Digital Supply Chain Standardized Data Exchange

July 17, 2019

movie labs

API Overview

Minutes

This was not in the original presentation

Notes within presentation are in red

Minutes

- The group discussed the topics captured in the attached presentation (slide 6 on)
  - Embedded in the presentation are notes taken during the meeting (in red)
- Changes will be incorporated into documentation at www.movielabs.com/md/api
- DSCA meetings will be scheduled at approximately 4-week intervals starting in approximately 4 weeks
- We will try to compile list of attendees
- Following slides contain decisions, action items and implementation approach

Decisions and Action Items

- **Decision: Topics in spec that are not specifically listed here are accepted as-is**
  - For example, OAuth approach is acceptable as documented
  - Decisions are subject to revision as people get into implementation
- **Action Item**: Investigate use of *HTTP 301 Moved Permanently* when an ALID changes.
  - Note that this helps with reconciliation when an ID has changed on the platform side, but does not address the studio changing IDs (which is the actual question at hand)
- **Decision**: TLS: TLS 1.2 required. TLS 1.3 allowed.
- **Decision**: Base URL: URL will *not* include studio as part of the path
  - Note: Access will be controlled via authorization mechanism
  - Note: This will require that ALIDs be globally unique to avoid collisions
- **Action Item**: How do we URL paths for special operations? Should we use ‘getcount’ or just ‘count’?
- **Decision**: Range GET need not be supported
- **Decision**: “Bulk Avails” (some form of Master Avail) required initially. However, this might only be used for testing where volumes are small.
- **Action Item**: Need to establish precise feeds
  - Three categories might be too much (based on polling intervals)
  - Might need to separate feeds based on data objects – more specific than general categories such as “Avails”
- **Non-Decision**: Still not clear resolution on JSON vs. XML
  - Editor feels there is substantial momentum behind XML based on current adoption of MDDF specifications in XML.
  - SPE is ready to go with XML as soon as anyone is ready to receive it
  - It was noted that it’s easier to find engineers and convince engineers to proceed if in JSON

Implementation Planning

- Minimum Viable Product (MVP)
  - Start with Avail plus status
    - Note that MEC was rejected, although editor still things that MEC would be much more practical (all studios have experience with MEC via UV or MA), and several platforms accept MEC. This will depend on who the partners are.
  - Full authentication (no stubs or shortcuts).
- Integration and Testing
  - Integration should be incremental with well-defined stages
  - “Stubs” should be part of the integration plan.
    - A stub might be as simple as picking a file from a directory, sending it through the API to a peer who receives it (via API) and writes the file to a directory.
  - Whether or not we have Bilateral testing or Virtual Plugfests will depend on the participants (TBD)
  - No decision on shared resources such as sample code
- Communications will be via email (for now)
  - Use existing list
- Meetings to be scheduled every 4 weeks, starting in about 4 weeks
  - Next meeting will review spec with updates based on decisions in this meeting

Digital Supply Chain Standardized Data Exchange

July 17, 2019

movie labs

API Overview

Agenda

- Meeting Introductions
- Goals for today
- API Project Introduction
- Technical Approach Overview
- Decisions to review, spec updates
  - General Approach
  - HTTP, TLS, Endpoints
  - Atom
  - XML vs. JSON
  - Authentication and Authorization
- Implementation Planning

Part 1: Context

Introduction

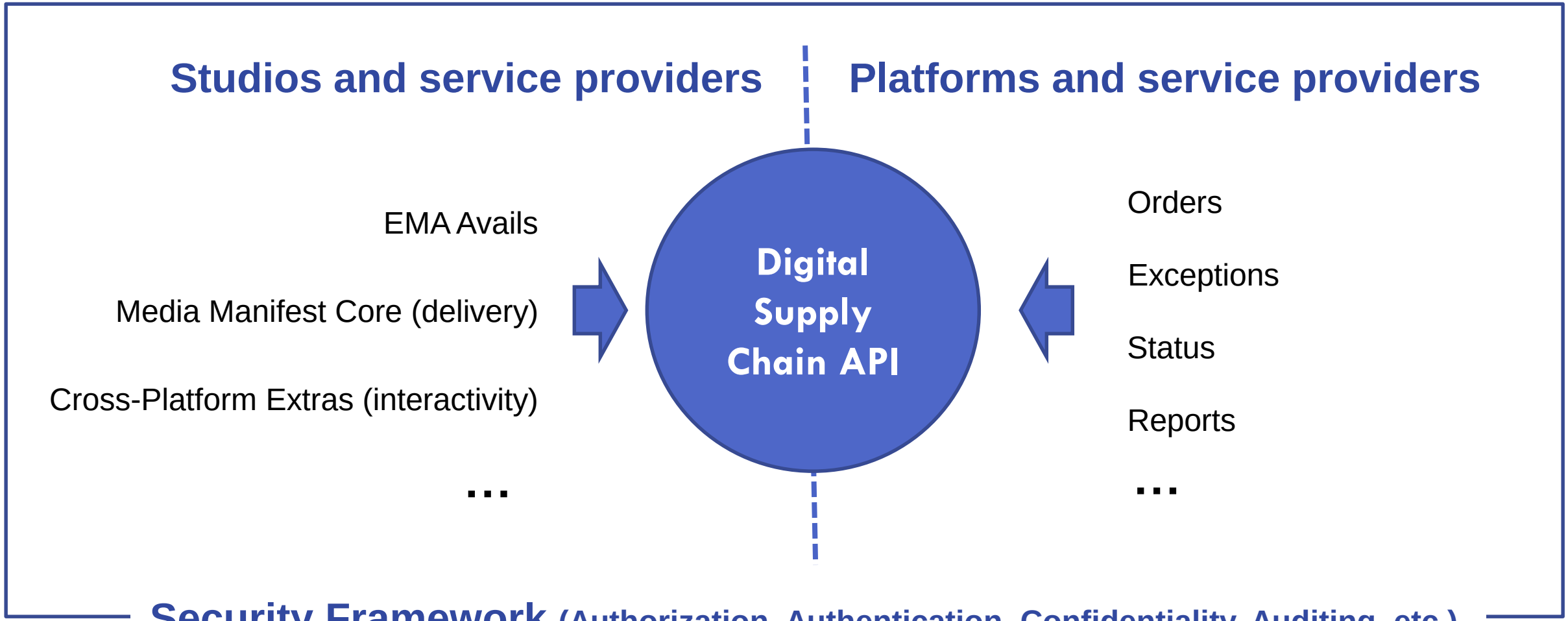
Motivation

- Platforms and Studios looking for opportunities to streamline operations and more tightly integrate supply chain
- Standards reaching critical mass of adoption
- Platform-specific portals can help studios self-manage content, but does not support automation
- Platform-specific APIs allows automation, but are non-standard and therefore non-scalable
- A standardized API is essential to maximize automation:
  - Reduce cost, reduce errors, reduce time to market, improve visibility, increase quality, and support business models that increase revenue

