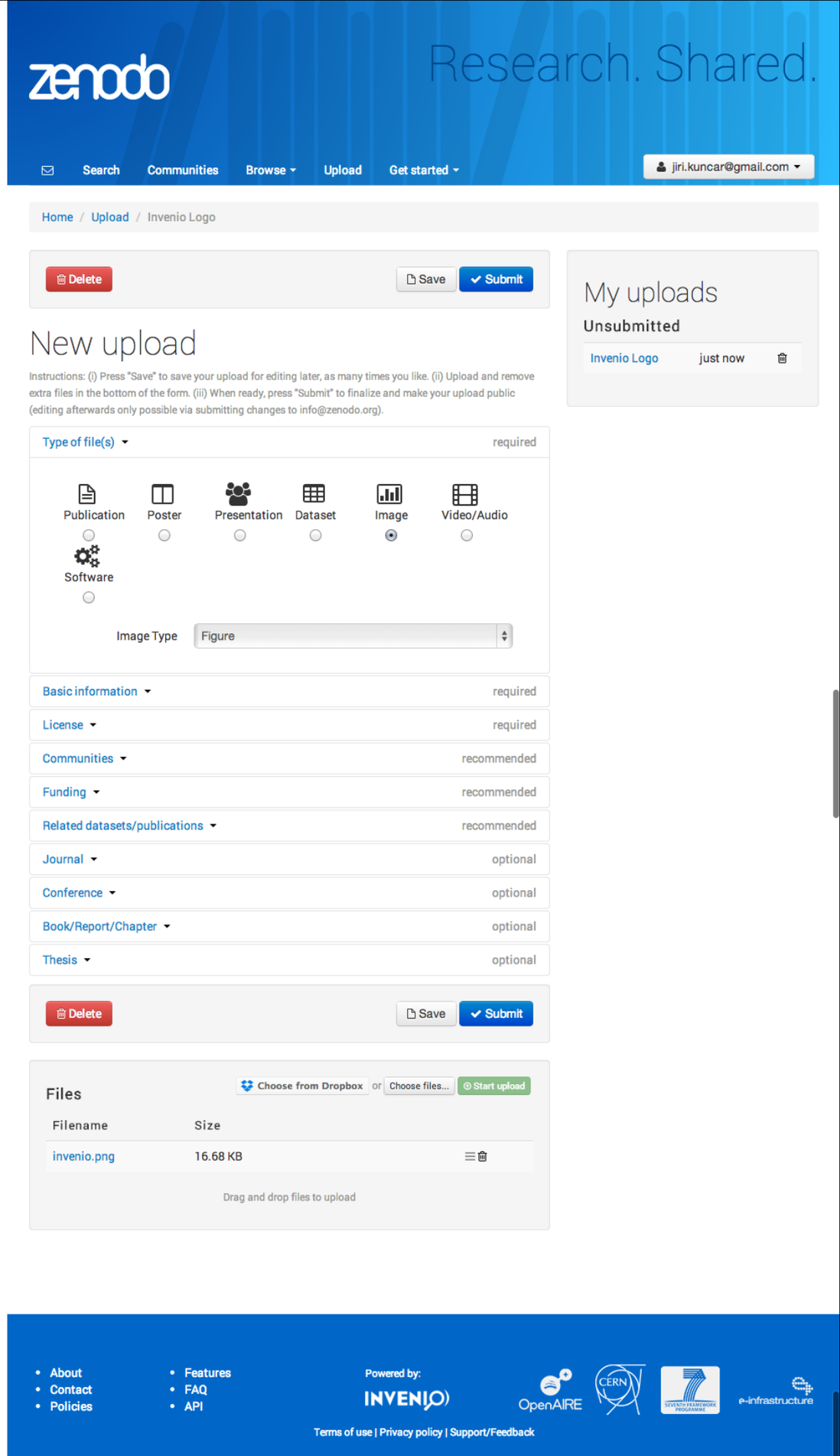
Depositions
in Invenio 2.0

Jiří Kunčar
jiri.kuncar@cern.ch
Lars Holm Nielsen
lars.holm.nielsen@cern.ch

Depositions

- New module to supersede WebSubmit
  - Workflows + Forms
- REST API
- Few assumptions:
  - Make easy things easy, and hard things possible.
  - Not connected to records.



