of interest
  - Software technologies and parts thereof, e.g., Hibernate
  - Software languages, e.g., SQL
  - Software artifacts, e.g., O/R mapping file
  - Software concepts, e.g., persistence
  - ...
- Relationships of interest
  - Conformance
  - Transformation
  - ...
A technology model for JAXB (XML-data binding in the Java platform)

Part 1: Technology break-down and concepts
A technology model for JAXB (XML-data binding in the Java platform)

Part 2: Type-level mapping
A technology model for JAXB (XML-data binding in the Java platform)

Part 3: Instance-level mapping
A technology model for JAXB (XML-data binding in the Java platform)

Part 4: Conformance
Vocabulary for technology modeling
Vocabulary overview

- **Entities** in software development
  - e.g.: Java, Python, J2EE, Django, Testing, Inheritance

- **Entity types** in software development
  - e.g.: Language, Technology, Concept

- **Relationships** in software development
  - e.g.: 
    - HelloWorld.java *elementOf* Java
    - Django *uses* Python

- **Relationship types** in software development
  - e.g., „elementOf“ or „uses“
Entities and types thereof
Entity types I / II

• Languages, e.g., *Python*

• Technologies, e.g., *Django web framework*

• System, e.g., an *information system*

• Artifacts, e.g., files or more concretely a *Python script*

• Fragments, e.g., a *Python function*

• Meaning, e.g., the *meaning of a Python function*

• Concepts, e.g., *Composition* or *inheritance*
Entity types II / II

- Resources, e.g., Wikipedia pages
- Request, e.g., the invocation of a certain service, tool, or script
- Response, e.g., the response returned upon a request
- Specifications, e.g., Java Language Specification
- Protocol, e.g., HTTP
- Standards, e.g., DVI
- Organizations, e.g., IBM or Siemens
- People, e.g., „Joe Programmer“ (at Siemens)
- Roles, e.g., Developer, Tester, or Manager
(Software) language entities

• Definition:

  • *An artificial language used in software development*

• Examples

  • Programming languages: *Java, Python, Ruby, …*
  
  • Query languages: *XPath, SQL, XQuery, …*
  
  • Transformation languages: *XSLT, SQL, ATL, …*
  
  • Modeling languages: *UML, SDL, BPMN, …*
More types of software languages

• Hypertext languages (HTML)

• Markup languages (XML)

• Configuration languages

• Annotation languages

• Template languages

• …
(Software) technology entities

• Definition:
  
  • A tool (in a very general sense) used in software development

• Examples
  
  • APIs and libraries: JDOM, JQuery, Swing, Tkinter, Twitter, …
  
  • Frameworks: JPA, Hibernate, Spring, Django, …
  
  • IDEs: Visual Studio, Eclipse, NetBeans, …
  
  • Platforms: .NET, Android, J2EE, Java, JRE, …
  
  • Language processors: javac, python, gcc, …
More types of software technologies

- Server, e.g., Web server
- Web browser
- Plugins
- Office software
- Operating systems
- Package portals, package manager
- App stores
- ...

© 2014, Ralf Lämmel, rlaemmel@acm.org
(Software) system entities

• Definition:

  • A deployed, implemented, or designed software system

• Examples

  • An information system for a specific startup
  • A web application complementing said information system
  • A web service making said system available through an API
  • A smartphone app complementing said web application
(Software) artifact entities

• Definition:

  • A physical artifact as part of a software system

• Examples

  • Files: source code, byte code, markup, ...
  
  • Directories as shallow or deep collections of files
  
  • Packages as logical collections
(Software) fragment entities

• Definition:
  • A part of a software artifact

• Examples
  • Classes in a Python script collecting many classes
  • Methods in a class of a Python script
  • A CREATE TABLE statement in SQL/DDL script
  • A method call in a Python script
Source code as a nested container

```python
def search(l, x):
    return searchInRange(l, x, 0, len(l)-1)

def searchInRange(l, x, min, max):
    if min>max:
        return False
    else:
        middle = min+(max-min)/2
        if x > l[middle]:
            # Search in right half
            return searchInRange(l, x, middle+1, max)
        elif x < l[middle]:
            # Search in left half
            return searchInRange(l, x, min, middle-1)
        else:
            # Found in the middle
            return True
```
(Software) meaning entities

