

Structuring the Scope: Enabling Adaptive and Multilateral Authorization Management

Bojan Suzic, Andreas Reiter and Alexander Marsalek

Graz University of Technology, Austria



Overview

- Introduction and Motivation
- Properties of Access Scopes
- Integrated Authorization Management
- Application in Use Scenario
- Conclusion

Introduction

Cloud services expose their resources and operations using Web APIs

Web APIs are applied to support core business of service providers

How can be security aspects of service use and resource sharing be managed?

Some issues:

- Obstacles due to proprietary interfaces and hard-wiring
- Interoperability of security controls across diverse organizations
- Provider-centric management of security in the cloud
- Capability of security controls

Managing and coordinating security of our assets hosted at other providers?

Motivational Scenario

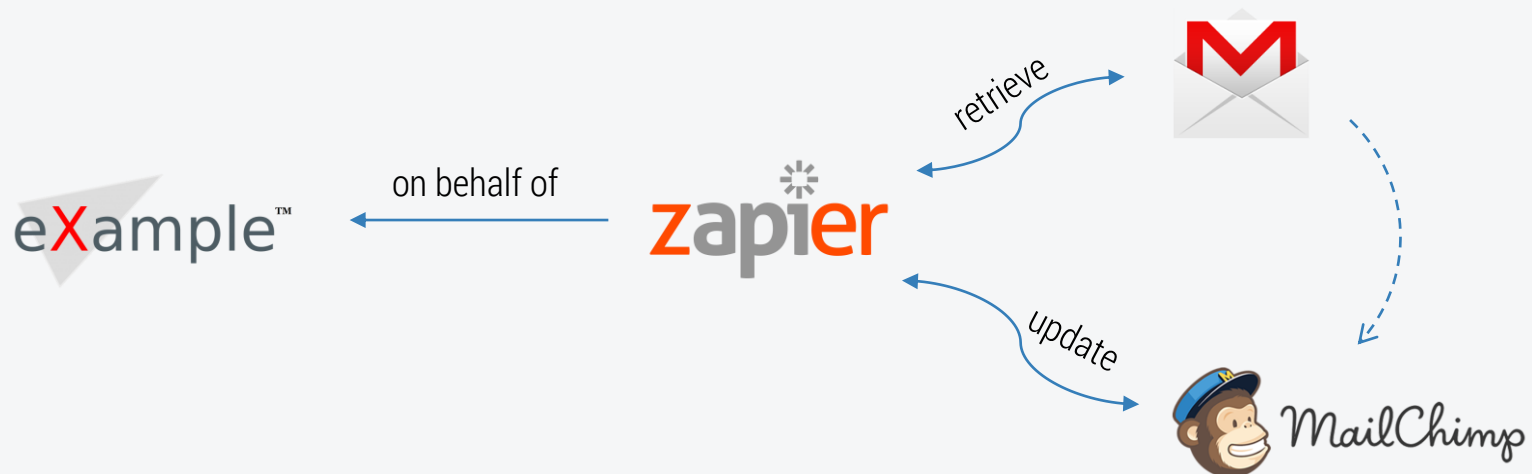
eXample Inc. uses *Zapier* to automate its tasks

Zapier connects data sources from *Gmail* and *MailChimp* on behalf of a customer

Web APIs (REST) typically applied to expose and share resources

Task:

- Periodically retrieve and extract email senders from recent emails at Gmail
- Add them as subscribers to a list at MailChimp



Authorization in the Cloud

Authorization: Zapier needs access to resources at both providers

Typical case relies on OAuth 2.0 Web Authorization Framework – RFC 6749

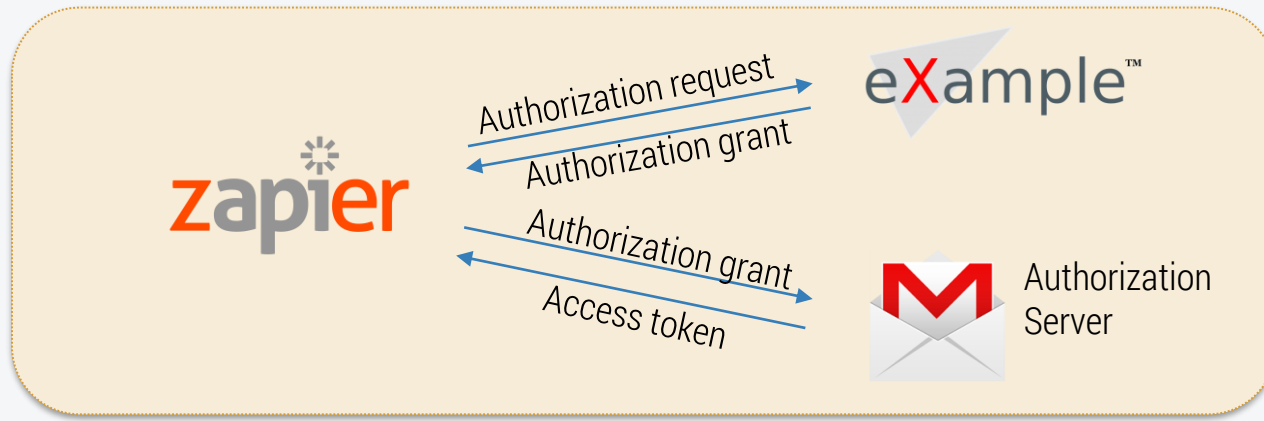
Primary concepts in OAuth 2.0:

