We *Publish*, You *Subscribe*

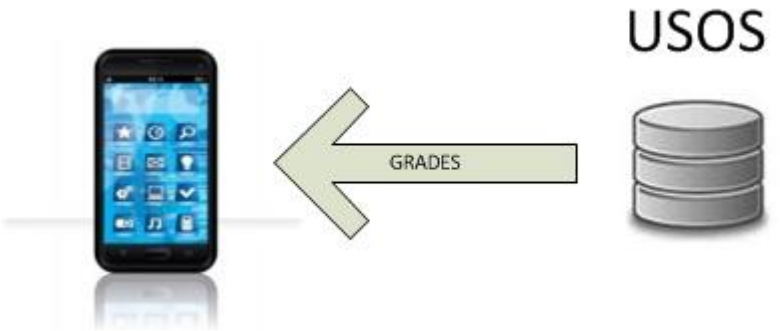
Hubbub as a Natural Habitat for Students and Academic Teachers

Janina Mincer-Daszkiewicz
University of Warsaw
jmd@mimuw.edu.pl

Umea, 2014-06-11

Agenda

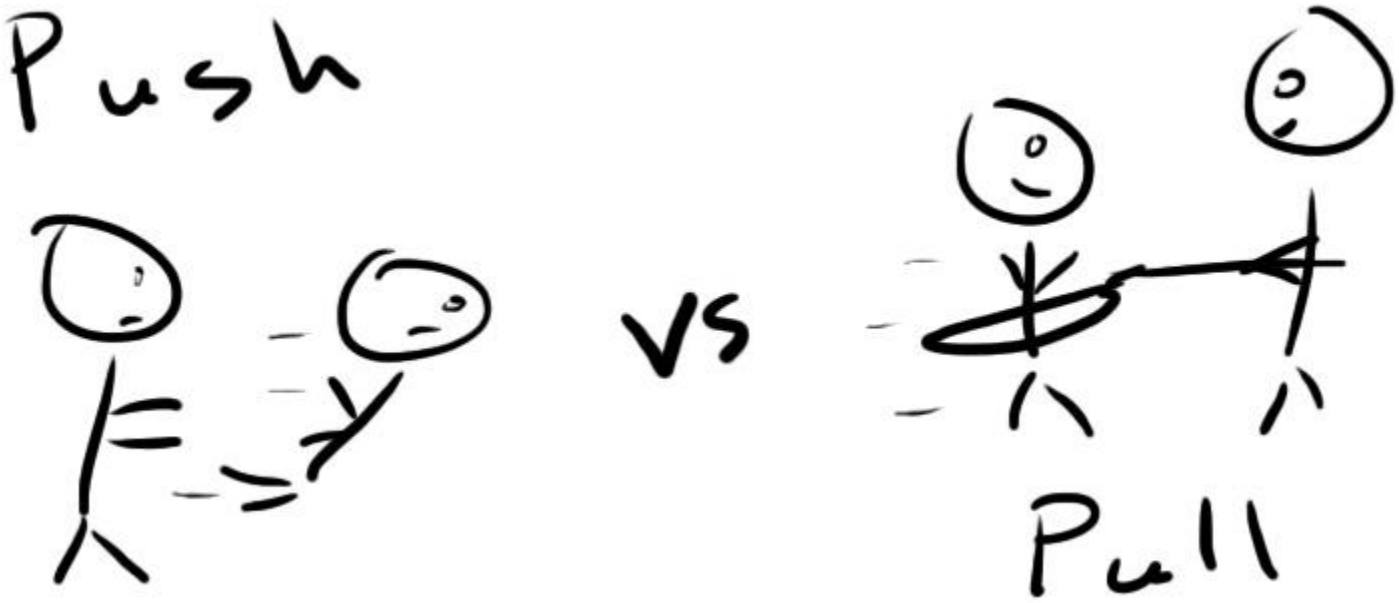
- Mobile applications based on PUSH (instead of PULL) paradigm
- PubSubHubbub protocol
- Real time notifications
- OAuth protocol
- USOS API



USOS stands for University Study-Oriented System
(used by over 40 HEIs in Poland)

