

State Machines to State of the Art  
*Smart, Efficient Design using REST and MVC*

Rowan Merewood, Lead Developer, Plusnet

# Motivation

Trying to be a better Software Engineer

# The Tools

HTTP, REST, CRUD, MVC,  
State Machines

# HTTP Protocol

A stateless way of interacting with resources

# HTTP Protocol

HEAD GET POST PUT DELETE  
TRACE OPTIONS CONNECT

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9>

# HTTP Protocol

HEAD GET POST PUT DELETE  
TRACE OPTIONS CONNECT

# HTTP Protocol

Safe



HEAD GET POST PUT DELETE  
TRACE OPTIONS CONNECT

GET and HEAD methods SHOULD NOT have the significance of taking an action other than retrieval. These methods ought to be considered "safe".

# HTTP Protocol

Idempotent



Methods can also have the property of "idempotence" in that (aside from error or expiration issues) the side-effects of  $N > 0$  identical requests is the same as for a single request

# What's Important?

User Experience  
*(of course!)*

# What's Important?

Safe == Read Only  
Idempotent == Predictable

# What went wrong?

Great Power → Great Responsibility?

# What was abused?

HTTP, Sessions, and Cookies – oh my!

# Safe?

<http://www.shop.com/buy.php?item=AstonMartin>

# Stateless?

I'll just press "Back" then...

# REST

Representational State Transfer

# REST

Nouns not verbs

# REST

`/getOrder?id=123` vs. `/order/123`  
`/addContact` vs. `/contacts/`  
`/updateArticle?id=abc` vs. `/article/abc`

# REST

REST is ideal for implementing over HTTP

# Design

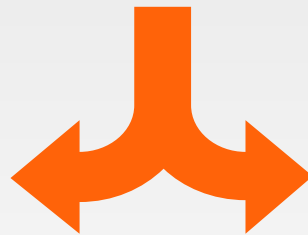
Enough front-end stuff,  
what's going to do the work?