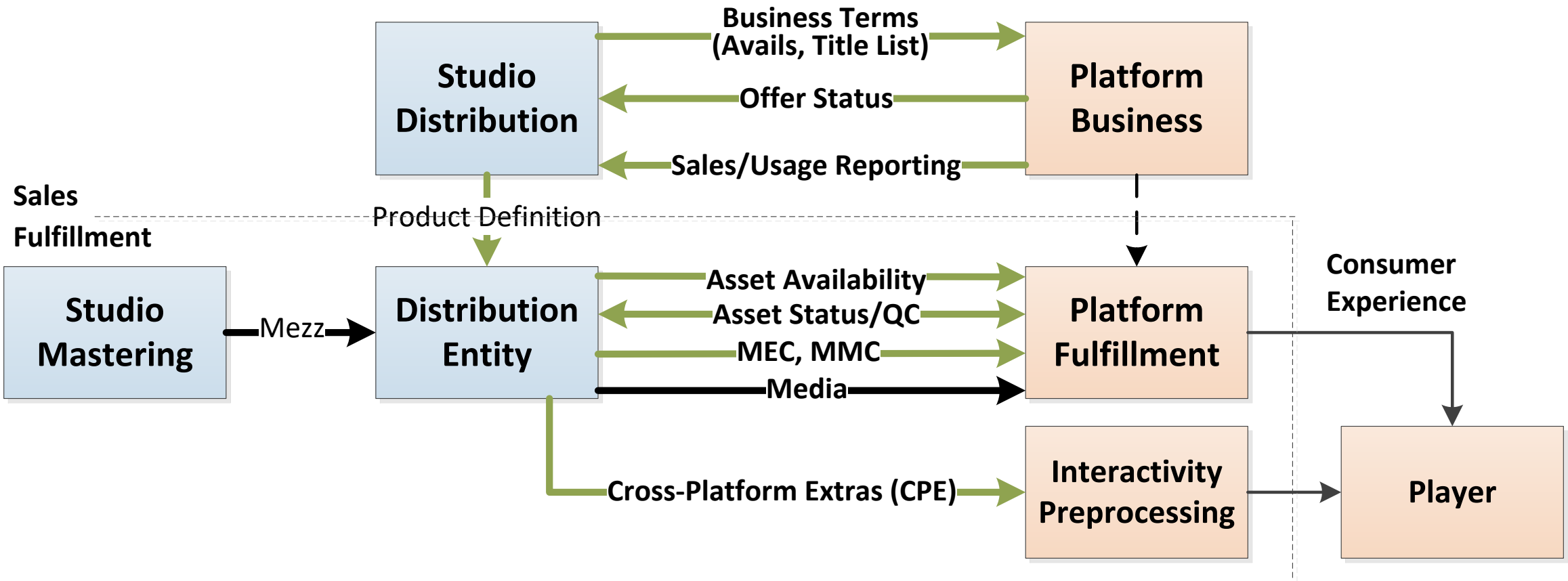
Challenges to Address

- Following considered in spec design
  - Uniformity
  - Security Policies
  - Platform Challenges
  - Studio Challenges
- ⇨ Continue to validate against platform and studio requirements

API Concept


Studios and service providers
Platforms and service providers
EMA Avails
Media Manifest Core (delivery)
Cross-Platform Extras (interactivity)
...
Digital Supply Chain API
Orders
Exceptions
Status
Reports
...
Security Framework (Authorization, Authentication, Confidentiality, Auditing, etc.)

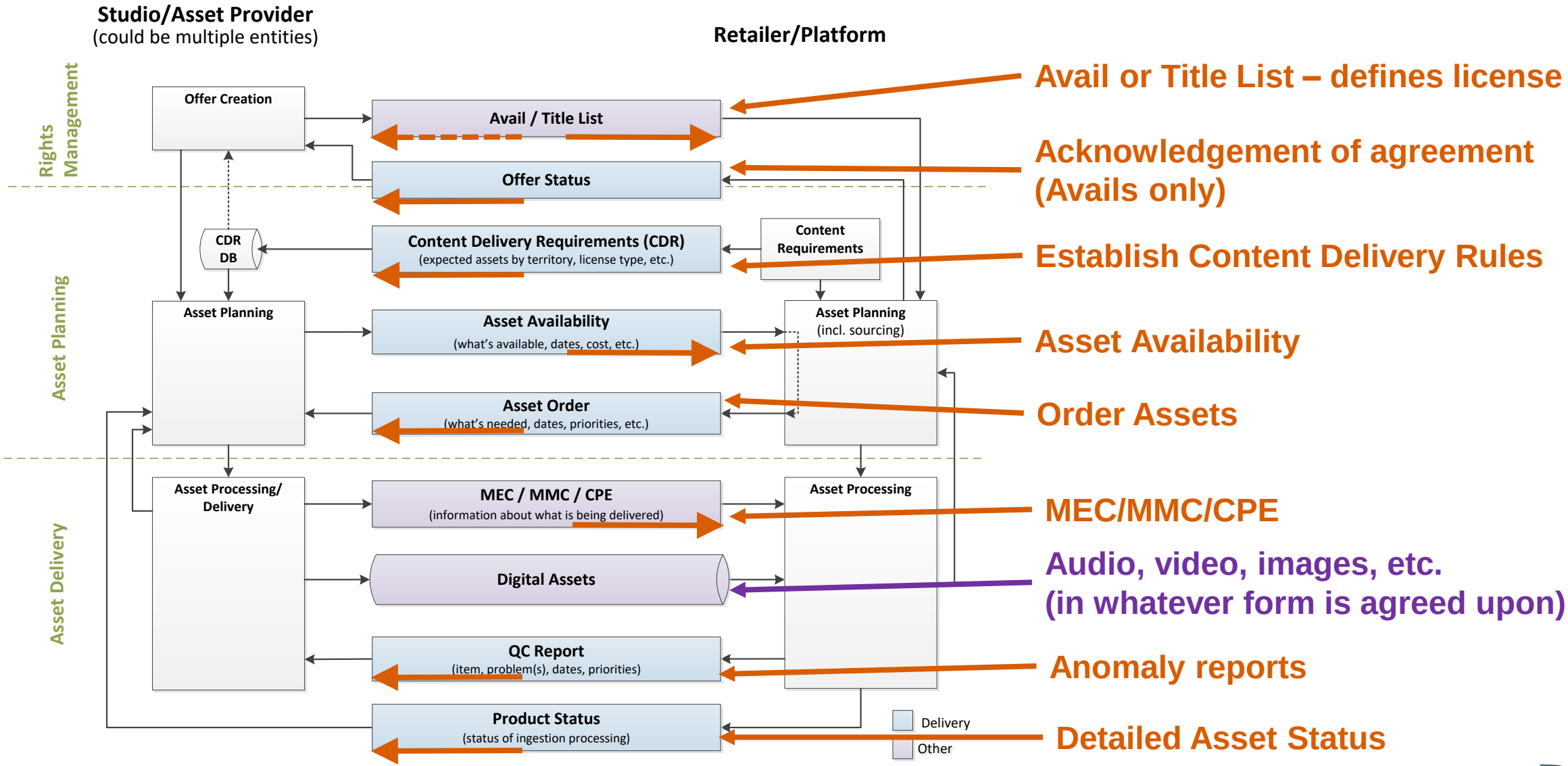
MovieLabs Digital Distribution Framework (MDDF)



