ESE CPCC Project

Michael Kleber, Clemes Krainer,
Andreas Schröcker, Bernhard Zechmeister

Department of Computer Sciences
University of Salzburg, Austria

January 25, 2012

Content

Outline

Introduction
Task

- simulation of physical helicopter swarms
- simulation of sensors
- abstraction of virtual vehicles (virtual helicopters)
- migration of virtual vehicles among flying physical helicopters

Introduction
Project Scope

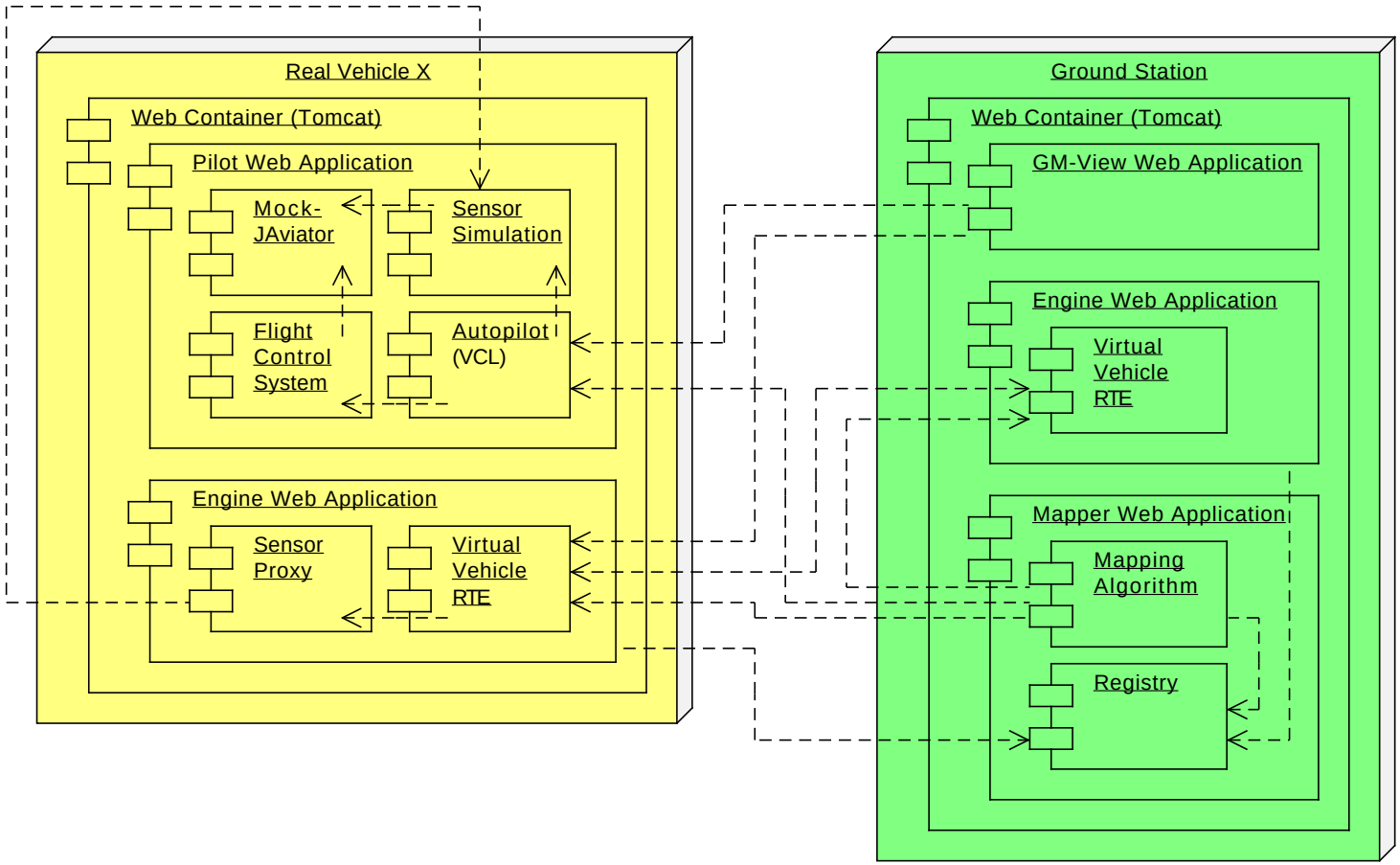
- real vehicles (physical helicopters) follow strict flight plans
- no network bandwith limits
- no processing power limits