Stating the problem

- Members of academic community, **students** and **academic teachers**, want to use smart phones and mobile Internet to access data stored in academic databases.
- **Providers** of student management information systems (**SMIS**) for higher education offer mobile applications to fulfill these needs.
- There is no question of whether to allow such access, the question and challenge is how to deliver information in **real time** (instantly), in a **user friendly** manner, without exposing university servers to **crashes** in peak hours.
- There is also a problem of data **confidentiality**. Nobody wants to jeopardize personal data, photos, grades, registrations, diplomas etc. stored in an academic information system.



http://www.sonlight.com/blog/2011/10/not-negative-the-positives-of-homeschooling.html

PUSH or PULL

- **USOS API** is a public API to **USOS**. Mobile applications may use USOS API to get access to information handled by USOS.
- However the standard **PULL** method does not scale well.
- Systems like *Facebook*, *Twitter*, *Flickr*, *Foursquare* base their **notification services** on a **PUSH paradigm**, where it is a server, not a client which starts communication. Clients may **subscribe** to be notified about the events of interest. Information **providers** send notifications in real time to a designated **hub**, from which they are further distributed to all subscribers.
- PUSH is not only more efficient, but also more natural for phone owners who are used to obtain SMSs without any activity on their side. This is the responsibility of the telecom service provider to locate the phone with the appropriate number and deliver SMS straight to it.

System of notifications follows the same idea

Number of events

- The academic community of the University of Warsaw consists of more than **50 thousand** students and more than **3.5 thousand** academic teachers.
- **Grades** are among the most interesting objects handled by any SMIS.
- In the most busy month, June 2013, there were **247,685 grade changes** concerning **40,248 users**.
- That means that for this one particular event type the system should be able to send to each client (mobile application) about **250 thousand notifications concerning about 40 thousand users in time shorter than one month**.

This is the efficiency we want to deliver

What hubbub has to do with that?

