From codebase to database: Automation via workflows

...

A way of writing customizable logic.

Agenda

- What is workflow as a concept
- Some examples of workflow
- Automate workflows using traditional coding techniques
- Problem with traditional techniques
- Design a technique using event based architecture to solve the problem

- Use Cases
- Benefits / Advantages over traditional methods
- Overview: our implementation of this technique
- Challenges we are facing with this design

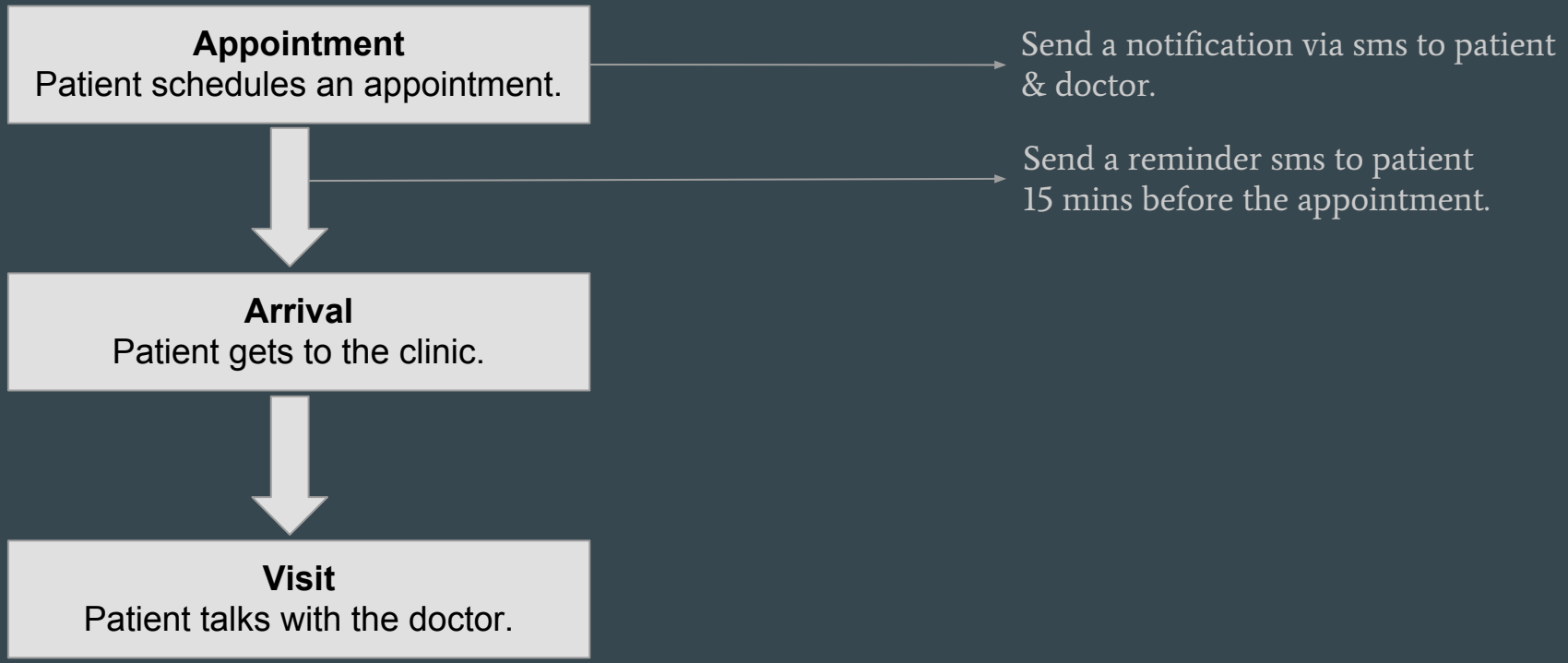
Workflow as a concept

- Dictionary meaning:
  The sequence of processes through which a piece of work passes from initiation to completion.
- It is a sequence of operations, declared as work of a person or group/organization.
- From a higher-level perspective, workflow may be considered a view or representation of real work.
- Workflows may be viewed as one fundamental building block to be combined with other parts of an organization's structure.

- History:
  One of the earliest usages of the term 'work flow' was in a railway engineering journal from 1921.

Sample Workflow

An example workflow for the purpose to this talk.



