A-SIT Technology
E-Government Innovation Center

30 August 2016 Study · Privacy & Crypto · 27 min read

Contextual Data Exchange

Archived. This content is no longer updated. It remains available for reference.

Source https://technology2.a-sit.at/en/publications/contextual-data-exchange/

We present a reusable data structure that addresses the issues of static, inflexible and practically non-interoperable authorization definitions. We first establish the structure that introduces enhanced expressivity, context-sensitivity and adaptability in descriptions of authorization constraints. In the next step we develop the supporting software component and the web-based interface for definition and inspection of access authorizations established using the proposed structure. Based on that, we present a demonstration prototype and describe the application of the proposed structure both in terms of emerging solutions and existing authorization framework.

1. Introduction

The emerging notion of data-driven economy sets the focus on data as an important asset in generating new knowledge and wealth in today's world. The adoption of technologies and concepts such as *smart cities*, *smart healthcare*, *smart government* and *smart energy* is expected to accelerate in coming years and decades, resulting in the transformation of economy and business models [1, 2]. The significance of this ongoing process has been acknowledged by government authorities as well, leading to specific actions aimed at supporting and facilitating the acceptance of data-driven business [2].

Big Data, large-scale data processing, and data exchange are recognized as some of the principal enablers of data-driven economy. In this work, we consider the data exchange as a process that quickly gains significance as the novel business models emerge. We focus on security aspects of data exchange across different entities and introduce a new perspective that addresses security challenges and issues present in existing standards and implementations. These issues prevent secure and privacy conforming data sharing in complex flows that involve many subjects, diverse resource types and different data sharing layers.

In this work, we present a novel approach for formalizing authorizations. Our approach enables authorization to be transparently reused across various entities and platforms, supporting the interoperability of security functions in complex and internetworked environments. First, we introduce the structural basis for modeling of authorizations using graph-based descriptions and semantic vocabularies. We then present the software framework that implements our proposal and shows its practical application in terms of a web-based application prototype for the definition of expressive and granular authorization

extents. Our proposal is a part of wider effort [1] which is intended towards establishing the conceptual framework and supporting software tools aimed at advancing the security of cross-domain data sharing and integration processes

1.1. Current state

By taking part in a broad range of processes, such as data sharing, data reuse, and data processing, numerous entities are benefiting from the generation of new knowledge or derivation of new value from existing datasets. Serving both as technology and business enablers, the paradigms such as cloud and APIfication [3] foster the establishment of further means to generate, share, process and exchange the data on a large scale in the web and cloud. Currently, OAuth 2.0 [4] is de facto mostly adopted and widely used framework for web-based authorizations. By applying the flows defined in this framework, many subjects worldwide are exchanging the data or providing authorization consents for cross-entity resource sharing and service consumption.

In our previous work [5, 6, 7] we have investigated the capability of OAuth 2.0 and its emerging UMA profile [8, 9], especially regarding the structuring of authorization consents and their reuse across diverse platforms and entities. In [5] we have elaborated five issues related to security, manageability, and interoperability of cross-organizational data sharing flows based on these protocols. These include the *arbitrary definition of access scopes*, *non-standardized approach of access scopes*, *coarse-grained permissions*, *detached authorizations* and *out-of-the-band process for requesting of permissions* .

In this work we provide a model that extends existing access scopes in OAuth 2.0 and UMA, enabling expressive, adaptable, granular and dynamic definition of scope-based authorizations for data sharing and services consumption. With this proposal, we aim to address identified issues and provide a significant step towards the unified framework for advancing the security of crossorganizational authorizations and data sharing.

1 We use working title *DAta Sharing and Processing framework (DASP)* to denote the work that addresses various perspectives and related challenges

1.2. Terminology

We interchangeably apply the terms *authorization extent*, *authorization scope* or simply *authorization* . These terms should be referred here as synonyms. In practical meaning, authorization extent is used to describe a more abstract concept of an authorization instance and its related properties. In our framework, we treat this concept as the reusable asset in broader interactions related to authorization in the inter-organizational environment. On the other hand, authorization scope conforms more to a practical application of our structure in OAuth or UMA authorization frameworks. In this case, the structure is reduced to comply with the concept of a scope, as foreseen in the respective frameworks.

In the following sections, we oft rely on terms such as a *client*, *resource owner*, and *service provider* . We refer to these terms as defined in OAuth 2 web authorization framework [5], referring to the client that accesses the resources owned by a resource owner and collocated at some service provider.

1.3. Outline

In the following chapter, we present the model of enhanced authorization extents. We first clarify the understanding of authorization extent and its role in security management. We then introduce a general model that establishes the structural elements and relationships used to structure authorizations. Following that, we define an authorization token, explain its particular elements and structuring flow. We conclude the chapter with the sample model and its explanation.

In the third chapter, we present the software library and the prototype developed to structure the authorizations for end users. We present the software architecture and its main components.

In the fourth chapter, we introduce the case scenario. We first define the authorization requirements, then we walk through the prototype interface, illustrate and explain particular steps. The prototype is used to present the capability of provided library and GUI tool.

Finally, we conclude this report and provide an outlook of further work.

2. Modelling Authorization Extents

In this section, we present the general conceptual model that enables the description of the token structure presented in this work. First, we present the general overview of provided classes, object and data properties. Then, we explain each of them, describing their role and integration in the framework.

Our work presented in this section builds on the work outlined in our initial technical report [3] and published workshop paper [4]. While these publications addressed the application of interoperable security policies in cross-entity context, in this work we focus on the application of client-provided authorization requests that conform to the flows in currently broadly applied standards such as OAuth 2.0 and UMA. These authorizations may be transformed to security policies in later stages, as described in [4].

In this work, we reuse service descriptions provided in referenced works. We furthermore refine and extend the main vocabulary presented in the initial work. Based on these changes we establish DASP (DAta Sharing and Processing) Framework as the subsequent iteration of our previous work.

2.1. What is an authorization extent?

Authorization extent is a concept that describes a degree of authorization that is requested by the client and consented by the resource owner. This refers to the authorization as the multidimensional concept, referring both the resources, actions, operations, and their consisting elements, using different levels of granularity and expressivity. In this work, the authorization extent is furthermore considered to refer to a specific context that defines its validity and applicability under different situations. The definition of expressive, contextual and dynamic authorizations we see as a novel contribution not present in existing frameworks.