Representative Use Cases

- Avails submission
- Content ordering
- Fulfillment status
- QC feedback
- Storefront status (Avails Status)
- Data Asset delivery/asset updates (Asset Ordering and Delivery)
- Manifest updates
- POS/revenue reporting
- Anything else we come up with

Model represents multiple *superimposed* workflows



Process

- ✓ Identify use cases for systematic data exchange
- ✓ Define message exchange framework
- ✓ Find first movers
- ⇨ Obtain consensus and establish a standard before fragmentation occurs
- Describe benefits to ecosystem partners to encourage adoption

Tech. Approach

Summary of Approach

- API model: RESTful
  - Studio to Platform (e.g., avail, metadata)
  - Platform to Studio (e.g., asset order, financial reporting)
  - Does POST return 301, redirecting to the resource with status.
  - Rejected: other approaches, including SOAP
  - ~~Consider GraphQL?~~
- Status: Atom/RSS
  - Offer processing status (both studios and platforms)
  - Atom meets requirements for our workflows
  - Rejected
    - RESTful polling (highly inefficient for this application)
    - Reverse channel – too much overhead setting up reverse channel. Not the right model
    - Notification – all solutions are proprietary
- Authorization: OAuth2

Key protocols – Foundation of API

- HTTP
  - Data communication
  - Basis for RESTful interface and Atom
- TLS – Prevents outsiders from seeing or altering data as it transits between parties
- OAuth2
  - Ensures only authorized parties access data
  - Provides means for 3rd parties (service parties) to act on behalf of studios or retailers
- Atom
  - Standard means of publishing ‘news’ about updates

Specification Summary

- Uses industry standards
  - Similar to APIs offered by major internet company
- Key document sections
  - HTTP, TLS
  - Authentication and Authorization
    - Studio, Retailer and Service Provider Authentication (Roles change depending on use case)
    - OAuth2 Authorization
  - Endpoints
  - RESTful Web Service
  - Atom
    - Feeds provide prioritized information about updates
  - Use Cases

http://www.movielabs.com/md/api/

Support for Use Cases

- Need to choose initial use cases; for example:
- Avails
  - XML Avails is basis for data exchange
  - Studios post or retrieve AvailsList objects to Retailers
  - Posting a new object is equivalent to a Full Extract
  - API supports Master Avails, but assumption is that only changes will be posted
    - Earlier decision was to defer ‘bulk update’ which is needed for Master Avails
  - Status mechanism provided to inform studio when changes occur (either normal processing or exceptions that require attention)
    - Details needs to be worked out on this
- Media Entertainment Core (MEC)
  - Based on MEC Core Metadata object
  - Same mechanisms as Avails
- Other use case-specific details can be provided, but intention is to focus on Avails and MEC first

Why you should be an early implementer

- General principle:
  - A spec is not *complete* until there is at least one implementation
  - A spec is not *interoperable* until there are at least two implementations
- As parties move closer to implementation, refinements will be needed
- In practice, early implementers have a large voice in specification refinement
- We need parties to sign up for implementation

Next Steps

- Specification
  - Complete specification
  - Develop support material
    - Best Practices
  - Reference implementations?
- Use cases/MVP
  - Detail *specific* use cases (Avails? MEC? Other?)
  - Priority and timing for other use cases
- Start implementing
  - No spec survives first contact with implementers

Part 2: Technical Discussion

Decisions

- General Approach
- HTTP, TLS, Endpoints
- Atom
- XML vs. JSON
- Authentication and Authorization

movie labs

Highlights of Approach

- Information flows to retailers in a RESTful interface
  - “Pragmatic REST*”
- Status is returned in same RESTful interface
  - However, because status rarely changes, polling every object (even with caching) is very inefficient
  - So, retailers publish Atom feeds which instruct studios which objects (resources) to GET
    - Uses Atom Syndication Format and some aspect of Atom Publishing Protocol (AtomPub)
- Security provided by TLS 1.2 and Oauth 2
  - Recommend TLS 1.3 ASAP once available
  - Minimum 1.2

*Thanks to Brian Mulloy of apigee for some useful guidance

HTTP

All connections

- TLS 1.2 (RFC 5246), *TLS 1.3?*
  - Via HTTPS endpoint
  - Servers should use TLS Cert from well known CA
  - Clients should ensure certificate is valid (not expired, valid chain of trust, no on CRL, etc.)
- Assume good hygiene (e.g., use persistent HTTP connections)

Base URL

- Each retailer publishes endpoints relative to a Base URL
- All endpoints are relative to these Base URLs
- A Base URL includes
  - Domain (e.g., api.sofaspuds.com)
  - Content Provider—studio or delegate(s)
  - A version of form v*x* where *x* is the version
- General agreement not to include studio in URL
- Example:

  `api.sofaspudfilms.com/mddf/v1/craigsmovies.com`

Base URL Issues

- **Question:** *Should studio be part of the path?*

  Putting Content Provider in the URL makes access control simpler, and avoids issues such as ALID collision. Assuming ownership changes result in completely new Avails. However, studio mergers (or splits) can complicate structure.