A simplistic workflow from medical services domain

Automated workflow via Traditional coding technique

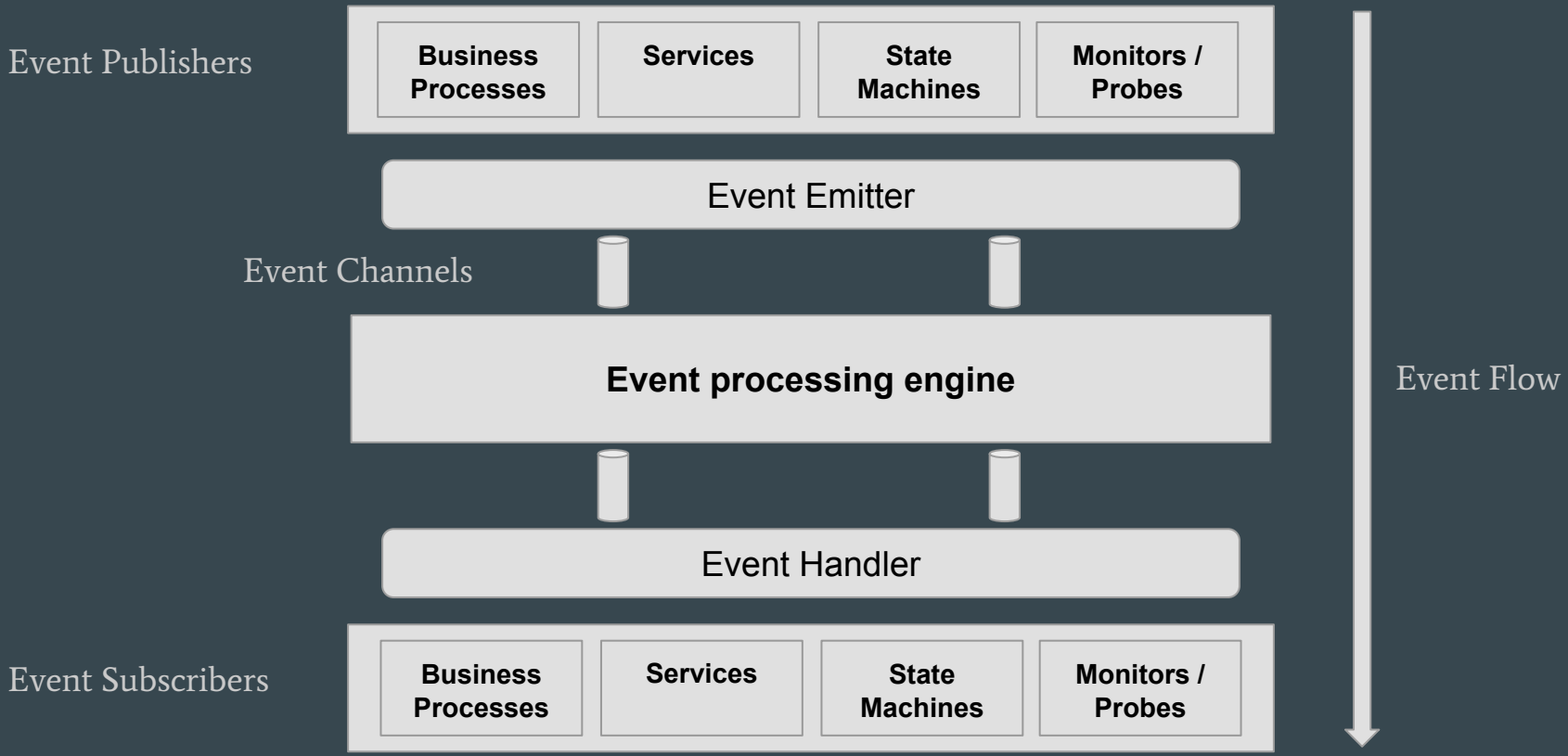
```ruby
class Appointment < ActiveRecord::Base
  belongs_to :doctor
  belongs_to :patient

  after_create :send_sms_to_doctor_and_patient
  after_create :remind_patient_before_hand

  def remind_patient_before_hand(appointment)
    MailerWorker.perform_at(
(appointment.scheduled_time - 15.minutes),
appointment.patient_id)
  end
end
```

Problems with this method:

- Suppose we try to sell this to different doctors, and found out that every doctor's workflow is little bit different.
- How can we achieve this kind of customizability in our application to cater for different requirements ?
- Suppose, we came up with settings feature where a set of settings can be enabled/disabled to handle different requirements. (but is it a feasible solution ?)
- So now we will design a solution based on event driven architecture, which is fully customizable and can handle requirement differences easily.

Event driven architecture (EDA)

- It's a software architecture pattern promoting the production, detection, consumption of, and reaction to events.
- Key aspects:
  1. Type of interaction: Event-driven
  2. Initiator: Event
  3. Participants: Open-ended
- Some principles of EDA:
  1. "Real-time" events as they happen at the producer
  2. One-way "fire-and-forget"
  3. Immediate action at the consumers
  4. Informational ("someone logged in"), not commands ("audit this")



EDA (generic block diagram)

Design a system using EDA to solve our problem

System comprised of following entities

System's primary focus should be on the domain & domain logic.

- Domain events:
  - A Domain Event is something that happened that the domain expert cares about.
  - By exposing relevant Domain Events on a shared event bus we can isolate cross cutting functions to separate system.
- Conditions:
  - Often domain logic accompanies with some conditional scenarios.
  - Through this entity, we decide whether to execute a given action after evaluating defined conditions. Conditions may be optional.
- Actions:
  - A way of reacting to domain events in the system.
  - We can perform any activity in the system when an event is triggered.

