RPC, RMI, SOA

Martin Kuba, ÚVT MU

Komunikace v distribuovaných systémech

- z hlediska synchronnosti
  - **synchronní** – volající strana zastaví a čeká, dokud nedostane odpověď
  - **asynchronní** – volající strana pokračuje v práci, na příchod odpovědi je upozorněna
- z hlediska zajištění doručení zprávy
  - **transientní** (pomíjivá)
  - **persistentní** (vytrvalá)
- webové služby obvykle používají transientní synchronní komunikaci, ale lze použít všechny kombinace

RPC - Remote Procedure Calls

- distribuované systémy komunikují zasíláním zpráv
- vhodná abstrakce – RPC - Remote Procedure Calls
- např. DCE RPC, SUN RPC, …
- synchronní volání *požadavek-odpověď*
  - volaná procedura, parametry předávané hodnotou
  - návratové hodnoty
- IDL – Interface Definition Language
- klient a server *stubs*
  - volané jako běžné funkce v daném prog. jazyce
  - zajišťují marshalling/serializaci, komunikaci, unmarshaling/deserializaci

RMI - Remote Method Invocation

- distribuované *objektově-orientované* systémy potřebují předávat parametry odkazem
- distribuovaný objekt má **stav**, metody – **interface**, a **implementaci**
- Remote Method Invocation - např. CORBA, Java RMI, DCOM
- binární protokoly, Object Request Broker
- Java RMI umí předat objekt – stav i implementaci metod

SOA - Service Oriented Architecture

- RMI funguje jen v systémech pod centrální správou, neškáluje na Internet-size
  - synchronní komunikace neškáluje
  - *tight coupling* - verzování a evoluce jsou obtížné
  - distribuovanost nelze schovat (partial failure)
- SOA – Service Oriented Architecture
  - služby mají definovaný interface
  - interface je popsán zprávami, ne operacemi na datových typech
  - služby lze nalézt (např. v adresáři)

SOA (2)

- rozdíl mezi OO a SOA
  - přehrávač CD poskytuje službu přehrání CD
  - různá kvalita služby ve walkmanovi nebo HiFi věži
  - v objektově-orientovaném přístupu by každé CD bylo dodáno s vlastním přehrávačem, ze kterého by nešlo vyjmout
- SOA patrně lépe odpovídá způsobu, jímž jsou organizovány lidské aktivity
- *loose coupling* - nezávislá evoluce klientů a služeb provozovaných různými organizacemi

Web services

Martin Kuba, ÚVT MU
PA160 lecture, spring 2020

A web service is a software system designed to support interoperable machine-to-machine interaction over a network.

(W3C, Web Services Glossary)

Glossary

URL - Uniform Resource Locator

HTTP - Hypertext Transfer Protocol

HTML - Hypertext Markup Language

XML - Extensible Markup Language

GUI - Graphical User Interface

CGI - Common Gateway Interface

SSL/TLS - Secure Sockets Layer/Transport Layer Security

REST - Representational State Transfer

JSON - JavaScript Object Notation

AJAX - Asynchronous JavaScript and XML

Brief web services history

1989 - World Wide Web invented

1991 - HTTP 0.9 specified

1992 - Internet at Masaryk University :-)

1993 - first GUI web browser Mosaic

1993 - CGI interface for executing programs

1995 - JavaScript introduced by Netscape

1996 - SSL 3.0

1998 - XML 1.0

1998 - SOAP 1.1 by Microsoft

2003 - SOAP 1.2 by W3C (never used)

2004 - WS-Interoperability Basic Profile

Brief web services history (2)

2000 - REST defined by Roy Fielding

2001 - JSON format invented

2004 - GMail and Google Maps

2004 - Web 2.0, wikis, mash-ups

2005 - AJAX (Asynchronous JavaScript)

2005 - Yahoo! offers JSON web services

2006 - OpenID 2.0

2008 - HTML5 (First Public Working Draft)

2010 - OAuth 1.0

2010 - mobile devices with Android

2012 - OAuth 2.0

Brief web services history (3)

2013 - responsive web design as answer to mobile devices with differing screen sizes

2006-2013 - cloud computing (Amazon 2006, Microsoft 2008, Google 2013)

2014 - HTML5 finalised

2014 - OpenID Connect

2015 - HTTP/2, JSON Web Tokens

2016 - OpenAPI (Swagger)

