User-centered Security Management of API-based Data Integration Workflows

Bojan Suzic
Institute for Applied Information Processing and Communications
Graz, Austria
Email: bojan.suzic@iaik.tugraz.at

***Abstract*—One of the consequences of the present adoption of cloud-based services among organizations is the increasing rate of outsourcing of business and technical functions to third parties. The recent approaches such as *cloud integration platforms (iPaaS)* facilitate this trend even further. In this scenario, users' resources distributed across different cloud systems are accessed, shared and processed completely in the cloud, at third-party premises, effectively transferring the execution of entire business processes to the cloud. In this work, we approach security challenges and issues that arise from data and resource integrations of such scale. Our contribution aims at advancing privacy and confidentiality in collaboration flows of distributed, cross-domain systems. We focus on the perspective of resource owners, enabling automated, structured discovery and security orchestration of their resources hosted at various cloud premises. We furthermore consider the perspective of integration clients that access and process distributed resources on behalf of resource owners, providing the model for discovery, integration and fine-grained constraints of sharing requests. Our contribution is examined on a basis of the focused prototype that allows proxy-based integration with existing systems and web authorization protocols.**

I. Introduction

The fast-paced development and broad adoption of cloud services enabled the enterprises to transform their businesses and practices, taking advantages of lowered entry barriers and accelerated innovation processes introduced by the new paradigm. By relying on component reuse and delivering the benefits of economy of scale even to small enterprises, cloud-based approaches contributed to the creation of new business and service models, strengthening the ties and dependence between different organizations. However, these developments and widespread acceptance of cloud model caused other impacts as well. Some of their consequences can be observed through multiplied complexity of organizational processes and additional security concerns raised with the adoption of new models. This especially applies to the overall governance and protection of personal information, corporate assets and processes in a highly-connected and dependable environment.

In this work we present our ongoing contribution that advances the security and legal conformance of cross-system integrations. Our approach takes advantage of rich expressivity of semantic technologies, putting them in the context of cross-domain data protection and security governance, aimed both to organizations and end users. We apply our contribution in the form of service, policy and request models in the use case that considers both emerging approach of *Integration Platforms as a Service* (iPaaS) and broadly adopted web authorization protocols. By developing and testing an initial prototype in a focused use-case, we examined the flexibility and seamless integration potential of our framework.

Outline: this work is organized as follows. In the second section, we revisit related work, establish the problem and state our contribution. We then introduce proposed approach, providing the high-level overview of our model, followed by detailed consideration of its building blocks. In fourth section we describe our prototype implementation. In the subsequent section we discuss our approach, followed by the conclusion and research roadmap.

II. Background and Related Work

A. Integration platforms

The approach of cloud-based integration got broader attention recently, as the products focused on integration and management of cloud services started to gain traction. The emergence of these services, however, does not imply the establishment of a new discipline. *Enterprise integration*, in its various forms, has been present for more than a decade [1]. Following the emergence of *iPaaS*, analysts tried to establish and define this service model. Pezzini et al. contributed in this direction, identifying *iPaaS* as a suite of cloud services enabling *development, execution and governance of integration flows connecting any combination of on-premises and cloud-based processes, services, applications and data within individual, or across multiple organizations* [2].

The analysis of functional and organizational aspects, as well as the detailed overview of integration platforms, technologies and challenges have been provided in [3]. Raj investigated challenges for SaaS and XaaS integrations, identifying *dynamic nature of SaaS interfaces*, *dynamic characteristics of metadata of SaaS solutions* and *data quality and integrity issues*. Other contributions that established the concepts and challenges in cloud-based service integrations have been provided in the works of Kleeberg et al. [4] and Baude et al. [5]. As they deal both with the integration of enterprise systems and processes, integration platforms often overlap or share similar issues with other concepts. These include *Cloud Brokerage*, *B2B Integration*, as well as *Enterprise Application Platforms* and *API Management*.

