ORCiD-Workshop: Überblick

Torsten Bronger
Zentralbibliothek, Forschungszentrum Jülich
t.bronger@fz-juelich.de

1. Ziele

- Möglichst viele Mitarbeiter, mindestens alle publizierenden, haben eine ORCiD.
- Die Zentralbibliothek kennt alle ORCiDs der Mitarbeiter.
- Die ORCiDs sind in unserer Personen- und Publikationsdatenbank hinterlegt.
- Die Nutzer können ihre Publikationen aus unserem System bequem an ihr ORCiD-Profil übermitteln.

2. Plan

- Mitglied bei ORCiD werden (z.B. über Konsortium). ORCiD-Mailingliste abonnieren.
- Zugang zur Sandbox holen.
- Programmieren gegen die Sandbox.
- Zugang zum ORCiD-Produktivsystem holen und mit dem eigenen Account nochmal testen.
- Massen-Email-Versender entwickeln und aktivieren, d.h. in regelmäßigen Abständen erinnern.

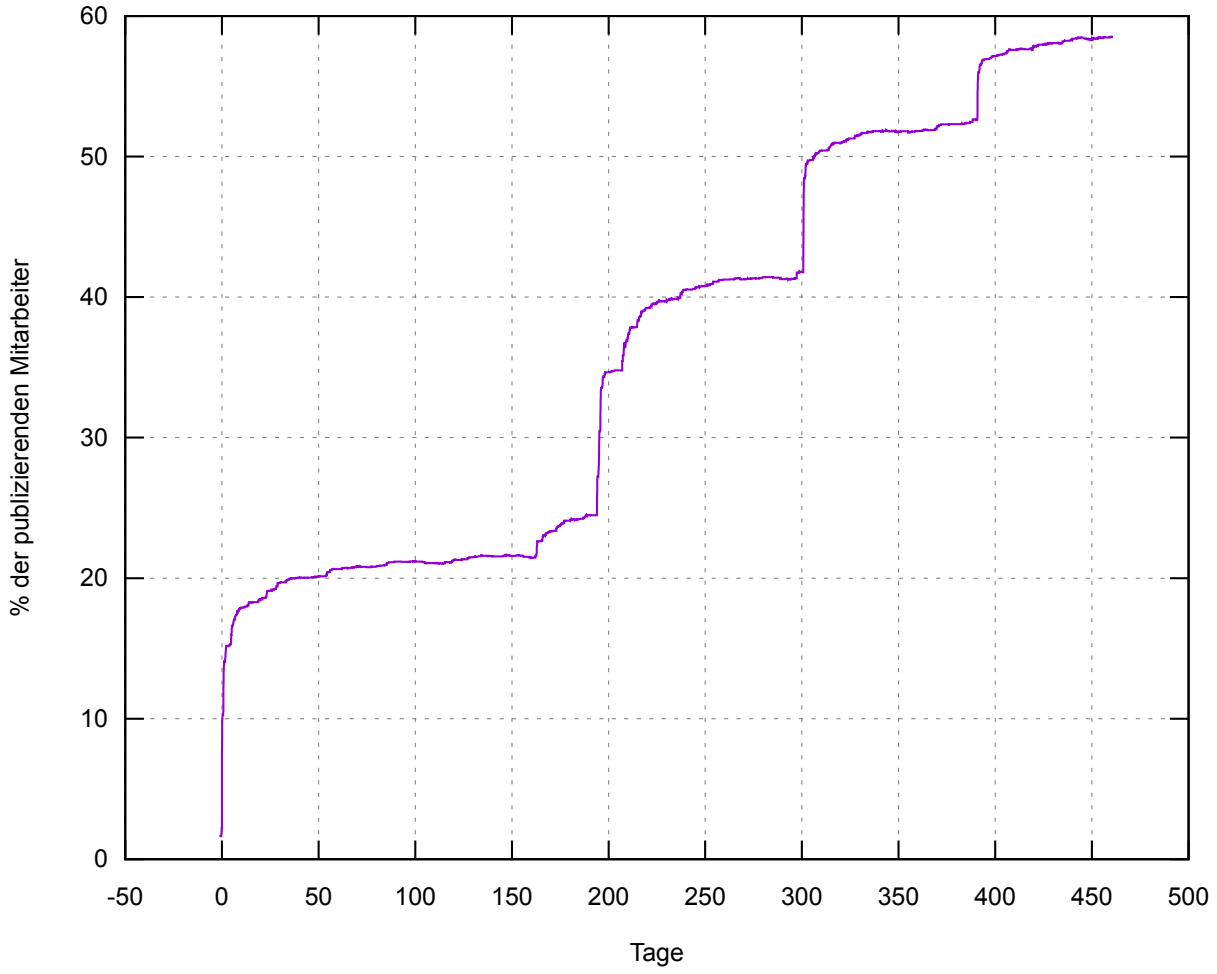
3. Reminder-Emails

- Fishing-Befürchtungen adressieren: Signieren, Kontaktdaten angeben, auf Intranet-Webseite mit dem Email-Text verweisen.
- Nicht alle Emails auf einmal verschicken.

4. ORCiDs sammeln

- Nutzer müssen selber ORCiD anlegen, wir helfen nur dabei.
- Adressaten für die Email gut wählen. Im FZ Jülich: Alle Mitarbeiter in der ersten Runde, dann nur die mit mindestens einer Publikation. Ausschließen von bereits vorhandenen ORCiDs.
- Externe Datenbanken in Betracht ziehen (Datenschutz klären).
- Erfolg messen. Im FZ Jülich: ORCiD-Quote zeitabhängig loggen.

5. ORCiD-Quote im FZ Jülich



6. Upload von Veröffentlichungen

- Nutzer bekommt Liste mit seinen Veröffentlichungen.
- Checkboxen sind per Default markiert, außer, wenn der Nutzer nur Herausgeber war.
- Vor dem Upload holen wir und alle Put-Codes von ORCiD → Unterscheidung POST/PUT möglich.
- Eigentlicher Upload geht in den Hintergrund, weil er zu lange dauert.
