STATE IN THE STATELESS WORLD

HELLO

Luka Muzinic
@lmuzinic

Working in a remote team of three software engineers, able to offer outsourcing and consulting services, leadership of development teams and code reviews. Managing everything from application architecture to infrastructure. Delivering projects that are documented, covered with tests, with automated provisioning of local development virtual machines and production servers.

WHY WE NEED STATE?

THE PROBLEM

STATUS OF ENTITIES

Our entities can have life of their own, they start out one way and then after series of events they end up different. We want to store that information.

EASY PEASY

JUST ADD A PROPERTY

```php
/**
 * @var string
 */
private $status;

/**
 * @return string
 */
public function getStatus()
{
    return $this->status;
}

/**
 * @param string $status
 */
public function setStatus($status)
{
    $this->status = $status;
}
```

EASY PEASY

JUST CHANGE A PROPERTY AND SAVE IT

```php
$em = $this->getDoctrine()->getManager();
$rep = $em->getRepository(Entity::class);

$entity = $rep->find(1);

$entity->setStatus('paid');

$entityManager->flush();
```

BUT THERE ARE ALSO SOME BUSINESS RULES

"I should be able to publish an article only after lector says it is OK"

"Warehouse should not ship customer orders until they have been fully paid"

"Support can promote users if they have verified their email address"

HOW TO ADD BUSINESS RULES?

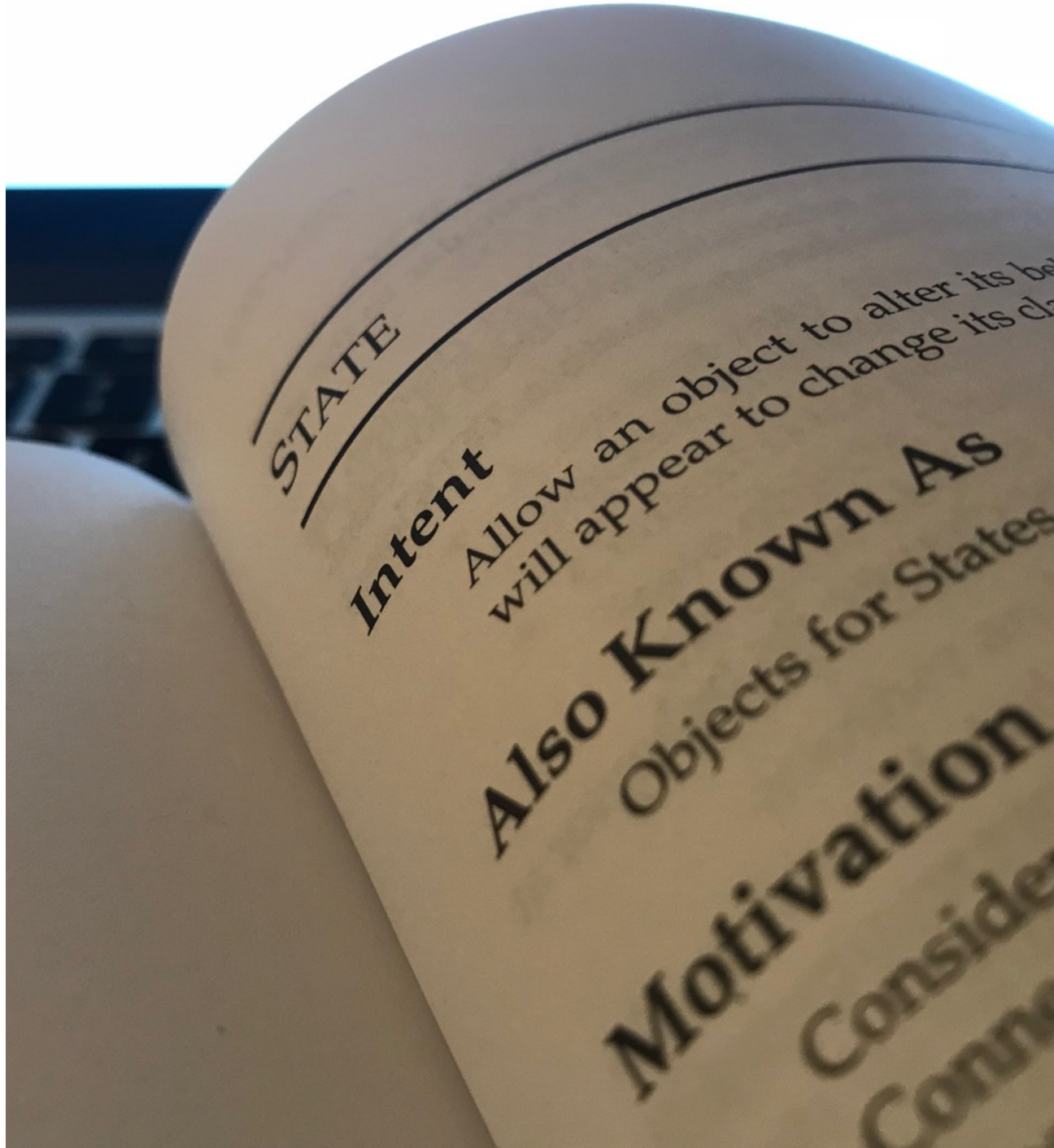
There must be a clever way to include business rules in our system.