- Resource owner, resource server, authorization server, client
- Initiate *authorization flow* to obtain access credentials
- *Access token* – most commonly used access credential
- *Access scope* – determines the extent of permissions given to the agent

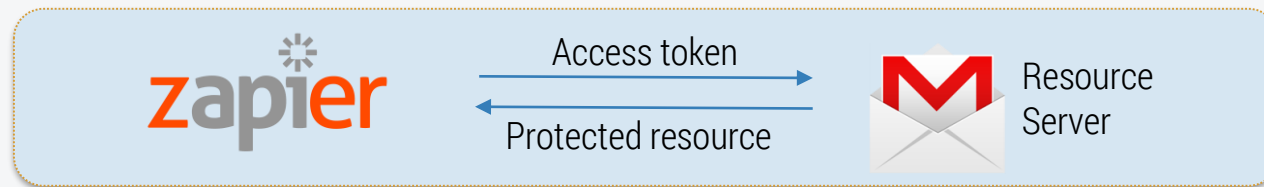


Authorization Flows

Obtaining access token (initially)

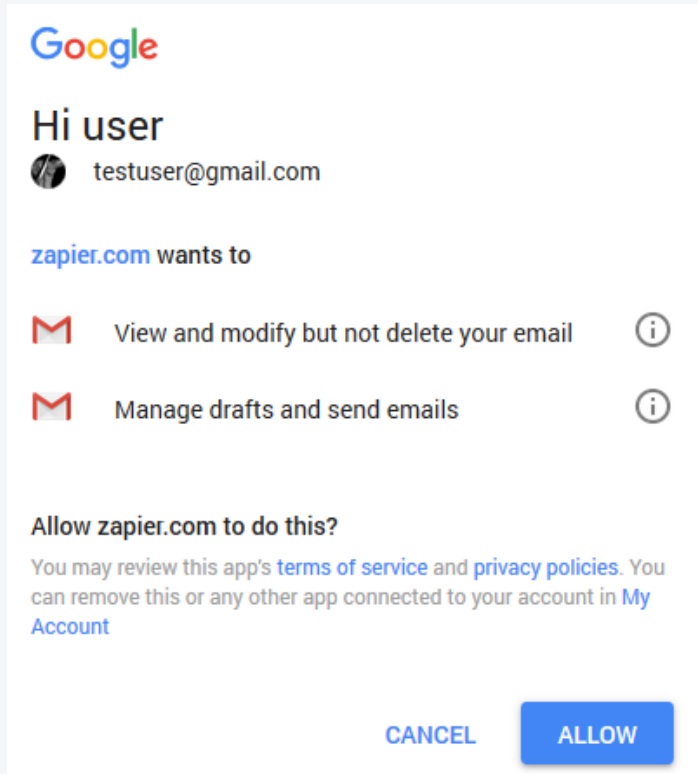


Retrieving resource or performing operations (repetitive)



The same flow is applied in the case of MailChimp as well

Obtaining Consent - Zapier



Resource owner is presented with the interface to review and allow the permissions given to the *client*


Permissions are abstracted as a *scope*

Scopes requested by Zapier:

- *gmail.compose*
- *gmail.modify*

Both scopes provide broad range of operations over all instances of subsumed resources

Obtaining Consent - MailChimp



Connect Zapier to your account

Zapier makes it easy to connect 100s of web applications directly to Mailchimp, no coding required!

