7363 - Web-basierte Anwendungen
4750 – Web-Engineering

Eine Vertiefungsveranstaltung

Wettkampf-Simulation

Modellierung

Empfohlene Parameter

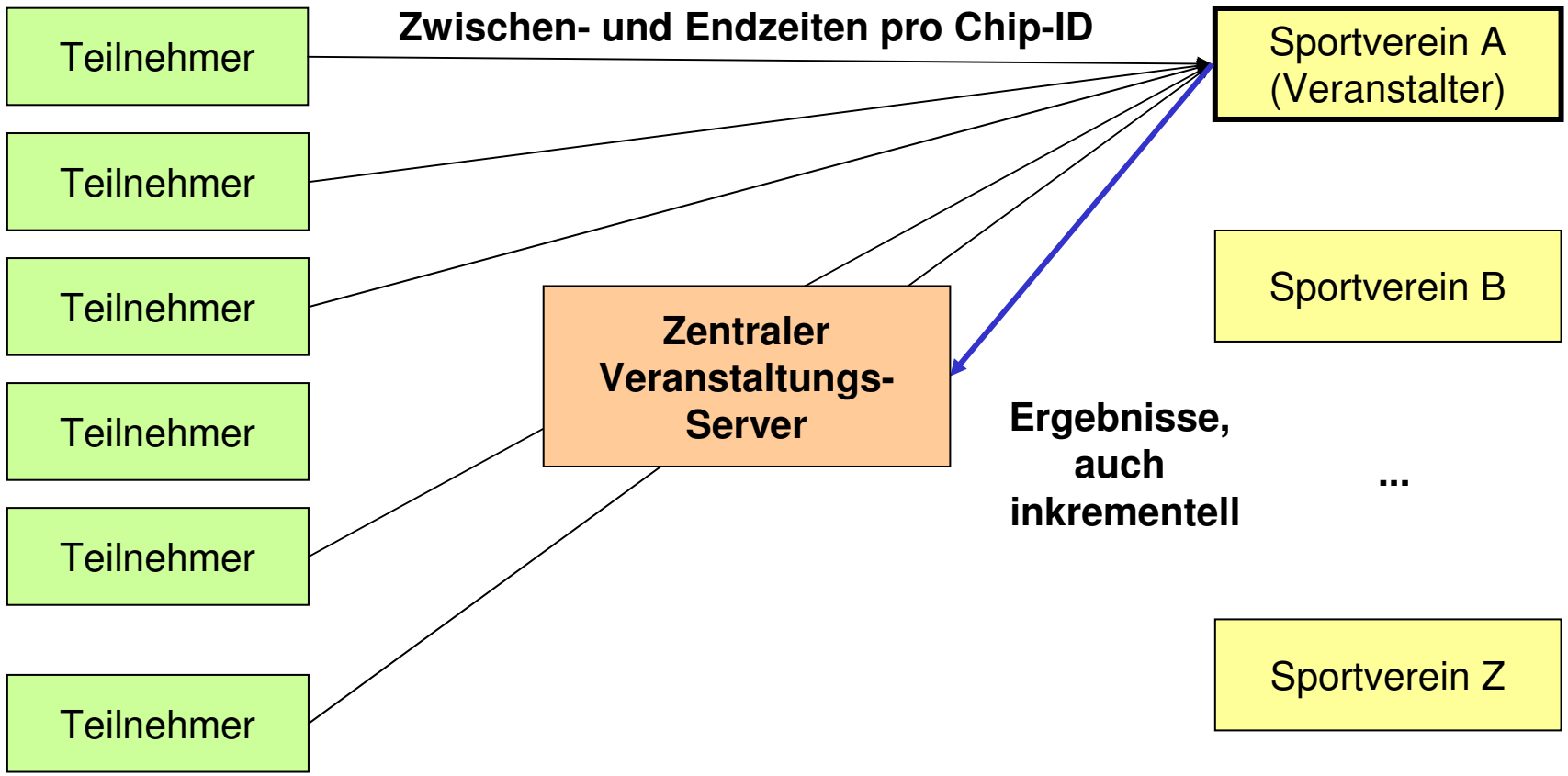
Überlegungen zur Datenübertragung

Ziele

- Ziele
  - Vermittlung physikalischer Grundlagen für die Implementierung der Wettkampf-Simulation
  - Parameter-Empfehlungen