WTForms

Celery

SQLAlchemy

Deposition technology

CERN

Features

- **Custom widgets**
  (upload type/CKEditor)
- **Auto-complete**
  (support for multi-field)
- **Form fields**
  (combine multiple forms into one to create structure and hierarchy, keywords, creators)
- **Actions**
  (pre-reserve DOI)
- **Tags like fields**
  (grants, communities)
- **Configurable file upload**
  (PLUpload with chunking)
- **Instant saving on server**
  (Pause deposition process and come back later)
- **Hide/show fields**
  (based on other input field, e.g. type of files)

INVENIO

CERN

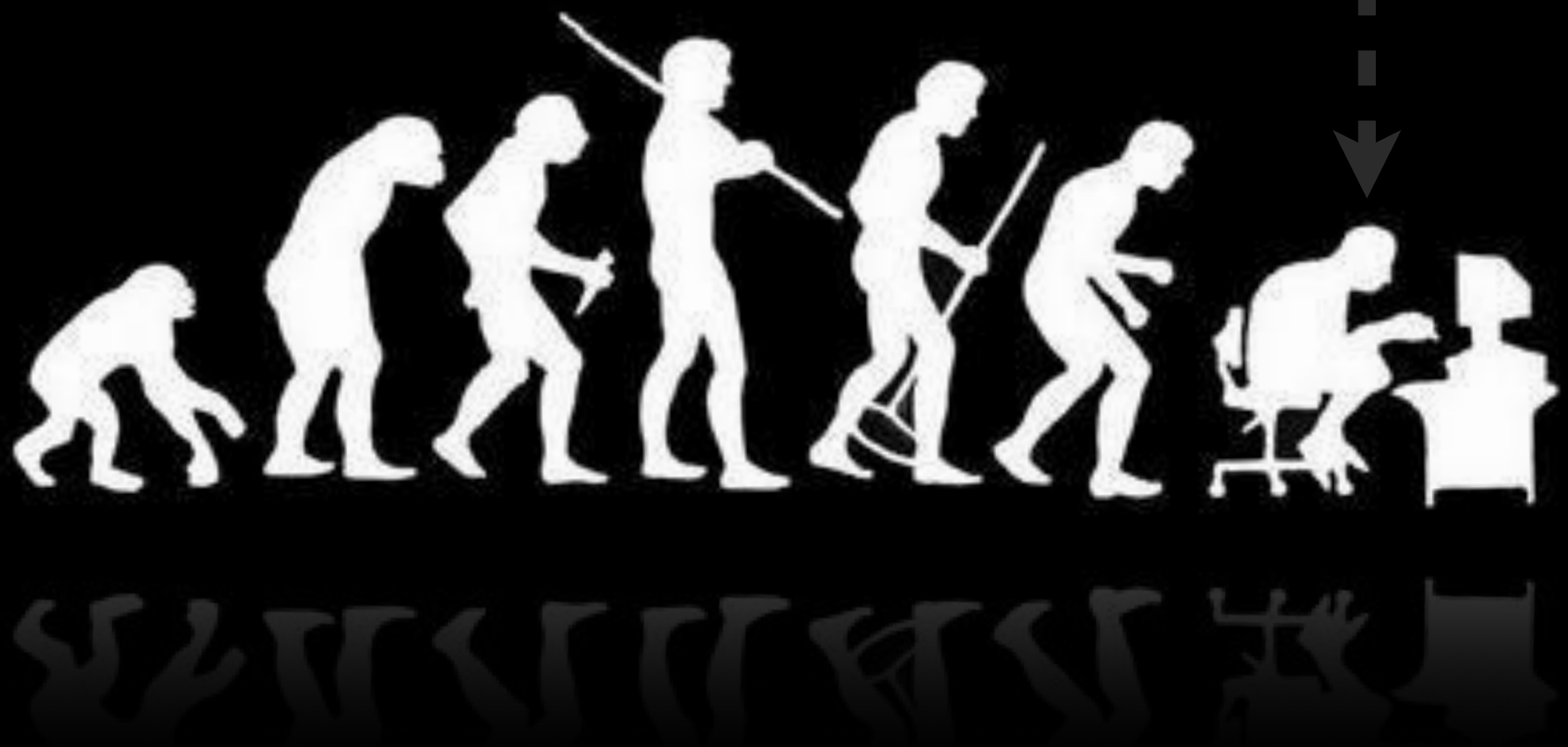
Workflow

List of files

Deposition

List of metadata objects

~~Record~~



Developer needed

Workflow

- **Deposition type**
  (e.g. article, preprint, image, ...)
- **ZENODO:**
  everything is one type
- **Simple case:**
  run task X1,..., Xn
  (branching/merging also supported)
- **Example**:
  1) Render form and wait for input
  2) Upload record
- **Basic tasks available**, but needs more (e.g. approval worflow)

```
class upload(DepositionType):
    workflow = [
        # Render the form and wait
        # until it is completed
        render_form(draft_id='_default'),
        # Create the submission information
        # package by merging data from all
        # drafts - i.e. generate the recjson.
        prepare_sip(),
        # Reserve a new record id
        create_recid(),
        # Post process generated recjson
        # according to needs in ZENODO
        process_sip(),
        # Generate MARC based on recjson
        # structure
        finalize_record_sip(),
        # Seal the SIP and write MARCXML file
        # and call bibupload on it
        upload_record_sip(),
        # Schedule background tasks.
        run_tasks(),
    ]
    name = "Upload"
    name_plural = "Uploads"
    api = True
    draft_definitions = {'_default': ZenodoForm}
```

Form

- A way to collect and validate metadata (deposition has a list of metadata drafts)
