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.
Different kinds of software models

- **Data models** (e.g., to be implemented in a database)
- **Structural models** (e.g., to be implemented in software)
  - Class diagrams (to model state and relationships)
  - Package diagrams (to group classes)
- **Behavioral models** (e.g., to be implemented in software)
  - Sequence diagrams (to define specific „scenarios“)
  - Activity diagrams (to define general „workflows“)
  - State diagrams (to define object states and transitions)
  - Transformation models (in MDE and model transformation)
- **Technology models?**
Data model of an HRMS

<xs:schema ...

  <xs:element name="company"> … </xs:element>
  <xs:element name="department"> … </xs:element>
  <xs:complexType name="employee"> … </xs:complexType>
  <xs:element name="name"> … </xs:element>
  <xs:element name="address"> … </xs:element>
  <xs:element name="salary"> … </xs:element>

</xs:schema>
Structural model of an HRMS
Structural model of a *Polls* app
Behavioral model of a metro’s turnstile
Technology model for YACC-based parsers
Software models vs. technology models

• **Software** models
  - Structure and behavior of the software system

• **Technology** models
  - *Entities* related to technology usage
  - *Relationships* between those entities
Technology modeling

• Entities

• Entity types

• Relationships

• Relationship types
Technology modeling

- **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
in technology models
Entity types I / III

- 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
- Function, e.g., a mathematical function; a meaning
- Concepts, e.g., Composition or inheritance
Entity types II / III

- Resources, e.g., *Wikipedia pages*
- Request, e.g., the *invocation of a certain service or tool*
- Response, e.g., the *response returned upon a request*
- Function, e.g., a *function for a semantic meaning*
- Action, e.g., an implied *effect on some artifact*
Entity types III / III

• 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

Specific entities given as examples (as instances of entity types)
Let’s look at some major entity types in more detail — thereby encountering interesting subtypes.
(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 categories 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 categories of software technologies

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

• Definition:

  • A mathematical function arising as the meaning of some artifact

• Examples

  • The semantics of a Python function
  • The I/O behavior of a program
  • The I/O behavior of a tool as part of a technology
(Software) action entities

• Definition:
  • A side-effect arising as the meaning of some artifact

• Examples
  • 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 in technology models
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.
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
```
Relationship types 'uses'

- A system (an artifact) uses a language.
- A system (an artifact) uses a technology.
- A system (an artifact) uses a concept.
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/implemetations/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 function.

• An artifact (a fragment) defines an action.

• A technology defines a function.

• 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.
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
Django

as of 15 May (emphasis changed; elisions applied)

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

as of 15 May (emphasis changed; elisions applied)

Features; Components: The core Django MVC framework [...] consists of an object-relational mapper which mediates between data models (defined as Python classes) and a relational database ("Model"); a system for processing requests with a web templating system ("View") and a regular-expression-based URL dispatcher ("Controller").
Django
as of 15 May (emphasis changed; elisions applied)

Also included in the core framework are:

• A lightweight and standalone web server for development and testing.
• A form serialization and validation system which can translate between HTML forms and values suitable for storage in the database.
• A template system that utilizes the concept of inheritance borrowed from object-oriented programming
• A caching framework which can use any of several cache methods.
• [...]
A low-level view on the Polls app

A database

The model

The view

Python code, HTML, and templates

Templates for views
- 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 ∈ SQLITE3-IMAGE (a language we made up)
• mysite : Directory
  • __init__.py : File ∈ Python
  • manage.py : File ∈ Python
• media : Directory
• polls : Directory
  • __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 : Directory
  • admin : Directory
  • polls : Directory
    • detail.html : File ∈ HTML
    • index.html : File ∈ HTML
    • results.html : File ∈ HTML
• urls.py : File ∈ 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: request and response for DB schema

- schemaRequ : Request # for database schema
- schemaResp : Response # for database schema
- schemaRequ ∈ Bash # a shell script
- schemaResp ∈ SQL # 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
```

Don’t bother about details: these are Python (Django) classes for the business data of the Polls app.
The view for „end users“

```python
def index(request):
    latest_poll_list = Poll.objects.all().order_by('-pub_date')[: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})
```

Don’t bother about details, but a typical view loads or saves data, and renders data as HTML via a template.
The view for „admins“ according to Django

```
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

**mysite/polls/models.py:**
```python
class Poll(models.Model):
    ...
```
```
class Choice(models.Model):
    ...
```

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

{% 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><br />
{% 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 $\text{DjangoTempl} \supset \text{HTML}$.

• Extra constructs:
  
  • Python expressions $\{\{ \ldots \} \}$ 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 DjangoTemplate
• 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
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
The End