

OPM/PL

Round-Trip Development with OPM and Prolog

Mayer Goldberg Guy Wiener

Ben-Gurion University

Feb. 3, 2010

Outline

- 1 Introduction
- 2 Problems
- 3 OPM
- 4 OPM/PL
- 5 Example
- 6 Summary

Outline

1 Introduction

2 Problems

3 OPM

4 OPM/PL

5 Example

6 Summary

Software Development Methodologies Today

Agile

- Rapid development
- Focus on code
- Testing
- Avoids “ceremony”
- Discards information

Model-Based

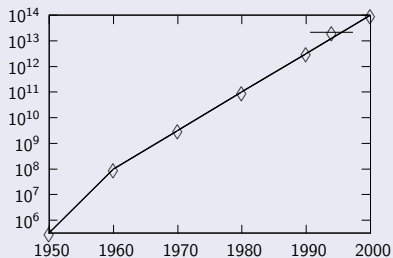
- Model before code
- Code generation
- Validation
- Requires “ceremony”
- Not always applicable

Real-Life Situations

Software Projects

- A lot of code
- Partial model
- Some tests
- Mental models

LOC in DoD Projects



Outline

- 1 Introduction
- 2 Problems**
- 3 OPM
- 4 OPM/PL
- 5 Example
- 6 Summary

Wordpress

- A web application for blogs and small web sites
- Runs on a standard platform
- 80K LOC
- Open-source
- Extensive manual
- Partial documentation
- Considered best-of-breed
- Moto: "CODE IS POETRY"

The screenshot shows the WordPress.org website. The header includes the WordPress logo and the text "WORDPRESS.ORG". Navigation links for "Home", "About", "Extend", "Docs", "Blog", "Forums", and "Hosting" are visible. The main content area features a post titled "WordPress Birthday Party" by Matt Fied under the "Events" category, dated May 25, 2008. The post text mentions a party in San Francisco and provides details for RSVPing. A sidebar on the right lists various categories such as "Development", "Documentation", "Events", "Focus", "General", "Hosting", "Meta", "Newsletter", "Releases", "Security", "Store", "Switchers", "Widgets", and "WordCamp".

Figure: A Wordpress blog

Code Mass

- The size of software projects increase **exponentially** over time
 - Test **add** mode code, even up to 2:1
 - Comments also grow in size
- ⇒ It is impossible to understand a software by reading its code

Code Riddle #1

Is this piece of PHP code related to Posts?

```
<h3 class="storytitle">
  <a href="<?php the_permalink() ?>">
    <?php the_title(); ?>
  </a></h3>
<div class="storycontent">
  <?php the_content('more...'); ?>
</div>
```

Don't know...

Code Riddle #1

Is this piece of PHP code related to Posts?

```
<div class="post" id="post-<?php the_ID(); ?>">
<h3 class="storytitle">
  <a href="<?php the_permalink() ?>">
    <?php the_title(); ?>
  </a></h3>
  <div class="storycontent">
    <?php the_content('more...'); ?>
  </div>
</div>
```

Maybe: It is in a div with a class post

Code Riddle #1

Is this piece of PHP code related to Posts?

```
<?php while (have_posts()) : the_post(); ?>
<div class="post" id="post-<?php the_ID(); ?>">
<h3 class="storytitle">
  <a href="<?php the_permalink() ?>">
    <?php the_title(); ?>
  </a></h3>
  <div class="storycontent">
    <?php the_content('more...'); ?>
  </div>
</div>
<?php endwhile; ?>
```

Yes! It comes after a call to the function `the_post`. But what does this function do?