Username

Password

MailChimp does not apply scopes

The given permissions include
all operations over every resource

No compartmentalization applied

Broad Permissions

Requirements from use cases:

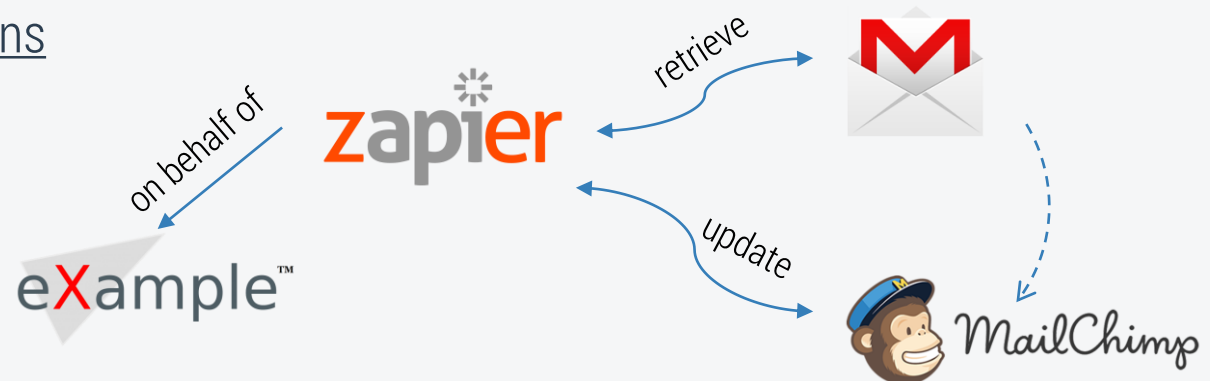
- Gmail: (1) retrieving a list of recent messages and (2) the value of *From: field* from the *header* of these *messages* needed
- MailChimp: (3) adding an entity to a particular subscriber list

The problem with *broad permissions*:

- Zapier allowed to retrieve and manage all messages in an account
- This includes managing drafts, sending or temporarily deleting messages
- Can execute any API operation at MailChimp

Potentially leads to numerous security and privacy risks

Applies to most integrations



Properties of Access Scopes

Unilateral definition

Invariable

Unstructured

Out-of-the-band

Coupled

Context insensitive

Established by the service provider

Designated as a predefined set

Imposed to other entities

Excluding resource owners and clients

Properties of Access Scopes

Unilateral definition

Invariable

Unstructured

Out-of-the-band

Coupled

Context insensitive

Statically determined

Immutable sets of permissions

Typically do not change in production

Properties of Access Scopes

Unilateral definition

Invariable

Unstructured

Out-of-the-band

Coupled

Context insensitive

Defined as opaque strings

Cannot be decomposed

Authorization extent cannot be derived

Discovery of supported or provided authorizations not possible

Dynamic definition not supported

Properties of Access Scopes

Unilateral definition

Invariable

Unstructured

Out-of-the-band

Coupled

Context insensitive

The scope extent communicated non-transparently
Described in service documentation (for developers)
Applications cannot interpret the scope

Properties of Access Scopes

Unilateral definition

Invariable

Unstructured

Out-of-the-band

Coupled

Context insensitive

Specific to the service

May reflect business model or view of SP

Cannot be decomposed

Predefined set with built-in properties

Properties of Access Scopes

Unilateral definition

Invariable

Unstructured

Out-of-the-band

Coupled

Context insensitive

Cannot express attributes of resources, environment or involved parties

The same parameters apply to all contexts (end-users, resources, target environment)

Integrated Authorization Management

Supporting integrated authorization management:

Granular specification of authorizations

Claiming acceptable constraints

Context-dependent enforcement

Selective and transformational sharing

Scalable management

Contribution

Defining management flows

- Supporting cooperative and adaptive authorization management

Defining supporting vocabularies

- Describing requests, responses, contextual properties and resource restrictions
- Describing access control and OAuth 2.0 entities

Establishing authorization descriptor

- Relies on vocabularies
- Supports granular, instructive and expressive specification
- Structuring authorization requirements and grants
- Applicable beyond single organization

Management Flows

Defining management flows:

- (1) Exposing the service descriptor
- (2) Determining the request scope
- (3) Requesting authorization
- (4) Refining authorization extent
- (5) Transforming into security policy
- (6) Inspecting authorization descriptor

Provider exposes service description

Includes available resources, their structure and organization

Management Flows

Defining management flows:

- (1) Exposing the service descriptor
- (2) Determining the request scope**
- (3) Requesting authorization
- (4) Refining authorization extent
- (5) Transforming into security policy
- (6) Inspecting authorization descriptor

Client retrieves service model and decides the extent of required permissions

Finding intersection between security and functional goals

Considers exposed resources, applicable constraints and supported operations

Management Flows

Defining management flows:

- (1) Exposing the service descriptor
- (2) Determining the request scope
- (3) Requesting authorization**
- (4) Refining authorization extent
- (5) Transforming into security policy
- (6) Inspecting authorization descriptor

Client generates authorization request

Expresses its acceptable range of permissions and constraints

Deliver request *interactively* or *asynchronously*

Management Flows

Defining management flows:

- (1) Exposing the service descriptor
- (2) Determining the request scope
- (3) Requesting authorization
- (4) Refining authorization extent**
- (5) Transforming into security policy
- (6) Inspecting authorization descriptor

Resource owner inspects and refines the request

Interactive request: inspected using owner's client involved in the flow

Asynchronous request: on the side of service provider

Management Flows

Defining management flows:

- (1) Exposing the service descriptor
- (2) Determining the request scope
- (3) Requesting authorization
- (4) Refining authorization extent
- (5) Transforming into security policy**
- (6) Inspecting authorization descriptor

After consent by resource owner is obtained

Server-side transformation into security policy

Considers target system and environment

Management Flows

Defining management flows:

- (1) Exposing the service descriptor
- (2) Determining the request scope
- (3) Requesting authorization
- (4) Refining authorization extent
- (5) Transforming into security policy
- (6) Inspecting authorization descriptor**

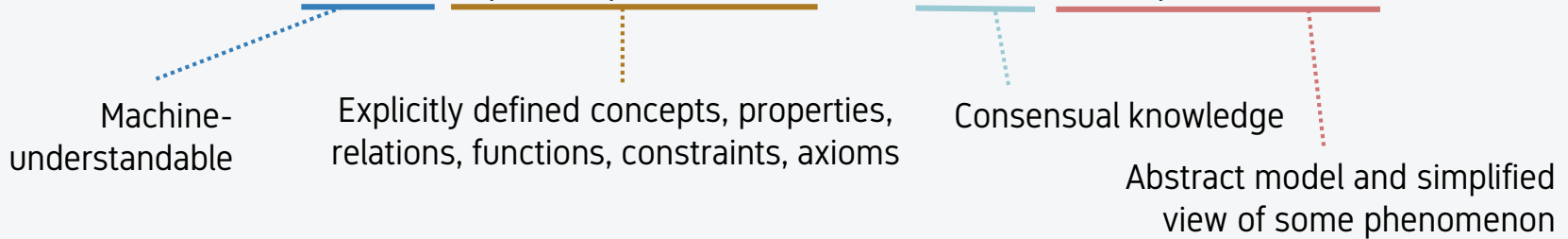
Optionally providing authorization descriptor back to the client

Allows the client to determine the degree of provided (redacted) permissions

Vocabularies

Uses semantic vocabulary as a building block, establishing a

formal, explicit specification of a shared conceptualization



$$\Omega = (C, R, E, I)$$

C – classes (unary predicates)

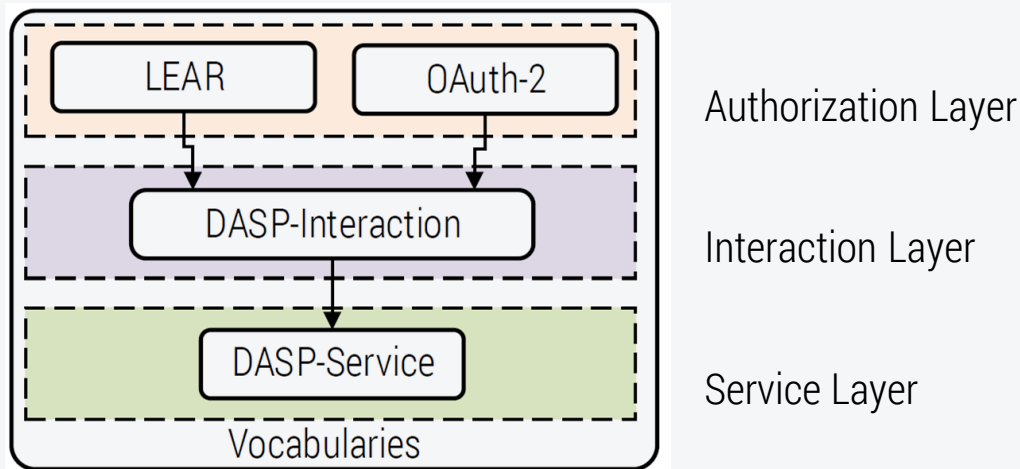
R – relations (binary or higher predicates)

E – explicit instances of classes and relations

A - axioms

Vocabularies

Organizing vocabularies in layers according to their role in the process



Concepts in vocabularies serve as *terminological knowledge* (T-Box)