• Definition:
  • The meaning of an artifact, a technology, or a part thereof

• Examples
  • **Functions**
    • The semantics of a Python function (e.g., performing serialization)
    • The I/O behavior of a program (e.g., transforming XML content)
    • The I/O behavior of a tool as part of a technology (e.g., code generation)
  • **Actions**
    • The meaning of a specific method call in a program
    • The meaning of a database update
(Software) concept entities

• Definition:
  • A concept from the broad domain of software development

• Examples
  • A programming technique such as iteration or recursion
  • A modeling principle such as inheritance or composition
  • A design pattern such as Composite or Visitor
  • A classifier for software languages or technologies
  • A general capability of a software system, e.g., persistence
(Software) resource entities

• Definition:
  • A resource for any entity above in the sense of Linked Data

• Examples
  • A Wikipedia page for a software language
  • A portal for a software technology
  • A repository URI for a system
  • A fragment locator URI for a source code fragment
  • A LinkedIn URI for a developer
Relationships and types thereof
Relationship types 'part of'

- An artifact (a file) is part of a system.
- A fragment is part of an artifact.
- A language is part of another language.
- A technology is part of another technology.
- A concept is part of a concept.
Relationship types 'uses'

• A system (an artifact) uses a language.

• A system (an artifact) uses a technology.
Relationship types for languages and models thereof

- An artifact is an **element of** a language.
- A language is a **subset of** another language.
- An artifact **conforms to** an artifact.
- An artifact **defines** a language.
- An artifact **corresponds to** another artifact.
Model of technology usage for .NET’s xsd.exe
http://worker.101companies.org/MegaModels/implementations/xsdClasses/
Relationship types for classification

- A concept is a (specialization) of another concept.
- A concept is an instance of another concept.
- A language is an instance of a (classification) concept.
- A technology is an instance of a (classification) concept.
Relationship types for meanings & Co.

• An artifact (a fragment) **defines** a meaning.

• A technology **defines** a meaning.

• A technology **supports** a protocol.

• A technology **implements** a specification.

• An artifact (a fragment) **implements** a concept.
Relationship types for data flow & access

• An artifact is transformed into another artifact.

• A request yields a response.

• A language is the **domain of** a function.

• A language is the **co-domain of** a function.

• A function **maps** one artifact to another.

• An action **reads** an artifact.

• An action **writes** (as in „modifies“) an artifact.

As a shortcut, we may use the artifact right in place of its meaning.
Relationship types for persons

• A person **has developed** a system.

• A person **knows of** a language.

• A person **knows of** a technology.

• A person **knows of** a concept.
Benefits of technology modeling

- Raise the level of abstraction compared to ...
  - file system view
  - build management view
- Recognize all technologies and languages explicitly
- Understand concepts behind artifacts
- Understand relationships between artifacts
- Link model elements to artifacts or resources
Modeling Django

We use the Polls app as the running example.

Find the code here:

https://github.com/rlaemmel/mysite
Python Web frameworks

- Django
- web2py
- Flask
- Bottle
A low-level view on the Polls app

A database

The model

The view

Python code, HTML, and templates

Templates for views
Types of artifacts: How useful is that?

- db_sqlite3 : File
- mysite : Directory
  - __init__.py : File
  - manage.py : File
  - media : Directory
- polls : Directory
  - __init__.py : File
  - admin.py : File
  - models.py : File
  - tests.py : File
  - views.py : File
- settings.py : File
- templates : Directory
  - admin : Directory
  - polls : Directory
    - detail.html : File
    - index.html : File
    - results.html : File
- urls.py : File
• db_sqlite3 : File $\in$ SQLite3-IMAGE (a language we made up)
• mysite : Directory
  • __init__.py : File $\in$ Python
  • manage.py : File $\in$ Python
  • media : Directory
  • polls : Directory
    • __init__.py : File $\in$ Python
    • admin.py : File $\in$ Python
    • models.py : File $\in$ Python
    • tests.py : File $\in$ Python
    • views.py : File $\in$ Python
  • settings.py : File $\in$ Python
• templates : Directory
  • admin : Directory
  • polls : Directory
    • detail.html : File $\in$ HTML
    • index.html : File $\in$ HTML
    • results.html : File $\in$ HTML
• urls.py : File $\in$ Python

Languages of artifacts: How useful is that?

