EXPOSING

A FLEXIBLE, COMPOSABLE & EXTENSIBLE

REST API

Thierry Delprat

td@nuxeo.com

https://github.com/tiry/
AGENDA

• Quick introduction
  ▪ *provide some context*

• API design constraints & principles
  ▪ *explain the problem we want to solve*

• Building Nuxeo API
  ▪ *REST + Automation + Composition*

• Design consequences
  ▪ *price of flexibility*
SOME CONTEXT

What we Do and What Problems We Try to Solve
we provide a **Platform** that **developers** can use to
build **highly customized** Content Applications

we provide **components**, and the **tools** to assemble them

everything we do is **open source**

https://github.com/nuxeo

**various customers** - **various use cases**

- Track game builds
- Electronic Flight Bags
- Central repository for Models
- Food industry PLM

me: developer & CTO - joined the Nuxeo project 10+ years ago
Nuxeo Platform

Repository

SEARCH

CRUD

REST API

Nuxeo Repository

Content Model

Security

Versions

Life-Cycle

Blobs

Services

- Workflows, Conversions, Diff, Notifications, Activity ...
WHY API IS KEY FOR US

- Nuxeo Repository is a **backend**
  - Portals, Mobile Apps, ERP, CRM ...

- API is **UI**
  - for the developers
  - HTML5/JS
In Nuxeo architecture **everything is a plugin**
- **Nuxeo Server can provide a single service or 100's of services**

Everything is **configurable**
- **Logic and Data Structures depends on configuration**
API CHALLENGE

Expose a Platform: not an application

"One API"

but

Multiple combinations of services, plugins and Domain Models

developers using the platform want to expose the API of their Application
NEED TO FIND A SOLUTION

One Platform ➡ "One API"
REST API
DESIGN PRINCIPLES
what we want to have
BE EFFICIENT

- Avoid **round trips**
  - Get all needed data in one call
  - Resolve some data on the server side

- Avoid fetching **too much data**
• **Adapt** to the server side **configuration**
  - Domain model definition

• **Adapt** to client side **requirements**
  - Provide data for the screen mapping
  - Application can have **different flavors**
• Work between **transaction boundaries**
  - do all the work in one call

• Ensure **isolation**
  - Other users should not see inconsistent data

• Maintain **encapsulation**
  - Client should not make assertion on server implementation

*Client consumes a service, it does not build the service.*
BE EXTENSIBLE

- Expose *any meaningful business API*
  - Make API clean and application maintenance easy

- **Adapt API granularity** to the target Applications
  - one API behind each single button

*We can not build the target Business API: users/devs will do it*
BALANCE CLIENT/SERVER ROLES

"one-size-fits-all" does not work

→ Client driven

CLIENTS
Ask for the data they need
Use custom API

"open bar" seems too messy

→ Server controlled

SERVER
Manage the meta-model
Choose what API is exposed
(versioned software artifact)
• Chunked & Out of band upload

• Cachable and Seekable download
BE SENSIBLE

• Do not lose our soul
  ■ fight to keep the dynamicity of the platform!

• No REST integrism
  ■ Useful is more important than Beautiful

• Dogfooding is key
  ■ if this is not good enough internally, this is not good

• Building API is part of the development cycle
  ■ adding http API should never be a task for later
Actually, just the chronology has been adjusted!
BUILDING THE REST API

Exposing Resources
EXPOSING RESOURCES

Expose Use Cases !?

PLATFORM ➔ Target use cases are not defined

Expose the Domain Model !?