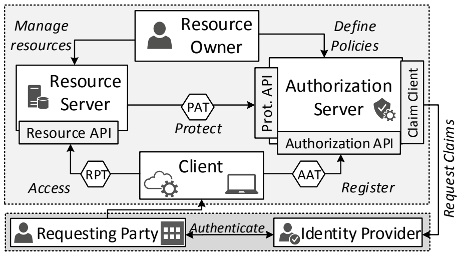

Fig. 1: Resource sharing using UMA flows

B. Protocols and Architectures for Web Authorization

In the focus of our work are data exchanges performed on the web, using HTTP protocol and interfaces. These interfaces are typically built according to *RESTful* architectural style, which currently represents one of the major approaches for web APIs. This can be observed by analyzing the directory maintained at *ProgrammableWeb*, where the vast majority of registered web APIs are based on RESTful style [6]. One of the major approaches to protect these interfaces is *OAuth 2.0*, a web authorization protocol that enables clients to access protected resources on behalf of a resource owner [9]. In OAuth 2.0 scenario, a *resource owner* authorizes a *client* to make requests to resources hosted by *resource server*. These interactions are governed by *authorization server*, which issues access tokens to the client on the basis of a resource owner consent. Currently, OAuth 2.0 is dominantly adopted approach for web API protection. Building on OAuth 2.0, *UMA* [10] represents an emerging protocol that further refines its flows, processes and APIs, focusing on a protection of resources in distributed environments. Derived from an initial work of Machulak [7], UMA introduces *resource owner policies* and *distributed architecture* by separating authorization and resource server and defining additional APIs for that purpose [8]. Fig. 1 depicts the architectural model of UMA.

Another approach that deals with authorization, considering distributed and enterprise-oriented perspective, is *XACML* [11], an XML-based declarative language standardized by OASIS. XACML provides the means to specify access control policies based on an extensive set of built-in data types, functions, combining algorithms and supported profiles [11]. Primary elements of XACML are *rule*, *policy* and *policy set*. Due to its focus towards intra-enterprise processes and complex infrastructure, XACML is rarely applied in large-scale API integrations [6].

C. Integration scenario

A typical integration platform scenario reuses the building blocks from a general API integration, extending the process to span across different entities and platforms. In its base form, this scenario encompasses the use of organizational accounts at third party providers, with the goal to execute predefined tasks. The related flow is commonly realized using Web APIs exposed by *service providers* and secured using widely adopted mechanisms, such as OAuth 2.0 protocol [9]. However, the complexity added by *iPaaS* in comparison to standard approaches is manifold. First, they rely on the execution of automated task batches organized in *workflows*, assuming the subsequent accesses to multiple *external* systems, in a *context-dependent* manner. The parts of this process are, contrary to typical API integrations, completely executed in premises of different and unrelated *third party organizations*. They both *consume* and *process* client's data in a range of *heterogeneous* systems, out of the client's control. The scope of these interactions raises the issue of effective management, as well as *secure data propagation* and overall *accountability* of interactions executed across various organizations and clouds.

In Fig. 2 we show an example activity performed by an integration platform in the cloud. The *workflow* that represents organizational business process is executed in the scope of *cloud integration platform* that connects to *on-premise* organizational systems, various cloud services and external organizations. In this example, the platform connects to organizational *Live* email account, retrieves and processes the messages and then, according to predefined triggers, consumes the interface on *Salesforce* and uploads data to organizational *GDrive* storage. The last steps of the workflow relate to controlled data exchange with external entities.

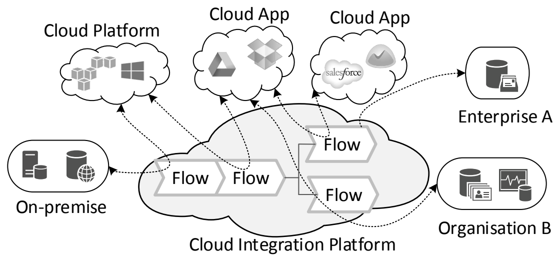

Fig. 2: Workflow-based integration of services

D. Problem statement and contribution

The traditional *enterprise integration* model has been customized for on-premise setups, dealing with the security requirements from intra-organizational perspective. The transition to the cloud and broad application of *API-based integrations* increased the attack surface, raising the need to consider inter-organizational communications as well. Commonly applied protocol for such integrations, OAuth 2.0, defines only *coarse-grained*, *static* and *platform-specific access scopes*, which do not provide adequate means to govern accesses in multiple connected environments. Its UMA profile introduces the concepts of *security policies* and *distributed enforcement*, but misses defining their structures and semantics, effectively preventing the interoperability between different providers.