- Workflow decides to render form
- Filters, validation, auto complete, widget, processors
- Field enclosures
- **Extended WTForms Form** (to support AJAX saving etc.)

```
class ZenodoForm(WebDepositForm):
    title = fields.TextField(
        description='Required.',
        default="Untitled",
        placeholder="Start typing...",
        # Process input data (e.g. trim string)
        filters=[strip_string,],
        # Validation of data
        validators=[validators.required()],
        # Auto-complete anything
        autocomplete=fancy_auto_complete,
        # Specify custom widgets
        widget=widgets.MyFancyInput,
        # Use processors to update other fields
        # after validation
        # (e.g. fetch DOI information)
        processors=[]
    )
    # Field enclosures: list of
    # name/affliation dicts
    creators = fields.DynamicFieldList(
        fields.FormField(CreatorForm))


class CreatorForm(WebDepositForm):
    name = fields.TextField()
    affiliation = fields.TextField()
```

Task

- A unit of work
- Should be made in reusable manner
- Tasks can halt the workflow
- Can render anything they like

```
def render_form(draft_id='_default'):
    def _render_form(obj, eng):
        # Get an easy interface to
        # the workflow object
        d = Deposition(obj)

        # The metadata object:
        draft = d.get_or_create_draft(draft_id)

        if draft.is_completed():
            # Continue to next task
            # if the draft is already completed.
            eng.jumpCallForward(1)
        else:
            # Get the form
            form = draft.get_form(
                validate_draft=draft.validate)

            # Tell webdeposit what to render
            # (i.e. you can render anything
            # you like - or use the defaults)
            d.set_render_context(...)
            d.update()

            # Halt workflow in the current step
            eng.halt(
               'Wait for form submission.')
    return _render_form
```