# The Problem

Booking train tickets

# Analysis

What's my resource?  
Tickets

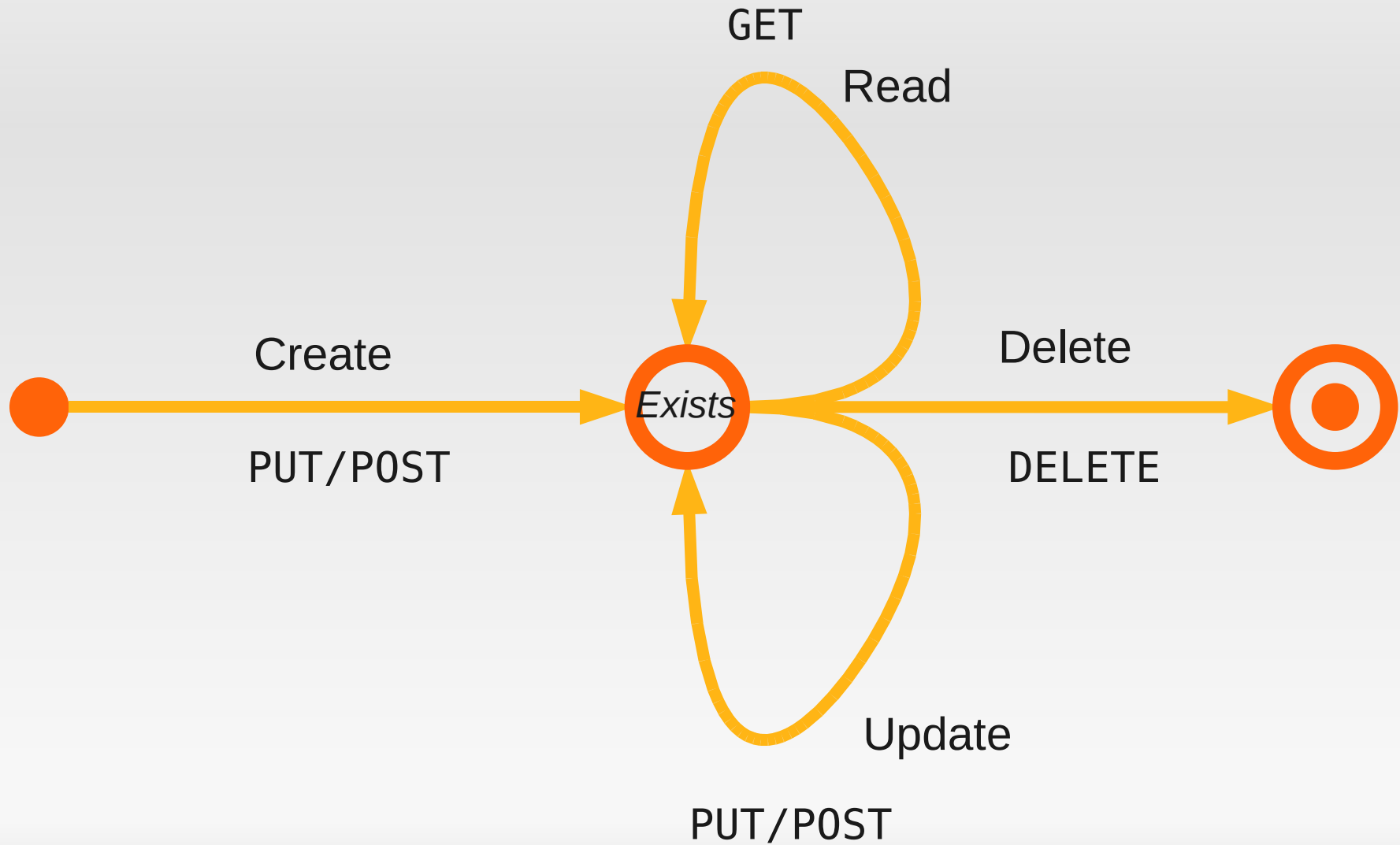


Origin, Destination, Departs, Class, Price

# CRUD

Create, Read, Update, Delete

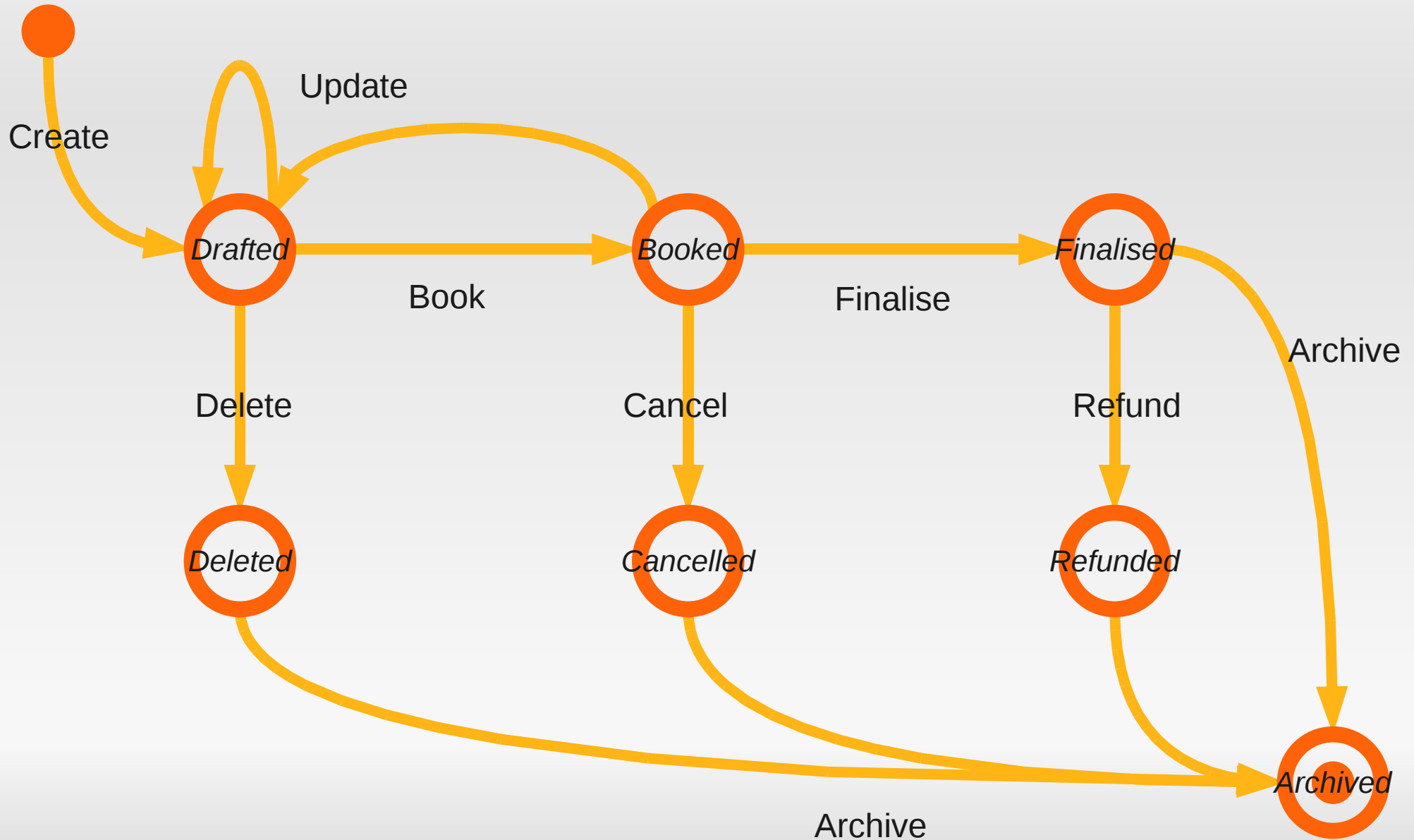
# CRUD



# Analysis

Create, Read, Update, Delete,  
Book, Cancel, Finalise, Refund, Archive

# Analysis



# Code

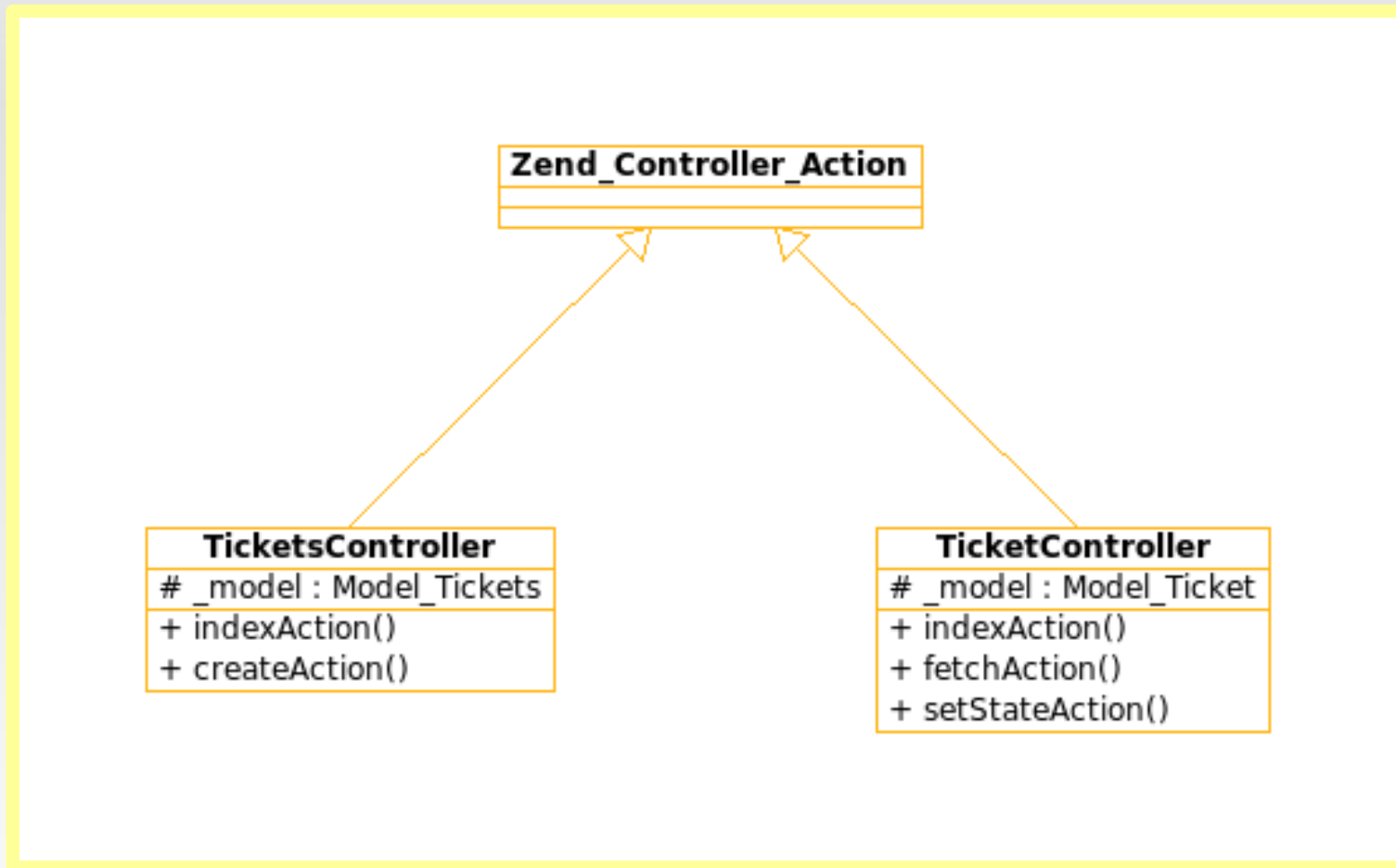
Rowan On Rails

*(I am so, so sorry!)*

# URLs

- `/tickets`
- GET: list tickets
- POST: create ticket
  
- `/ticket`
- GET: display ticket form
  
- `/ticket/123`
- GET: display ticket
  
- `/ticket/123/booked`
- POST: state transition

# Controllers



# Routing

```
$router = $frontController->getRouter();

$route = new Zend_Controller_Router_Route(
    'ticket/:ticketId',
    array(
        'controller' => 'ticket',
        'action'      => 'fetch',
    )
);
$router->addRoute('ticketFetch', $route);

$route = new Zend_Controller_Router_Route(
    'ticket/:ticketId/:stateHandle',
    array(
        'controller' => 'ticket',
        'action'      => 'changestate',
    )
);
$router->addRoute('ticketChangestate', $route);

$frontController->setRouter($router);
```