http://hubbubclub.org/info/who-we-are/

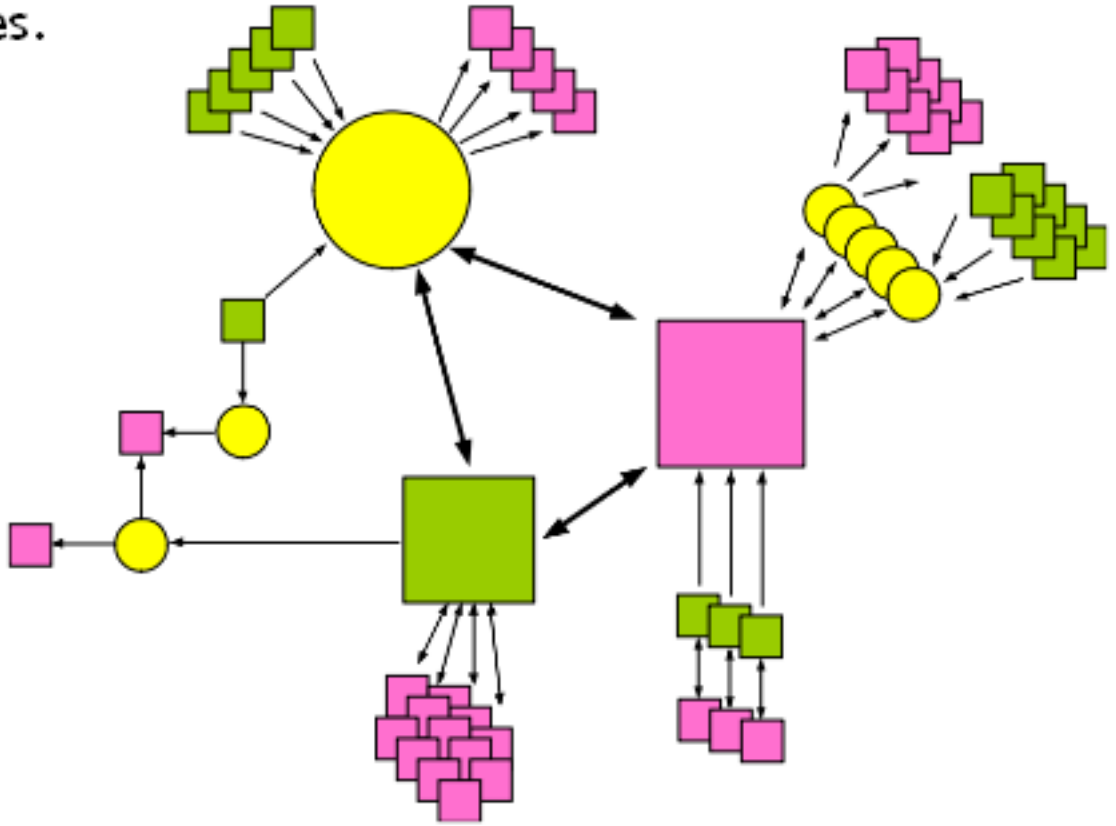
PubSubHubbub protocol

- **Open protocol** used by parties which want to communicate in a **publish-subscribe** manner.
- In order to provide a secure path, subscribers should **share a secret** with the hub, to be used by the hub to compute an HMAC signature that will be sent to the subscriber.
- A client chooses an **object** he wants to observe, indicates object **attributes** of interest, **URL** and a **verification token**.
- The system first verifies the subscription by making request to the given URL, with the verification token attached. Subscription is confirmed when the server answers properly to this request. The purpose of this verification is to **prevent DDoS attacks** by making sure that the client controls the server available under the given URL.

PubSubHubbub protocol – cont.

- Whenever one of the indicated attributes changes the value, the notification system makes **HTTP POST** to the given URL. The request (in **JSON** format) does not contain new values of the attributes.
- It has an extra HTTP X-Hub-Signature header with the HMAC checksum of the request body calculated using SHA1 algorithm with the key being the shared secret known to the hub and the client (private key from the OAuth protocol).
- No **sensitive data** is sent in notifications. If the client wants to obtain the new values, a new request should be made.
- **Facebook**, **Flickr**, **Foursquare**, **Instagram** implement real time notifications in a similar way.
- **Twitter** delivers Streaming API — users connect to endpoints from which they read potentially infinite stream of data.

The future is distributed: There will be big hubs, many small hubs, and tons of publishers and subscribers. Publishers, subscribers, and hubs may play multiple roles.



Publishers

Subscribers

Hubs

https://code.google.com/p/pubsubhubbub/

USOS API

- **USOS API** is a standard **REST-like** interface to data gathered in USOS, publicly available, well documented, with guaranteed backward compatibility.
- Single USOS API installation consists of three functional parts:
  - Web services (API methods) available for applications.
  - Mini-portal for programmers, with public documentation (in English) of all methods. This portal is also used for generating keys used to identify applications with the server.
  - Administration panel for students and academic teachers. Users have access here to the list of applications, to which they have granted access to their USOS data.
- API methods are gathered in modules, e.g. *users*, *courses*, credits, *exams*, *grades*, *geo*, *mailing*, *oauth*, *photos*.
- Results of methods are delivered in XML or JSON format.

http://apps.usos.edu.pl/developers/api/

OAuth protocol in USOS API

- **OAuth** (*Open* standard for *Authorization*) is an open protocol for clients (usually applications) to access resources (such as confidential data) on behalf of a resource owner (usually end user).
- In a classical model of client-server authentication, the client uses its credentials (*username* and *password*) to gain access to its resources located on the server.
- In OAuth model, the client (which is not the owner of the resource, but only acts on his behalf) requests access to the resources which are controlled by their owner, but are located on the server. The client has to get permission from the owner first. It is expressed in the form of an **access token**. The purpose of the token is to avoid the situation when the owner has to share his password with the client. Unlike passwords, tokens may be issued with scope and time constraints (and cancelled at any time).
- OAuth is used by Facebook, GitHub, Google, Microsoft, PayPal, Twitter, Yahoo!, Flickr, Foursquare, Instagram, LinkedIn, and many others.