Automate the sample workflow with this event-driven system

- We will create a workflow as:
  - Triggering Event:
    - New appointment scheduled event.
  - Action:
    - Send a notification sms to doctor & patient with appointment time.
    - Send a reminder to patient 15 mins before appointment time.
- Let's name this workflow as 'notification_workflow'

- We can have a workflow model to save our custom workflows to database.
- Instead of sending notification directly in appointment's callback, we now publish appointment scheduled(created) event in it.
- Consider we have some method to subscribe a workflow to any domain event. Using that we can now register our notification_workflow to appointment scheduled event.
- We can implement some sort of Event processing engine which will find all the workflows registered on a triggered event & can execute them all.
- We can provide a GUI interface for creating & editing workflows. This way every health provider can create custom workflows according to his needs.

```ruby
class Worflow < ActiveRecord::Base
  has_many :conditions
  has_many :actions

  def execute
    # evaluate conditions & execute actions
    ...
  end
end

class Appointment < ActiveRecord::Base
  belongs_to :doctor
  belongs_to :patient
  after_create :new_appointment_created_event
end

register(notification_workflow,
:new_appointment_created)

# some method in Event processing engine.
def execute(worflow_id)
  workflow = Workflow.find(workflow_id)
  workflow.execute()
end
```

Use cases for EDA

- Notifications module
  - Notifications can be a cross cutting concern in many systems, which makes it a perfect contestant for using EDA.
  - For example: A system consisting of monitoring & reporting service
    - Monitoring service - want's to send notification when some metric goes above threshold value.
    - Reporting service - send different types of reports via notifications.
- Implement hooks within some service
  - For example: A popular code hosting platform 'Github' provides a webhook service.
    - It allows external services to be notified with a POST request, when certain events happen within your repository.

Let me select individual events.

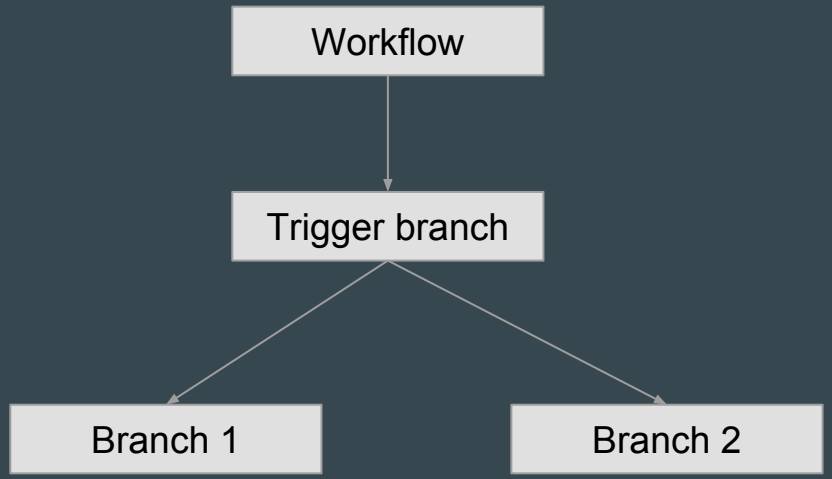
- [ ] **Commit comment**
  Commit or diff commented on.
- [ ] **Create**
  Branch or tag created.
- [ ] **Delete**
  Branch or tag deleted.
- [ ] **Deployment**
  Repository deployed.
- [ ] **Deployment status**
  Deployment status updated from the API.
- [ ] **Fork**
  Repository forked.
- [ ] **Gollum**
  Wiki page updated.
- [ ] **Issue comment**
  Issue comment created, edited, or deleted.
- [ ] **Issues**
  Issue opened, edited, closed, reopened, assigned, unassigned, labeled, or unlabeled.
- [ ] **Member**
  Collaborator added to a repository.
- [ ] **Page build**
  Pages site built.
- [ ] **Public**
  Repository changes from private to public.
- [ ] **Pull request**
  Pull request opened, closed, reopened, edited, assigned, unassigned, labeled, unlabeled, or synchronized.
- [ ] **Pull request review comment**
  Pull request diff comment created, edited, or deleted.
- [x] **Push**
  Git push to a repository.
- [ ] **Release**
  Release published in a repository.

Advantages of EDA over traditional methods

- Fully Customizable
- Highly Scalable
- Easily extensible
- Efficiently handle crosscutting concerns by decoupled system.

High level overview of ‘our’ implementation of EDA

- Binary tree structure
- Introduced notion of branches
- Branch can be action branch or condition branch
- Workflow has one compulsory branch called trigger branch
- Action branches are always present at leaf nodes of the tree



Challenges we are facing in this system

- Activities performed in workflow actions can trigger other events implicitly triggering other workflows subscribed to that event which can result into a loop of workflows triggering each other leading to deadlock conditions.
  How this problem can be solved efficiently ? (Think)

  Open for discussions !

References

- Wikipedia
- http://www.slideshare.net/stnor/event-driven-architecture-3395407

Thank You