CONFIGURABLE ➔ Target Domain Model is unknown

Expose raw technical resources !
Exposing simple resources

- **Documents**
  - GET /repo/{repoId}/path/{docPath} HTTP 1.1
  - GET /repo/{repoId}/id/{docId} HTTP 1.1

- **Users & Groups**
  - GET /user/{userName} HTTP 1.1
  - GET /group/{groupName} HTTP 1.1
  - GET /directory/{directoryName}/{entryId} HTTP 1.1

- **Tasks & Workflows**
  - GET /workflowModel/{modelName} HTTP 1.1
  - GET /workflow/{workflowInstanceId} HTTP 1.1
  - GET /task/{taskId} HTTP 1.1
### EXPOSE SIMPLE RESOURCES

**path**: Access documents by their path

<table>
<thead>
<tr>
<th>Method</th>
<th>Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/path/{docPath}</td>
<td>Find a document by its path</td>
</tr>
<tr>
<td>PUT</td>
<td>/path/{docPath}</td>
<td>Updates a document by its path</td>
</tr>
<tr>
<td>DELETE</td>
<td>/path/{docPath}</td>
<td>Deletes a document by its path</td>
</tr>
<tr>
<td>POST</td>
<td>/path/{docPath}</td>
<td>Creates a document by its parent path</td>
</tr>
<tr>
<td>GET</td>
<td>/repo/{repold}/path/{docPath}</td>
<td>Find a document by its path</td>
</tr>
<tr>
<td>PUT</td>
<td>/repo/{repold}/path/{docPath}</td>
<td>Updates a document by its path</td>
</tr>
<tr>
<td>DELETE</td>
<td>/repo/{repold}/path/{docPath}</td>
<td>Deletes a document by its path</td>
</tr>
<tr>
<td>POST</td>
<td>/repo/{repold}/path/{docPath}</td>
<td>Creates a document by its parent path</td>
</tr>
</tbody>
</table>

**id**: Access documents by their id

<table>
<thead>
<tr>
<th>Method</th>
<th>Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/repo/{repold}/path/{docPath}</td>
<td>Find a document by its path</td>
</tr>
<tr>
<td>PUT</td>
<td>/repo/{repold}/path/{docPath}</td>
<td>Updates a document by its path</td>
</tr>
<tr>
<td>DELETE</td>
<td>/repo/{repold}/path/{docPath}</td>
<td>Deletes a document by its path</td>
</tr>
<tr>
<td>POST</td>
<td>/repo/{repold}/path/{docPath}</td>
<td>Creates a document by its parent path</td>
</tr>
</tbody>
</table>
EXPOSE SIMPLE RESOURCES
GET /nuxeo/api/v1/path/movies/star-wars HTTP/1.1

```json
{
  "entity-type": "document",
  "repository": "default",
  "uid": "5b352650-e49e-48cf-a4e3-bf97b518e7bf",
  "path": "/movies/star-wars",
  "type": "MovieCollection",
  "isCheckedOut": true,
  "title": "Star Wars",
  "facets": ["Folderish"
}
```

Server returns a minimal payload
Client need to control what **data schemas** are sent
ADAPTATIVE MARSHALING

- Control what **data schemas** are sent to the client

```plaintext
GET /nuxeo/api/v1/path/movies/star-wars HTTP/1.1
X-NXProperties dublincore, common

{
  "entity-type": "document",
  "repository": "default",
  "uid": "5b352650-e49e-48cf-a4e3-bf97b518e7bf",
  "path": "/movies/star-wars",
  "type": "MovieCollection",
  "isCheckedOut": true,
  "title": "Star Wars",
  "properties": {
    ...
    "common:icon": "/icons/movieCollection.png",
    "dc:description": "Star Wars collection",
    "dc:creator": "tiry",
    "dc:modified": "2015-10-22T02:12:59.07Z",
    "dc:lastContributor": "tiry",
    "dc:created": "2015-10-22T02:12:59.07Z",
    "dc:title": "Star Wars",
    ...
    "dc:contributors": ["tiry", "system" ]
  },
  "facets": [ "Folderish"
  ]
}
```
• Client may require more data
  - get Document **children** at the same time
  - get the **breadcrumb** data
  - get **thumbnail** or **preview** url
  - ...