2018 - TLS 1.3

My definition of a web service

web service client communicates with a web server providing a web resource identified by a URL, using HTTP protocol (optionally secured by TLS) exchanging messages in XML or JSON formats

this definition covers

- SOAP/WSDL services
- REST APIs
- dynamic web pages using AJAX

SOAP/WSDL web services

- SOAP was **S**imple **O**bject **A**ccess **P**rotocol
- WSDL is **W**eb **S**ervice **D**escription **L**anguage
- technology for remote procedure calls using exchange of XML messages
- preferred in the enterprise world
- used in API of the Czech eGovernment's "Data Boxes"
- WS-Interoperability Basic Profile needed to ensure interoperability
  - requires SOAP1.1
- many WS-* extensions

SOAP call

```
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body>
  <m:GetPrice xmlns:m="https://www.w3schools.com/prices">
    <m:Item>Apples</m:Item>
  </m:GetPrice>
</soap:Body>

</soap:Envelope>
```

SOAP response

```
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body>
  <m:GetPriceResponse xmlns:m="https://www.w3schools.com/prices">
    <m:Price>1.90</m:Price>
  </m:GetPriceResponse>
</soap:Body>

</soap:Envelope>
```

SOAP/WSDL web services (2)

- started as XML-based Remote Method Invocation protocol
- changed to Remote Procedure Call protocol (no objects - SOAP is not abbreviation now)
- introduced own type system
  - big problems with compatibility
- later replaced by XML Schema type system
- main lesson - remote interfaces should be defined by ***messages***, not ***operations***

SOAP versus REST

- enterprises prefer complicated stack
  - XML
  - SOAP, WSDL, WS-Interoperability
  - WS-* (WS-Security, WS-Addressing, ...)
  - persistent connections - queues
  - RPC based
  - complex tools and frameworks, need an IT department
- Internet crowd prefers simplicity
  - JSON
  - web APIs described as HTTP requests to URLs
  - AJAX in browsers
  - transient connections - TCP/IP, HTTP
  - scalable using REST

Web APIs

- well-known APIs
  - Google APIs (Calendar, GMail, Maps, ...)
  - Facebook API
  - Twitter API
  - based on HTTP+TLS+JSON+OAuth
- third party clients
  - web, mobile (Android, iOS, ...), desktop, embedded
- OAuth
  - developer registers an **application** at API provider
  - user authorises the application to use certain operations in the **API**, giving the application a **token**
  - **application** uses the **token** to use the **API** on behalf of the user

JSON - JavaScript Object Notation


```
{
   kind: "calendar#events",
   etag: "\"GZxpEFttRDAOmLHnWRxLHHWPGwk/vpPwPyIKi2CubgzCWOVY8MIHGPo\"",
   summary: "EGI.eu Events",
   updated: "2013-04-22T06:00:02.000Z",
   timeZone: "Europe/Amsterdam",
   accessRole: "reader",
 - items: [
    - {
         kind: "calendar#event",
         etag: "\"GZxpEFttRDAOmLHnWRxLHHWPGwk/Z2NhbDAwMDAxMjY5ODQ0NDcwMDkzMDAw\"",
         id: "vs17ehlthhfrlgke0a0o98hors",
         status: "confirmed",
         htmlLink: https://www.google.com/calendar/event?eid=dnMxN2VobHRoaGZybGdrZTBhMG85OGhvcnMgZXZlbnRzQGVnaS5ldQ,
         created: "2010-02-12T08:47:42.000Z",
         updated: "2010-03-29T06:34:30.093Z",
         summary: "EGEE to EGI Transition Meeting for User Community and Operations",
         description: "A focus on the transition of the EGEE NA2, NA3 and NA4 activities to the EGI era with significa
         followed by more general transition of EGEE operations to NGI operations from Tuesday afternoon. A detailed a
         /conferenceDisplay.py?confId=1",
         location: "Nikhef",
       - creator: {
            email: "steven.newhouse@egi.eu",
            displayName: "Steven Newhouse"
         },
       - organizer: {
            email: "events@egi.eu",
            displayName: "EGI.eu Events",
            self: true
         },
       - start: {
            dateTime: "2010-03-01T13:00:00+01:00"
         },
       - end: {
            dateTime: "2010-03-03T12:00:00+01:00"
         },
         visibility: "public",
         iCalUID: "vs17ehlthhfrlgke0a0o98hors@google.com",
         sequence: 0
      },
```