STATE PATTERN

INTENT

Allow an object to alter its behaviour when its internal state changes. The object will appear to change its class.

http://wiki.c2.com/?StatePattern



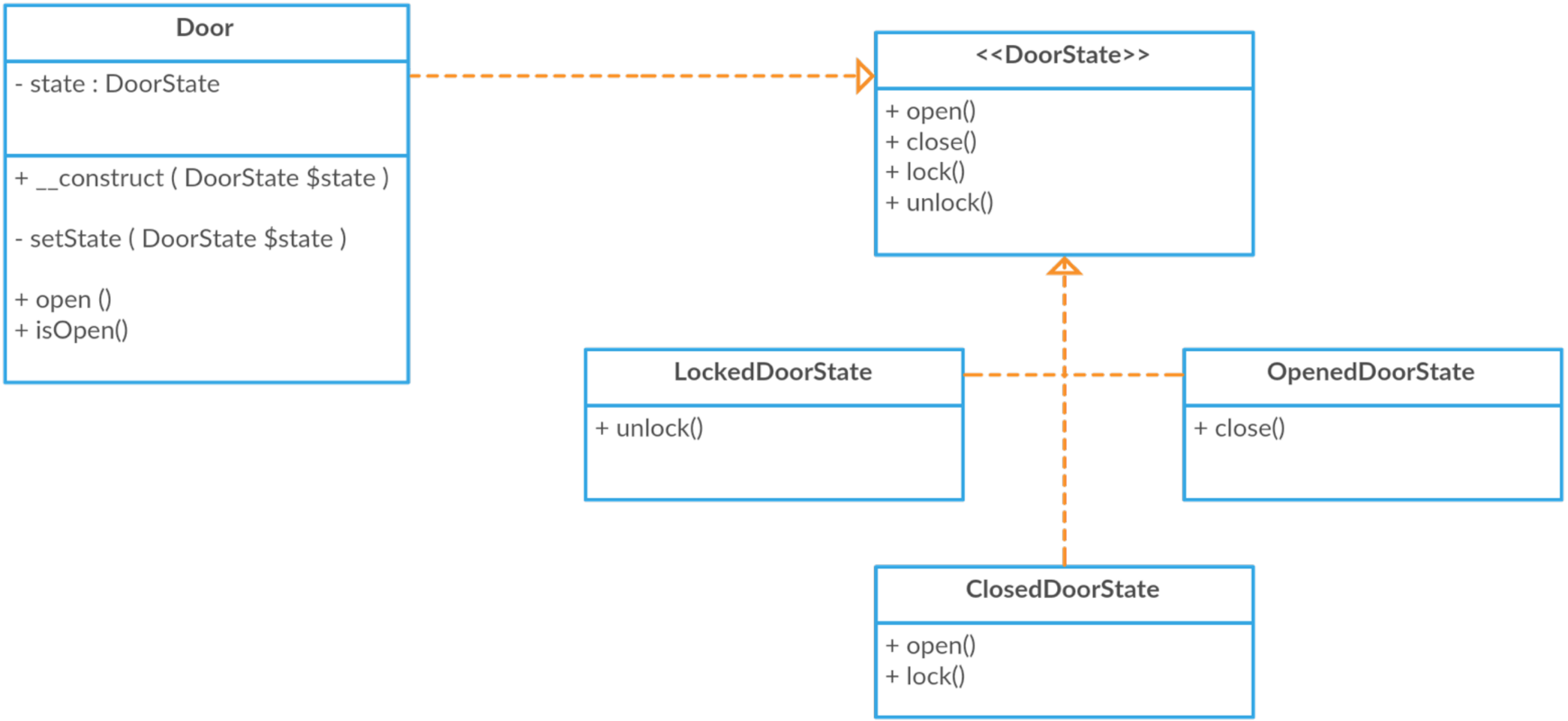
STATE PATTERN

USAGE

The State pattern can be used, for instance, to implement a Finite State Machine efficiently and elegantly. This approach can be useful when implementing business processes or workflows.

https://github.com/sebastianbergmann/state

STATE PATTERN



STATE PATTERN

```
interface DoorState
{
    public function open();

    public function close();

    public function lock();

    public function unlock();
}
```

Start off with a simple interface listing all possible transitions

STATE PATTERN

```
abstract class AbstractDoorState implements DoorState
{
    public function open()
    {
        throw new IllegalStateTransitionException;
    }

    public function close()
    {
        throw new IllegalStateTransitionException;
    }

    public function lock()
    {
        throw new IllegalStateTransitionException;
    }

    public function unlock()
    {
        throw new IllegalStateTransitionException;
    }
}
```

Make all transitions throw exceptions by default

STATE PATTERN

```
class LockedDoorState extends AbstractDoorState
{
    public function unlock()
    {
        return new ClosedDoorState;
    }
}

class ClosedDoorState extends AbstractDoorState
{
    public function open()
    {
        return new OpenDoorState;
    }

    public function lock()
    {
        return new LockedDoorState;
    }
}
```

Define states as factories to other states

STATE PATTERN

```php
class Door
{
    private $state;

    public function __construct(DoorState $state)
    {
        $this->setState($state);
    }

    public function open()
    {
        $this->setState($this->state->open());
    }

    public function isOpen()
    {
        return $this->state instanceof OpenDoorState;
    }

     ...

    private function setState(DoorState $state)
    {
        $this->state = $state;
    }
}
```