To describe services or interactions we instantiate them as *assertions* (A-Box)

Authorization descriptor – a graph-based structure, instantiates concepts from vocabularies

Conforms to descriptions and capabilities announced by services

Roles: AuthorizationRequest, AuthorizationResponse, ErrorResponse

Exposing Service Description

Given a service vocabulary $\Omega_{(s)} = \{C, R, \varepsilon, I\}$

Service provider exposes a service description

$$M = \{C_M, R_M, E_M, I_M\} \mid C_M \subseteq C, R_M \subseteq R, I_M \subseteq I \text{ and } \forall e \in E_M, e \in C_M \vee e \in R_M$$

Provided as RDF, JSON-LD or Turtle

Service description typically includes:

- Exposed resources and intents (actions)
- Relations between resources and actions
- Parameters and URL mappings for entities
- Organization of resources (consisting elements)
- Supported operations in the service (transformative)
- Extraction rules for resources or their elements

Exposing Service Description

Example in Turtle:

- (1) References vocabularies
- (2) Initializes service and exposes its resources and intents
- (3) Refining hierarchy of resources
- (4) Specifying extraction rules (semantic lifting)

```
@prefix : <http://www.daspsec.org/o/gmail-api#> .
...
@base <http://www.daspsec.org/o/gmail-api> .
<http://www.daspsec.org/o/gmail-api> rdf:type owl:Ontology ;
  owl:imports <http://www.daspsec.org/o/dasp-service/0.3>, ... .
:GmailService rdf:type dasp-service:Service ;
  dasp-service:hasAction :RetrieveEmail .
:EmailResource rdf:type dasp-service:Resource ;
  dasp-service:hasElement :MessageHeader , ... .
:RetrieveEmail dasp-service:Action ;
  dasp-service:affectsResource :EmailResource ;
  dasp-service:hasURLPath:1 :U_URLDesignator .
...
:U_URLDesignator dasp-service:StaticPathElement ;
  dasp-service:hasElementValue "gmail"^^xsd:string .
:U_MessageIdAPIElement rdf:type dasp-service:DynamicPathElement .
:MessageHeader rdf:type dasp-service:Element ;
  dasp-service:hasElement :FromField , ... .
:FromField rdf:type dasp-service:Element .
:MessageHeaderExtractor rdf:type dasp-service:ElementExtractor> ;
  dasp-service:hasElement :MessageHeader ;
  dasp-service:isProducedByAction :RetrieveEmail ;
  dasp-service:hasJSONPathContentExtractionRule "$.payload.headers"^^xsd:string .
:FromFieldExtractor rdf:type dasp-service:ElementExtractor ;
  dasp-service:hasElement :FromField ;
  dasp-service:hasElementContainer :MessageHeader ;
  dasp-service:hasJSONPathContentExtractionRule "$..?(@.name==\"From\")"^^xsd:string .
```

Consuming Service Description

Accessing agent consumes service descriptions to structure authorization request:

- Retrieve service descriptor

$$D \leftarrow \langle \text{remote service} \rangle$$

- Derive exposed services

$$S \leftarrow s_d \in D \mid s_d.\text{instanceOf}(\text{DASP-Service:Service})$$

- Retrieve exposed resources of a service (optionally)

$$R \leftarrow \text{res} \in D \mid s \xrightarrow{\text{hasResource}} \text{res}$$

- Retrieve supported actions (optionally)

$$A \leftarrow \text{act} \in D \mid \text{res} \xrightarrow{\text{hasAction}} \text{act}$$