- *Answer: No*

Published Avails Endpoints (relative to Base URL)

- /avails
  - Bulk Avail operations – Yes for initial implementation. Might need filters (e.g., territory).
  - e.g., `api.sofaspuds.com/mddf/craigsmovies.com/v1/`**`avails`**
- /avails/{ALID} where {ALID} is the value of an ALID
  - Individual Avail operations
  - `api.sofaspuds.com/mddf/craigsmovies.com/v1/`**`avails/md:alid:eidr-s:B65C-7EC9-1F9F-D611-F84F-0`**
  - `api.sofaspuds.com/mddf/craigsmovies.com/v1/`**`avails/md:alid:eidr-s:B65C-7EC9-1F9F-D611-F84F-0?Region=US`**
  - `api.sofaspuds.com/mddf/craigsmovies.com/v1/`**`avails/md:alid:eidr-s:B65C-7EC9-1F9F-D611-F84F-0?TransactionID=12345`**
- /avails_atom
  - Gets service document. Other endpoints provided in service document for feeds.
  - `api.sofaspuds.com/mddf/craigsmovies.com/v1/`**`avails_atom`**
- Do we need endpoints for Special queries: e.g., `…/getcount/avails_data`?
  - Yes, but should it be ‘count’ or ‘getcount’
- Note: Distinct endpoints would be provided for MMC, CPE, etc.

HTTP

- XML and JSON
  - Clients *must* include "`Accept: application/xml`"
  - Servers must support XML
  - Do we allow JSON?
    - Client *may* include "`application/json`"
    - Servers *may* support JSON
- ETag
  - Servers must include "`ETag:`" in GET responses.
  - Clients include "`If-None-Match:`" in GET requests
  - Servers respond with 304 if matched

HTTP Responses

Code	Interpretation
200 OK	Request received and processed.
201 Created	Object created (POST). Location information is returned
206 Partial Content	Indicates more items available in range GET
304 Not Modified	Content has not been modified since previous request with same ETag
400 Bad Request	Improperly formed request (e.g., bad XML)
401 Unauthorized	Client failed to authenticate properly and cannot access resource. Can be repeated after authentication.
403 Forbidden	Client authenticated properly, but is attempting to access a resources for which the client does not have rights. Do not repeat request
404 Not Found	Attempting to access a resource that does not exist
5xx Server Error	Try again

Note that caches and proxies might return additional codes.
4xx conditions return error status
Other 4xx conditions are returned as 400 with error information.

Error Status

- Conceptual example:

```
<?xml version="1.0" encoding="UTF-8"?>
<Error>
      <HTTPResponse>400</HTTPResponse>
      <ErrorCode>XMLValidation</ErrorCode>
      <ErrorMessage>Avails XML Document does not comply with Avails v2.2
Schema</ErrorMessage>
      <Resource>api.sofaspuds.com/mddf/v1/craigs.movies.com/avails/md:alid:e
idr-s:abcd-1234-abcd-1234-abcd-m</Resource>
      <MoreInfo>Missing required field AvailsLicensor</MoreInfo>
      <Ref>1234abcde</Ref>
</Error>
```

- Note the inclusion of
  - HTTP responses can include codes not returned (e.g., 409, 410)
  - Developer-friendly error messages
  - Transaction reference for debugging (retrieving logs, etc.)

HTTP GET for multiple resources

- GET to /avails is assumed to act across all Avails
- Without further qualification, all Avails returned in one Avails XML document
- Range-GET need not be supported
- Sorted GET requests not supported
  - Resources are returned in an undefined order
- General Queries not supported
  - Should they be? If so, which ones? (title, EIDR, etc.)
  - If supported, include query string in resource GET

HTTP GET for multiple resources

- Pagination Supported
  - GET to /avails returns an Avails object containing all Avails for that studio (one record). [Support Range GET? Not initially]
  - Partial request are done using query string with
    - `offset` – offset from first record (0 = first record)
    - `limit` – Maximum number of records that can be returned (note that only on the last request will < limit resources be returned)
  - GET `api.sofaspuds.com/mddf/v1.0/avails?offset=0&limit=100`
  - If more items are available, HTTP Response is 206 [Is this legal?]
  - Alternatives
    - Provide a special query to determine how many Avails are present (/avails_data/getcount)
    - Partial response: …/avails?fields=ALID,Licensor,AvailType,ShortDescription
- Partial Response not supported?

Authentication and Authorization

Authentication

- Clients (APIs) authenticated with client_id, client_password
  - Clients present id/password
  - Obtained from Portal
  - Note that a Client can act on behalf of multiple Resource Owners (e.g., service provider servicing multiple studios)
  - Note: This is different from OAuth2 client identifier
- Users authenticated
  - Any authentication method is acceptable
  - Two Factor Authentication recommended
  - Needed by Resource Owners to obtain Authorization Grant
  - Needed by Client implementers to obtain client_id/client_password

Authorization

- Authorization Grant -- a secret presented to obtain an Authentication Token and, optionally, a Refresh Token
- Authentication Tokens – A secret presented to access Resources
  - Includes ‘scope’ of what can be accessed
  - Short lifespan reduces impact of compromise
  - Can be revoked
- Refresh Tokens – A secret presented to obtain Authentication Token
  - Facilitate short-life Authentication tokens (can always refresh)
  - More secret than Authentication token
  - Can be revoked

Obtaining Authorization (part 1)

- OAuth2 provides multiple methods for obtaining Access Tokens (RFC 6749)
  - Authorization Code Grant (4.1)
    - Method used in MDDF API
  - Implicit Grant (4.2)
    - Not sufficiently secure (access tokens too long-lived), not supported by most implementations (?)
  - Resource Owner Password Credentials Grant (4.3)
    - Not supported because it would require sharing passwords between organizations (weak security practice)
  - Client Credentials Grant (4.4)
    - Not supported because Client credentials do not necessarily identify Resource Owner

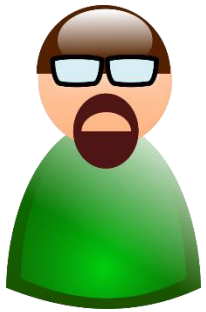
Obtaining Authorization Grant (part 1)

- Model is for a user to obtain Authorization Grant through a Portal
  - For example, a studio logs into a retailer portal to get Authorization information
- In all cases
  - User must provide credentials (username/password) to obtain Authorization Grant
  - User specifies the scope (e.g., avails, metadata) of the authorization

Obtaining Authorization Grant (part 2)

- OAuth2 supports multiple methods for acquiring access tokens
  - Authorization code