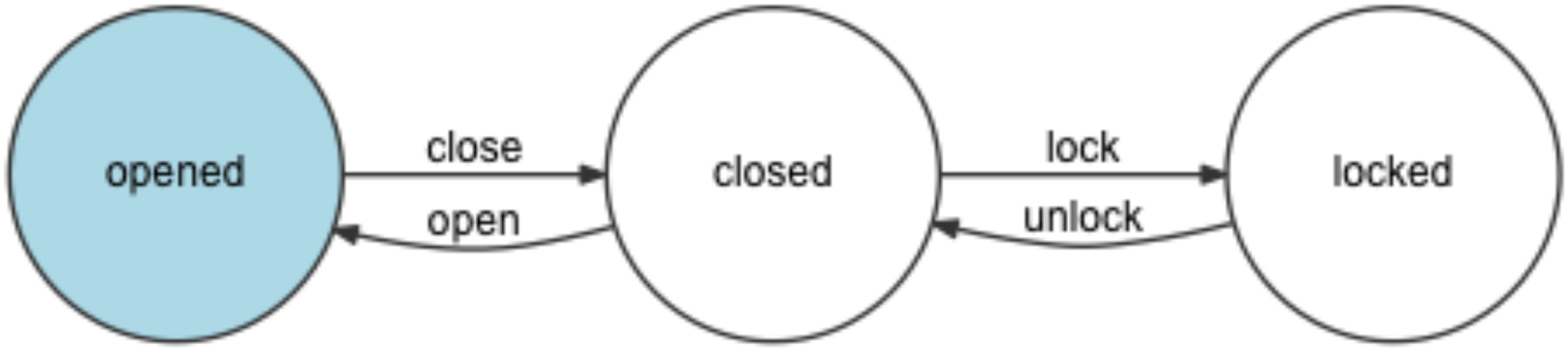
Add state to your entity

BUT MY ENTITIES ARE ALREADY BIG!

Shoving all that inside your entity makes them fat.

IS THERE A BETTER WAY?

STATE MACHINES



Defining states and allowed transitions between them

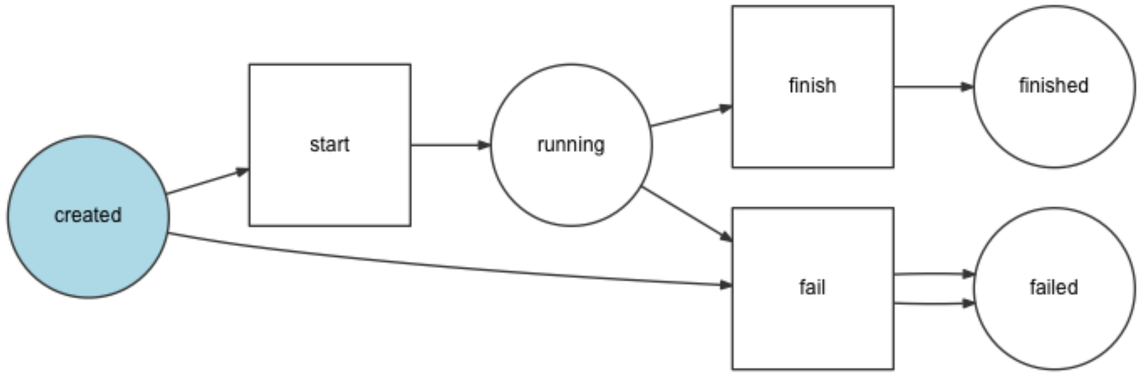
STATE MACHINES PACKAGES

```
composer require symfony/workflow

composer require winzou/state-machine

composer require yohang/finite
```

Pick one

SYMFONY WORKFLOW CONFIG

```yaml
framework:
    workflows:
        workload:
            type: 'state_machine'
            marking_store:
                type: 'single_state'
                arguments:
                    - status
            supports:
                - AppBundle\Entity\Workload
            places:
                - created
                - running
                - finished
                - failed
            transitions:
                start:
                    from: created
                    to: running
                finish:
                    from: running
                    to: finished
                fail:
                    from: [created, running]
                    to: failed
```



SYMFONY WORKFLOW USAGE

```
$workload = new AppBundle\Entity\Workload();

$stateMachine = $this->container->get('state_machine.workload');

if ($stateMachine->can($workload, 'start')) {
    $stateMachine->apply($workload, 'start');
}
```

Focus on what needs to be done, not if it can be done!

SYMFONY WORKFLOW EVENTS

```php
class Workflow
{
    public function __construct(
        Definition $definition,
        MarkingStoreInterface $markingStore = null,
        EventDispatcherInterface $dispatcher = null,
        $name = 'unnamed'
    )

...
```

Cool, EventDispatcher is there!