- For actions: derive affected resources, their elements and exposed operations

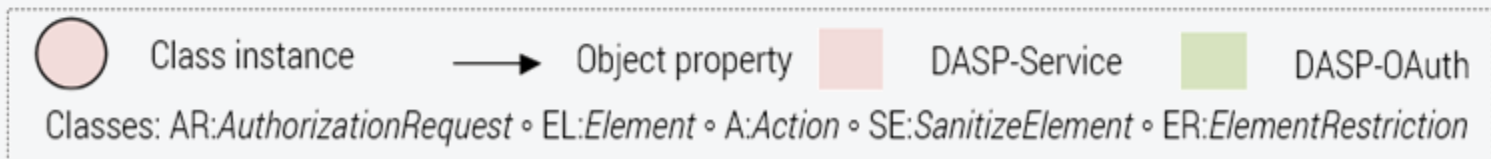
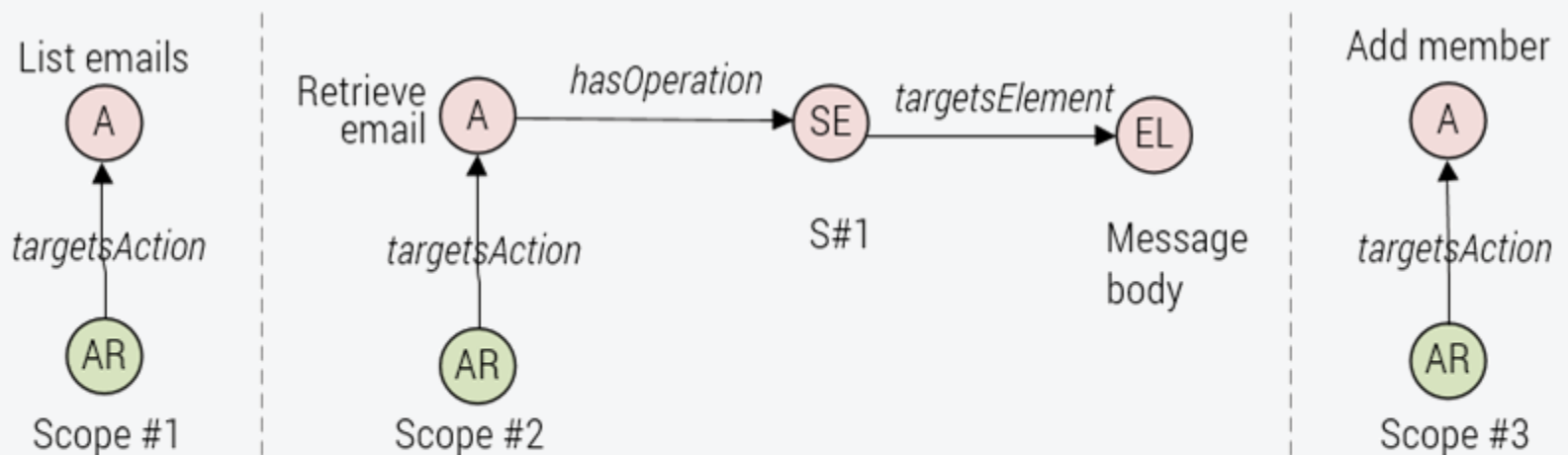
$$\text{act} \xrightarrow{\text{affectsResource}} \text{res}, \text{res} \xrightarrow{\text{hasElement}} \text{el}, \text{act} \xrightarrow{\text{hasOperation}} \text{op}$$

- Determine requested actions/resources and applicable operations and initialize a new scope

Structuring Authorization Request

Structured scopes for three cases (accessing agent):