Obtaining Authorization Code



Resource Owner
User

User logs into Portal

User requests Authorization Code

Provides client_id *{abc123}*

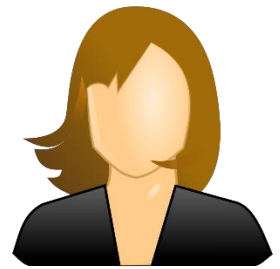
Portal delivers Authorization Code

{Zw80Wcwzliwk0ayb}

Portal

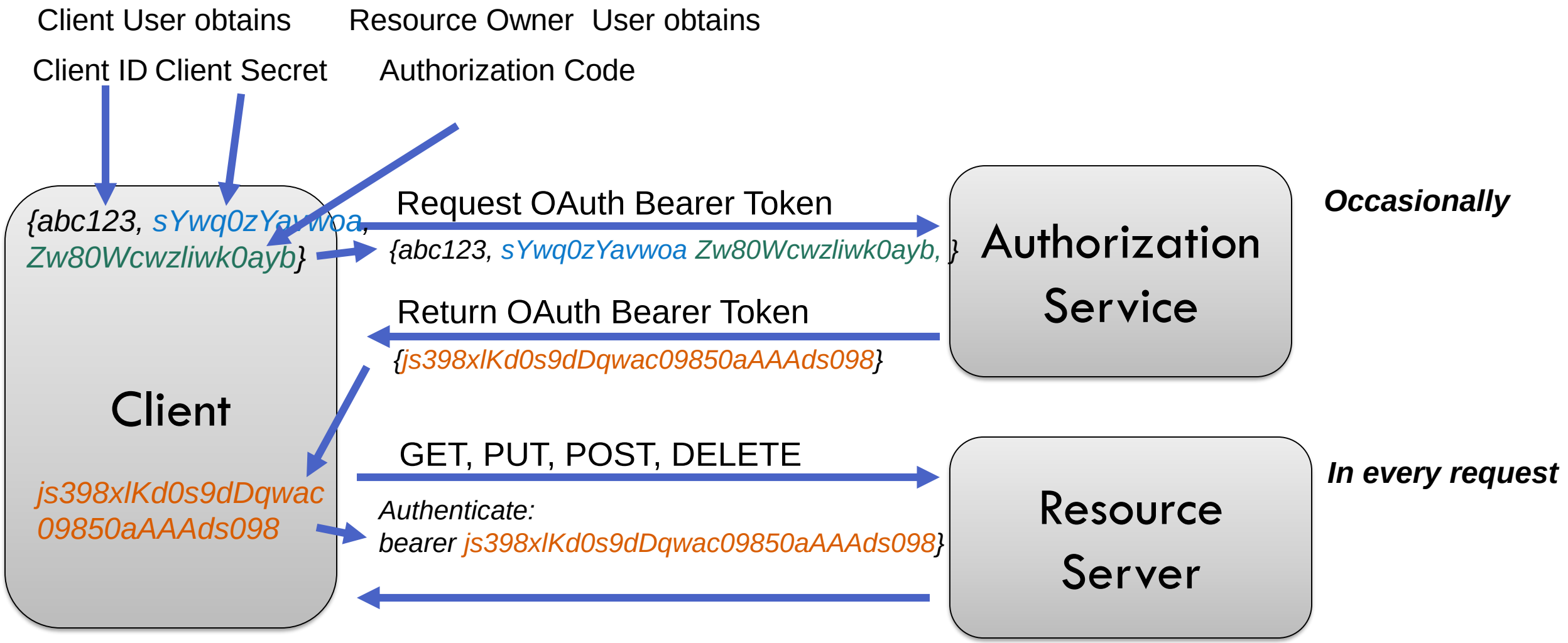
Resource Owner delivers client-specific Authorization Code to Client via secure channel

{Zw80Wcwzliwk0ayb}

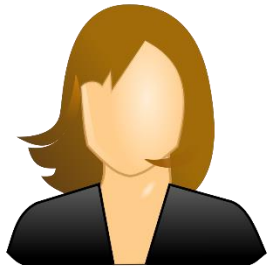


(repeat for each Client who will be authorized)

Authorization Process



Obtaining Client ID and Password



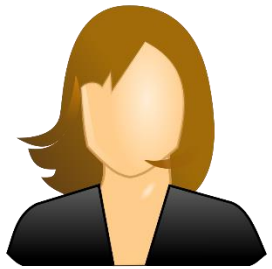
Client User

User logs into Portal

Portal delivers Client ID and password

{abc123, sYwq0zYavwoa}

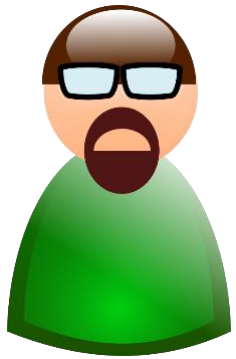
Portal
(Auth Server)