Is this HTML, proper?
Issues

• What is the schema underlying the database image?
• What are the roles of the different python scripts?
• How do code and database relate to each other?
• The HTML files are not plain HTML files, actually.
• What technologies are used by the app?
Issues

• What is the schema underlying the database image?

• What are the roles of the different python scripts?

• How do code and database relate to each other?

• The HTML files are not plain HTML files, actually.

• What technologies are used by the app?
In need of new entities: \textbf{request} and \textbf{response} for DB schema

- \texttt{schemaRequ} : Request \# for database schema
- \texttt{schemaResp} : Response \# for database schema
- \texttt{schemaRequ} $\in$ Bash \# a shell script
- \texttt{schemaResp} $\in$ SQL \# \texttt{CREATE TABLE} statements
schemaRequ : Request # for database schema

~ $ pwd
/home/rlaemmel/mysite
~ $ python manage.py sql polls

schemaResp : Response # for database schema

```
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;
```
Issues

• What is the schema underlying the database image?

• **What are the roles of the different python scripts?**

• How do code and database relate to each other?

• The HTML files are not plain HTML files, actually.

• What technologies are used by the app?
Concepts behind the many 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*
Concepts

- Model: the data / business logics part MVC
- View: the user interface part of MVC
- Router: a variation of a controller (part of MVC)
- Configuration: configuration of a component or a system
- Initialization: initialization of a component or a system
- Administration: administration of a system
- Testing: test of an artifact or a system
The *model*

```python
from django.db import models
import datetime

class Poll(models.Model):
    question = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')
    def __unicode__(self):
        return self.question
    def was_published_today(self):
        return self.pub_date.date() == datetime.date.today()
    was_published_today.short_description = 'Published today?'

class Choice(models.Model):
    poll = models.ForeignKey(Poll)
    choice = models.CharField(max_length=200)
    votes = models.IntegerField()
    def __unicode__(self):
        return self.choice
```

These are Python (Django) classes for the business data of thePolls app.
The view for „end users“

```python
def index(request):
    latest_poll_list = Poll.objects.all().order_by('-pub_date')[0:5]
    t = loader.get_template('polls/index.html')
    c = Context({'latest_poll_list': latest_poll_list,})
    return HttpResponse(t.render(c))

def detail(request, poll_id):
    p = get_object_or_404(Poll, pk=poll_id)
    return render_to_response('polls/detail.html', {'poll': p},
                              context_instance=RequestContext(request))

def results(request, poll_id):
    p = get_object_or_404(Poll, pk=poll_id)
    return render_to_response('polls/results.html', {'poll': p})
```

A typical view loads or saves data, and renders data as HTML via a template.
The view for „admins“ according to Django

```python
class ChoiceInline(admin.TabularInline):
    # Another more spacious option
    # class ChoiceInline(admin.StackedInline):
    model = Choice
    extra = 3

class PollAdmin(admin.ModelAdmin):
    fieldsets = [
        (None, {'fields': ['question']}),
        ('Date information', {'fields': ['pub_date'], 'classes': ['collapse']}),
    ]
    inlines = [ChoiceInline]
    list_display = ('question', 'pub_date', 'was_published_today')
    list_filter = ['pub_date']
    search_fields = ['question']
    date_hierarchy = 'pub_date

admin.site.register(Poll, PollAdmin)
```

These views are standardized by Django: they allow us to do basic data management for polls and choices.
The *router* (map URLs to views)

```python
from django.conf.urls.defaults import patterns, include, url
from django.contrib import import admin
admin.autodiscover()

urlpatterns = patterns('',
    url(r'polls/$', 'mysite.polls.views.index'),
    url(r'polls/(?P<poll_id>\d+)/$', 'mysite.polls.views.detail'),
    url(r'polls/(?P<poll_id>\d+)/results/$', 'mysite.polls.views.results'),
    url(r'polls/(?P<poll_id>\d+)/vote/$', 'mysite.polls.views.vote'),
    url(r'^admin/', include(admin.site.urls)),
)
```

Regular expression for URLs with parameters  
Python function for a particular view
Issues

• What is the schema underlying the database image?