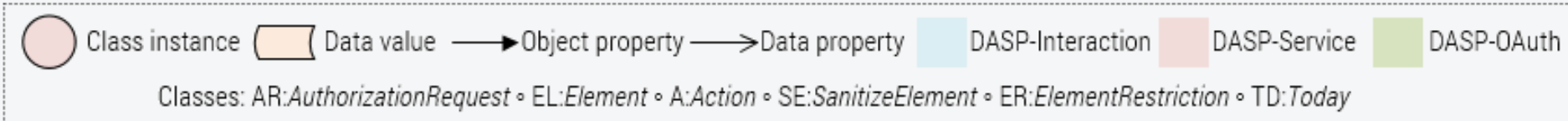
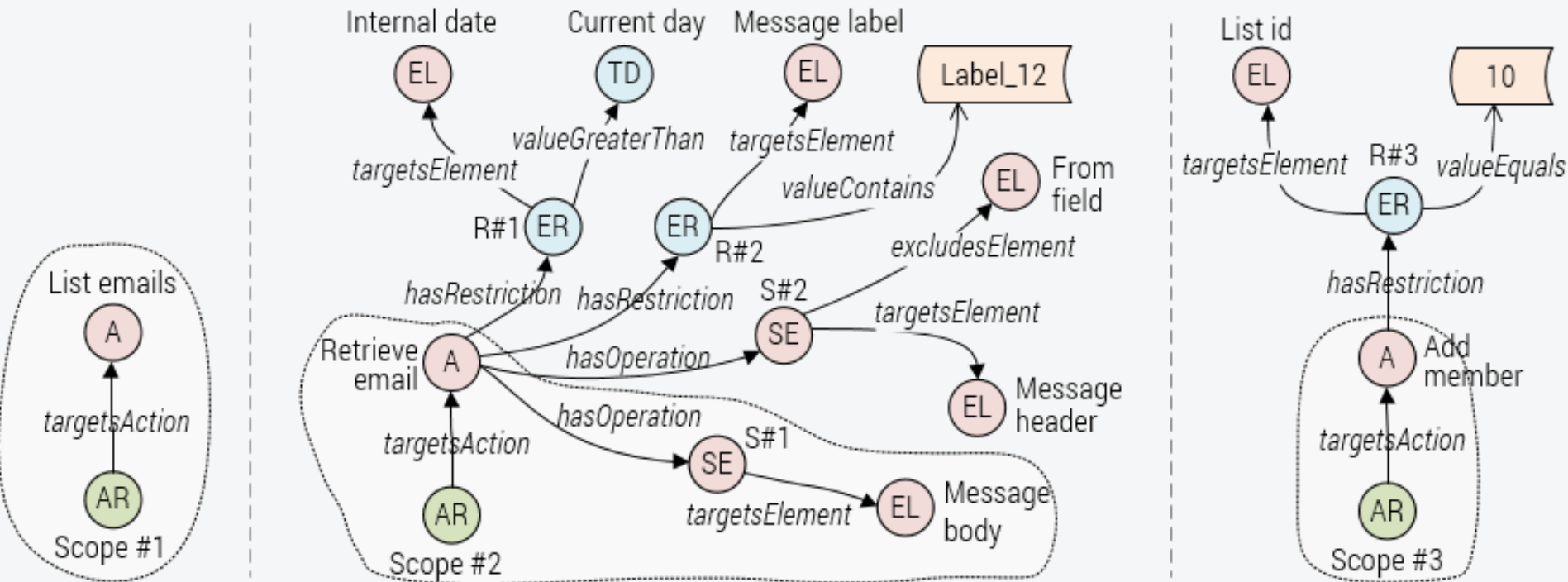
- *Partially cooperative* client – provides focused, but non-optimally constrained request
- Gmail: (1) retrieving a list of recent messages and (2) the value of From: field from the header of these messages needed
- MailChimp: (3) possibility to add an entity to a particular subscriber list



Structuring Authorization Request

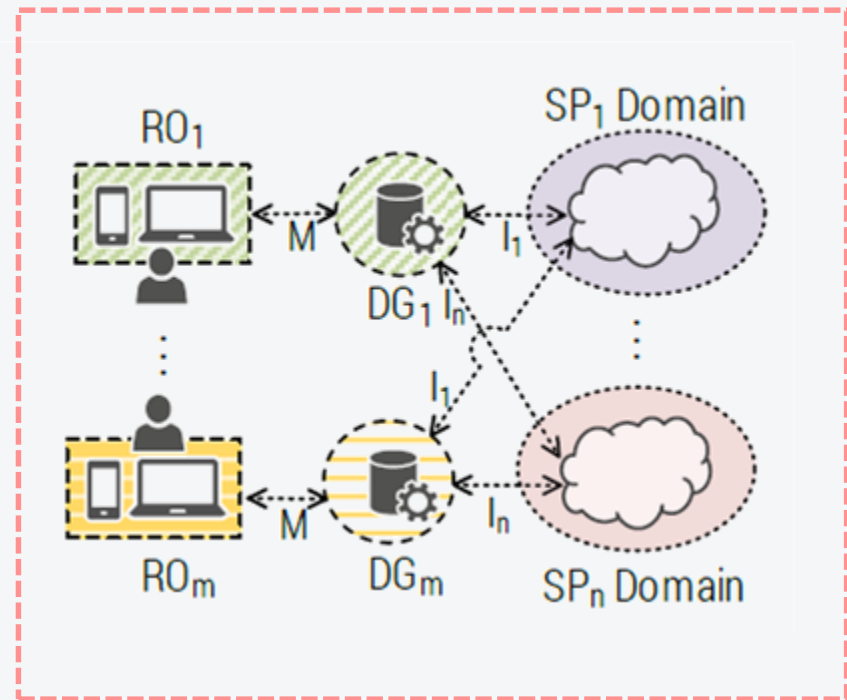
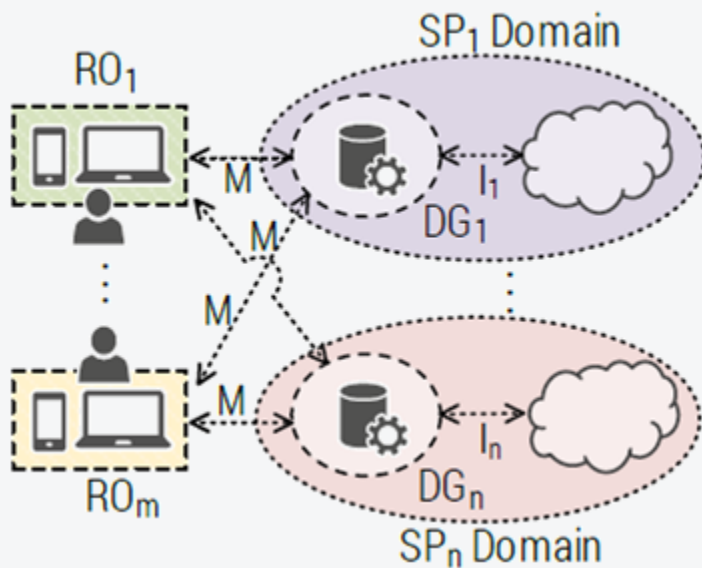
Structured scopes for three cases (redacted by the resource owner):