Granularity in the sense of this work refers to the ability to reference resources at different abstraction and instantiation levels that correspond to resource type, its subtype, particular instance or its property. The granularity in this work further refers to the resource representation that depends on particular context and

includes additional data elements [2] . Similarly, expressivity refers to the ability to establish and describe these interrelationships in a way that enables representations on an arbitrary level of complexity, while maintaining consistency and low integration overhead.

One of the main challenges in describing authorization extents is the level of details that need to be provided to clients. In standards, such as OAuth 2 and UMA, the structure and meaning of authorization scopes are left to the arbitrary definition of providers, preventing automated reuse, structuring and reasoning over these entities in cross-domain and web-wide context [6, 8].

This non-definition in these standards is left for the purpose – it is hard to catch descriptions of resources across a diverse range of the systems and integrate them in a long-term applicable standard that aims to be adopted in a broader scope. One of the related challenges is the syntactic perspective that dominates in a typical protocol or datatype standardization process.

In our proposal we approach this challenge by relying on an additional layer that enriches syntaxfocused structures with an additional semantic layer that enables flexible definition and exchange of concepts while maintaining syntactical requirements. In this work, we further aim to address the overall challenge by providing the framework that enables structuring and reusing of authorization extents on a large scale, allowing an automated authorization and security policy management across the diverse range of platforms.

2.2. General model

Figure 1 provides a simplified graphical overview of refined DASP-Core vocabulary for definition and enforcement of authorization extents. This description corresponds to TBox data model [11], representing abstract classes and their relationships.

The particular implementations instantiate these classes and establish their inter-relations by reusing provided object properties and by following predefined constraints in the form of ranges and domains. Additionally, data properties are defined by reusing and refining our DASP-Core vocabulary [7, 6] and general properties defined in RDFS vocabulary [12].

2 Consisting parts which are a part of the resource in a whole. For instance, an API resource is delivered as a monolithic (atomic) data structure, usually as a JSON or XML document. The constituting elements of this resource are considered as *data elements* in our work, which envisages contextual evaluation or resource transformation based on its properties and/or the properties of its consisting parts.

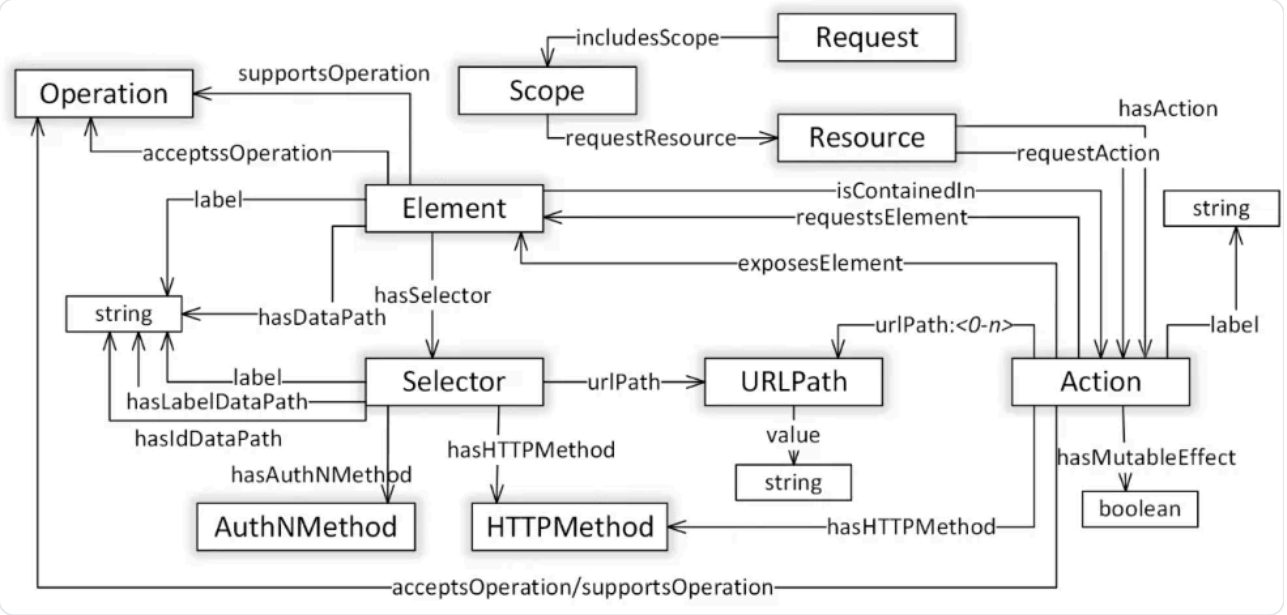


Figure 1: TBox-model defined in DASP-Core vocabulary

2.3. Definition of an authorization token

Following the model provided in Figure 1, the definition of authorization token starts from the `Request` class, which describes a request coined by the client and sent to the service provider or resource owner for authorization. This `Request` may contain one or more authorization `Scopes` . Each scope structures an authorization extent that describes requires and acceptable `Resources`, `Actions`, `Elements` and `Operations [3]` .

Starting from the particular `Scope`, it references the requested `Resources` . Each resource represents the resource exposed by the service provider to the clients. The resource is hence an abstract representation of something that the service provider designates as a resource or an asset. In a standard case, this resource is accessible and referenced via RESTful API and can be subjected to different `Actions` . In terms of RESTful paradigm, each Action may correspond to a particular set consisting of an `HTTP method`, supported operations on `Action` and a range of `Elements` exposed or processed in the form of that `Action` .

While the service provider references exposed actions in the form of `hasAction` property, clients structure authorization request by referencing these actions using `requestAction` property. This way, the distinction is made between available and requested actions, depending on the perspective (requesting client or providing service).

In the further process, the client may additionally refine its authorization request by providing more parameters on requested `Elements` or accepted operations on a level of `Action` or `Element` . In this sense, an `Element` corresponds to the referenceable data record provided with the execution of an `Action` . In the case of RESTful resources, an `Element` may represent a consisting part of API response provided over particular `Action` .

2.4. Description of vocabulary elements

In the following tables we provide the short descriptions of object classes, properties and data properties of provided vocabulary. These correspond to a TBox model [11] that supports the creation of authorization scopes.

3 We apply monospaced typeface to refer to the classes and properties provided in used vocabulary By instantiating these classes and their relationships in the form of directed graph [12, 13], the clients define the authorizations requested from resource owners or service providers.