- Upload findet parallel, aber gedrosselt statt. Nur 4 Verbindungen gleichzeitig, 24 Publikationen pro Sekunde.
- Kann mittlerweile deutlich schneller und einfacher gemacht werden.

7. Nutzerwünsche zum Upload von Veröffentlichungen

- Schalter, um alle Checkboxen anzuschalten
- Schalter, um alle Checkboxen auszuschalten
- Möglichkeit, nur alle Publikationen seit X anzuzeigen
- Möglichkeit, für jemand anderes die Publikationen hochzuladen

8. Implementation bei uns

- Der ORCiD-Server ist ein eigener kleiner Webserver, für den der Apache eine Extra-Regel hat.
- Dadurch sehr einfach zu entwickeln, Wahl der Werkzeuge völlig frei.
- Allerdings auch eigener Monitor nötig, damit Admins benachrichtigt werden, falls der ORCiD-Server ausfällt.
- Außerdem Nachahmung des Stils der Haupt-Website nicht perfekt.

FZJ-Code: github.com/bronger/invenio1-orcid

9. Neuigkeiten in der ORCiD-Welt

- Bis zu 100 Veröffentlichungen können nun in einem Rutsch hochgeladen werden.
- „Institutional Sign-In“.
- „Institutional Connect“ (beta).

ORCiD-Workshop: Die ORCiD-API

Torsten Bronger
Zentralbibliothek, Forschungszentrum Jülich
t.bronger@fz-juelich.de

1. Die ORCiD-API

- Klassisch: HTTP, ReST, JSON (oder XML).
- Man muß v2.0 benutzen, alles andere wird bald abgeschaltet.
- Traffic drosseln – die ORCiD-Server sind nicht sehr potent. Maximal 24 Publikationen pro Sekunde, maximal vier Verbindungen gleichzeitig. Mehrere hundert Papers *während* des Requests hochzuladen geht also nur schlecht. Zumal man parallele Prozesse beachten muß.
- HTTP 5xx ist ganz gerne mal ein Client-Fehler.
- Die ORCiD-Mailingliste ist sehr hilfreich, aber die große Zeitverschiebung bei kritischen Schritten berücksichtigen.