OAuth protocol in USOS API

- There is an analogy given by Eran Hammer-Lahav http://hueniverse.com/oauth/guide/:

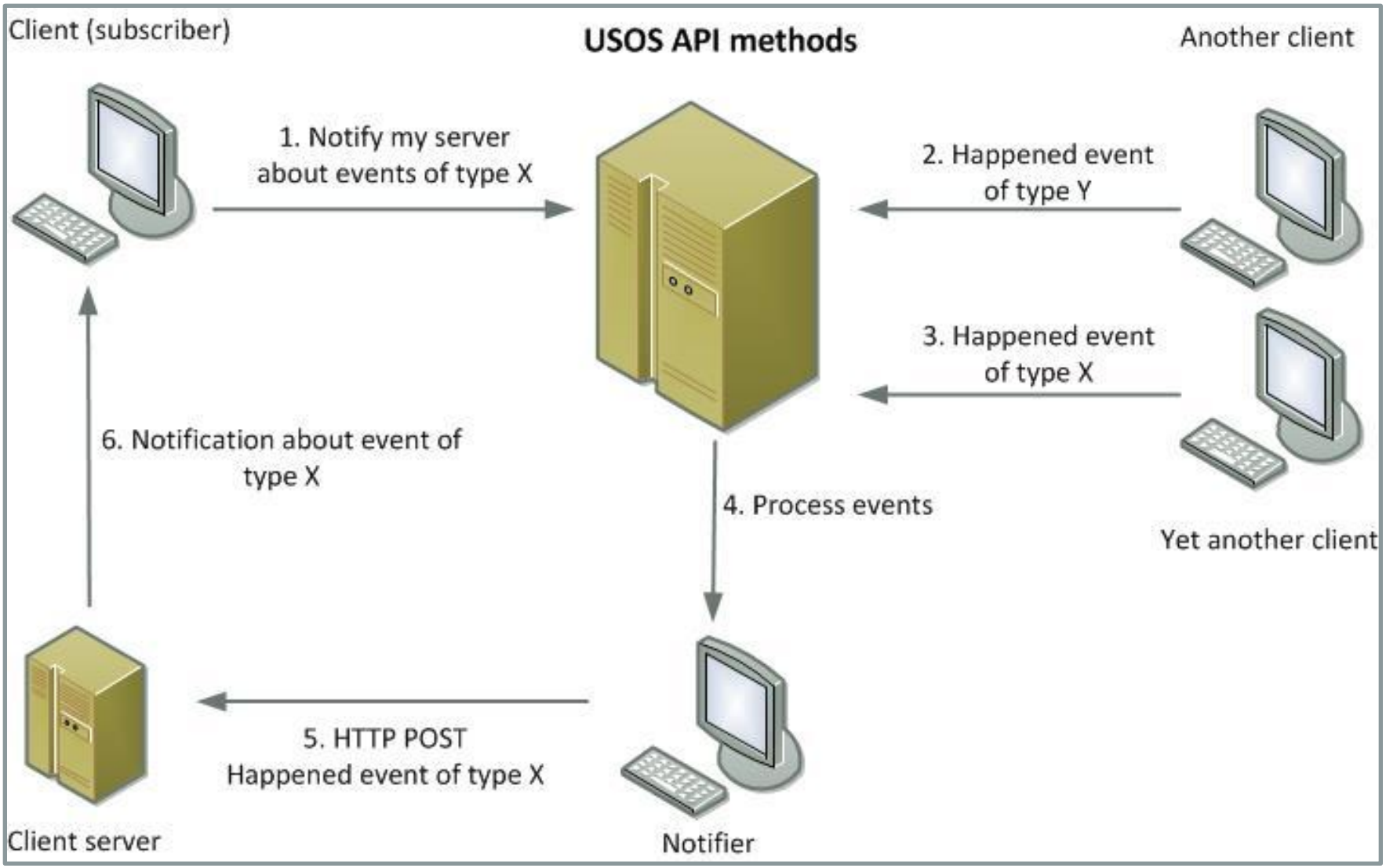
  *Many luxury cars come with a **valet key**. It is a special key you give the parking attendant and unlike your regular key, will only allow the car to be driven a short distance while blocking access to the trunk and the onboard cell phone. Regardless of the restrictions the valet key imposes, the idea is very clever. You give someone **limited access** to your car with a **special key**, while using **another key** to unlock everything else. [..] The decoupling of the resource owner's username and password from the access token is one of the most fundamental aspects of the OAuth architecture.*

- Academic databases store private data (like citizen identity numbers, names, birth dates etc.), confidential data (like information about handicapped students or social aid), data of business value (like mailing lists of students and staff members), data which is vulnerable for theft or destruction (like grades or thesis reviews).
- **USOS API** implements **OAuth** protocol to **protect** the data.