REST API

INVENIO

- Same code used for validation and processing
- **Assumptions:**
  Workflow can run in headless mode
  (i.e. your workflow tasks are API-aware)

```
# List depositions
>>> r = requests.get("https://zenodo.org/api/deposit/depositions")
# Create new upload
>>> r = requests.post("https://zenodo.org/api/deposit/depositions?
apikey=YOUR_API_KEY", data="{}", headers={"Content-Type": "application/json"})
# Publish
>>> r = requests.post("https://zenodo.org/api/deposit/depositions/%d/action/
publish?apikey=YOUR_API_KEY" % deposition_id)
```

CERN

Key functionality delegated to **deposition type**, hence can be overwritten by you.

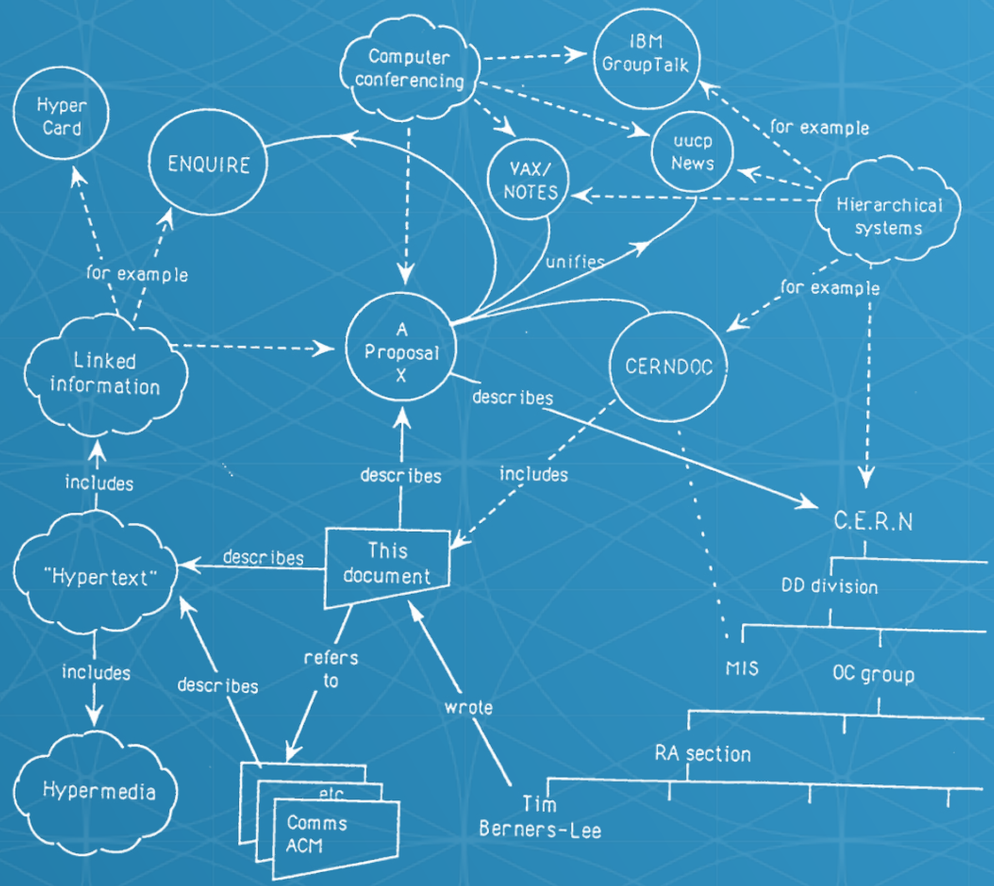
```
class upload(DepositionType):
    @classmethod
    def render_completed(cls, d):
        # how do I want to
        # render the completed upload
```

Extending

2000
2001
2002
2005
2006
2007
2008
2009
2010
2011
2012
2013

CERN

INVENIO



this is **next**...

Jiří Kunčar
jiri.kuncar@cern.ch
Lars Holm Nielsen
lars.holm.nielsen@cern.ch

CERN