Client User

User transmits client_id to Resource Owner

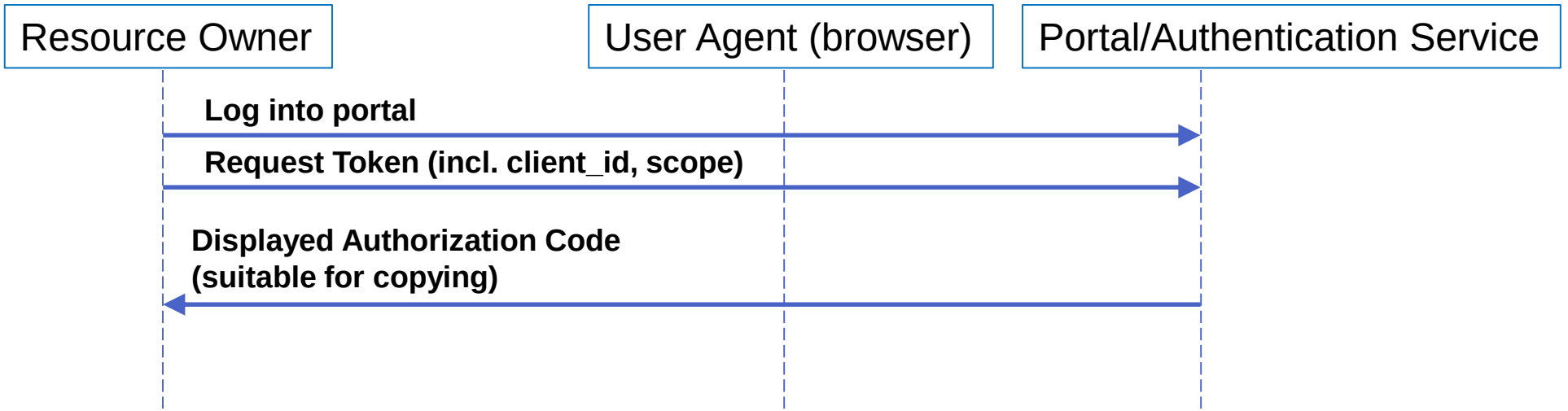
{abc123}



Resource Owner
User

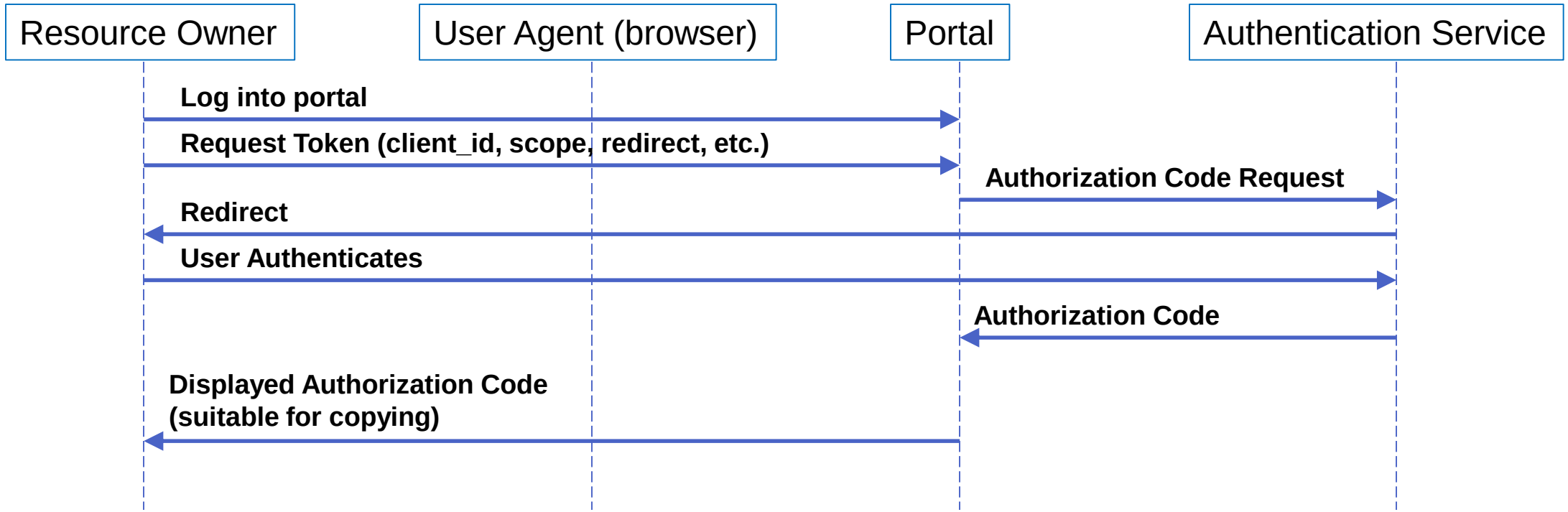
Authentication (Simple Portal model)

- Portal and Authentication Service integrated



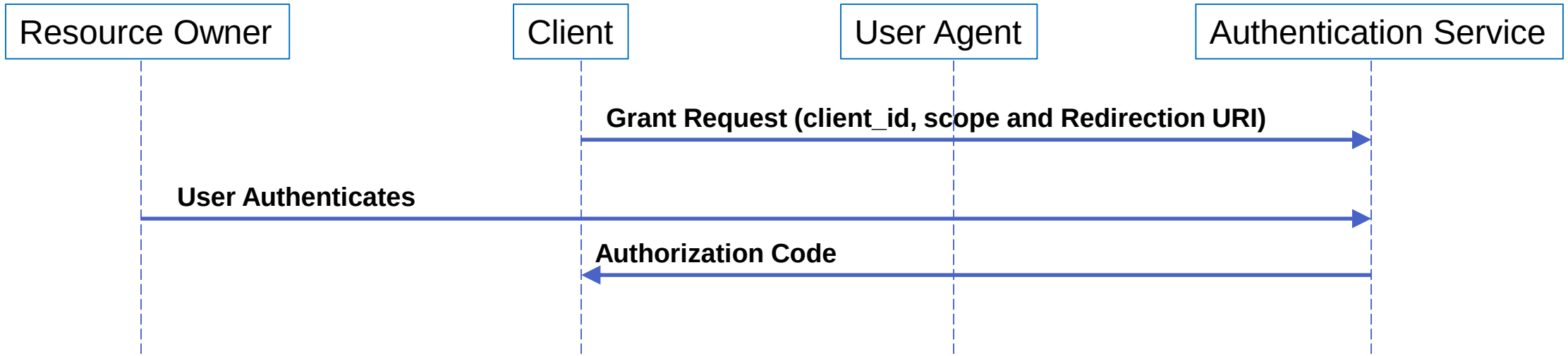
Authentication (Portal and RFC 6749 Auth Code Grant)