2. Voraussetzungen

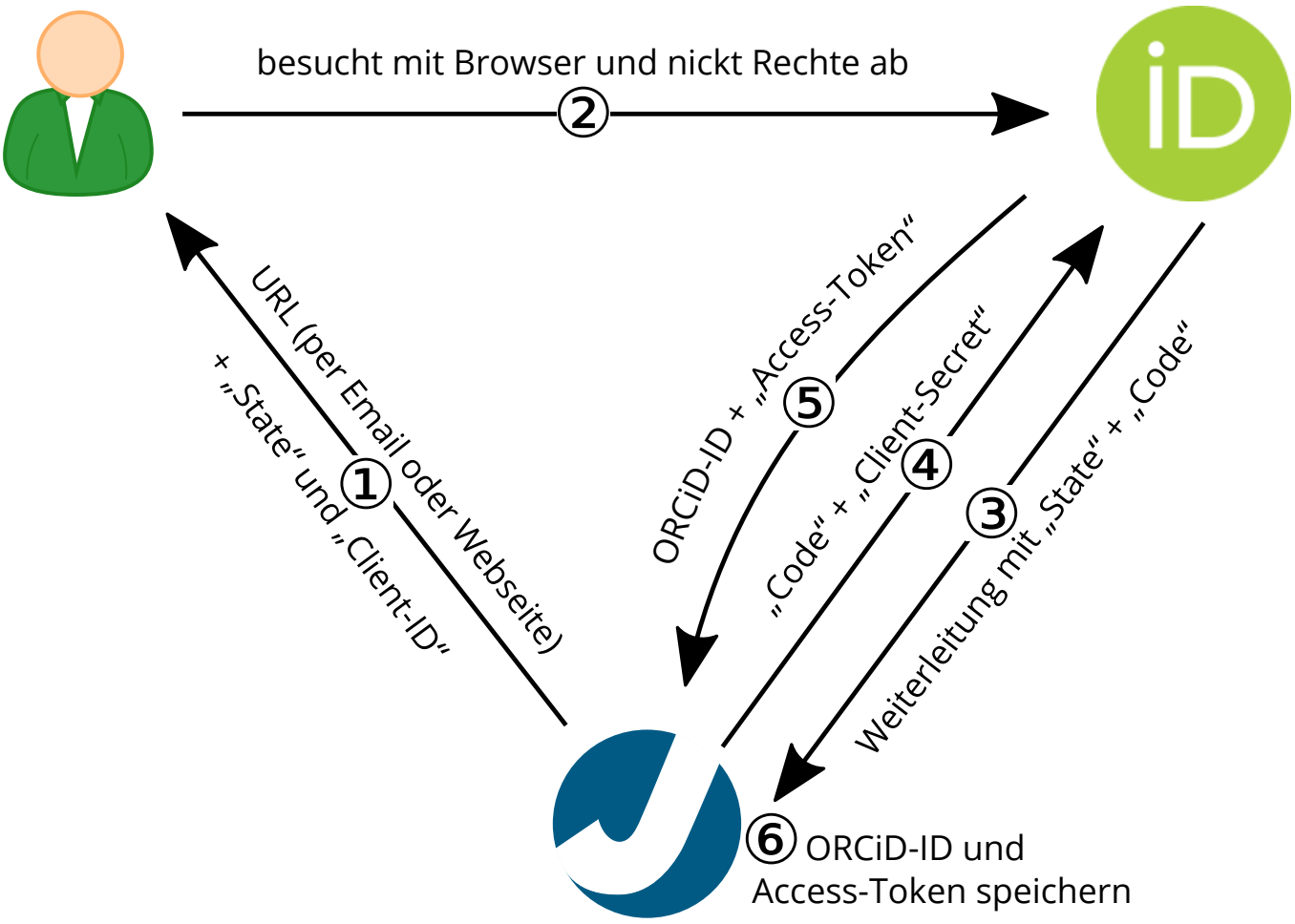
Man benötigt stets eine Client-ID und ein Client-Secret. Bekommt man von ORCID.