• Client ask for the data
  - using **Headers**
  - using Query String **parameters**
Marshaling registry is pluggable

custom **Enrichers** can be contributed

"How the data is fetched" is a server side matter
FETCHING CONTEXTUAL DATA

GET /nuxeo/api/v1/path/movies/star-wars?enrichers.document=thumbnail HTTP/1.1

X-NXenrichers.document: thumbnail

```
{
  "entity-type": "document",
  "repository": "default",
  "uid": "5b352650-e49e-48cf-a4e3-bf97b518e7bf",
  "path": "/movies/star-wars",
  "type": "MovieCollection",
  "isCheckedOut": true,
  "title": "Star Wars",
  "contextParameters": {
    "thumbnail": {
      "url": "/nuxeo/nxthumb/default/5b352650-e49e-48cf-a4e3-bf97b518e7bf/thumb:thumbnail/Small_photo.jpg"
    }
  },
  "facets": [
    "Folderish"
  ]
}
```
• **Resolve** entity fields
  - *pointing to a label*
  - *pointing to an other Document*
  - *pointing to a User*
  - *

Implicit JOIN
• Use client side parameter to know what to resolve
  ▪ **header**
  ▪ **QueryString parameter**

```
fetch.objectType=fieldToFetch
translate.objectType=fieldToTranslate
depth=children
```

• Can be **recursive**
  ▪ **client need to control that too!**
RETRIEVE LINKED DATA

HTTP / JSON

JAX-RS

Marshaling

Resolvers

Publisher

+Java

Resources centric API

Repository

Plugin

User / Group

Directory

Task

Workflow

Workflow Service

Plugin

 UserManager

Plugin

Java Service API & Java Extensibility model
• Change the **return type**
  - get only **ACLs** or **History** info about the Document
  - get the **tasks** associated to document

```
GET /nuxeo/api/v1/path/movies/star-wars@acl HTTP/1.1
```

```
GET /nuxeo/api/v1/path/movies/star-wars@audit HTTP/1.1
```

• Use your own **business object**
  - use **business Adapters**
    ◦ wrap document or documents
    ◦ provide custom marshaling

```
GET /nuxeo/api/v1/path/movies/star-wars@bo/MyBusinessObject HTTP/1.1
```
GET /nuxeo/api/v1/path/movies/star-wars@bo/MovieCollection HTTP/1.1

{
  "entity-type": "MovieCollection",
  "id": "5b352650-e49e-48cf-a4e3-bf97b518e7bf",
  "title": "Star Wars",
  "episodes": 7
}
- Sent as links
  - Digest
  - CDN

- Uploaded out-of-band
  - chunking
  - reference in JSON
• **Upload EndPoint**

```plaintext
POST /api/v1/upload/{batchId}/{fileIdx} HTTP 1.1  
X-Upload-Chunk-Index 0  
X-Upload-Chunk-Count 5
```

• **Reference** Blobs from JSON Payload

```plaintext
PUT /nuxeo/api/v1/path/movies/star-wars HTTP/1.1

{"entity-type": "document",  
"properties": {  
  {  
    "file:content": {  
      "upload-batch": "0b0061d48f69b072",  
      "upload-fileId": 0,  
      "type": "blob"  
    }  
  }  
}
```
ARE WE HAPPY WITH THAT?

- **Efficiency**
  - *we can get all data in one call*

- **Flexibility**
  - *we can configure the data we want*

- **Extensibility: partial**
  - *enrichers, resolvers & adapters are not always enough*

- **Coverage: poor**
  - *100+ services and only 5 endpoints*
  - *not everything is CRUD*
NEED A WAY TO MAP 100+ SERVICES

Without creating 100 endpoints!

Need an other paradigm!
AUTOMATION API

Exposing service API over HTTP
• **Build a coarse gained API** on top of service Java API
  ▪ select simple **Commands**

• **Shell like** commands!
  ▪ *each service can contribute*

> commandA(p1,p2) | commandB(p3,p4)
COMMAND SYNOPSIS

Command

Parameters

INPUT
(Doc, Blob, User ...)

Context
(User, Doc ...)

OUTPUT
(Doc, Blob, User ...)