Implementing notifications in USOS API

- There are two main problems to be solved when implementing the system of notifications: how to manage subscriptions and how to distribute notifications.
- **Subscriptions** are managed by USOS API, what means that the new API module had to be designed and implemented, with methods to subscribe/unsubscribe to particular events.
- **Notifications** may be distributed in two possible ways.
  - They might be sent when the event occurs. This solution is simple but potentially absorbs many resources.
  - Another possibility is to queue information about events and sent notifications later, grouping them by event type.

  The second solution was chosen.
- The hub is a **special daemon** which has access to USOS API tables where **subscriptions** and **events** are stored. The daemon periodically browses the tables for new records and distributes notifications to the clients.



General overview of the system architecture

Implementing notifications in USOS API

- The newly developed module ***services/events*** contains the following methods:
  - *subscribe_event* — subscribe to events of a given type.
  - *unsubscribe* — unsubscribe from events.
  - *subscriptions* — list the consumer subscriptions.
  - *notifier_status* — get information on the status of the notification daemon.
- **Event types** are paths to methods like ***services/grades/grade*** which change data in the database.
- Client (mobile application) subscribes to a given type of events through ***subscribe_event*** method.
- Client's request is **validated** to make sure that it is in fact under control of the server described by the URL delivered in the invocation. Providing the validation was successful, a new subscription is created.
- USOS API **stores subscriptions** in the local table.

Implementing notifications in USOS API

- Assume that an event of the chosen type occurred.
  - USOS API checks it the client is **authorized** to receive the notification.
  - The client's server receives an **HTTP request** with one or more notifications. The request method is POST. The request body is in JSON format
- There are two **security requirements** concerning the system of notifications:
  - Protecting sensitive data

    Solved by the OAuth protocol. The problem of possible leaks of sensitive data is solved in a tricky way. Instead of sending sensitive data the notification daemon sends only an information about the event.
  - Ensuring authenticity of notifications received by subscribers

    Solved by the PubSubHubbub protocol.