The same Google Cal event in XML


```
- <entry>
  - <id>
      http://www.google.com/calendar/feeds/events%40egi.eu/private/full/vs17ehlthhfrlgke0a0o98hors
    </id>
    <published>2010-02-12T08:47:42.000Z</published>
    <updated>2010-03-29T06:34:30.000Z</updated>
    <category scheme="http://schemas.google.com/g/2005#kind" term="http://schemas.google.com/g/2005#event"/>
  - <title type="text">
      EGEE to EGI Transition Meeting for User Community and Operations
    </title>
  - <content type="text">
      A focus on the transition of the EGEE NA2, NA3 and NA4 activities to the EGI era with significantly reduced EC funding during the firs
      to NGI operations from Tuesday afternoon. A detailed agenda is available - https://www.egi.eu/indico/conferenceDisplay.py?confId=1
    </content>
    <link rel="alternate" type="text/html" href="https://www.google.com/calendar/event?eid=dnMxN2VobHRoaGZybGdrZTBhMG85OGhvc
    <link rel="self" type="application/atom+xml" href="https://www.google.com/calendar/feeds/events%40egi.eu/private/full/vs17ehlthhfrlg
  - <author>
      <name>Steven Newhouse</name>
      <email>steven.newhouse@egi.eu</email>
    </author>
  - <gd:comments>
      <gd:feedLink href="https://www.google.com/calendar/feeds/events%40egi.eu/private/full/vs17ehlthhfrlgke0a0o98hors/comments"/>
    </gd:comments>
    <gd:eventStatus value="http://schemas.google.com/g/2005#event.confirmed"/>
    <gd:where valueString="Nikhef"/>
    <gd:who email="events@egi.eu" rel="http://schemas.google.com/g/2005#event.organizer" valueString="events@egi.eu"/>
    <gd:when endTime="2010-03-03T12:00:00.000+01:00" startTime="2010-03-01T13:00:00.000+01:00"/>
    <gd:transparency value="http://schemas.google.com/g/2005#event.opaque"/>
    <gd:visibility value="http://schemas.google.com/g/2005#event.public"/>
    <gCal:anyoneCanAddSelf value="false"/>
    <gCal:guestsCanInviteOthers value="true"/>
    <gCal:guestsCanModify value="false"/>
    <gCal:guestsCanSeeGuests value="true"/>
    <gCal:sequence value="0"/>
    <gCal:uid value="vs17ehlthhfrlgke0a0o98hors@google.com"/>
  </entry>
</feed>
```

AJAX

- **A**synchronous **J**avaScript **A**nd **X**ML
- (Ajax was a Greek mythological hero)
- AJAX does not need XML, uses JSON mostly
- enabled by introduction of **XMLHttpRequest** JavaScript object to web browsers around the year 2006
- asynchronous request to web server
- response processed in JavaScript
- same-origin policy (protocol,host,port)
- Cross-origin resource sharing (CORS)

REST

- **Re**presentational **S**tate **T**ransfer
- software **architecture style** for creating scalable web services
- invented by Roy Fielding, author of HTTP 1.1
- resources identified by URIs
- representations of resources as JSON, XML or other formats
- uses HTTP methods GET, PUT, DELETE and POST for manipulating resources

REST API Descriptions

- API described in human natural language
  - e.g. “image can be changed by HTTP PUT request to /image/{imageID} with the image in request body”
- WSDL 2.0 defined in 2007, but never used
- OpenAPI since 2016
  - machine-processable description of REST interfaces
  - a form of IDL (Interface Description Language)
  - written in YAML language, which is a more human-readable superset of JSON

OpenAPI

- “machine-readable interface files for describing, producing, consuming, and visualizing RESTful web services”
- developed since 2010 as **Swagger**, renamed to **OpenAPI** in 2016
- version 3.0.0 released in 2017
- latest version 3.0.2 released in 2018
- API description in file **openapi.yml**
- tool **OpenAPI Generator** can generate clients in about 40 programming languages

```yaml
openapi: 3.0.2
info:
  title: My awesome API
  version: 1.0.0
  description: Just an example of OpenAPI description
servers:
  - url: 'https://my.example.org/api/v1'
components:
  schemas:
    User:
      type: object
      properties:
        id: { type: integer }
        firstName: { type: string }
        lastName: { type: string }
  responses:
    UserResponse:
      description: returns a User
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/User"
  parameters:
    id:
      name: id
      description: numeric id
      schema:
        type: integer
      in: query
      required: true
paths:
  '/getUser':
    get:
      operationId: "getUser"
      summary: "returns a User for a given id"
      parameters:
        - $ref: '#/components/parameters/id'
      responses:
        '200':
          $ref: '#/components/responses/UserResponse'
```