• What are the roles of the different python scripts?

• How do code and database relate to each other?

• The HTML files are not plain HTML files, actually.

• What technologies are used by the app?
In need of new entities: request and response for DB synchronization

- `syncRequ` : Request # for database sync
- `syncResp` : Response # for database sync
- `syncRequ` ∈ Bash # a shell script
- `syncResp` ∈ Text # verbose log
syncRequ : Request # for database sync

~ $ pwd
/home/rlaemmel/mysite
~ $ python manage.py syncdb

syncResp : Response # for database sync

Creating tables ...
Creating table auth_permission
...
Creating table django_admin_log
Creating table polls_poll
Creating table polls_choice

You just installed Django's auth system, which means you don't have any superusers defined. Would you like to create one now? (yes/no): yes
Username (Leave blank to use 'rlaemmel'): rlaemmel
...
Superuser created successfully.
Installing custom SQL ...
Installing indexes ...
No fixtures found.
How do code and database relate to each other?

Relationships:
- syncRequ **reads** mysite/polls/models.py
- syncRequ **reads** db.sqlite3
- syncRequ **writes** db.sqlite3
- schemaResp **corresponds to** mysite/polls/models.py

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

### mysite/polls/models.py:
```
class Poll(models.Model):  
  ...  

class Choice(models.Model):  
  ...  
```
A hidden language for administration

• Remember
  • python manage.py sql polls
  • python manage.py syncdb

• There exist more such administrative commands.

• We designate a language **DjangoAdmin** ⊂ **Bash**.
Issues

• What is the schema underlying the database image?

• What are the roles of the different python scripts?

• How do code and database relate to each other?

• The HTML files are not plain HTML files, actually.

• What technologies are used by the app?
The template for the *index* view

```html
{% if latest_poll_list %}
  <ul>
    {% for poll in latest_poll_list %}
      <li><a href="/polls/{{ poll.id }}">{{ poll.question }}</a></li>
    {% endfor %}
  </ul>
{% else %}
  <p>No polls are available.</p>
{% endif %}
```
The template for the *detail* view

```html
<h1>{{ poll.question }}</h1>

{% if error_message %}<p><strong>{{ error_message }}</strong></p>{% endif %}

<form action="/polls/{{ poll.id }}/vote/" method="post">
{% csrf_token %}
{% for choice in poll.choice_set.all %}
    <input type="radio" name="choice" id="choice{{ forloop.counter }}" value="{{ choice.id }}" />
    <label for="choice{{ forloop.counter }}">{{ choice.choice }}</label>
{% endfor %}
<input type="submit" value="Vote" />
</form>
```
The template for the *results* view

```html
<h1>{{ poll.question }}</h1>

<ul>
{% for choice in poll.choice_set.all %}
  <li>{{ choice.choice }} -- {{ choice.votes }} vote{{ choice.votes|pluralize }}</li>
{% endfor %}
</ul>

<a href="/polls/{{ poll.id }}">Vote again?</a>
```
A language for templates

• We designate a language **DjangoTempl ⊃ HTML**.

• Extra constructs:
  • Python expressions {{ ... }} evaluating to HTML
  • Loops over Python data to return HTML
Issues

• What is the schema underlying the database image?
• What are the roles of the different python scripts?
• How do code and database relate to each other?
• The HTML files are not plain HTML files, actually.
• What technologies are used by the app?
Referenced python modules

- webapp **uses** django.db is **part of Django and implements database access**
- webapp **uses** django.test is **part of Django and implements testing**
- webapp **uses** django.template is **part of Django and implements DjangoTempl**
- webapp **uses** django.http is **part of Django and supports (the) HTTP (protocol)**
- webapp **uses** django....
- webapp **uses** datetime is **part of PythonRuntime**
- webapp **uses** os is **part of PythonRuntime**
Summary: some day we will have profound models of technologies as opposed to just white papers, wikipedia articles, and blog posts.