SYMFONY WORKFLOW EVENTS

```php
$stateMachine->can($workload, 'start');
```

GUARD

workflow.guard
workflow.[workflow name].guard
workflow.[workflow name].guard.[transition name]

SYMFONY WORKFLOW EVENTS

```php
class WorkloadSubscriber implements EventSubscriberInterface
{
    public function guardStart(GuardEvent $guardEvent)
    {
        ...

        $guardEvent->setBlocked(true);
    }

    public static function getSubscribedEvents()
    {
        return [
            'workflow.workload.guard.start' => ['guardStart']
        ];
    }
}
```

Block it if necessary!

SYMFONY WORKFLOW EVENTS

```
$stateMachine->apply($workload, 'start');
```

LEAVE

workflow.leave
workflow.[workflow name].leave
workflow.[workflow name].leave.[transition name]

TRANSITION

workflow.transition
workflow.[workflow name].transition
workflow.[workflow name].transition.[transition name]

ENTER

workflow.enter
workflow.[workflow name].enter
workflow.[workflow name].enter.[transition name]

ENTERED

workflow.entered
workflow.[workflow name].entered
workflow.[workflow name].entered.[transition name]

COMPLETED

workflow.completed
workflow.[workflow name].completed
workflow.[workflow name].completed.[transition name]

ANNOUNCE

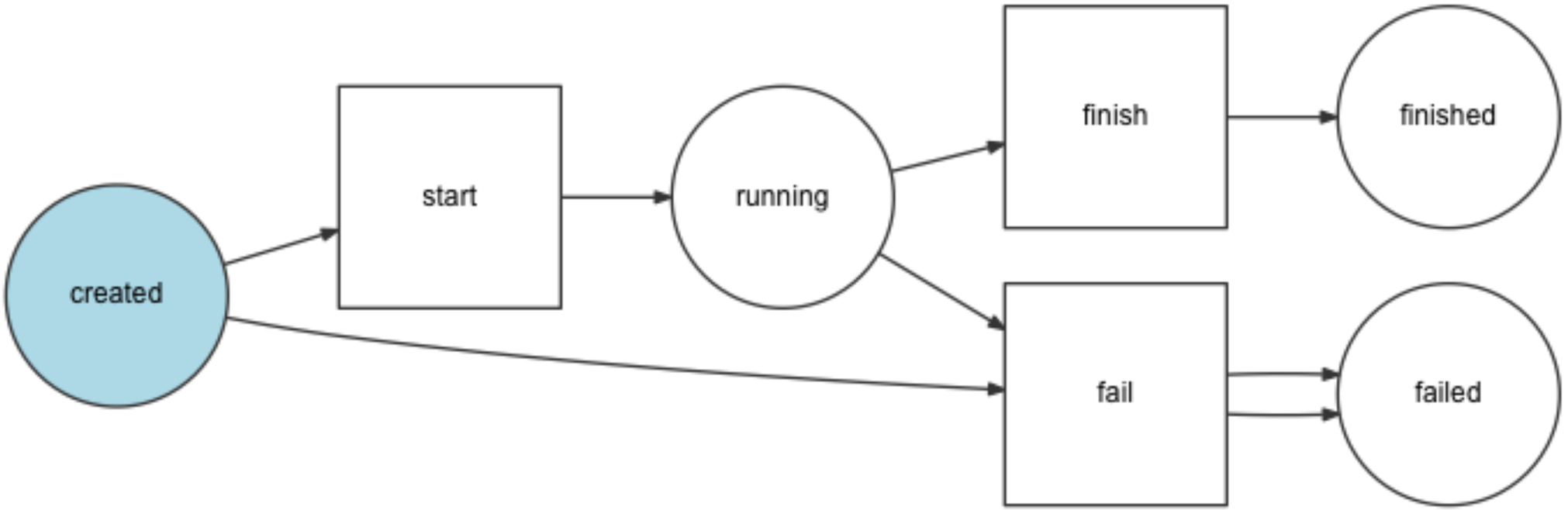
workflow.announce
workflow.[workflow name].announce
workflow.[workflow name].announce.[transition name]