Java client library generated by OpenAPI Generator

```java
/**
 * returns a User for a given id
 *
 * @param id numeric id (required)
 * @return User
 * @throws ApiException If fail to call the API, e.g. server error
 */
public User getUser(Integer id) throws ApiException {
    ApiResponse<User> resp = getUserWithHttpInfo(id);
    return resp.getData();
}
```

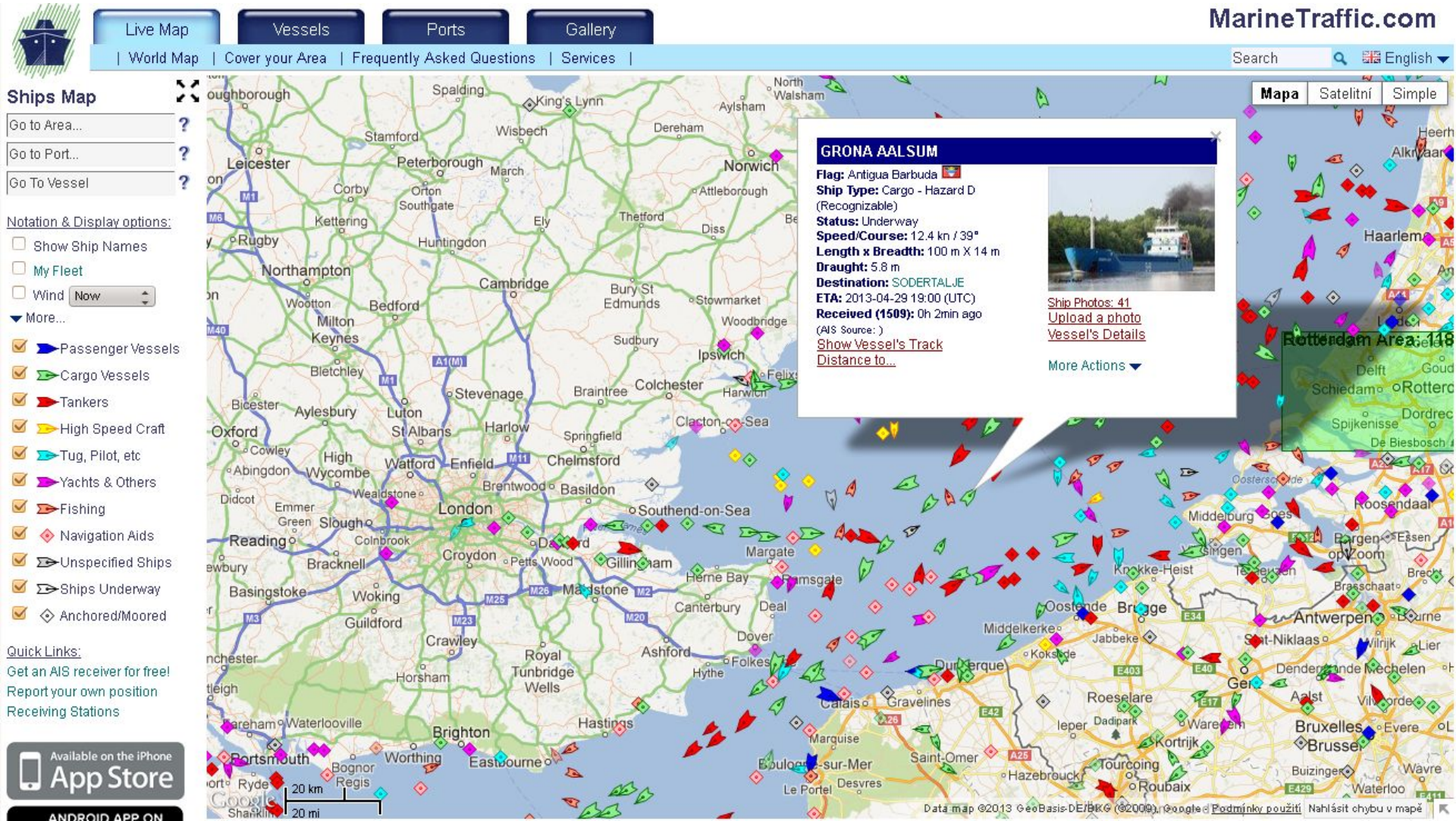
Python client library generated by OpenAPI Generator

```python
class DefaultApi(object):
    """NOTE: This class is auto generated by the swagger code generator program.

    Do not edit the class manually.
    Ref: https://github.com/swagger-api/swagger-codegen
    """

    def __init__(self, api_client=None):
        if api_client is None:
            api_client = ApiClient()
        self.api_client = api_client

    def get_user(self, id, **kwargs):  # noqa: E501
        """returns a User for a given id  # noqa: E501

        This method makes a synchronous HTTP request by default. To make an
        asynchronous HTTP request, please pass async_req=True
        >>> thread = api.get_user(id, async_req=True)
        >>> result = thread.get()
```