(nuxeo)
Commands as REST resources

- **GET** to retrieve definition
- **POST** to execute
GET /nuxeo/api/v1/automation/Document.PageProvider HTTP/1.1

HTTP/1.1 200 OK
Content-Type: application/json

{
    "id":"Document.PageProvider",
    "label":"PageProvider",
    "description":"Perform a query ...",
    "signature": ["void", "documents"],
    "params": [ 
        {"name":"page",
        "type":"integer",
        "required":false
        },
        {"name":"query",
        "type":"string",
        "required":false,
        },
        ...
    ]
}
Operation *Repository.PageProvider* (PageProvider)

**Description**
Perform a query or a named provider query on the repository. Result is paginated. The query result will become the input for the next operation. If no query or provider name is given, a query returning all the documents that the user has access to will be executed.

**General Information**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>currentPageIndex</td>
<td></td>
<td>integer</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>documentLinkId</td>
<td></td>
<td>string</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>language</td>
<td></td>
<td>string</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>maxResults</td>
<td></td>
<td>string</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>namedParameters</td>
<td>Named parameters to pass to the page provider to fill in query variables.</td>
<td>properties</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>page</td>
<td></td>
<td>string</td>
<td>no</td>
<td>NXQL</td>
</tr>
<tr>
<td>pageSize</td>
<td></td>
<td>integer</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>providerName</td>
<td></td>
<td>string</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>query</td>
<td></td>
<td>string</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>queryParams</td>
<td></td>
<td>stringlist</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>sortBy</td>
<td>Sort by properties (separated by comma)</td>
<td>string</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>sortInfo</td>
<td></td>
<td>stringlist</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>sortOrder</td>
<td>Sort order, ASC or DESC</td>
<td>string</td>
<td>no</td>
<td>ASC, DESC</td>
</tr>
</tbody>
</table>

**Signature**
- **Inputs**: void
- **Outputs**: documents
RUN AN OPERATION

POST /nuxeo/api/v1/automation/Document.PageProvider HTTP/1.1
Content-Type: application/json+nxrequest
{
  "params":
   {
     "query": "select * from Note",
     "page": 0,
   }
}

HTTP/1.1 200 OK
Content-Type: application/json
{
  "entity-type": "documents",
  "pageIndex": 0,
  "pageSize": 2,
  "pageCount": 2,
  "entries": [
    {
      "entity-type": "document",
      "repository": "default",
      "uid": "3f76a415-ad73-4522-9450-d12af25b7fb4",
      ...
    },
    ...
  ]
}
• Share the **marshaling layer** and **extension**
  - *Enrichers, Resolvers are available too*

• **Compose** Resources and Automation API
  - *Pipe Resources as input for Automation Operation*

```bash
> cat /doc/path/somedoc | command(p3,p4)
```
POST /nuxeo/api/v1/path/somePath/@op/Blob.ToPDF HTTP/1.1

HTTP/1.1 200 OK
Content-Type: application/pdf
...

RESOURCES > AUTOMATION
ARE WE HAPPY WITH THAT?

- Efficiency
- Flexibility
- Coverage
  - *all services and plugins can contribute*
- Extensibility
  - *good, but limited to Java Developers*
- Consistency
  - *still no way to align Application Transactions*
MORE COMPOSITION

assemble API blocks without having to code

build business API
COMPOSABLE API

Expose the API that matches client needs
• Tailor the API to match application requirements
  - one API behind every action / button

• Allow business analysts or UI developers to tailor the API
  - define what API is exposed
    ○ UI & Workflow needs
AUTOMATION CHAIN

- Assemble operations in a chain
- Pipe Output / Input
- Give it a name
- Call and execute within a single transaction

One Transaction
One Context
Server side assembly
ASSEMBLING CHAINS

**Documentation**

- Conversion
- Document
- Execution Context
- Execution Flow
- Fetch
- Files
- Local Configuration
- Notification
- Push & Pop
- Scripting
- Services
- User Interface
- Users & Groups
- Workflow Context

**Blob.RunConverter**
- Add

**Concatenate PDFs**
- Add

**Convert To PDF**
- Add

**Convert to given mime-type**
- Add

**Render Document**
- Add

**Render Document Feed**
- Add

The context document represents either the current document in the user interface or the target document of a repository event.