Code Riddle #2

Q: Is this SQL code related to categories?

```
SELECT tr.object_id
FROM $wpdb->term_relationships AS tr
INNER JOIN $wpdb->term_taxonomy AS tt
ON tr.term_taxonomy_id = tt.term_taxonomy_id
WHERE tt.taxonomy IN ($taxonomies)
AND tt.term_id IN ($terms)
ORDER BY tr.object_id $order
```

Code Riddle #2

Q: Is this SQL code related to categories?

```
SELECT tr.object_id
FROM $wpdb->term_relationships AS tr
INNER JOIN $wpdb->term_taxonomy AS tt
...
```

A: Yes

- Wordpress does not have “TAG” or “CATEGORY” tables
- Instead, both are stored as **terms**. A term can belong to different **taxonomies** – Either categories or tags.
- Rationale: **Extensibility** (E.g, support tags for links)

Code Riddle #3

Q: What is the purpose of this table?

```
CREATE TABLE $wpdb->postmeta (  
    meta_id bigint(20) NOT NULL auto_increment,  
    post_id bigint(20) NOT NULL default '0',  
    meta_key varchar(255) default NULL,  
    meta_value longtext,  
    PRIMARY KEY (meta_id),  
    KEY post_id (post_id),  
    KEY meta_key (meta_key)  
)
```

Code Riddle #3

Q: What is the purpose of this table?

```
CREATE TABLE $wpdb->postmeta (  
    ...  
)
```

A: Plugins

- Wordpress is extensible by **Plugins**. There are over 2000 plugins.
- Plugins may attach simple data to a post by using this table
 - Map coordinates, rating, polls, footnotes. . .
- This fact is not documented in the table creation code

Models

A model represents the requirements and design of the software

- Abstract
- Concise
- platform-independent
- Verify-able

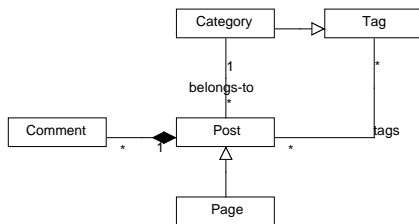


Figure: The WordPress Model

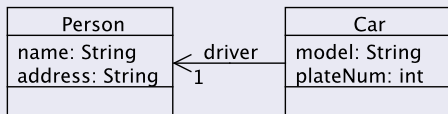
Lie in the Comments

Reedy Green, "How to Write Unmaintainable Code"

You dont have to actively lie, just fail to keep comments as up to date with the code.

Model Rot

Model



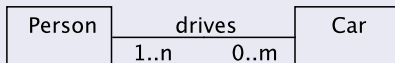
Code

```
class Person {
    String name;
    Address addr;
}

class Car {
    String model;
    Company maker;
    String plateNum;
    List<Person> drivers;
}
```

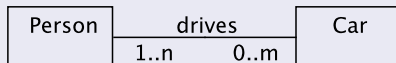
Abstraction Gaps

Bi-directional Association



Abstraction Gaps

Bi-directional Association

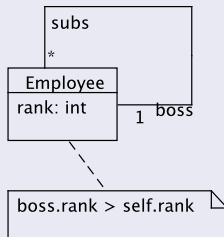


Possible Implementations

- A list in each class
- A list with pairs
- A pair of hash-tables
- A relational database table
- A relation in an objects database
- A file
- A directory with files
- A combination of the above

Distances in Code

A model



Employee.java

```
class Employee {  
    int rank  
    Employee boss;  
    List<Employee> subs;  
}
```

Some stored procedure

```
SELECT * FROM Employees  
WHERE rank > $my_rank
```

The code is not a 1:1
translation of the model

Not Everything is Object-Oriented

Q: Which of the following is **not** a Wordpress class?

① Snoopy

② Post

③ POP3

④ WP_Object_Cache

Not Everything is Object-Oriented

Q: Which of the following is **not** a Wordpress class?

① Snoopy

② **Post**

③ POP3

④ WP_Object_Cache

A: Post!