SYMFONY WORKFLOW IN TEMPLATES

lmuzinic/web-luka.muzinic.net	2d44505310	1	lmuzinic	2017-04-03 20:13:12	2017-04-03 20:13:14	details
lmuzinic/web-luka.muzinic.net	66823fe034	1	lmuzinic	2017-04-03 20:21:33	2017-04-03 20:21:35	details
lmuzinic/web-luka.muzinic.net	95c2d9eb3e	2	lmuzinic	2017-04-25 11:43:14		details stop

```
{% if workflow_can(workload, 'fail') %}
    <a class="btn btn-small btn-danger" href="#">stop</a>
{% endif %}
```

SYMFONY WORKFLOW DUMP




```
$ bin/console workflow:dump workload | dot -Tpng -o graph.png
```

SYMFONY WORKFLOW TEST COVERAGE

```php
public function setUp()
{
    self::bootKernel();

    $this->container = self::$kernel->getContainer();
    $this->workload = new Workload();
    $this->workload->setStatus('running');
    $this->stateMachine = $this->container->get('state_machine.workload');
}
```

SYMFONY WORKFLOW TEST COVERAGE

```php
public function testRunningWorkloadCanNotStart()
{
    $result = $this->stateMachine->can($this->workload, 'start');
    $this->assertFalse($result);
}

public function testRunningWorkloadCanFinish()
{
    $result = $this->stateMachine->can($this->workload, 'fail');
    $this->assertTrue($result);
}

public function testApplyingStartOnRunningWorkloadThrowsException()
{
    $this->setExpectedException('Symfony\Component\Workflow\Exception\LogicException');
    $this->stateMachine->apply($this->workload, 'start');
}

public function testApplyingFailOnRunningWorkload()
{
    $this->stateMachine->apply($this->workload, 'fail');
    $this->assertEquals('failed', $this->workload->getStatus());
}
```

SYMFONY WORKFLOW TEST COVERAGE

```
bash-4.3# vendor/bin/phpunit --testdox

s\AppBundle\Workload\Running
 [x] Running workload can not start
 [x] Running workload can finish
 [x] Running workload can fail
 [x] Applying start on running workload throws exception
 [x] Applying fail on running workload
```

POSSIBLE USAGES

BINARY

enable/disable
open/closed

PUBLISHING

articles (draft, approved, published, archived)

PAYMENTS

subscriptions (active, pending renewal, expired)
order (ordered, shipped, canceled, returned, refunded)

GAMES

character levels (peasant, beginner, warrior, lord)
action state (running, jumping)