- Redaction can be done in *interactive* or *asynchronous* flow



Deployment Models

Data-security Gateway - *provider-centric* and *user-centric* deployment models
Implements security evaluation and enforcement using provided vocabularies



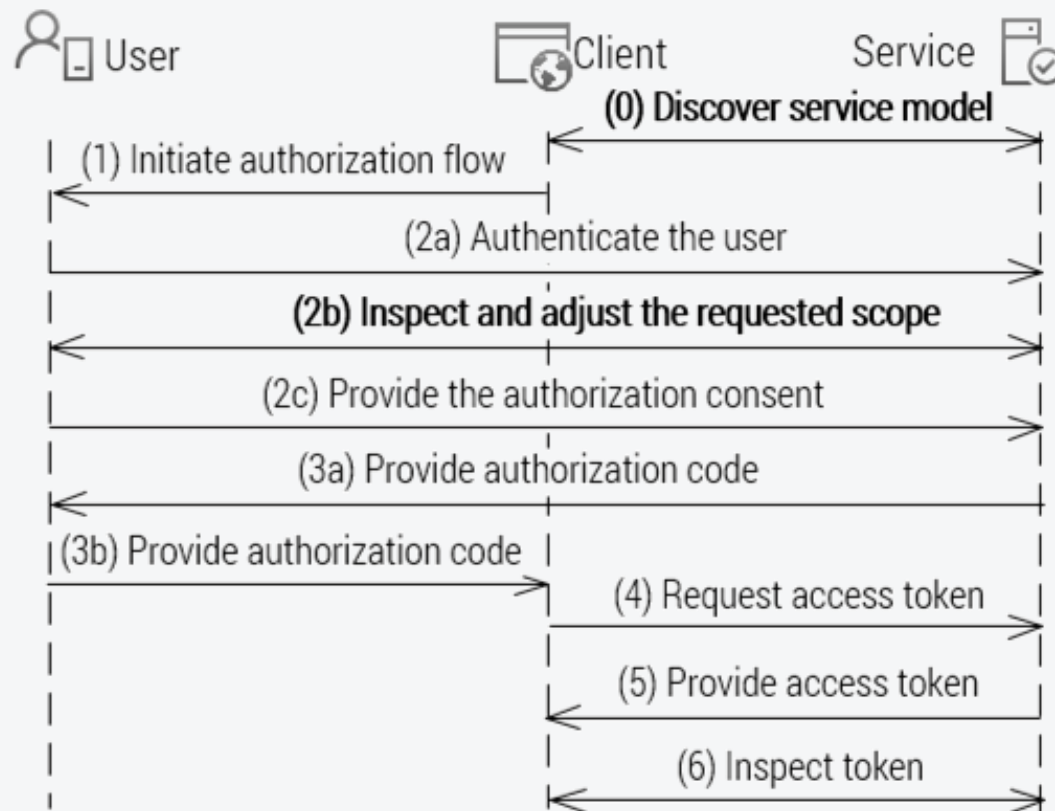
Related work: <https://demo.a-sit.at/am/>

Integration with Other Frameworks

Aim – protocol-agnostic approach that scales beyond a single environment

Integration into OAuth 2.0 – additional steps (0 and 2b)

Authorization descriptor provided as Base64 encoded string



Conclusion

Observed issues:

- Under-specification leading to low management capability
- Semantic vs syntactic interoperability

Goal:

- Advancing manageability of security controls
- End-to-end integration and reuse of security controls
- Application beyond a single protocol (OAuth)

Approach:

- Introducing lightweight interoperability layer to connect different environments
- Decoupling security controls from service providers and associating them with service models
- Providing self-dereferenceable and transparent structures for resource- and context-aware management of authorizations

Any questions?



Thanks for your attention!