Class	Description
`Action`	*Action is an entity that represents the activity that can be executed on a resource*
`Element`	*Element is a consisting part of a resource*
`HTTPMethod`	*Represents standard HTTP methods.*
`Operation`	*References the operation that can be executed over the particular element or action*
`Request`	*Describes HTTPMethod supported by the action*
`Resource`	*Resource is a basic type that represents retrievable resource by the means of API*
`Scope`	*Denotes the structured scope that is included in the authorization request*
`Selector`	*Selector represents a way to retrieve instances of the Element that can be later used to represent or reason over its properties.*
`URLPath`	*URLPath describes a part of an URL. Several URLPaths concatenated in their original order form a complete URL Path.*
`AuthNMethod`	*Authentication method (abstract class for future developments)*

Table 1: Class types defined in DASP-Core vocabulary

Property	Description
`supportsOperation`	*Denotes operations that can be executed on entity (exposed by service)*
`acceptsOperation`	*Denotes operation that can be executed on entity or its part, which is acceptable by the client requesting authorization*
`exposesElement`	*References the action that is supported by the resource*
`isContainedIn`	*Reverse property that references the action exposing an Element*
`hasAction`	*Describes HTTPMethod supported by the action*
`hasHTTPMethod`	*References parts of URL path that is used to initiate action. The complete URL path consists of its parts (1...n)*
`hasURLPath(1-…)`	*Denotes the structured scope that is included in the authorization request*
`includesScope`	*Denotes the resource that is requested with authorization request*
`requestResource`	*References the resource requested in the authorization request (referenced by scope)*
`requestAction`	*Denotes the action that is requested in the authorization request (referenced by resource)*
`requestElement`	
`hasMutableEffect`	*States if the action has mutable effects on its exposed Elements*

Table 2: Object properties defined in DASP-Core vocabulary

Property	Description
`hasDataSegment`	*Describes data path segment of an Element*
`hasDataPath`	*Describes data path used to retrieve an Element (subproperties such as XML and JSON Path)*
`hasXMLPath`	*Describes XML path used to retrieve an Element*
`hasJSONPath`	*Describes JSONPath used to retrieve an Element*

Table 3: Data properties from DASP-Core vocabulary

2.5. Sample model

Figure 2 provides a graphical representation of a sample service model based on DASP-Core vocabulary [12, 7, 6]. This model is realized as an ABox description [11, 13] of particular service and its endpoints.

By looking at this example we may observe that the service provider exposes a `Resource` labeled as a message. This resource supports retrieval `Action`, which is characterized using HTTP GET request and executed on URL described by `urlPath` properties. As the action has `mutableEffect` set as false, this implies that its execution does not change target resource on the service provider. Furthermore, this implies that supported operation may be executed in a post-processing step, prior to the delivery of data to the client.

In the further description, the `id` element is joined to the `Selector`, a class that enables live retrieval of available options. Its instantiation is practically used by the resource owners to retrieve available

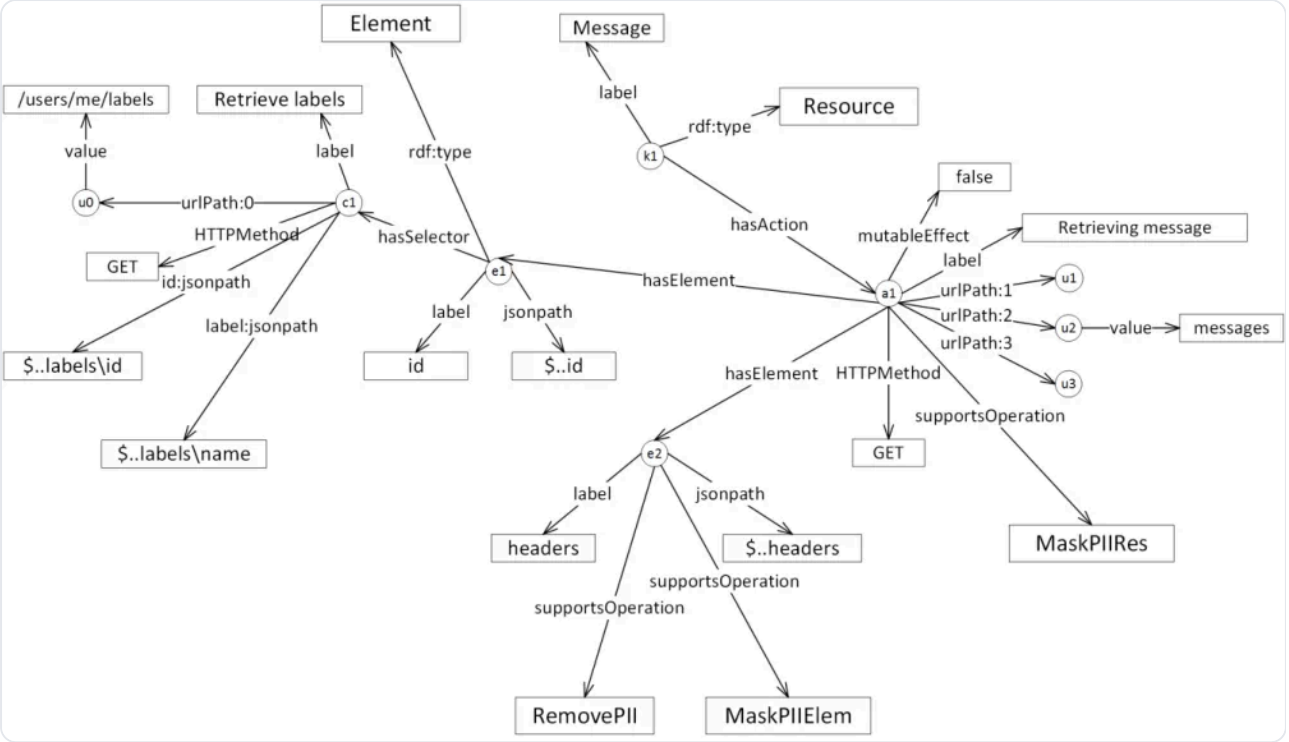


Figure 2: Sample ABox-based instantiation of resource description The described resource (message), when accessed using retrieval action, exposes two elements in its response. The first element is a `header` of the message, and the second its `id` . Hence, in this description, `Elements` provide a way to describe and alter the monolithic representation of action's response. By providing this information, we enable clients to semantically and structurally assess the provided results and restrict their actions on particular parts that are relevant to clients' use cases. The value of this element is accessible by applying data path property, instantiated as JSON Path or XML Path vector.

resource options or their instantiations, enabling the restriction of the authorized access down to a particular instance. Similarly, the references to `URLPath` elements in the `Action` descriptions may be used by the clients to autonomously retrieve requested `Resources` during the operational phase, once the authorization has been consented by the resource owner.

