Decoupling with (Domain-)Events

SymfonyCon Warsaw

Benjamin Eberlei (@beberlei)

December 2013

passion for software quality

Me

- Working at Qafoo

passion for software quality

We offer trainings and consulting to help our customers create high quality code.

`http://qafoo.com`

- Doctrine Developer
- Symfony Contributor
- Twitter `@beberlei` and `@qafoo`

Outline

Motivation

Example

Events

Technical Implications

Domain Events

Empirical Observation

Just using Symfony and Doctrine does not prevent us from creating high coupling and legacy code.

Our Mistakes

Thinking in technical use-cases

.. leads to huge controllers

Our Mistakes

One model to rule them all

.. leads to monolithic object-graphs

Our Mistakes

No abstraction of business rules

.. prevents reusability of code

passion for software quality

What would we prefer?

- Small controllers
- Reusable business logic
- Recombinable business logic
- Decoupled bundles

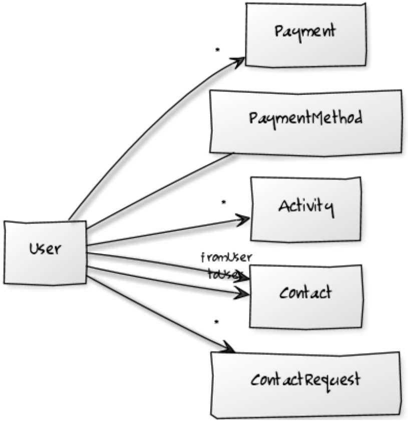
passion for software quality

Outline

Example: Establish Contact

- `User` A requests `Contact` to `User` B
- Contact request triggers:
  - notification mail
  - activity stream item
- Accept or Decline contact requests
  - with notifications again
- Accepting contact notifies other systems, example:
  - Synchronization of data
  - Indexing
  - Payment
  - Metrics/Logging

Example: Establish Contact



Example: Establish Contact

Demo

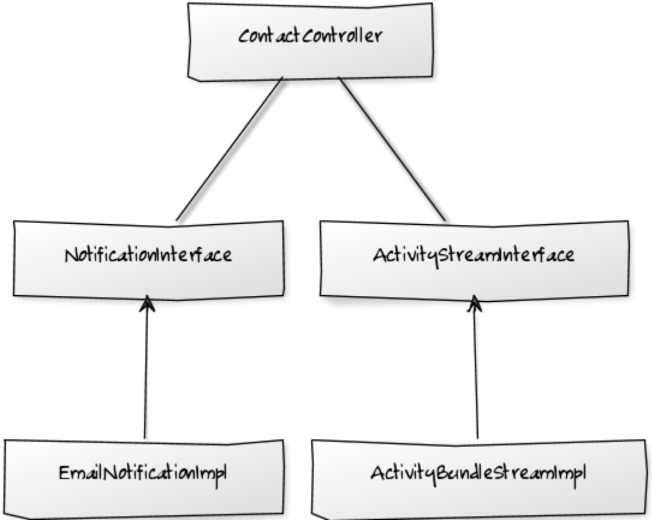
Questions

- Does creating services solve the problem?
- How many dependencies has the controller?
- To which bundles do the controller dependencies link?
- To which bundles do the entity dependencies link?
- How would you enable/disable/recombine features?

Fixing Service Dependencies

- Apply Dependency Inversion (SOLID principles)
- Use interfaces for operations/services
- Bundle that uses operation provides interface
- Other bundle implements that interface

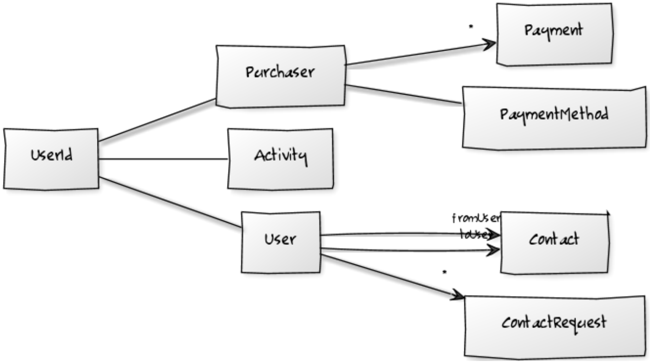
Example: Establish Contact



Fixing Entity Dependencies: Bounded Contexts

A BOUNDED CONTEXT delimits the applicability of a particular model so that team members have a clear and shared understanding of what has to be consistent and how it relates to other CONTEXTS. *(*Evans in Domain-Driven Design*)*

Example: Establish Contact



Problems

That still leads us with

- a hardcoded process/workflow
- too many dependencies in the controllers
- cognitive overload, everything is important

Outline

passion for software quality

What are Events?

An event can be defined as a significant change in state (Wikipedia*)*

Using Events

- State change makes the event happen
- Application creates Message Object for the event
- Message Object is passed to a dispatcher/publisher
- Listeners are notified of the event

Allows decoupling compontents even more!