- The major concepts in Wordpress **are not represented as classes**, but as API functions
- Rationale:
 - Simplicity** Web designers are not programmers
 - Brevity** In-lined PHP code clutters the HTML page
 - Compatibility** The object system changed in PHP 5
- Most of WP classes are bridges to external services

What We Would Like to Have

- Summarize the code automatically
- Add information **incrementally** by need
- Link the code and the model
- Use models when possible (code generation, validation)
- Round-trip: Add generated code to the summary

Outline

- 1 Introduction
- 2 Problems
- 3 OPM**
- 4 OPM/PL
- 5 Example
- 6 Summary

OPM

Object-Process Methodology (Dori 2002)

Features

- Simple: ~ 20 basic declarations
- Single diagram type
- Dual representation: Textual and graphical
- Scaling operators

Language Elements

Entities

- Objects
- Processes
- States

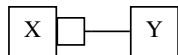
Relations

- Built-in
- Tagged

OPD

Object-Process Diagrams

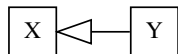
Entities and Relations



X exhibits Y



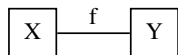
X consists Y



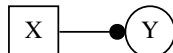
Y extends X



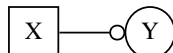
X instance Y



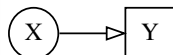
Tagged relation f



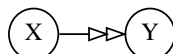
X handles Y



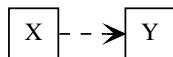
Y requires X



X yields Y



X invokes Y

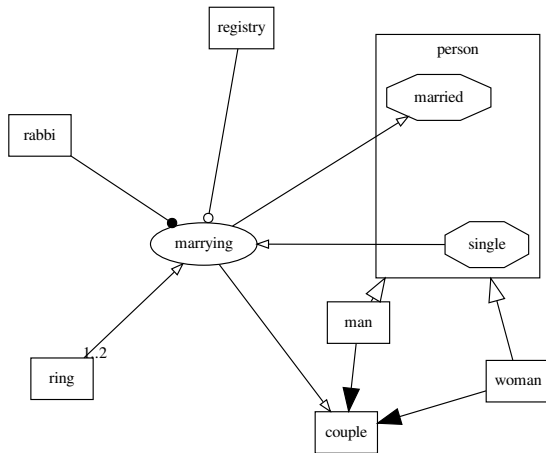


X implements Y

- Objects are boxes
- Processes are ellipses
- Relations are arrows

OPD

Example Diagram



example5

OPL

Object-Process Language

- A textual format for OPM
- Composed of sentences
- Each sentence is a declaration or a relation
- Declarations are required for entities and tagged relations
- Relations include a subject, the relation name, and arguments

OPL

Example Paragraph

```
object person states single, married.  
object man, woman, couple, registry, rabbi, ring.  
process marrying.  
man, woman extends person.  
couple consists man, woman.  
marrying changes person from single to married.  
marrying consumes ring(1,2).  
marrying yields couple.
```

Outline

- 1 Introduction
- 2 Problems
- 3 OPM
- 4 OPM/PL**
- 5 Example
- 6 Summary

OPM/PL

A lightweight, interactive environment for round-trip modeling.

- Based on the Object-Process Methodology
- Implemented in Prolog

Features

- Write models
- Query and visualize models
- Summarize code (through language libs)
- Link code summary to model
- Generate code (through language libs)

Writing Models

- OPL paragraphs are Prolog programs
- Declarations and relations are expanded to predicates

Prolog predicates for the example model

```
object(person).
object(man).
object(woman).
...
process(marrying).
state(person::single).
state(person::married).

extends(man, person).
extends(woman, person).
consists(couple, man).
consists(couple, woman).
consumes(marrying, ring).
produces(marrying, couple).
...
```

Queries

- Developers can query the model using Prolog.
- All the declarations and relations are predicates.
 - Aux. predicates include transitive relations.