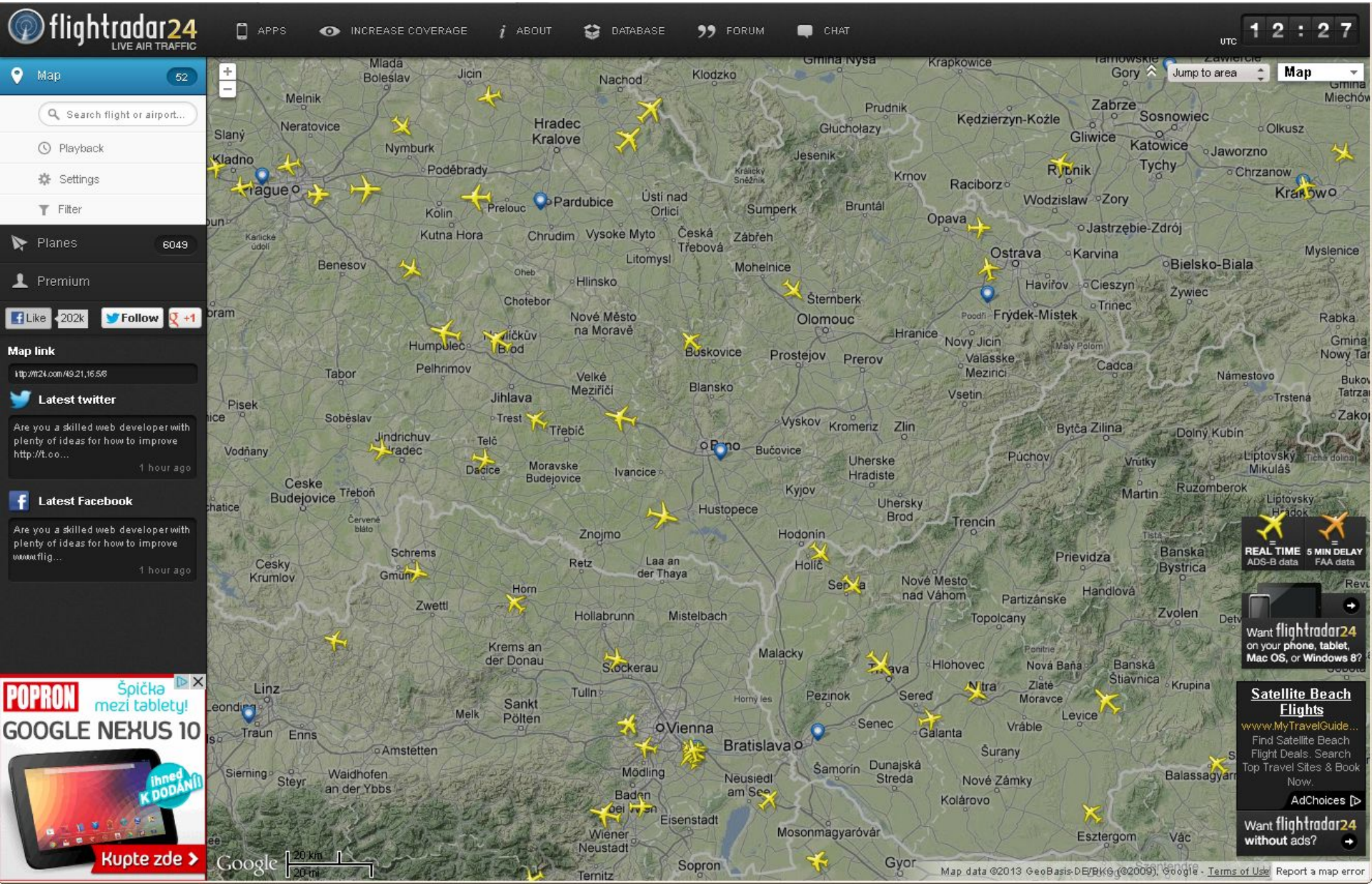
Mash ups

- combine data from various sources
- typically a Google map with some geospatial data
  - ships - http://www.marinetraffic.com/
  - aircrafts - http://www.flightradar24.com/

www.marinetraffic.com



www.flightradar24.com



Federated identity

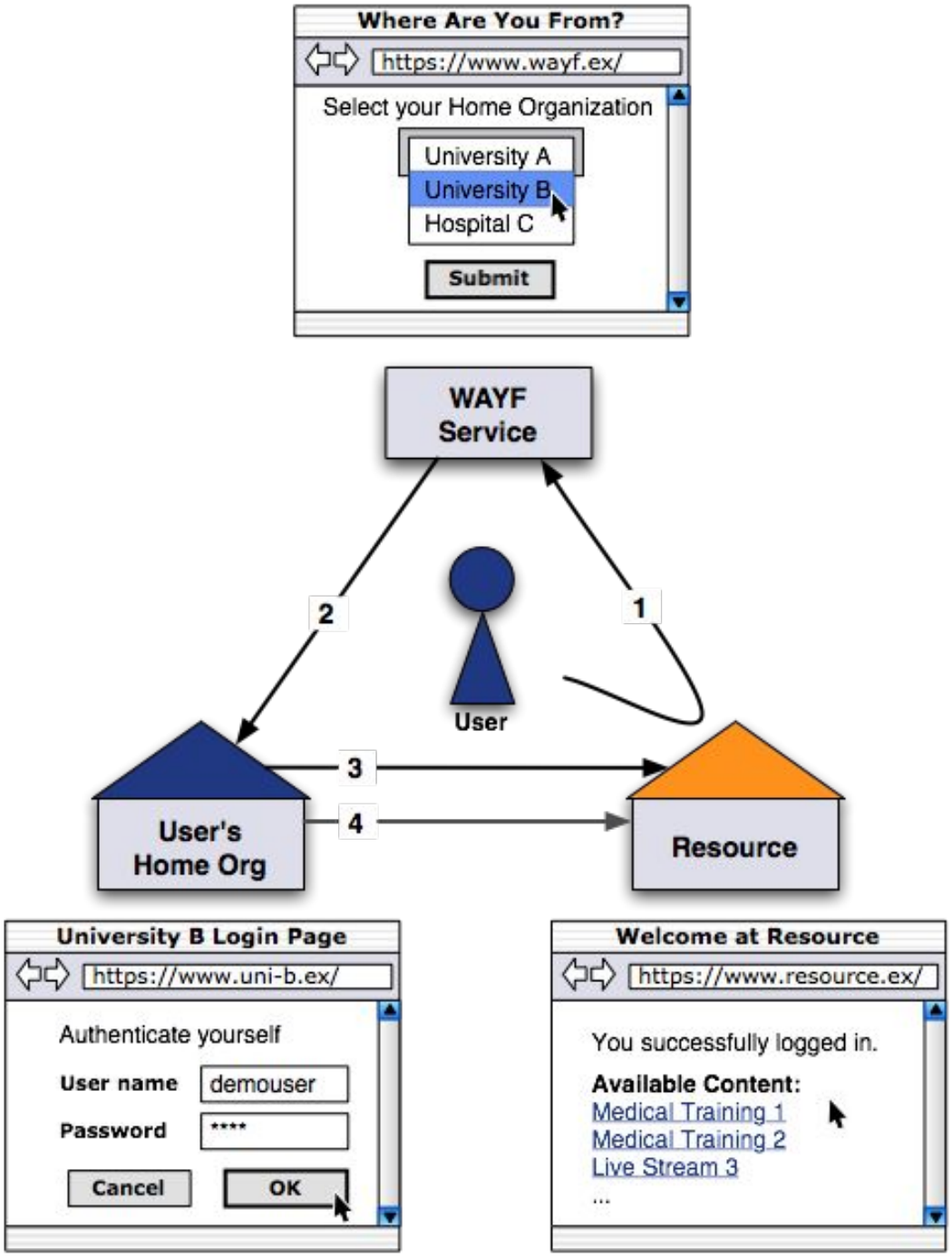
- many authentication mechanisms were developed for the web
  - username+password (hard to remember)
  - X509 digital certificate (complicated to get)
  - digest, Kerberos etc. (not much support in browsers)
- users forget passwords to rarely used accounts
- in federated identity, account from one organisation can be reused at others
- protocols and identity providers:
  - OpenID - MojeID.cz, anybody (obsolete)
  - SAML - in academia, Microsoft O365, Google Apps
  - OAuth - Google, Facebook, Twitter, ...
  - OpenID Connect - mix of OpenID and OAuth