- Auch für die Sandbox wird das benötigt.
- Auch für die „Public API“ wird das benötigt.

Unsere Konfiguration:

```
ORCID_CLIENT_ID = APP-****************
ORCID_CLIENT_SECRET = ********-****-****-****-************
SECRET_KEY = ************************************************
ORGANIZATION_NAME_REGEX = (Forschungszentrum|research center|research centre)\\s+j(ü|u|ue)lich
ORCID_AFFILIATION = {\"name\": \"Forschungszentrum Jülich\", \
                     \"address\": {\"city\": \"Jülich\", \
                                   \"region\": \"Nordrhein-Westfalen\", \
                                   \"country\": \"DE\"}, \
                     \"disambiguated-organization\": \
                     {\"disambiguated-organization-identifier\": \"28334\", \
                      \"disambiguation-source\": \"RINGGOLD\"}}
```

3. OAuth 2



4. Anmerkungen zur Implementierung

„State“ ist bei uns das Tupel (Nutzer-ID, Nutzer-ID || SECRET, Redirect-URL).

Das Access-Token muß sicher gespeichert werden. Verschlüsselt, aber nicht als Hash. Bei uns: Mehrfach genutztes One-Time-Pad (*räusper*).

Nach erfolgreichem OAuth 2 findet Authorisierung bei jedem HTTPS-Request wie folgt statt:

```
Authorization type: Bearer
Access token: <Access-Token für diesen Nutzer>
```

Wenn es nicht um das Profil einen bestimmten Nutzers geht: Client credientials (`client_id` und `client_secret`) werden bei POST/PUT im Body übergeben, sonst nur `client_id` im Query String.

5. Webserver

`/orcid`
Bei „Code“:

1. ORCiD und Access-Token speichern
2. Redirect auf `/orcid`

ORCiD vorhanden: Hochladen anbieten
Sonst: Anbieten, sich mit ORCiD zu verbinden

`/orcid/submit`

1. Publikationsliste anzeigen
2. Benutzer Publikationen auswählen lassen
3. Hochladen zu ORCiD
4. Redirect auf `/orcid`

6. Demo

```
#!/usr/bin/python3

import requests, subprocess, json
```

```
response = requests.post(
    "https://orcid.org/oauth/token",
    data={"client_id": open("id").read(),
          "client_secret": open("secret").read(),
          "grant_type": "authorization_code",
          "code": subprocess.check_output(["ssh_zb", "cat", "/tmp/bronger_code"]).decode(),
          "redirect_uri": "https://juser.fz-juelich.de/orcid"
    })

result = json.loads(response.text)
print("\nORCiD: {}\nAccess-Token: {}".format(result.get("orcid"), result.get("access_token")))
```

7. Wichtige Endpunkte

- POST https://api.orcid.org/oauth/token: Um einen Authorization-Code in ein Access-Token umzutauschen.
- GET https://api.orcid.org/v2.0/<ORCiD-ID>/activities: Um alle „Aktivitäten“ eines Nutzers in einem Rutsch zu bekommen. Vor allem die Veröffentlichungen und die Affiliation.
- POST https://api.orcid.org/v2.0/<ORCiD-ID>/employment: Um eine Affiliation zu einem Nutzerprofil hinzuzufügen.

Dabei im HTTP-Header:

```
Accept: application/orcid+json
Content-Type: application/orcid+json
```

… oder halt XML, wenn einem das lieber ist.

8. Wichtige Endpunkte II

- POST https://api.orcid.org/v2.0/<ORCiD-ID>/work: Um eine Veröffentlichung zu einem Nutzerprofil hinzuzufügen.
- PUT https://api.orcid.org/v2.0/<ORCiD-ID>/work/<put-code>: Um die Veröffentlichung <put-code> des Nutzerprofils zu updaten.