3. Prototype

In this section, we describe the library for management of authorization extents and the prototype of web-based application used to structure and manage authorizations based on DASP framework. Both of these assets were developed in the course of this project with the purpose to demonstrate the overall concept and provide a basis for its evaluation.

3.1. Common library

The common library used for manipulation of authorization tokens provides the basic functionality for the instantiation, management, search and extraction of authorization tokens. This library consists of Java classes and packages [4] that provide interfaces for management of tokens and additional tools for manipulation, transformation and processing of RDF and OWL data [12, 13].

Additionally, the library includes vocabularies that contain abstract resource models and sample service and token models encoded in RDF/XML and JSON-LD [14] formats.

3.2. Management interface

The prototype implementation provided in the course of this project is developed using Java/Scala Play Framework 2.4 [5] The project depends on and includes additional external libraries, including the family of `Apache Derby`, `JSONLD-Java`, `Apache Jena`, `SemWeb4J`, `Jersey Security`, `Jersey Core` and `Jayway JSONPath` families of dependencies.

Figure 3 describes the architecture of the prototype system. This architecture includes the model provided by `Play Framework`, which provides MVC-based, flexible and efficient application framework for rapid development of scalable and reactive applications.

The `Router` is an entry point for users and applications, providing a RESTful interface coupled with application `Controllers`, each dedicated to specific endpoint and functionality. The `Controllers` reuse the common library, which provides different `Models` and utility tools for manipulation of authorization tokens. `Controllers` furthermore rely on different `Views`, used to render pages for end-users and provide client-based `JavaScript` functions for manipulation and API calls. The views are based on `Twirl framework`, which relies on `Scala` constructs for efficient rendering and reuse of resources. The responses to end users are served both by `Controllers` and `Views`, depending on particular functionality.

3.3. Functionality

The prototype in this section provides the web application that enables users to construct and inspect authorization tokens based on DASP framework's conceptual model. By using this system users are able to construct expressive, granular and context-sensitive authorization tokens that can be used as authorization requests in the scope of different protocols, including OAuth 2 [5], UMA [9], DASPspecific and other interactions.

4 Packages `daspcommon` and `model` in archive 5 http://playframework.org/ The details on functionality and particular actions supported in the prototype are further described on the basis of example case scenario provided in Section 4.

Figure 3: Architecture of Ctx-Gateway

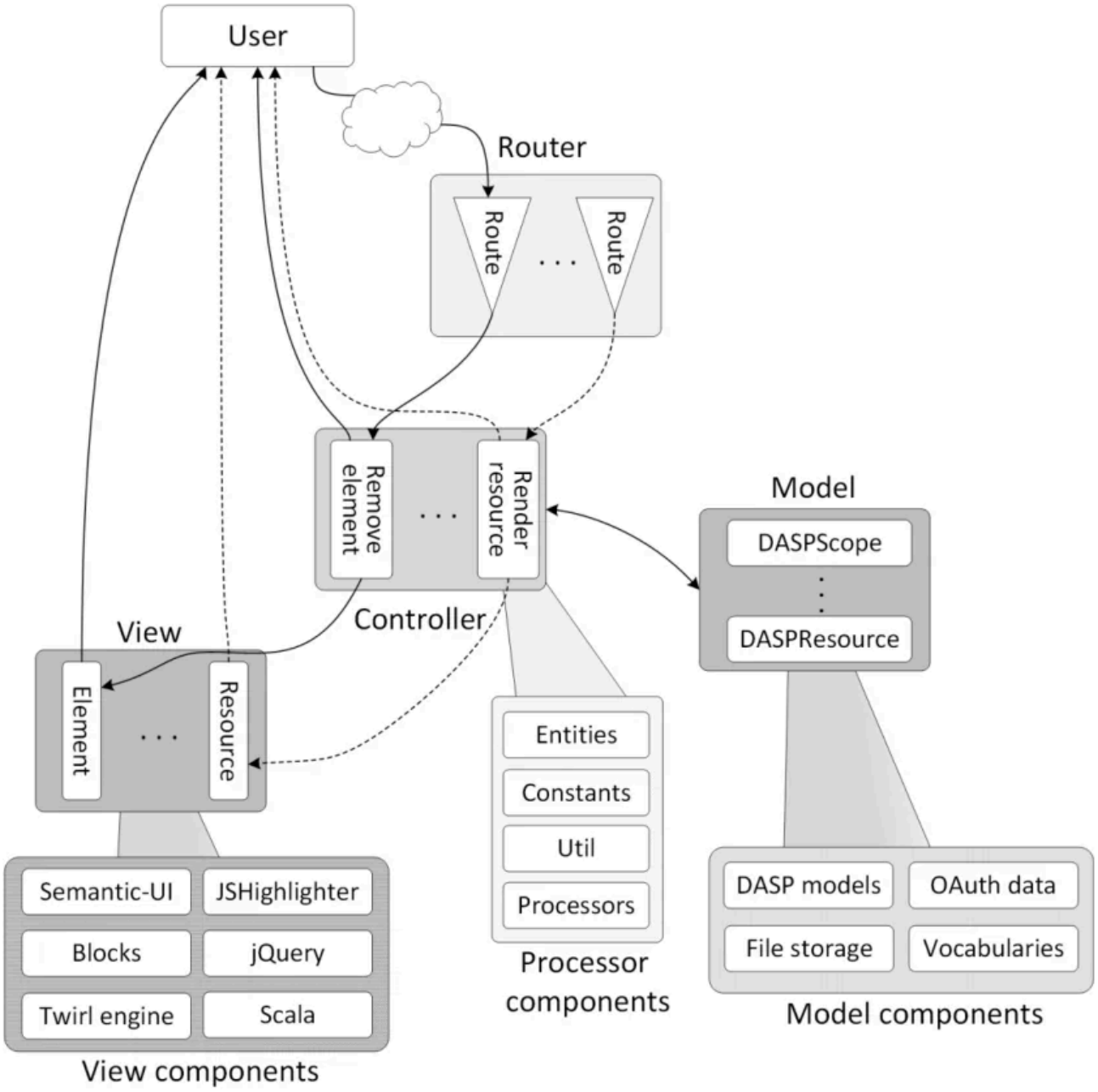
4. Sample Case Scenario

In this section, we present a case study of defining an authorization extents using DASP framework's components that can be employed to describe authorization scopes in various protocols, such as OAuth 2.0 or UMA.