OpenID versions 1 and 2

- obsolete
- introduced the idea of decentralized authentication protocol
- users were identified by URLs
- anybody could run an identity provider
- problem of trust
- only large identity providers like Google were trusted by service providers

SAML

- **S**ecurity **A**ssertion **M**arkup **L**anguage
- introduced in 2001
- provides **web browser single sign-on**
- SAML document is XML containing user attributes signed by an identity provider
- trust between identity providers (IdP) and service providers (SP) is established using **federations**
- a federation publishes list of trusted IdPs and SPs complying with federation's policy
- WAYF - Where Are Your From? service


Where Are You From?
https://www.wayf.ex/
Select your Home Organization
University A
University B
Hospital C
Submit
WAYF Service
2
1
User
3
4
User's Home Org
Resource
University B Login Page
https://www.uni-b.ex/
Authenticate yourself
User name
demouser
Password
****
Cancel
OK
Welcome at Resource
https://www.resource.ex/
You successfully logged in.
Available Content:
Medical Training 1
Medical Training 2
Live Stream 3
...

OAuth

- open standard for **authorization**, commonly used as a way for Internet users to authorize websites or applications to access their information on other websites but without giving them the passwords
- can be also used for authentication
- more in separate slides

OpenID Connect

- promoted as third version of OpenID
- authentication layer built on top of OAuth 2.0
- OAuth used for authorization
- standardized **UserInfo API**
- OpenID used for user data items (email, full name, etc.)

JWT - JSON Web Tokens

- RFC 7515 - JSON Web Signature
  - <header>.<payload>.<signature>
  - all 3 parts are base64-encoded, safe for URLs
  - <header> is JSON metadata identifying signing key
- RFC 7519 - JSON Web Tokens
  - JWS with JSON payload

JSON Web Token example

https://jwt.io/

Encoded PASTE A TOKEN HERE

```
eyJraWQiOiJyc2ExIiwiYWxnIjoiUlMyNTYifQ.e
yJzdWIiOiJtYWt1YiIsImlzcyI6Imh0dHBzOlwvX
C9teS5leGFtcGxlLm9yZ1wvIiwiZXhwIjoxNTY2M
zAyOTc4LCJpYXQiOjE1NjYzMDI5MTgsImp0aSI6I
jZhYzA3M2E2LTUwOTAtNDkyZS1hMmYzLTI0ZjQwN
zEzYWRjNCJ9.GvVyT_6YOKdjVk57o2sWUn3KYjtK
D0R8TBDeTemn_3BOV28oOD2mUE0lsU0xe3L0uHCb
_zS6tmG02I-
G_sDDbFkaaHoee6V8rrRBDOpqMNTorEdb75n3BrX
sYFQ7IUKa-
1JKx9fm6tHE1AQaksXoKlAoA4FCvZ5V8RBDg8-
cY9h5ixfZU4gg0xBZayo_hGcGz6HBtes9qq2PA5V
WwDhDAGZpdOWuLB44sl5CWuLQIfFzHUEgG2tsG-
k8FOnfaNUirWi0psrOd96EGVGkxgBrPV0peD4A_D
AN4qHKm3fPd3034vPemIZ_WtTxVlTarRBYX8fSan
7x5ZBxLP-s9rsV8g
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "kid": "rsa1",
  "alg": "RS256"
}
```

PAYLOAD: DATA

```
{
  "sub": "makub",
  "iss": "https://my.example.org/",
  "exp": 1566302978,
  "iat": 1566302918,
  "jti": "6ac073a6-5090-492e-a2f3-24f40713adc4"
}
```

VERIFY SIGNATURE

```
RSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
```