Erinnerung

- Das Szenario: Wettkampf



Physikalische Modellierung

- Starterfeld: Jeder Starter mit Eigenschaften
  - x Aktuelle Position in Metern
  - v Aktuelle Geschwindigkeit in m/s
  - chipID
  - startNr
  - weitere Attribute …

- Simulation einer Zeitscheibe Δt:

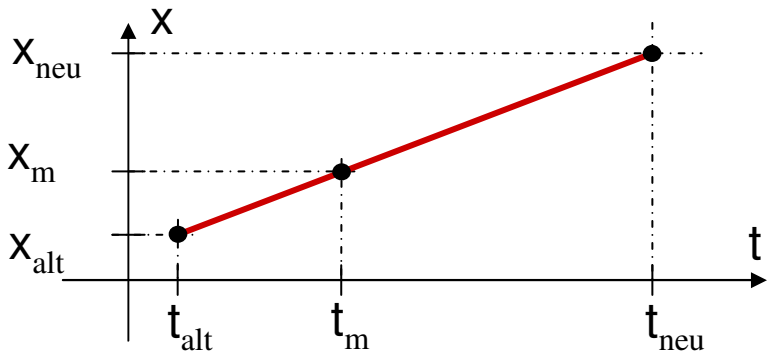
  Für t = 0, Δt, 2* Δt, … t_{max} :
  - Für alle Starter:
    - x → x + v * Δt
    - v → v + Δv(…) # Zu Δv siehe unten
    - Zwischenzeit(en) ermitteln
  - t += Δt
  - sleep Δt_{sim} # z.B. 30 sec für simulierte 5 Minuten

Physikalische Modellierung

- Zwischenzeitmessung: Einfach durch lineare Interpolation

  Sei x_m ein Messpunkt, etwa $x_m = 20000.0$ (km-Marke 20)

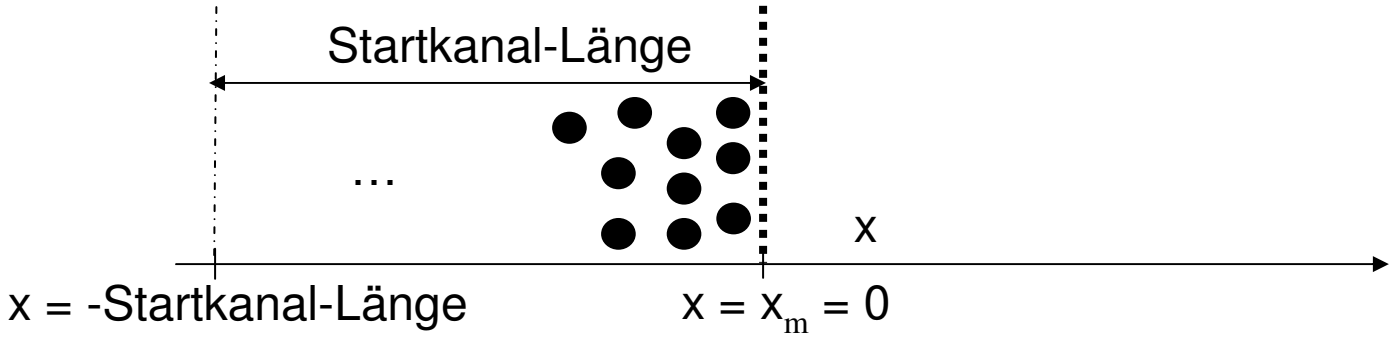
  – if $x_{alt} <= x_m$ and $x_{neu} > x_m$
    $t_m = t_{alt} + (x_m - x_{alt}) / v$
  – (chipID, t_m, x_m) melden



- Hinweise
  – Innerhalb einer Zeitscheibe können auch mehrere x_m durchlaufen werden, etwa km 20 und 21.1 (Halbmarathon) innerhalb 5 Min. Berücksichtigen Sie diesen Fall!
  – Damit nicht für jeden Starter für jede Zeitscheibe jede Messmarke getestet werden muss:
    - Sortieren Sie die Messmarken nach ihrer Entfernung
    - Führen Sie pro Starter einen Index, in dem Sie speichern, welche Messmarke als nächste erwartet wird, und testen Sie erst ab dieser Marke.
    - Keine weiteren Tests, wenn Index > Index der letzten Marke (im Ziel!)