4.1. Granular authorization for multiple resources

We first present a GUI interface walkthrough needed to define an authorization extent with the following requirements defined in natural language:

o *Requested is the access to resources Message label and Email message, explicitly requiring the particular extent of provided data and its context-specific transformations. These*



transformations are executed both on the level of returned data set and its particular consisting elements. This general requirement is refined with the following resource-specific explicit requirements and acceptable transformations:

Message label:

o *Allow the retrieval of existing labels, adding new labels and updating of existing labels* o *When applying Message label retrieval, explicitly accept the operation Contextual filtering of complete returned dataset* o *When applying Message label retrieval, explicitly request the following data fields (elements) of Message label to be included in provided dataset: Total unread messages and Total Messages* o *Allow Total unread messages data field of resulting data set to be cleaned prior to delivery* Email message:

o *Allow retrieving email messages in general* o *Explicitly request the following data fields (elements) of Email message to be included in the response: Thread id, Internal date and Message snippet* o *Allow specific operation of content removal to be executed over Thread id data field* This request, therefore, represents a complex authorization extent that asks for access to two different resource types, with supporting different activities for each of them. Furthermore, this request states explicit data requirements for each of these resources. In a standard case, the returning data set of both resources is provided as a whole, containing the information that is not relevant to use case as well.

By stating explicit data requirements and acceptable data transformations, both on the level of complete resource and its consisting data elements, the requesting client communicates its required acceptable access rights in several dimensions. In the further workflow these requirements may be accepted by the resource owner fully, or further constrained to allow for other transformations or restrictions according to the preference of a resource owner and available options at the service provider.

4.2. User-initiated structuring of the requested authorization

In this section, we take the requirements presented in the previous section and present how they can be structured by using the provided web-based tool.

For usability reasons and to gain a practical overview of definition phases and requirements, this activity is organized in the five steps, as presented in the following figures. Although the transition between the steps is visually presented as a linear process, the prototype supports arbitrary refinement and selection of the steps in a non-linear manner with limited functionality [6] .

6 For instance, we cannot select acceptable operations on elements if the elements were not selected in the previous step. However, we can allow user to traverse back to select additional elements and refine existing selections at each step.

In the first step, as presented in Figure 4, the user selects requested resources. These resources are exposed by the service provider and read by the application. The requested resources are presented in the lower part of the screen as white boxes.

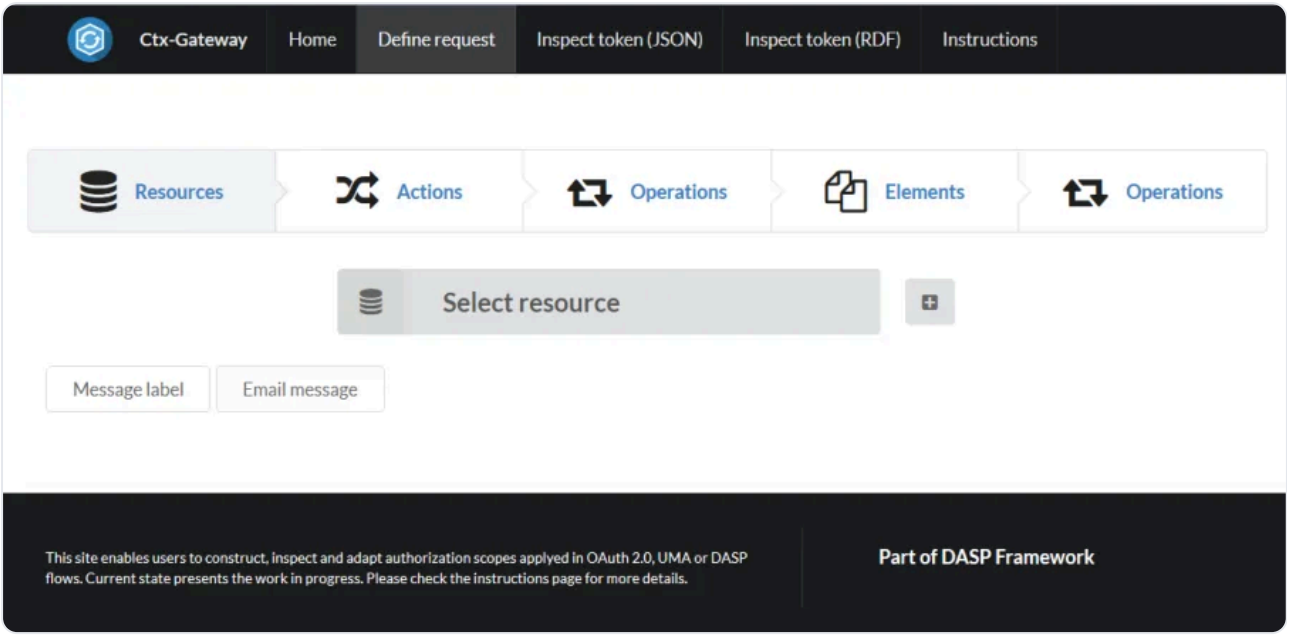


Figure 5: Selecting actions for each requested resource In the following step the user chooses the acceptable operations over each resource set. These operations may be executed in the first phase of a request [7], or in the second request phase [8] . Furthermore, they include the whole request or response set, which corresponds to what is exposed 7 For instance, the PUT/POST data sent to API by the client may be subjected to a pre-processing, prior to its provision to the target service 8 The response of target service may be post-processed, prior to its delivery to the client