Introduction
Applied Technologies

- HTTP as protocol for sensor abstraction and data exchange
- Java as programming language
- software implemented as web applications
- Apache Tomcat as web server and servlet container

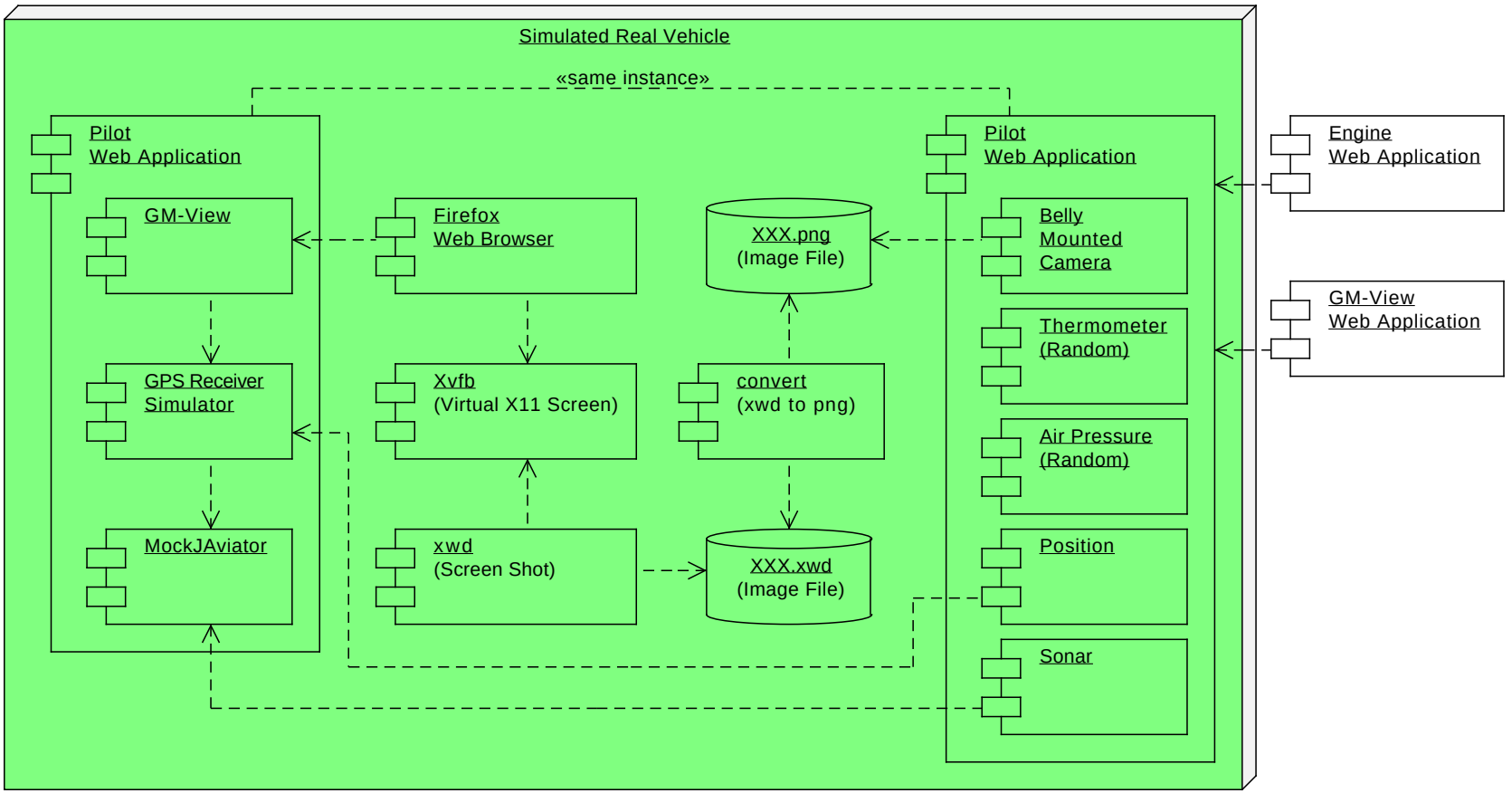
Outline

Introduction
System Overview



Introduction

Sensor Simulation



Outline

Real Vehicles
Vehicle Configuration

```
plant.simulated = true
plant.type = MockJAviator
plant.listener = udp://localhost:9011
plant.location.system.type = gpssim
plant.location.system.listener = tcp://localhost:9012
plant.location.system.update.rate = 10

controller.simulated = true
controller.type = JControl

pilot.type = JPilot
pilot.name = Pilot One
pilot.controller.connector = udp://localhost:9014
```