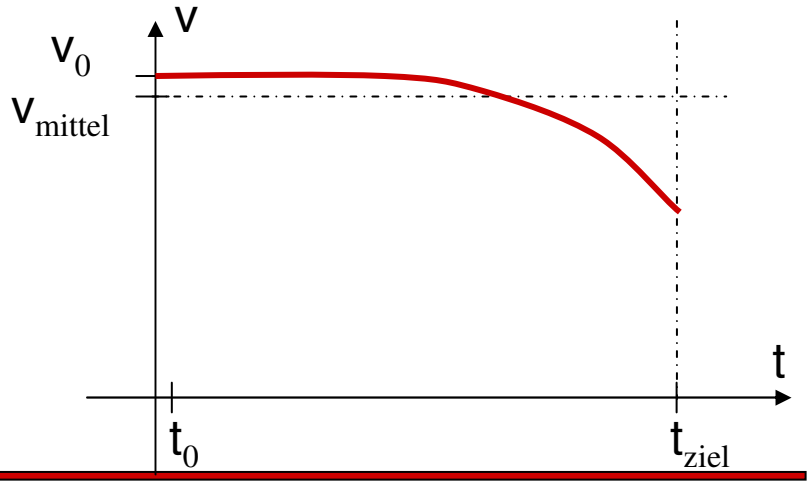
Physikalische Modellierung

- Netto- und Bruttozeiten
  - Initialisieren Sie die Starterpositionen x auf Zufallswerte aus einem Intervall [-Startkanal-Länge, 0.0]
  - Startschuss = $t_0 := 0$
  - Erster Messpunkt = Startlinie
    - Für dieses m gilt: $x_m = 0$
  - Gemessen & gemeldet werden nur Bruttozeiten
    - Nettozeiten: $t_{m,\,net} = t_m - t_{x=0}$



Physikalische Modellierung

- Variation der Geschwindigkeiten
  - Variante 1: Individuell verschiedenes, aber konstantes v
    - $\Delta v = 0$. Langweilig!
  - Variante 2: v variiert in jeder Zeitscheibe um Zufallswert
    - $\Delta v = \Delta v_{max} * (2 * rand - 1)$ # rand = Zufallszahl aus [0, 1]
    - Einfach, aber unrealistisch
  - Variante 3: v erst konstant, dann immer schneller fallend
    - z.B. per Parabel modelliert
    - $\Delta v = \Delta v(t)$ oder $\Delta v = \Delta v(x)$
    - Realistischer
    - Für Ambitionierte!



Wettkampf-Simulation

- Modellparameter (Empfehlungen)
  - **Anzahl Starter: ca. 1000**
    - Nur wenige müssen namentlich gemeldet sein,
    - aber für alle sollten Startnr. und Chip-ID bekannt sein
  - **Länge des Startkanals: 100 … 500 m**
  - Geschwindigkeiten
    - **v_{min} = ca. 8 km/h, v_{max}= ca. 18 km/h, Δv_{max}= 1 km/h**
    - Erinnerung: Durch 3.6 teilen ergibt m/s
  - Zeiten
    - **Δt = 5 min = 300 s, t_{max}= 5 h** (entspricht ca. 42.195 km / v_{min})
    - **Δt_{sim} = 20 s = Δt/15.**
      - Ein 5-Stunden-Wettkampf wird also in 20 Minuten simuliert. Das reicht für unsere Zwecke während der Abnahme.
      - Zum Testen empfohlen: **Δt_{sim} = 5 s**

Wettkampf-Simulation

- *On-line* Demo
  - Ruby-Programm laufsimu.rb:
    - Diskussion des Quellcodes (teils nach Diskussion der REST-Folien)
  - Demo-Lauf

- Hinweis für Ihre Projektabnahme
  - Nach Demonstration von Vereinsregistrierung, …, Teilnehmermeldungen sollten Sie einen Wettkampf simulieren.
  - Während der Wettkampf läuft, werden Zwischenzeiten an den zentralen Server geleitet.
    - Übertragen Sie NICHT jeden Messwert einzeln, sondern z.B. alle Messwerte aus einer Zeitscheibe mit einem Aufruf.
    - EIN Simulationsprogramm für ALLE Mess-Stationen genügt!
  - Zwischenergebnisse (temporäre Ergebnislisten) werden dann während des noch laufenden Wettkampfs (per Browser) abgerufen!
    - Sinn: Verfolgung des Rennverlaufs !