- Portal implements RFC 6749, Section 4.1 Authorization Code Grant

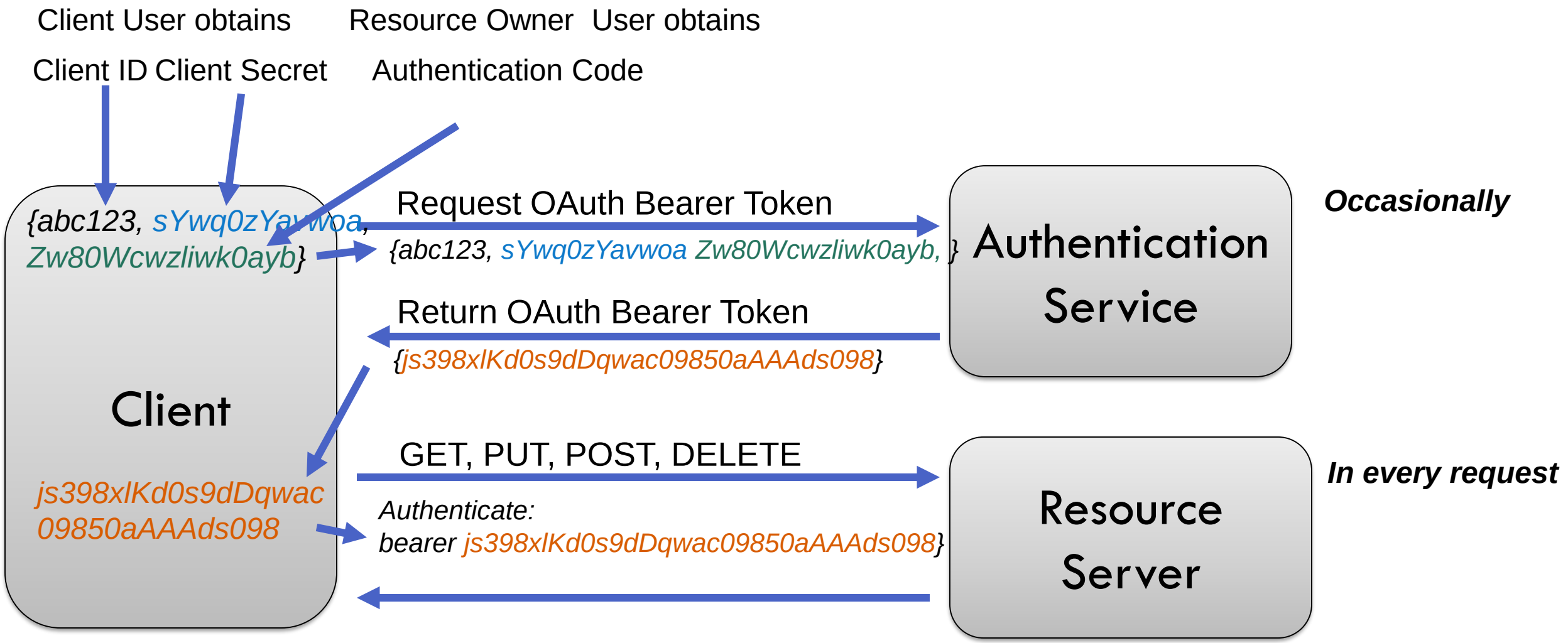


Authentication (RFC 6749 Auth Code Grant)