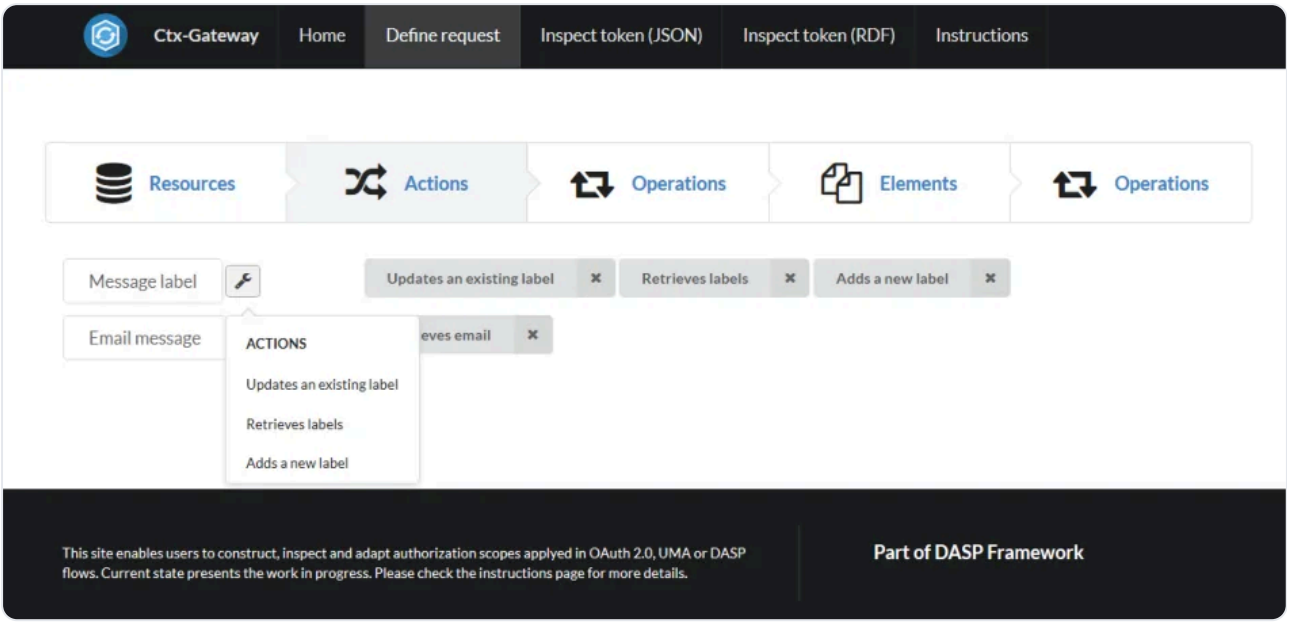


Figure 4: Selecting resources After the initial selection of resources is done, the user continues with the second step, to select requested actions over each of requested resources. These actions are presented next to each selected resource in the form of a dropdown with selectable elements, as shown in Figure 5. The selected actions are presented in the right part of the screen.

or requested from the API. The example list of available operations and their active selection for Message label and contextual filtering of its response is presented in Figure 6.

By explicitly stating a set of acceptable operations, the user (client) confirms that their execution is acceptable for its use case. One of the typical scenarios for this case is the filtering of redundant data that is normally provided by the service provider, but which is not relevant for connecting client. This way, the data sharing process can conform to the *principle of least privilege* [15], allowing the minimal *information footprint* to be provided to the accessing client.

By organizing supported transformative operations and enhancing the expressivity of security policies exposed to the clients (or resource owners), using this approach service providers can allow *context-specific execution* of acceptable operations, enabling the dynamic and selective data transformation that relates to a particular client, environment, data parameter, or another variable.

Figure 6: Selecting operations applicable over each action Following the selection of acceptable operations, the clients may explicitly request the data fields that have to be included in each interaction. Referred as *elements*, these data fields represent an abstraction of data sets that applies additional granularity level that is not provided in standard API descriptions.

By following the example of the email message, the provider may allow an API operation to return the whole message, available as a JSON structure that consists of several message fields, such as header, time, sender, recipient, spam-score etc. This structure is normally provided as a whole, allowing clients to retrieve only the whole data and not its particular needing parts.

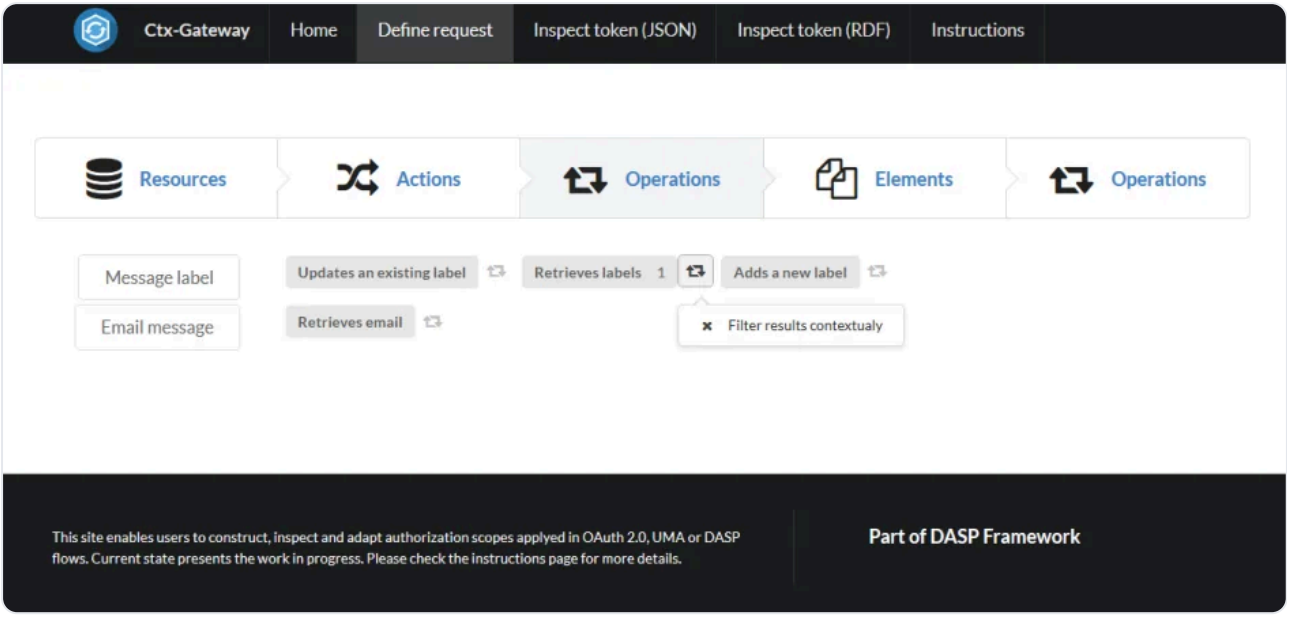


Figure 7: Selecting required elements for each action In our solution we enable service providers to expose descriptions of such data sets and to allow the clients to express the requirement for particular parts (data fields) of each resource on an additional level of granularity, without the need to re-engineer API interfaces or introduce additional options. Following this possibility and the example of the email message, the clients are able to explicitly state the requirement to be provided with message header or the list of senders, and to ignore other parts of the resource. Based on this requirement, the resource owner or service provider may include automated removal or processing of other parts of the data set, specifically for each accessing client and context.