Outline

passion for software quality

Properties of Event Messages

- No return value?
- No way to stop execution/propagatoin?
- Failure does not affect the event emitter?
- Asynchronous?
- Serializable?

Complexity through Event-based Architecture

- ACID transactions
  - Requires two-phase commit between datastorage and event dispatcher
  - Leads to complicated transaction management
- BASE transactions
  - Eventually Consistent data storage
  - Enables ”pull task”-based systems
  - Developers responsibility to handle failures

Outline

Discussion between Developers

Developer A: Where do you handle the case to send an email for a newly established contact between users?

Developer B: In the `ContactRequestCreatedPrePersistListener` when the `prePersist` Event happens.

Discussion between Developers

Developer A: How do you differentiate between a contact request triggerd by the user and batch import?

Developer B: By using a `kernel.request` listener that sets a boolean flag on the `ContactRequestCreatedPrePersistListener` for being inside a web-request. Only if the flag is `true` the email is sent.

A note on Symfony and Doctrine Events

Symfony and Doctrine events solve problems in the framework and database persistence contexts.

Hiding your business rules in them **will cause pain.**

Domain Events

”Something happend that domain experts care about”

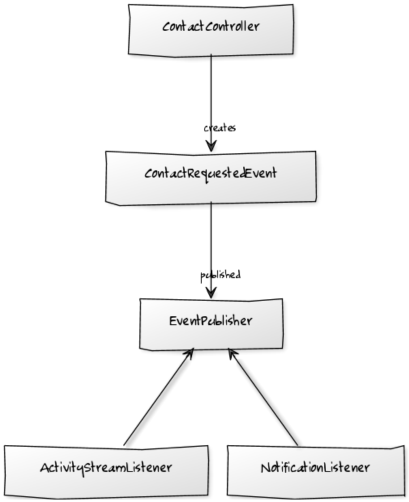
- Events for state changes in the model
  - in terms of the business
  - tied to the execution of use-cases
  - explicit in code

Event Storming

Example: Establish Contact

- `ContactRequested`
- `ContactEstablished`
- `ContactConfirmed`
- `ContactDeclined`
- `ContactImported`

Approach 1: Procedural



Approach 1: Procedural

```php
<?php
class ContactController
{
    public function requestAction(Request $request)
    {
        // ...
        $eventDispatcher = $this->get('event_dispatcher');
        $eventDispatcher->publish(
            'contact.requested',
            new ContactRequestedEvent($contactRequest)
        );
        // ...
    }
}
```

passion for software quality

Approach 1: Procedural

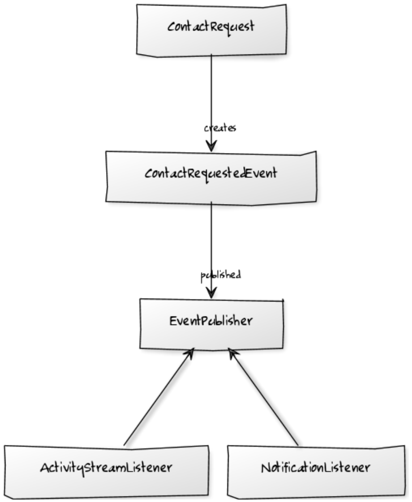
Benefits:

- Loose coupling
- Works with getters/setters and forms
- Can be easily added to existing (CRUD-)application

Drawbacks:

- Requires access to `EventDispatcher` in many places
- No two-phase commit between DB+Events
- Easy to forget triggering same event in other actions/commands

Approach 2: Events as state



Approach 2: Events as state

```php
<?php

class ContactRequest
{
    private $events = array();

    public function __construct($fromUser, $toUser)
    {
        $this->fromUser = $fromUser;
        $this->toUser = $toUser;

        $this->events[] = new ContactRequestedEvent($this);
    }
}
```

Approach 2: Events as state

```php
<?php

class ContactRequest
{
    private $events = array();

    public function confirm()
    {
        $this->confirmed = true;

        $this->events[] = new ContactConfirmedEvent($this);
    }
}
```

Qafoo
passion for software quality

Approach 2: Events as state

Benefits:

- Loose coupling
- Events are always created when state changes
- No dispatcher required in the model code
- Dispatching of events IFF storage tx successful
- Enables Event Sourcing

Drawbacks:

- Requires deep integration into Doctrine
- Does not work well with forms (unless..)

Conclusion

Decoupling can be achieved with

- interfaces for dependency inversion
- cutting the assocations between entities
- using events to communicate between bundles

Further Readings

- `http://www.whitewashing.de/2013/07/24/doctrine_and_domainevents.html`
- `http://www.whitewashing.de/2013/06/24/bounded_contexts.html`
- `http://www.whitewashing.de/2013/06/27/extending_symfony2__controller_utilities.html`
- `http://verraes.net/2013/04/decoupling-symfony2-forms-from-entities/`

passion for software quality

THANK YOU

https://joind.in/talk/view/10369

Rent a quality expert
qafoo.com