Some sample queries

```
?- object(X).
```

```
X = person ;
```

```
X = man ;
```

```
...
```

```
?- extends(X, person).
```

```
X = man ;
```

```
X = woman
```

```
?- relation(R).
```

```
R = consists ;
```

```
R = extends ;
```

```
...
```

```
?- relation(R),
```

```
    call(R, couple, man).
```

```
R = consists.
```

Example Queries

Use cases

```
use_case(Proc) :-  
    process(Proc),  
    object(User),  
    handles(User, Proc),  
    \+ (process(Top),  
        consists(Top, Proc)).
```

Conversion functions

```
convert(Proc, From, To) :-  
    process(Proc),  
    consumes(Proc, From),  
    yields(Proc, To),  
    \+ requires(Proc, _),  
    \+ handles(_, Proc),  
    \+ (consumes(Proc, X), X \= From),  
    \+ (yields(Proc, Y), Y \= To).
```

Views

- **Views** are predicates that return a sub-set of facts from the model
- Graphical and textual libraries take a view as an argument and generate a diagram or a paragraph

Example view – A system diagram

```
sd(Term) :- ...
```

```
?- sd(X).
```

```
X = process(marrying) ;
```

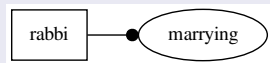
```
X = object(rabbi) ;
```

```
X = handles(rabbi, marrying);
```

```
process marrying.
```

```
object rabbi.
```

```
rabbi handles marrying.
```



Parsing and Generating Code

- Programming languages libs parse code into OPM model and generate code from models
- Current languages: Erlang, SQL (in progress)
- The **implements** relation links the code elements and their specification
- Details in example

Outline

- 1 Introduction
- 2 Problems
- 3 OPM
- 4 OPM/PL
- 5 Example**
- 6 Summary

A Development Task

- The software includes a server that handles login requests.
- Implemented in Erlang.
- Some operations are too slow.

Server Code

```
-module(login_srv).  
-export([start/0, login/2,  
        last_attempt/1, failed_attempts/1]).
```

```
-record(user,{username, password, name}).  
-record(login_ok,{username, time}).  
-record(login_fail,{username, password, time}).
```

```
start() -> ...
```

```
login(Username, Password) -> Users = get(users), Log = get(log),...
```

```
last_attempt(Username) -> Log = get(log),...
```

```
failed_attempts(Username) -> Log = get(log),...
```

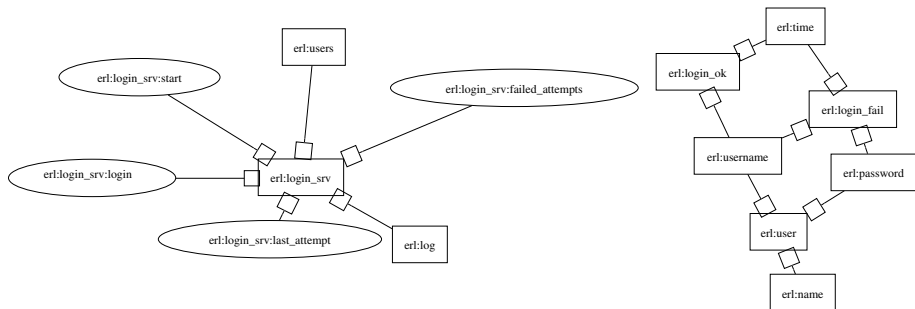
Code Summary

OPL

```
object erl:users, erl:username, erl:user, erl:time, erl:password,  
    erl:name,erl:login_srv, erl:login_ok, erl:login_fail, erl:log.  
process erl:login_srv:start, erl:login_srv:login,  
    erl:login_srv:last_attempt, erl:login_srv:failed_attempts.  
erl:login_fail exhibits erl:username, erl:time, erl:password.  
erl:login_ok exhibits erl:username, erl:time.  
erl:login_srv exhibits erl:users, erl:log,  
    erl:login_srv:start, erl:login_srv:login,  
    erl:login_srv:last_attempt, erl:login_srv:failed_attempts.  
erl:user exhibits erl:username, erl:password, erl:name.
```