9. Struktur einer Affiliation

```
{"name": "Forschungszentrum Jülich",
 "address": {"city": "Jülich",
             "region": "Nordrhein-Westfalen",
             "country": "DE"},
 "disambiguated-organization":
 {"disambiguated-organization-identifier": "28334",
  "disambiguation-source": "RINGGOLD"}}
```

Das ist dann die Nutzlast des POST-Requests.

Die Ringgold-Nummer nutzen wir (neben einer Regex auf den Namen), um zu detektieren, ob die Affiliation schon im Profil hinterlegt ist. (ORCiD-IDs für Affiliations wären hier natürlich praktisch, sind aber erst in Planung seitens ORCiD.)

10. Put-Codes

Put-Codes sind die nicht-geheime interne ID von Objekten in der ORCiD-Datenbank.

Zwei Möglichkeiten, an die Put-Codes zu kommen:

- Beim Anlegen von Dingen in ORCiD selber speichern. Der Put-Code wird beim Anlegen nämlich zurückgegeben.
- Sich die bei Bedarf von ORCiD holen. Für Veröffentlichungen geht das so:

1. Alle Veröffentlichungen eines Nutzers holen. 2. source-client-id.uri muß mit der eigenen URI übereinstimmen. 3. SOURCE_WORK_ID enthält dann die eigene interne ID der Veröffentlichung. 4. Der Put-Code steht im Feld „put-code“.

11. Python-API

- Python hat ein eigenes ORCiD-Binding: github.com/ORCID/python-orcid
- Sehr gute Code-Qualität, mit Tests abgesichert.
- Wird aktiv von CERN-Leuten gewartet.

Beispiel:

```
import orcid
api = orcid.MemberAPI(ORCID_CLIENT_ID, ORCID_CLIENT_SECRET, sandbox=False)
publications = api.read_record_member(orcid_id, "activities", token)["works"]
```

12. Original-Dokumentation der ORCiD-API

- Ist ein wenig unübersichtlich.
- ORCiD nutzt Swagger, d.h. man kann im Browser einfach api.orcid.org/v2.0/ ansurfen.

13. Fehler, die man abfangen sollte

- Access-Token-Holen klappt nicht (wir versuchen das bis zu sechs Mal).
- Jedesmal, wenn Sachen im ORCiD-Profil geändert wurden: Authorisierung könnte entzogen worden sein.
- HTTP 409 nach einem PUT auf eine Veröffentlichung bedeutet: Veröffentlichung ist da, aber auf „privat“ gesetzt.
- Grundsätzlich ist der Fehlertext im Body der Response sehr hilfreich.

ORCiD-Workshop: ORCiD-Massen-Emails

Torsten Bronger
Zentralbibliothek, Forschungszentrum Jülich
t.bronger@fz-juelich.de

1. Zweck

Mittels regelmäßiger Massen-Mails wollen wir erreichen, daß alle unsere Leute, die publizieren, eine ORCiD haben *und* wir diese ORCiD auch kennen.

Eigenschaften dieser Mail:

- automatisch generiert
- behelligt einen einzelnen Mitarbeiter maximal einmal alle drei Monate
- freundlich formuliert, Englisch und Deutsch, weist auf größte Fehler hin
- enthält einen personalisierten Link, mit dem man bei ORCiD sich eine ID holen kann, und wir die dann kriegen
- Anti-Fishing-Beweis: Digitale Signatur, Verweis auf Intranet-Seite

2. Auswahl der Empfänger

- ist aktiver Mitarbeiter
- hat noch keine ORCiD bei uns hinterlegt
- hat mindestens eine Publikation
- hat in den letzten drei Monaten keine Email von uns bekommen
- steht nicht auf der Whitelist

3. Versendevorgang

- Täglicher Cronjob, morgens.
- Drosselung auf eine Mail alle 10 Sekunden, um unseren ORCiD-Server zu entlasten.
- S/MIME-Signatur.
- Nutzt den SECRET_KEY, um den „State“ zu berechnen. Sonst keine Überschneidung mit dem ORCiD-Code.
- Aber: Natürlich viel Interaktion mit unserer Publikationsdatenbank.

FZJ-Code (fast): github.com/join2/join2/blob/master/lib/python/invenio/orcid/send_orcid_mails.py