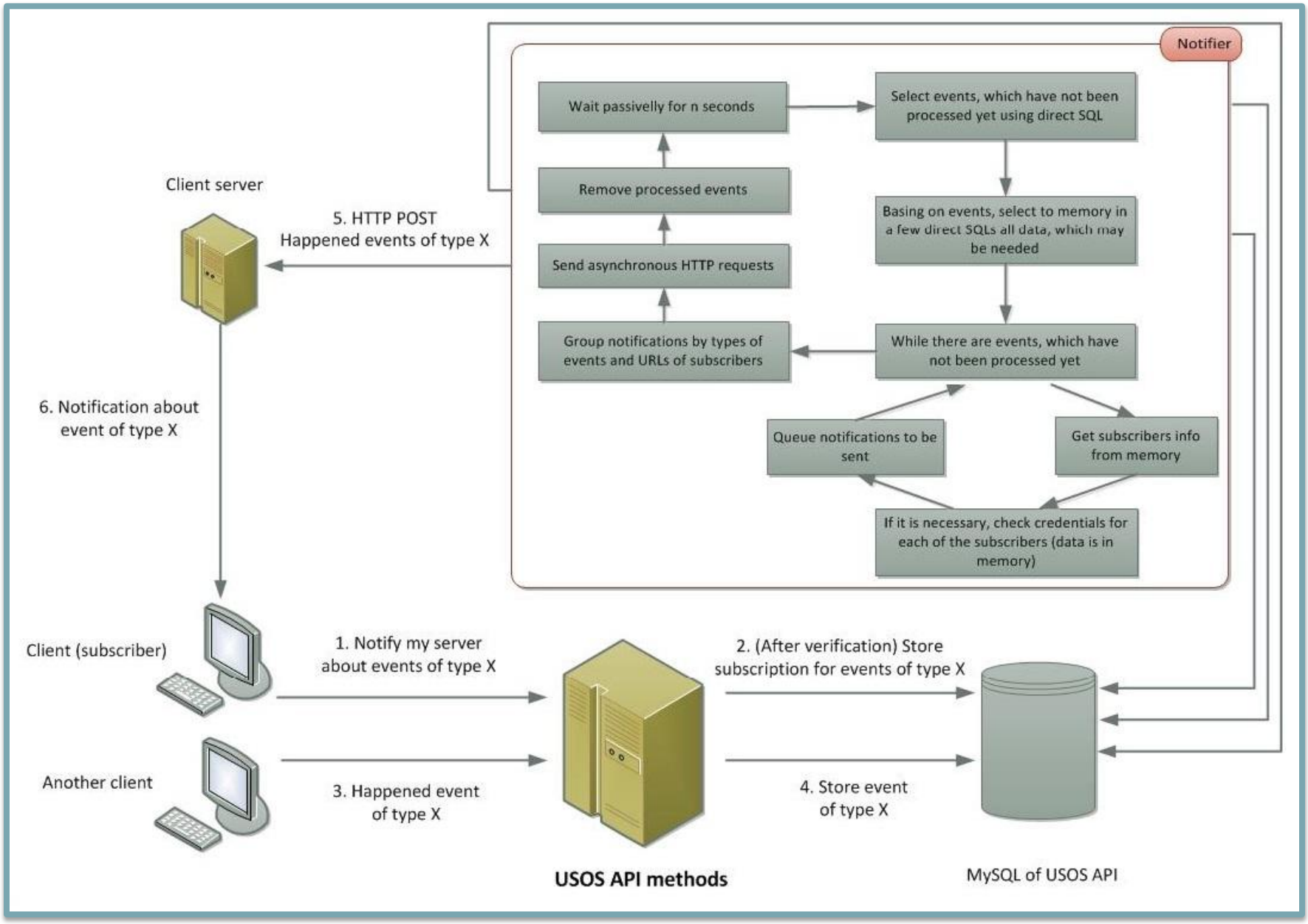
Tuning system performance

Test data sets	Trivial	Grouping	Caching	Direct SQL	Asynch. I/O	JSON	Extra tuning
3 clients, 10 users, 60 events	93.2	3.3	1.8	1.6	0.64	0.59	0.56
3 clients, 100 users, 600 events		19.6	3.7	2.5	1.59	0.99	0.72
3 clients, 1 000 users, 6 000 events		189.7	30.1	18.2	9.85	5.53	2.46
3 clients, 10 000 users, 60 000 events				180.8	92.6	51.1	17.62
3 clients, 40 000 users, 250 000 events						209.7	77

Test results for various data sets and implementation methods (times are given in seconds)

System architecture – summary

- A client makes subscriptions by invoking ***services/events/subscribe_event*** and provides a web server.
- Notifications are **HTTP requests** to that server.
- The system consists of new methods of USOS API for handling subscriptions and the **notification daemon** (which plays a role of the hub from the PubSubHubbub protocol).
- Subscriptions are established in the context of the client, i.e. one subscription may cover notifications on all data to which the client has access.
- Notification daemon wakes up periodically and processes events collected since last check.
- Sensitive data are not transmitted; the notification only signals the changes and includes information necessary to fetch them.
- Before sending notification about changes in a sensitive data of some user, permissions have to be checked; if the client has the OAuth access token for that user with sufficient privileges, the user can receive notification.
- HTTP requests with notifications have an additional header — X-Hub-Signature, which allows to verify the sender.
- Types of events are paths to methods returning a single record from the database; thanks to this, subscriber knows how to get the changed data.



System architecture with the algorithm performed by the notification daemon

Conclusions

- We designed a **system of notifications** being part of **USOS API**.
- It is based on the **PubSubHubbub** protocol, which allows to notify subscribers in real time about events originated from the student management information system.
- It uses **OAuth** protocol to ensure confidentiality of data access.
- Solution is not only **secure** and **user friendly**, but also **scales** well.
- New methods of USOS API and the notification daemon comprise a basic infrastructure necessary to develop **mobile applications** for interacting with the student management information system.
- These applications have yet to be designed and implemented (work in progress).

Acknowledgments

This paper is based on

the **Master thesis** of **Kamil Szarek**:

Event notifications system for USOSapi users

supervised by Janina Mincer-Daszkiewicz

Programming work was done by **Kamil**