# Actions

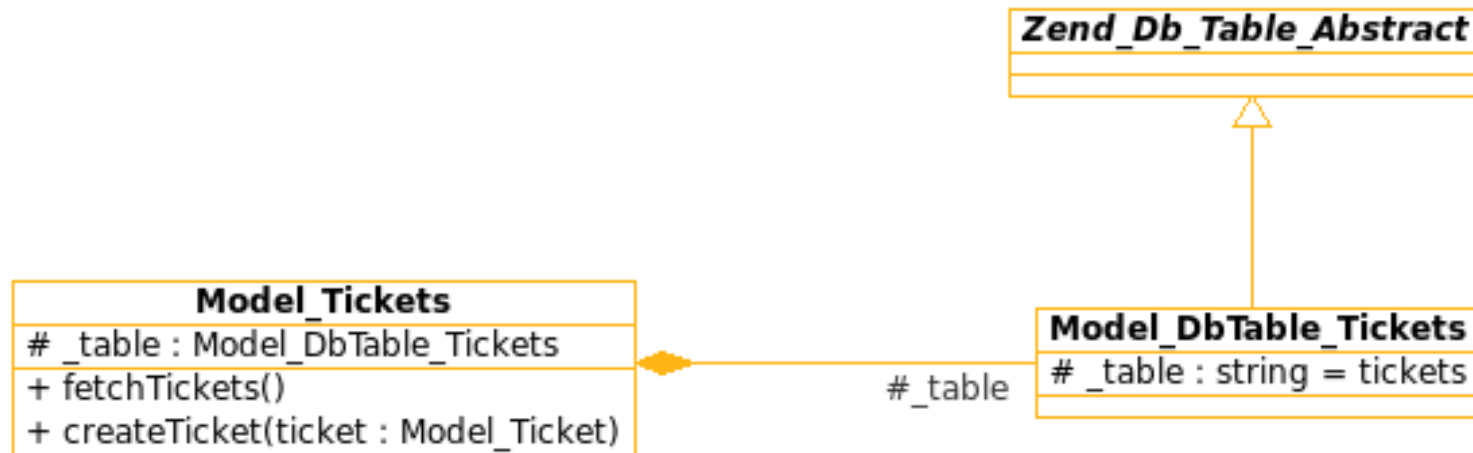
```
public function fetchAction()
{
    $model = $this->_getModel();
    $ticket = $model->fetchTicket($this->_getParam('ticketId'));
    $this->view->ticket = $ticket;
}

public function changestateAction()
{
    $model = $this->_getModel();
    $ticket = $model->fetchTicket($this->_getParam('ticketId'));

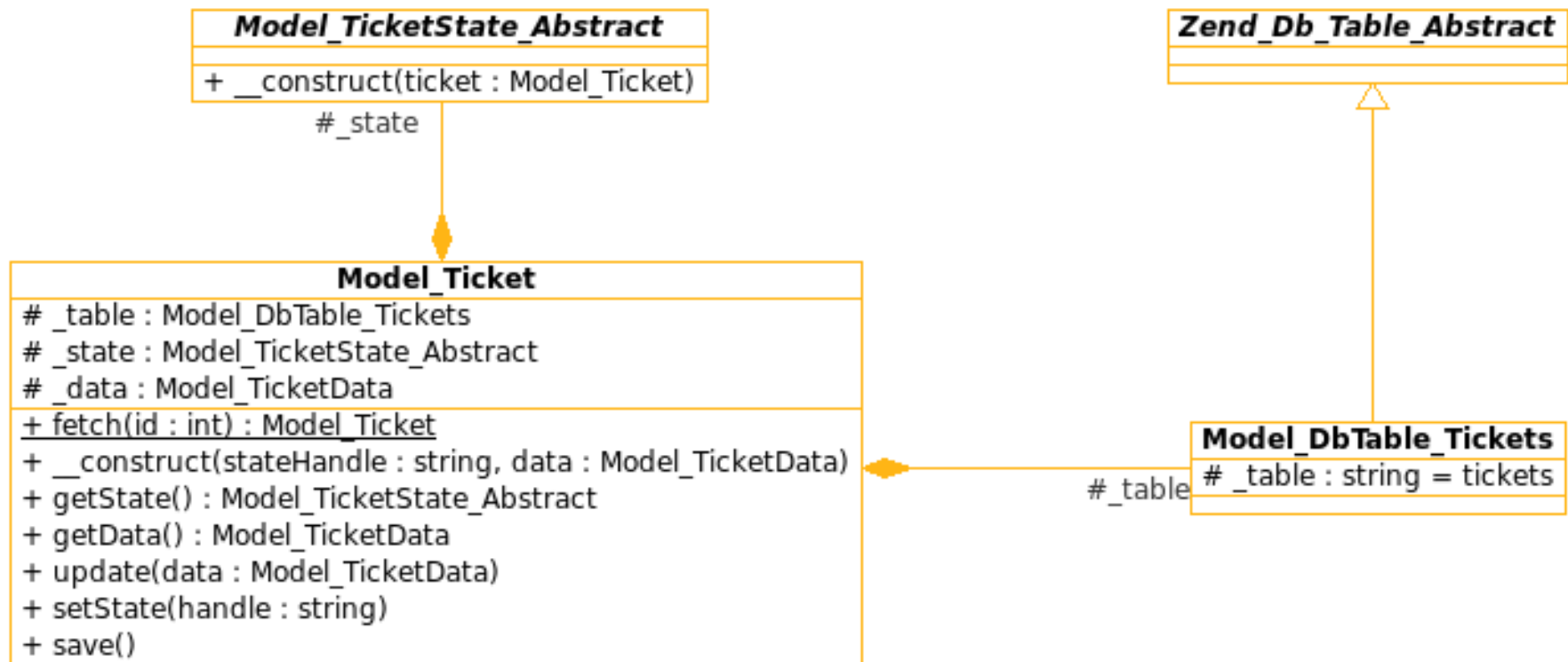
    if ($this->getRequest()->isPost()) {
        $model->setState($this->_getParam('stateHandle'));
        $model->save();

        $redirector = $this->_helper->getHelper('Redirector');
        $redirector->gotoRoute(
            array('ticketId' => $this->_getParam('ticketId')),
            'ticketFetch');
    }
}
```