Real Vehicles
Sensor Configuration

```
sensor.list = gps, temp, photo

sensor.gps.name = GPS receiver
sensor.gps.path = position
sensor.gps.uri = gps:///

sensor.temp.name = thermometer
sensor.temp.path = temperature
sensor.temp.uri = rand:///18/22

sensor.photo.name = belly mounted photo camera
sensor.photo.path = photo
sensor.photo.uri = x11:///:21
```

Real Vehicles

Vehicle Control Language

```
##
## @(#) real vehicle set course
##
go auto
takeoff 1m for 5s
fly to (47.82204197, 13.04086670, 20.00)abs precision 1m 2.0mps
fly to (47.82206088, 13.04092035, 20.00)abs precision 1m 2.0mps
fly to (47.82195102, 13.04488063, 20.00)abs precision 1m 2.0mps
hover for 20s
land
go manual
```

Outline

Vehicle Virtualization

Virtual Vehicle Program

- ability to suspend
- state is serialized
- information is persisted to file
- migration can be performed
- virtual vehicle can resume

Vehicle Virtualization

Virtual Vehicle Language

- list of commands
- command consists of a point and a list of actions
- point contains latitude, longitude, altitude
- specification of tolerance

Vehicle Virtualization

Virtual Vehicle Sample Program

Point 47.82201946 13.04082647 1.00 tolerance 12.3
Picture
Temperature

Point 47.82203026 13.04084659 25.00 tolerance 100
Temperature

Point 47.82211311 13.04076076 30.00 tolerance 1.2
Picture

Vehicle Virtualization

Scanner

- lookahead of one
- double and integers
- keywords and variables
- easy to add keywords - (prepared for adding if, else, while, for ...)

Vehicle Virtualization

Parser

- process symbols of scanner
- parser handles
  - command (position, actions)
  - position (point, tolerance)
  - point (lat. long. alt.)
  - actions
- error handling
  - throws parser exception with description
  - stop parsing

Vehicle Virtualization

Saving State

- java serialisation used
- file with state
- list of commands with actions serialized
- already collected data in seperate file

Vehicle Virtualization

Execution of VV

- VV are dispatched through
- execute function
- read state, execute, store state
- partial execution of commands supported

Outline

Mapper

- maps virtual vehicles to real vehicles
- invokes migration
- two components:
  - registration service
  - mapper

Mapper
Registration Service

- engine registers itself with registration service
- service fetches useful information:
  - sensors
  - waypoints

Mapper
Mapper

- cyclic
- fetches status of all virtual vehicles
  - next action point
  - and its actions
- status of all real vehicles
  - current position
  - next position
  - velocity
- two algorithms:
  - random mapping algorithm
  - simple mapping algorithm

Mapper
Simple Mapping Algorithm

for all virtual vehicles **do**

if virtual vehicle program is complete
then invoke migration to central engine

else find fastest real vehicle with at least one needed sensor
and distance CN to P < tolerance