USAGES IN THE WILD

sylius/sylius using **winzou/state-machine**

So far the list is short

TIPS AND TRICKS

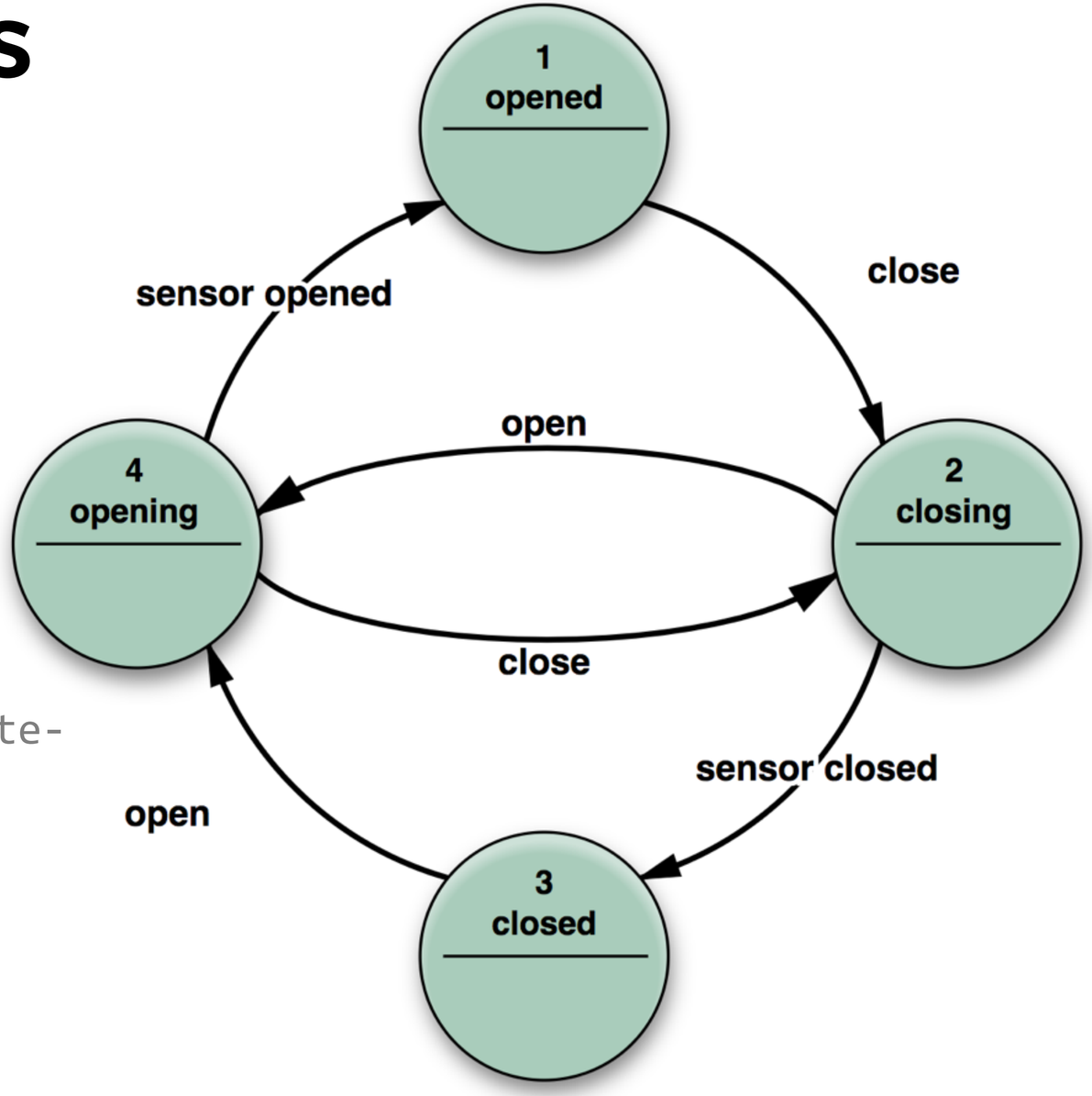
“Discuss it with your team/client”

“Treat them same as serialising”

“Leave a paper trail”

FINITE STATE MACHINES

```
# JS
npm install javascript-state-machine

# Python
pip install transitions

# Ruby
gem install state_machine

# PostgreSQL
http://felixge.de/2017/07/27/implementing-state-
machines-in-postgresql.html
```



WHAT DID WE LEARN?

RECAP

State machines are cool

STORE STATE SOMEWHERE

Models are good for storing state and bad for business logic how to transition between states.

Group that logic in one place, if possible. Preferably through state machines.

STATE PATTERN

The object will appear to change its class.

Look at sebastianbergmann/state as an excellent example of state pattern.

STATE MACHINES

Do not reinvent the wheel, use available packages if possible.

Test them as every other piece of code.

Enforce their usage throughout project.

QUESTIONS?

Luka Muzinic
@lmuzinic

luka.muzinic.net/talks

KTHXBAI