Wettkampf-Simulation

- Überlegungen zur REST-konformen Datenübertragung
  - Im DB-Sinn „besitzt“ ein Wettkampfteilnehmer mehrere Messwerte. Bsp.:

```
class Participant < ActiveRecord::Base
  has_many :measurements
end
class Measurement < ActiveRecord::Base
  belongs_to :participant
end
```

  - Zu den Attributen der Klasse „Measurement“ müsste dann neben „time_gross“, „chip_code“ und „location“ (oder: „location_id“) auch „participant_id“ gehören
- Probleme
  - „participant_id“ ist der Mess-Station bzw. dem Simulationsprogramm nicht bekannt. Diese Angabe muss vielmehr vom Veranstaltungs-Server rekonstruiert werden, und zwar aus dem Chip-Code (und der Wettkampf-ID)
  - Die Übertragung jeder einzelnen Messung mit REST ist ineffizient!

Wettkampf-Simulation

- Überlegungen zur REST-konformen Datenübertragung
  - Ausweg:
    - Schaffung eines neuen Modelltyps, etwa „MSet“ (Measurement set)
    - Kopfdaten: Veranstalter, Passwort sowie Wettkampf-ID
    - Positionsdaten: Einzelmessungen
  - Minimal-Controller dazu
    - Nur „create“, nur im XML-Modus unterstützen
    - Kein ActiveRecord-Modell zu „MSet“
    - Entweder: Direktes Verbuchen der Positionsdaten
    - Oder:
      - Zwischenspeichern als Dateien im Dateisystem (wie bei der „Abgabe“-Übung)
      - Verbuchen dieser Daten mit einem asynchron laufenden Prozess
  - Alternative:
    - MSet-Inhalt als BLOB speichern, CRUD-Möglichkeiten erhalten
    - Verbuchen der MSets durch separaten Prozess
  - Diskussion / Bessere Vorschläge?

Anhang: Code-Beispiele aus der Demo

- Übertragung der XML-Daten mit Net::HTTP

```
require "net/http"
require "uri"

# In Klasse „Measurements“:
 def send
    uri = URI.parse self.target_url
    Net::HTTP.start(uri.host, uri.port) do |conn|
      response = conn.post(uri.path, self.to_xml,
                           'Content-type' => 'text/xml')
    end
 end
#
# Bemerkungen: Fehlerbehandlung fehlt noch
```

Anhang: Code-Beispiele aus der Demo

- Erzeugung der XML-Daten mit „Builder“

```
require "rubygems"
require "builder"

# In Klasse „Measurements“:
def to_xml
  output = ""
  b = Builder::XmlMarkup.new(:target => output, :indent => 2)
  b.mdata do |x|
    x.club_id club_id
    x.passwd  passwd
    x.competition_id  competition_id
    x.measurements do |y|
     @data.each do |m|
        y.measurement do |z|
          z.chip_code  m.chipID
          z.time_gross m.t
          z.distance   m.x
        end
      end
    end
  end
  output
end
```

Anhang: Code-Beispiele aus der Demo

- Auswertung der XML-Daten mit „REXML“

```
class MeasurementsController < ApplicationController
  # POST /measurements.xml
  def create
    respond_to do |format|
      format.html { render :text => "HTML nicht erlaubt!" }
      format.xml  do
          doc  = REXML::Document.new(request.body.read)
          meas = REXML::XPath.match(doc, '//measurement')
          # Echo der Daten als Demo - Ort für Ihre Verbuchung?
          str = "Measurements received: #{meas.size}\n"
          meas.each do |m|
            ck = m.elements['chip_code'].text
            tg = m.elements['time_gross'].text.to_f
            di = m.elements['distance'].text.to_f
            str << "%10s %6.2f %6.2f\n" % [ck, tg, di]
          end
          render :text => str
      end
    end
  end; end
```