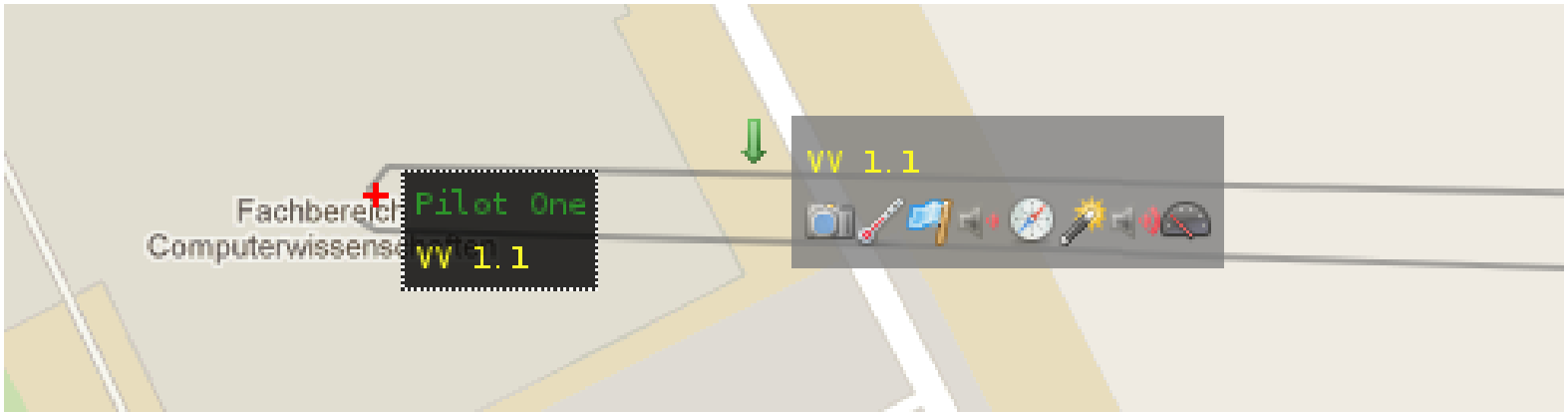
if found vehicle **then** invoke migration to it

Outline

Live Demonstration

Demo 1: Data Collection

- one flying real vehicle
- one virtual vehicle that collects data at four locations
- no migration

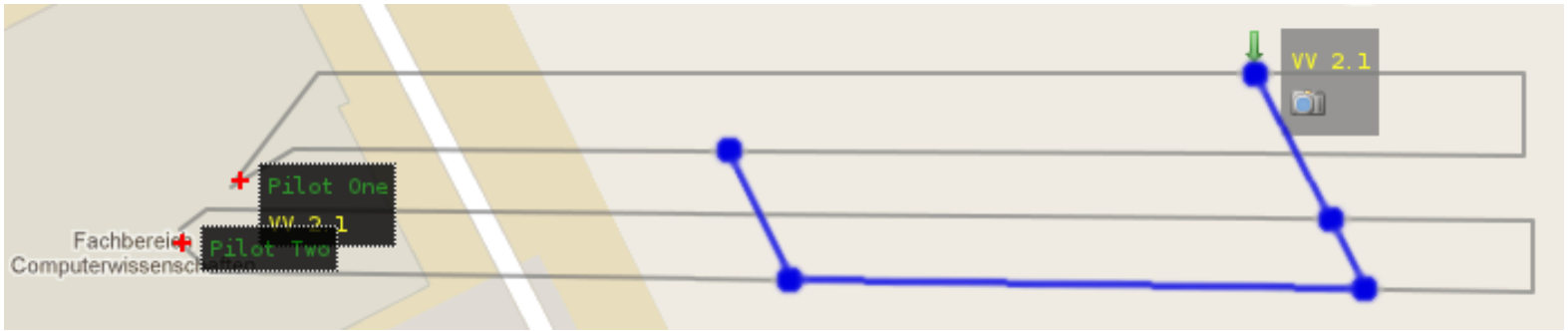


Outline

Live Demonstration

Demo 2: Virtual Vehicle Migration

- two flying real vehicles
- one virtual vehicle that collects data at five locations
- migration among both real vehicles

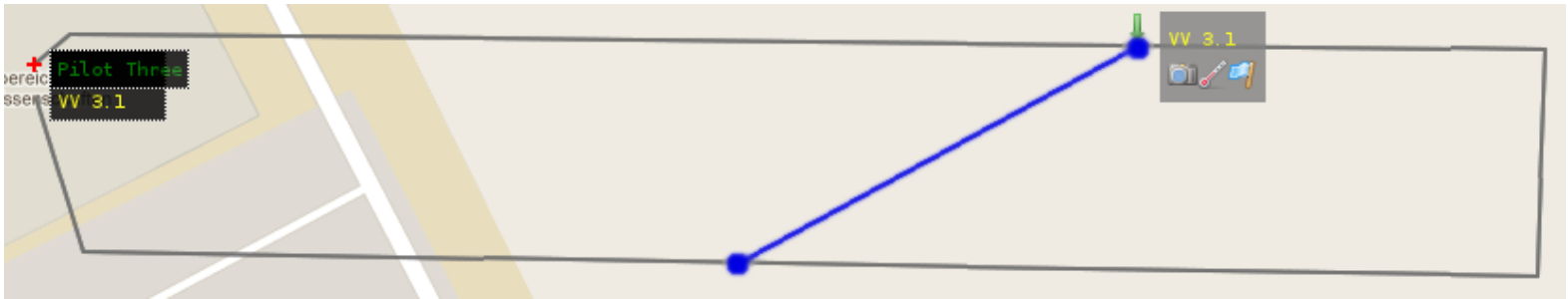


Outline

Live Demonstration

Demo 3: Real Vehicles with different Sensors

- three flying real vehicles carrying different sensors
- one virtual vehicle that collects data at two locations
- migration among all real vehicles