- Client implements RFC 6749, Section 4.1 Authorization Code Grant

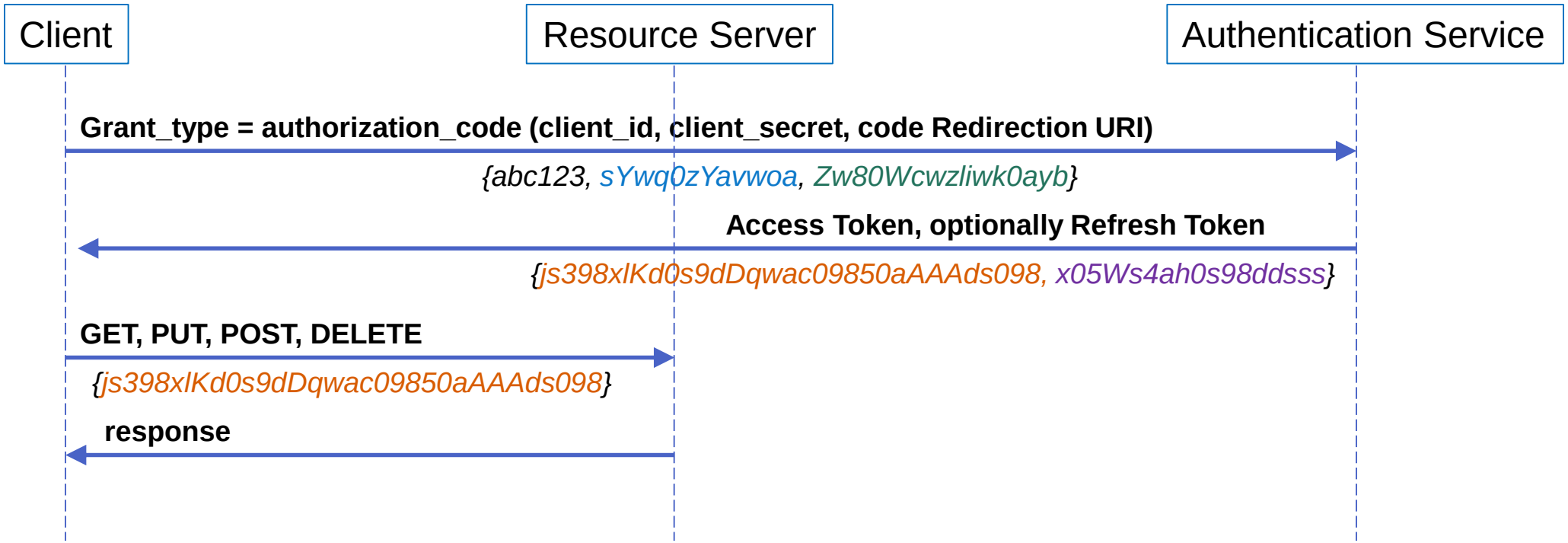


Authorization Process



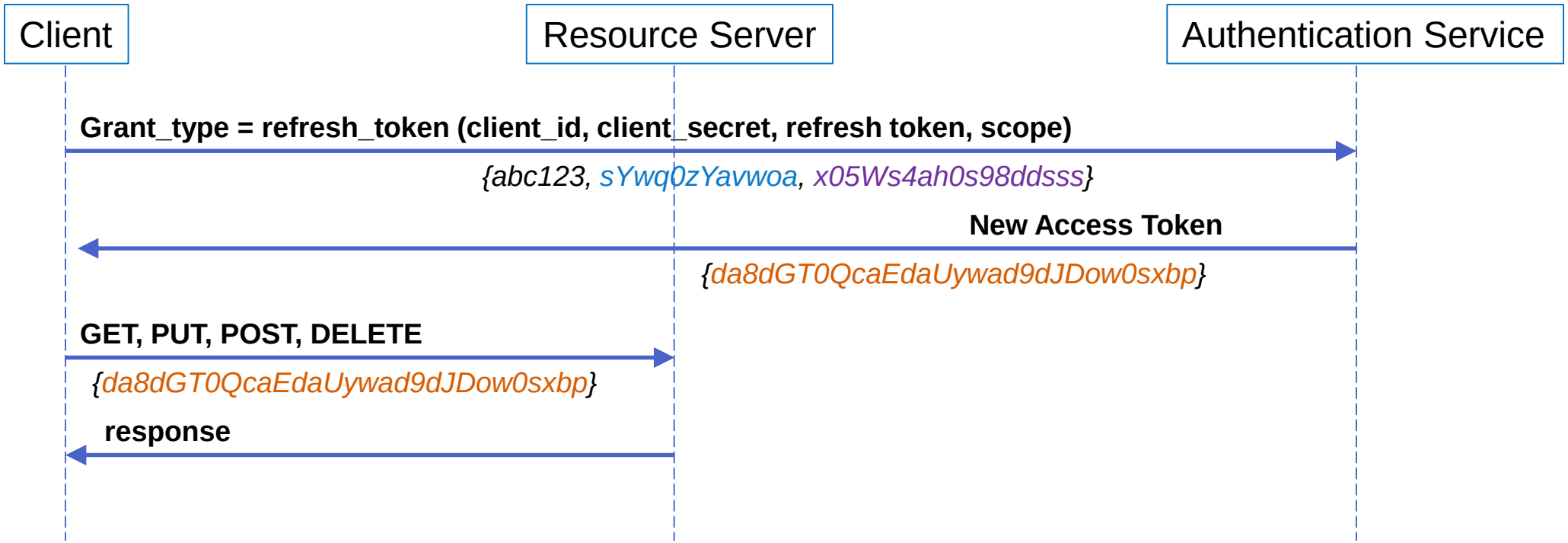
Access Token

- Client implements RFC 6749, Section 4.1 Authorization Code Grant



Refresh Token

- Client implements RFC 6749, Section 4.1 Authorization Code Grant



ATOM

Why Atom?

- Problem
  - To keep things properly RESTful and for consistency, a GET of a resource should return all useful information
  - So, polling Avails objects could return all useful information
  - And, caching can make polling more efficient
  - But, out of hundreds of thousands of Avail objects, only a few dozen are actively being processed
  - That means at least 3 orders of magnitude more polling queries are performed than necessary
  - It's better to know where the changes are so just those objects and be queried
- Solution
  - Each cloud service has something equivalent (or better), but these are proprietary.
  - Atom works and is a standard

Atom Feeds

- Persistent data is in resource; certain data duplicated in feed
  - Same information could be obtained by polling resources, although it's much less efficient
- Atom Publication Protocol (AtomPub)
  - Using "Service Document" as reference to all relevant feeds
- Atom Syndication Format – Feeds are created for each purpose, such as
  - Exception Feed – Feed of events that require intervention
  - Status Feed – Events related to background processing of resource data (e.g., Avails update validity checking)
  - Progress Feed – Updates of progress (e.g., storefront readiness)
- Feed characteristics (to be defined)
  - Update frequency and timeframes
  - Data Expiration

Atom Publishing Protocol (background)

- Using two parts of RFC 5023: Service Document and GET
- Service Document
  - “Service” – top level construct
  - “Workspace” – major functions. “Avails” is a workspace.
  - “Collection” – list of feeds associated with workspace
- Studio performs GET at well-known URL to retrieve Service Document
- AtomPub GET is RESTful, and all rules defined here for other HTTP GET operations apply
- Do we break up by data type? Leave to implementers.

Atom Service Document


```
GET https://api.sofaspuds.com/mddf/v1/avails_atom

<?xml version="1.0" encoding='utf-8'?>
<service xmlns="http://purl.org/atom/app#">
   <workspace title="Avails" >
      <collection title="Exception"
href="https://api.sofaspudfilms.com/mddf/v1.0/avails_atom/exception.atom
" />
      <collection title="Status" href="
https://api.sofaspudfilms.com/mddf/v1.0/avails_atom/status.atom" />
      <collection title="Progress" href="
https://api.sofaspudfilms.com/mddf/v1.0/avails_atom/progress.atom" />

   </workspace>
</service>
```

Atom Syndication Format *entry* (RFC 4287)

Element	Usage
title	Title of feed (“Exception”, “Status”, “Progress”)
link	Link to this feed
id	Unique ID for this entry
updated	Date and time when feed was updated
entry	One entry for each resource
entry/title	Optional: Title. e.g., ShortDescription from Avail
entry/link	Link with href attribute referring to resource
entry/id	ID for resource (typically ALID)
entry/updated	Date and time resource was updated
entry/category	TBD

Atom Syndication Format (RFC 4287)


```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
    <title>Avails Exception: craigsmovies.com</title>
    <link href="https://api.sofaspuds.com/mddf/v1.0/avails_atom/exception.atom" />
    <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa61</id>
    <updated>2017-03-16T03:43:02Z</updated>
    <entry>
        <title>Unbelievable Example Movie</title>
        <link href="https://api.sofaspuds.com/mddf/v1.0/avails_atom/md:alid:eidr-s:1234-5432-abcd-efgh-abcd-l”>
        <id>md:alid:eidr-s:1234-5432-abcd-efgh-abcd-l</id>
        <updated>2017-03-10T17:46:02Z</updated>
    </entry>
    <entry>
        <title>Sequel to Unbelievable Movie</title>
        <link href="https://api.sofaspuds.com/mddf/v1.0/avails_atom/md:alid:eidr-s:abcd-5432-abcd-efgh-abcd-l”>
        <id>md:alid:eidr-s:abcd-5432-abcd-efgh-abcd-l</id>
        <updated>2017-03-16T03:43:02Z</updated>
    </entry>
</feed>
```

XML vs. JSON

- XML is more rigorous
  - Well-defined
  - Can be validated
  - Can be signed (possibly important for Avails that are contractual documents)
  - Applicable standards are in XML
- JSON is easier to parse has nice language support, especially JavaScipt

Part 3: Implementation Planning

Implementation Planning

- MVP
  - What do we start with? **Avail plus status**? Metadata? Something else?
  - Do we do full authentication or find a shortcut for testing? **Full**
  - Do we do something "end-to-end" without integrating (e.g., pull files from a directory and drop into directory on the other side). **Do we stub out something? People like this idea**.
- Integration and Testing
  - Should be incremental. **Agreed in principle. Possibly stubs. Need to define.**
  - Bilateral testing or Virtual Plugfests? **Let's see who players are and where they are.**
- Shared resources?
  - Samples of authentication, authorization, REST and Atom
  - References to implementation resources (e.g., OAuth2 and Atom libraries)
- Communications
  - **Email**, Slack?, Dropbox?, Something else? Use existing list.
- Meeting type and frequency?
  - Call every 4 weeks. Cancel if no agenda items. Next call to review updates to spec based on this discussion.
- Other?

Thank You

Postel's Law

Be liberal in what you accept, and conservative in what you send