**Fetch > Context Document(s)**
- Edit

**Document > Update Property**
- XPath: dc:description
- save: true
- value: Changed

**Services > Tag Document**
- tags: Changed

**Document > Follow Life Cycle Transition**
- Value: validate

**Document > Snapshot Version**
- increment: Minor
- saveDocument: true

**Conversion > Convert To PDF**
- Edit

Drop the next operation.

See Online Help
IT DOES WORK!

- Business users & Front end developers leverage this:
  - to expose custom API for their UI
  - to build custom logic inside their Workflows
  - to add automatic processing (listeners)

- Actually it works almost too well:
  - users do awfully complicated things
  - chains calling chains calling chains ...
HAVE WE CREATED A MONSTER?
GO FURTHER THAN THE CHAIN MODEL

USE NASHORN TO ASSEMBLE OPERATIONS

- Operations remain the building blocks
- JavaScript is the glue code to assemble them

better control of the flow
function run(input, params) {

    var docs = Seam.getSelectedDocuments(input, {});
    /* Description: Fetch the documents selected in the current folder listing */

    if(docs.length>3){
        var index;
        for(index=0;index<docs.length;++index){
            Document.setProperty(input, {
                /*required:true - type: string*/
                'xpath': "dc:title",
                /*required:false - type: boolean*/
                'save': true,
                /*required:false - type: Serializable*/
                'value': "test"
            });
            WebUI.Refresh(input, {});
        }
    } else{
        WebUI.addMessage(input, {
            /*required:true - type: string*/
            'message': "DISPLAY IT",
            /*required:true - type: string*/
            'severity': "WARN"
        });
    }
}
ARE WE HAPPY WITH THAT?

- Efficiency
- Flexibility
- Coverage
- Extensibility
- Consistency
DYNAMIC AND COMPOSABLE API

What about the trade-off?
DYNAMIC API COMES WITH A PRICE

- **Documentation** is a challenge
  - maintain up to date and exhaustive

- **Clients** needs to deal with the dynamic aspect
  - data mapping
  - discover APIs

- **Debugging** API calls can be challenging

  *Introspection is a requirement*
**INTROSPECTION**

- **Trace** feature
  - *Allow client to retrieve traces of nested executions (dev mode)*

- **Introspect** the Command API
  - *GET retrieve definition + doc*

- Add REST API to **introspect configuration resources**
  - *Document types, Schemas*
  - *(Widgets & Forms)*

  ![Provide a Playground to test](#)

  *connect to a remote server*

  *introspect server API and structures*
Introspect Data Structures

http://nuxeo.github.io/api-playground/
Introspect Resources

HTTP://NUXEO.GITHUB.IO/API-PLAYGROUND/

GET /api/v1/path/{docPath}

Find a document by its path

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>docPath</td>
<td></td>
<td>Path of the document, ex: 'default-domain'</td>
<td>string</td>
</tr>
</tbody>
</table>

Click on Run for executing the request. You can also click on the "settings" icon for adding custom headers to your request. Then click on the "CURL request" tab for seeing the details of the request.
Introspect Command API
http://nuxeo.github.io/api-playground/

POST /api/automation/Document.Copy

Copy the input document into the given folder. The name parameter will be used as the copy name otherwise if not specified the original name will be preserved. The target folder can be specified as an absolute or relative path (relative to the input document) as an UID or by using an EL expression. Return the newly created document (the copy).

Input

document

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td></td>
</tr>
</tbody>
</table>
INTROSPECTION IN ACTION

Using introspection from Mule ESB DataSense
CLIENTS LIBS

Dynamic API → more complex to use !?

client libraries
NEXT CHALLENGES
UI INTROSPECTION

• **Introspect UI** components tree
  - *gather needed data*

• **Auto-configure** Nuxeo Calls
  - *schemas, enrichers, resolvers ...*

• **Post back** aggregated data
  - *reverse enrichers and resolvers ?*
SOME LINKS

http://nuxeo.github.io/api-playground/

https://university.nuxeo.io/

https://doc.nuxeo.com/display/NXDOC/REST+API

https://github.com/nuxeo

nuxeo-workshop-restapi
ANY QUESTIONS?

Thank You!

http://www.nuxeo.com/careers/