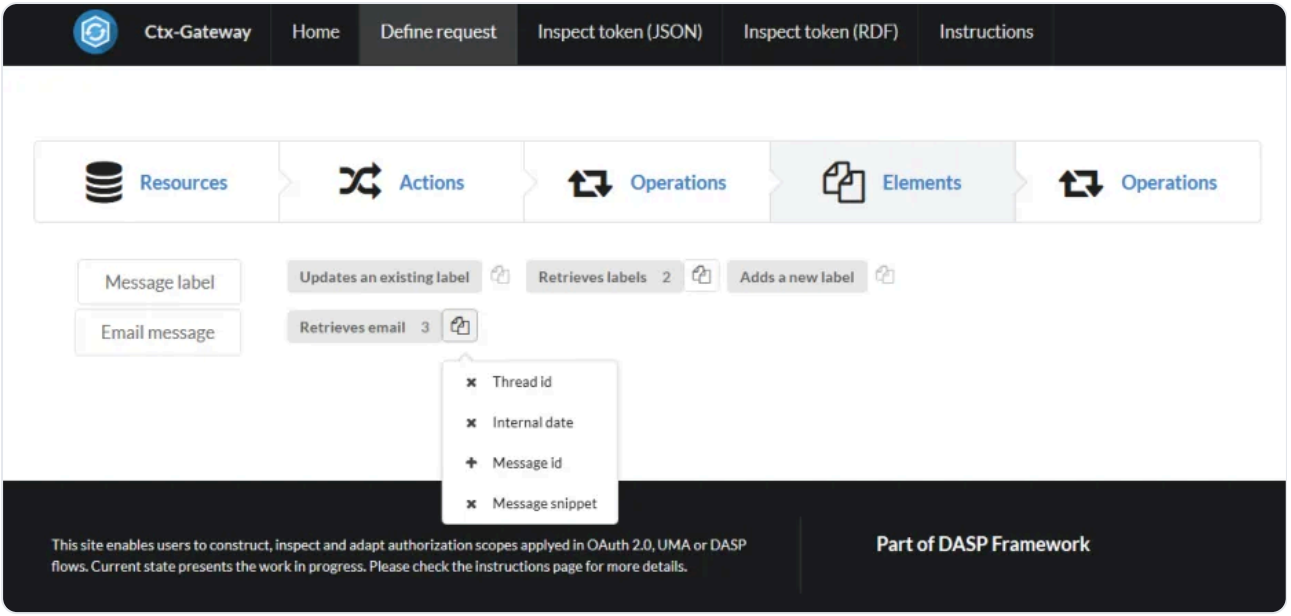
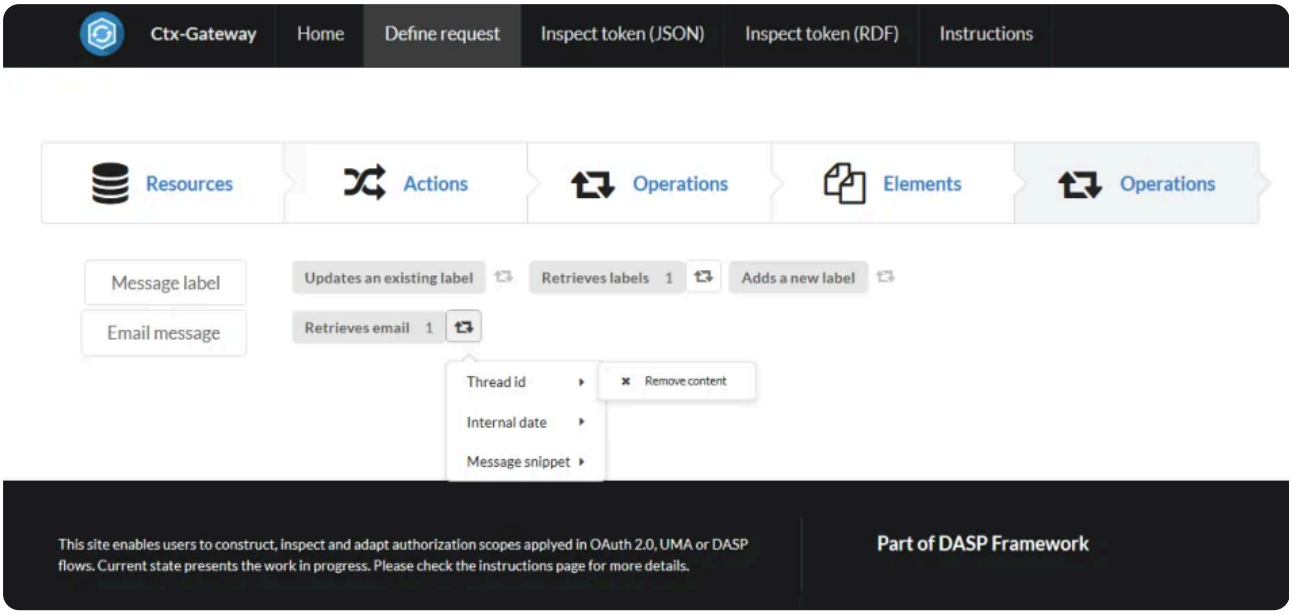


Figure 8: Selecting acceptable operations applicable over each element Finally, as illustrated in Figure 8, the user is able to check a range of supported operations for each requested element of each requested operation. In this step, the user chooses the operations that will be stated as acceptable operations for each element.

The interface presented on Figure 7 hence shows an illustrative list of selected and selectable elements that describe an email message. In this selection, the user explicitly states that required are *Thread id*, *Internal date* and *Message snippet*, as introduced in Section 4.1.



Compared to step 3, the difference is the level of granularity. While step 3 considers operations over complete requests and responses, this step refers to the operations available for particular data elements of each response.

The shaded boxes in Figure 8 imply that particular action does not contain elements or elements with related transformative operations to be executed in the authorization scope. This example illustrates the potential to expose and request the resources and their models in different descriptive levels and details

that conform to the particular deployment and services of each service provider.