Code Summary

OPD



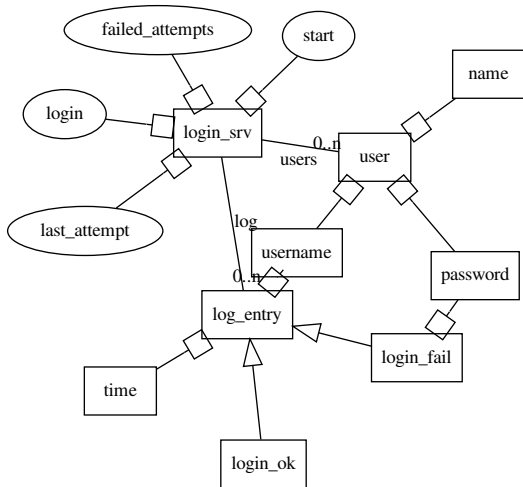
Enhanced Model

OPL

```
object username, user, time, password, name,  
  login_srv, login_ok, login_fail, log_entry.  
process start, login, last_attempt, failed_attempts.  
log_entry exhibits username, time.  
login_fail, login_ok extends log_entry.  
login_fail exhibits password.  
login_srv exhibits start, login,  
  last_attempt, failed_attempts.  
user exhibits username, password, name.  
relation users.  
relation log.  
login_srv users user(0,n).  
login_srv log log_entry(0,n).
```

Enhanced Model

OPD

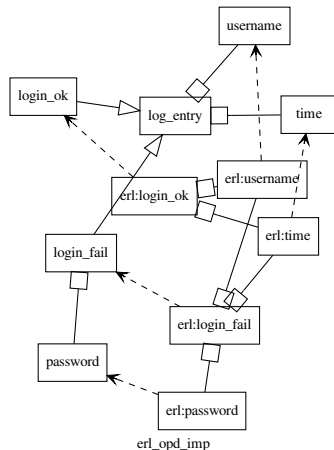


Specification and Implementation

```
implements(erl:X, Y) :- ...
```

```
view_imp(Term) :- ...
```

- ① The following objects:
log_entry, login_fail
and login_ok.
- ② All attributes of these
objects.
- ③ All implementation the
above objects.
- ④ All possible relations
between these objects.



Synchronizing the Model and Code

The **implements** relation can detect where the model and the code are not synchronized.

`\+implements(_, thing)` “thing” has no implementation

`\+implements(thing, _)` “thing” has no specification

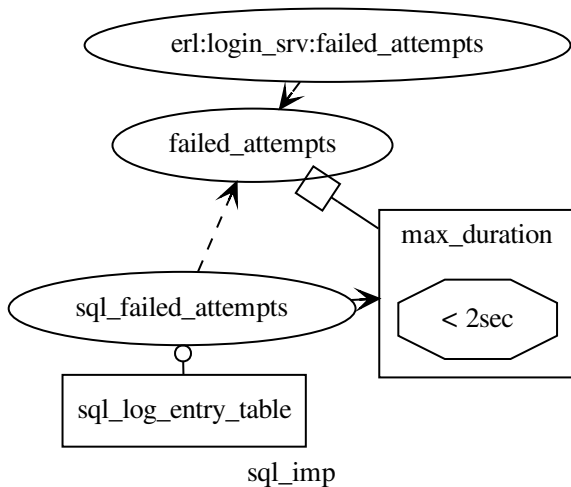
Code Generation

- Erlang lists are inefficient
- Generate SQL code instead, based on the model.

Generated SQL code for creating one table

```
CREATE TABLE log_entry (  
  col_log_entry_id int PRIMARY KEY,  
  col_log_entry_exhibits_username VARCHAR(30),  
  col_log_entry_exhibits_time VARCHAR(30),  
  col_log_entry_exhibits_login_ok int  
    REFERENCES login_ok(col_login_ok_id),  
  col_log_entry_exhibits_login_fail int  
    REFERENCES login_fail(col_login_fail_id));
```

Final Model



Outline

- 1 Introduction
- 2 Problems
- 3 OPM
- 4 OPM/PL
- 5 Example
- 6 Summary**

Future Work

- Examine OPM/PL in a real-life project
- Code sythesis

THANK YOU!