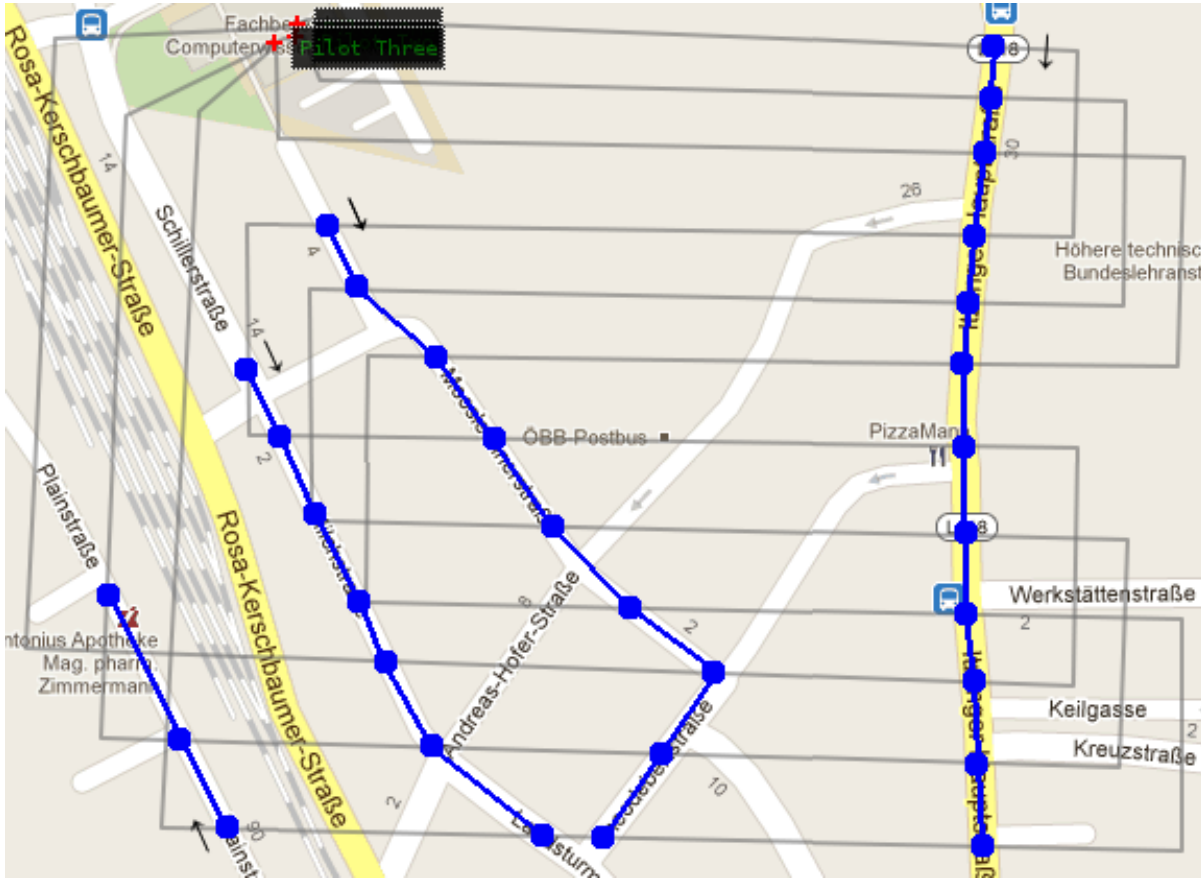


Outline

Live Demonstration

Demo 4: Complex Scenario

- three flying real vehicles
- four virtual vehicle that collect data
- migration among all real vehicles



Outline

Future Work

- more subtle mapping algorithms
- network traffic optimizations
- video sensor support
- extended geo-location
- flight plan generation based on virtual vehicle programs

Outline

Questions & Answers

Q

&

A