# Model



# Model

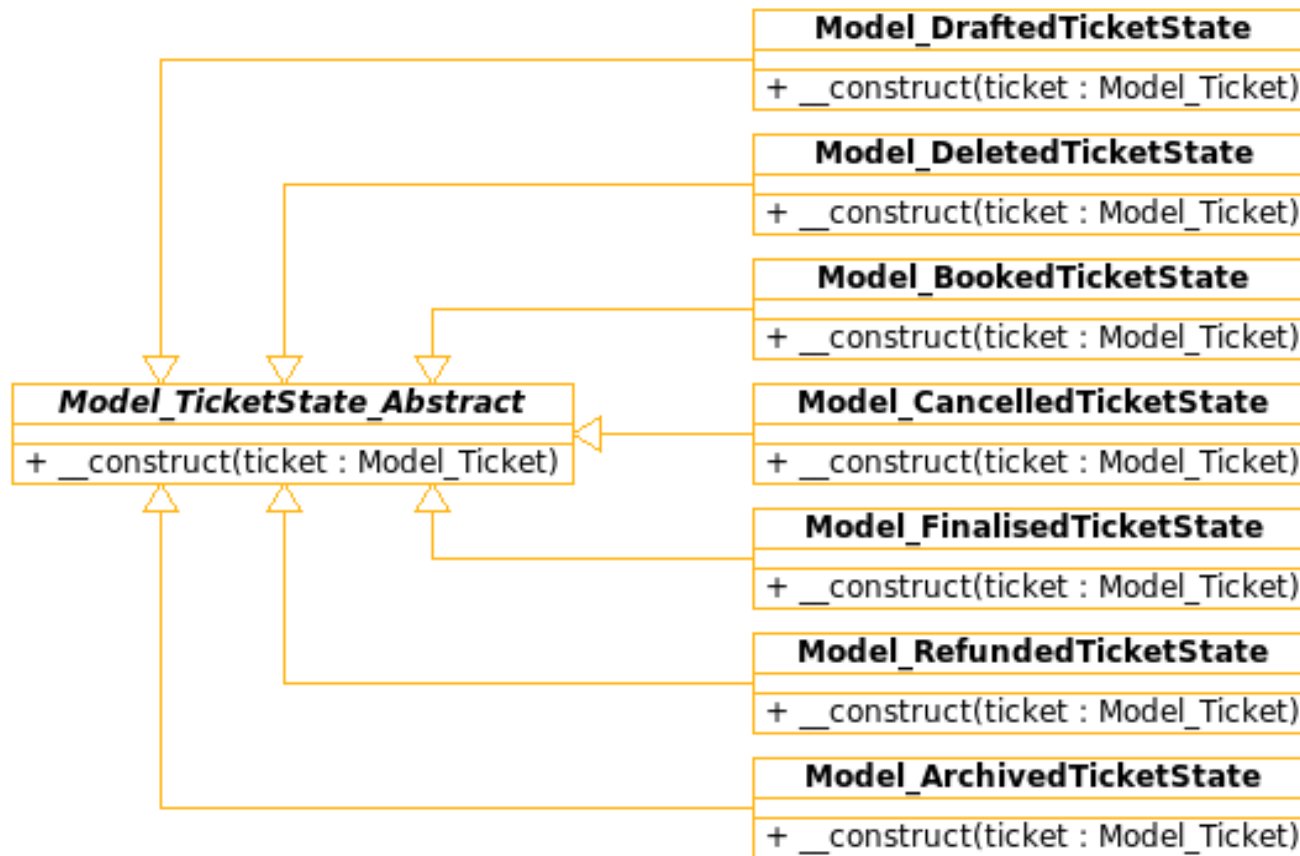


# Model

```
public function update(Model_TicketData $data)
{
    $this->setState('Draft');
    $this->_data = $data;
}
```

```
public function setState($handle)
{
    $classname = 'Model_' . $handle . 'TicketState';
    $state = new $classname($this);
    $this->_state = $state;
}
```

# States



# States

```
class Model_DraftedTicketState extends Model_TicketState_Abstract
{
    public function __construct(Model_Ticket $ticket)
    {
        $curState = $ticket->getState();

        if (!$this->isValidPrevState($curState)) {
            throw new Model_InvalidPrevStateException(
                'Tried to move to "Drafted" state from '.$curState);
        }
    }

    private function isValidPrevState(Model_TicketState_Abstract $state)
    {
        if (
            is_null($state) ||
            $state instanceof Model_DraftedTicketState ||
            $state instanceof Model_BookedTicketState
        ) {
            return true;
        }

        return false;
    }
}
```

# Benefits

Business logic solely in  
Model\_\*TicketState classes

# Benefits

Easy to add new states  
No change to MVC classes

# Benefits

Decoupled

I'm tempted to go hook this up to Zend\_Rest

# Cons

Lots of classes

# Cons

Method-calling overhead

# Cons

Potential to over-engineer!

# Questions

And possibly answers...