4.3. Machine-readable representation of authorization extent

In this section, we provide the machine-readable version of authorization request specified in Section 4.1 and structured using web-based graphical interface using the process presented in Section 4.2.



Figure 9: JSON-based representation of authorization extent By applying this example in the case of OAuth 2.0 and UMA, the relevant fields to be included in the access scope would be the part of the graph that starts with request:newscopre, including its children. Hence, we show how our structure can be reused among different frameworks and protocols.



Figure 9: shows the authorization extent included as an authorization scope in the access request of DASP framework. This structure includes additional nodes not presented in the steps executed in Section 4.3, which refers to specific representations presented in DASP framework that relate to the authorization request, and its included scopes. In this framework, each structure may consist of several requests, and each of them of several authorization extents (scopes), whose definition is out of the scope of this work.

The format presented in Figure 9 conforms to compacted form of JSON-LD [14]. This form allows the good level of human readability of resources, as well as their efficient processing and inclusion in automated systems, including the web-based exchange of linked data. Other potential formats can include RDF/XML family, which is typically used for graphs representations in the semantic web.

We now provide the explanations of the consisting parts of the structure, according to the labeling presented on Figure 9.

(1) This part defines main namespaces that will be applied in the graph.

(2) According to DASP framework, the graph starts with a new instance of `Request` class, defined in DASP-Core vocabulary. This request includes a new instance of `Scope` class, which corresponds to authorization extent (scope).

(3) The instance of a scope states requested resources, referencing the class instantiations from referred `Message` vocabulary. These are instances of `Resource` class, defined in DASP-Core vocabulary, and exposed at service provider as service description.

(4) This node describes requested `Label` resource, stating the actions requested in the authorization. These actions are instances of `Action` class, provided in the service description.

(5) Analogously, the requested action for `Message` resource is provided.

(6) This part further clarifies which elements are requested for action that relates to retrieval of email messages. The descriptions above extend existing model of `Message` [9] with statements over acceptable operations for exposed elements.

(7) Finally, the request states imported vocabularies from external parties. `DASP-Core` refers to the common vocabulary used to define requests and describe the scopes, while `Message` 9 Provided by the service provider refers to model exposed by service provider to describe its resources, interface, and functionality.

5. Conclusion

This technical report presents the following development iteration of the framework that integrates cross-system policy and resource management, enabling machine-to-machine awareness of resources distributed and shared across heterogeneous cloud systems. In the presented work we have focused on authorization descriptions which are used across a range of frameworks to ask and consent resource and service authorizations in resource sharing interactions.

We have presented a model of the structure that describes authorizations in an expressive, granular and interoperable manner. In order to demonstrate our approach and enable its further evaluation in a range of use cases, we have developed a library and web-based management interface used to structure and inspect proposed authorization structure.

By elaborating on particular process flows, user interface and structure representation, we have illustrated its capability and integration in target environments. By applying a sample case scenario we have shown how our structure enables the description of user requirements in complex scenarios that involve multiple parties and separate evolving platforms.

In the future work we aim to focus on other aspects of proposed framework, demonstrating the integration and application of proposed structure in a range of cases, including transformation to security policies, evaluation of these policies and integration with existing and diverse range of APIs.

6. References

[1] D. Vesset, H. D. Morris, B. Woo and N. Yezhkova, "World wide Big Data Technology and Services," IDC, 2012.

[2] B. Bilbao-Osorio, S. Dutta and B. Lanvin, "The global information technology report 2013," in *World Economic Forum*, 2013.

[3] European Commision, "Towards a thriving data-driven economy," European Commision, Brussels, 2014.

[4] J. Wettinger, U. Breitenbücher and F. Leymann, "Any2API - Automated APIfication," in *Proceedings of the 5th International Conference on Cloud Computing and Services Science*, 2015.

[5] D. Hardt, *The OAuth 2.0 authorization framework.,* 2012.

[6] B. Suzic, *Multidimensional Security Policies,* Graz: Zentrum für sichere Informationstechnologie - Austria (A-SIT), 2016.

[7] B. Suzic, "User-centered Security Management of API-based Data Integration Workflows,"

NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium, pp. 1233-1238, 2016.

[8] B. Suzic, “Securing integration of cloud services in cross-domain distributed environments,” in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, 2016.

[9] T. Hardjono, E. Maler, M. Machulak and D. Catalano, “User-managed access (UMA) profile of OAuth 2.0,” 2016.

[10] E. Maler, “Extending the Power of Consent with User-Managed Access: A Standard Architecture for Asynchronous, Centralizable, Internet-Scalable Consent,” in *Security and Privacy Workshops (SPW)*, 2015.

[11] R. Brachman, R. Fikes and H. J. Levesque, “Krypton: A functional approach to knowledge representation,” *Computer,* vol. 10, 1983.

[12] R. Cyganiak, D. Wood and M. Lanthaler, “RDF 1.1 concepts and abstract syntax,” W3C, 2014.

[13] W3C Owl Working Group, *OWL 2 Web Ontology Language Document Overview,* W3C, 2009.

[14] M. Sporny and L. Markus, *Json-ld 1.0-a json-based serialization for linked data,* W3C, 2014.

[15] F. B. Schneider, Least privilege and more, vol. Computer Systems, Springer, 2004.

[16] F. Schneider, *Least privilege and more,* Springer New York, 2004, pp. 253-258.

[17] A. Kertesz and S. Varadi, *Legal aspects of data protection in cloud federations,* Springer Berlin Heidelberg, 2014.

[18] G. Kelly and P. Hunt, *System for Cross-Domain Identity Management: Core Schema,* 2015.

[19] Kantara Initiative, *User managed access,* 2013.

[20] I. Salvadori and F. Siqueira, *A Maturity Model for Semantic RESTful Web APIs,* IEEE, 2015.

[21] L. D. Xu, *Enterprise systems: state-of-the-art and future trends,* IEEE, 2011.

[22] M. Pezzini and B. J. Lheureux, *Integration platform as a service: moving integration to the cloud,* Gartner, 2011.

[23] E. Rissanen, *eXtensible Access Control Markup Language (XACML) version 3.0,* OASIS, 2013.

[24] V. Hu, D. Richard Kuhn and D. Ferraiolo, *Attribute-based access control,* IEEE, 2015.

[25] R. Sandhu, *Role-based access control,* 1998.

[26] F. Liu, *NIST cloud computing reference architecture,* NIST, 2011.