Architecture of a Service

Diagram:
- Database Interface
  - Model
    - Web Interface
      - DB
Architecture of a Service

- RESTful Web Service
- Model
- ORM
- DB
Architecture of a Service

- RESTful Web Service
- Model
- ORM
- DB
REST is a software structure mainly used to produce machine readable contents using the natural way the Internet works.

- HTTP protocol
- Hypermedia formats

HTTP commands POST, GET, PUT and DELETE are used to create, delete or update resources (Similar to CRUD in database systems).
Main REST concepts
- **Addressibility** – resources uniquely identified by URI
- **Statelessness** – resource is the same, regardless of the chain of navigation to get to it
- **Connectedness** – every resource should be linked to by another resource
- **Uniform interface** – same set of methods to operate on all resources
- **Data is represented as resources**
- **Resources are addressed with a URI**
- **Many MIME types such as XML, JSON and YAML are supported**

http://www.bla.com/users/johnsmith
REST advantages over RPC web services

- Resources can be expressed using hyperlinks (URI: Unique resource Identifiers)
- No need to keep track of sessions
- Reduced server workload and response time due to caching
- Allows users to bookmark resources (the query to access the resources)
Architecture of a Service

RESTful Web Service

Model

ORM

DB
Object Relational Mapping

- Store and load data from an RDBMS into an Object Oriented Data Model
- Object Oriented Database

- Application programmer no longer needs to solve the Object-Relational Impedance Mismatch
Save to Database
Session = sessionmaker()
session = Session()
	newgroup = Group()
	newgroup.group_name = 'Reactor Workers'

session.save(newgroup)

Load from Database
query = session.query(Group).filter(Group.group_name == 'Reactor Workers')
Isn't that nicer than writing SQL?
Mappings between Classes and Tables are defined by the application/database programmer.

- **XML**
- **Programmatically**

In general:
- Classes -> Tables
- Properties -> Fields
class User
{
    private String name;
    private String phone;
}

class Group
{
    private String name;
    private List<User> members;
}

CREATE TABLE User ("Name" char(256) PRIMARY KEY, "Phone" char(20));

CREATE TABLE Group ("Name" char(256) PRIMARY KEY);

CREATE TABLE User_Group ("userid", char(256) PRIMARY KEY references User(Name), "groupid" char(256) PRIMARY KEY references Group(Name));
define a table to hold Group instances

group_table = Table('groups', metadata,
    Column('id', Integer, primary_key=True),
    Column('group_name', String(16), unique=True, nullable=False),
    Column('created', DateTime, default=datetime.now)
)

#create the table

metadata.create_all()

#bind the two together

mapper(Group,group_table)
Similarities between REST and ORM

- Map classes of objects into addressable, flattened space
- Have two separate parts, mapper and retrieval
  - Hide complexity of getting data
- Used in same environment
Goals

- **Ideal Goal**
  - Automatically define REST API and relational database tables from data models, creating 'persistent web objects' in a single click/operation

- Client layer which exposes web API as a set of shared objects – the same set that make up the data model on the server
Goals

- A more realistic goal:
  - Define REST APIs in the same way as ORM tables, with as little effort as possible, leveraging similarities/redundancies wherever they exist.
What have we done?

- Create prototypes using various existing frameworks
  - ORM
    - SQLAlchemy
    - Hibernate
    - Django
  - REST
    - POPO
    - CherryPy
    - Django
What is next?

- Choose elements from competing Python REST frameworks, and attempt to integrate them into the Django Web Platform
- Continue investigation into useful features in this field
Questions?