Additional details on OAuth 2.0 and UMA flows are available in our previous work [14], which demonstrated the non-conformance with *least privilege principle* [15] and numerous legal regulations [16]. In this sense we refer to *data minimisation principle* introduced by *Directive 95/46/EC* and *Regulation 45/2001* of European Commission, as well as related domain-specific opinions on *cloud* [17] and *IoT* [18], which present a legal motivation for our work.

In this work, we contribute with a model that deals with confidentiality and privacy issues of API-based data and resource integrations. We establish common *interoperability framework* for definition and exchange of *service models*, *policies* and *requests* in cross-domain collaborations. This model enriches security management of resources by introducing *inherent semantics* based on OWL [20] language and RDF Schema [21]. Based on these descriptions, we enable bidirectional model discovery between clients and servers, enabling them both to structure their requests and transform responses on a privacy and confidentiality conforming way, aiming at the establishment of *interoperable security* on the level of *services*, *collaboration requests* and *policies*.

III. Proposed Approach

We consider collaborative environments that host repetitive interactions between diverse entities, operating in various domains and jurisdictions. In this setting, user data or services are dispersed across these entities, being accessed and consumed as a part of various third-party workflows. In a particular case of cloud integration platforms, entire business processes can be outsourced to external entities, establishing both data sharing and processing out of organizational, physical control. Our proposed solution tackles the problem of data privacy and confidentiality in such processes. We rely on a model-based approach that consolidates and structures cross-service resource sharing by leveraging the semantic interoperability layer for cross-domain processes.

In the first step, we propose the modeling of services, policies and requests using common model vocabularies. The proposed approach enables the entities across interaction chain: (1) to derive the semantics and capabilities of the adjacent services, (2) to shape the requests that conform to exposed models, capabilities and their restrictions and (3) to ingest the delivered data in a structured way, provided with enhanced semantics and annotations. Additionally, our proposal enables the owners of resources collocated in the cloud: (1) to discover their resources on different systems and derive their capabilities, semantics and possible restrictions, and (2) to set security policies relevant for their resources in an interoperable way that is portable across different platforms. By relying on these capabilities, resource owners can employ additional services or tools for automated analysis, audit or orchestration of security-related properties of their distributed resources. In the following subsections, we provide more details on each building block of our model.

A. Exposing services and capabilities

This block establishes the structured model of services and resources exposed by the cloud provider. In a typical scenario, service providers present APIs that are specific to their use-cases and internal models, resulting with a broad range of different implementations and styles. This adds development and maintenance overhead globally, and impacts the interoperability between systems, as API clients need to be separately created for each exposed system and continuously maintained over its life cycle.

As APIs descriptions are provided mostly as *out-of-the-band contracts*, partially delivered in the form of human-readable documentation, practically the only possibilities for the developers to integrate remote functionalities and resources are to manually implement service-specific API clients or to rely on provided SDKs. These approaches, however, hinder the interoperability and automation of services on a broader scale, beyond the point-to-point integrations.

TABLE I: Information model concepts for resource sharing

Abstract Entity Class: *Service*	
Cloud Service, Email Service, Storage Service	The primary point in service provider's offer to clients. The members of this class denote different types of services that correspond to a particular use case.
Abstract Entity Class: *Resource*	
Object, Drive, Directory, File, Document, Media, Email, MsgBody, MsgHeader	Resource is a consumable entity exposed by service, which can be present in a domain and purpose-specific instantiations. The members of this class represent entities at different levels of abstractions, allowing specification of the granularity of resource sharing and application of (pre)processing operations at multiple abstraction levels.
Abstract Entity Class: *Operation*	
RemovePII, Encrypt, Mask	Operations are actions that can be executed on resources prior to their sharing. They effectively provide a resource view adjusted to a particular context.

TABLE II: Object properties

Property	Description
exposes	Entity that service exposes to clients using web API
contains, isPartOf	Relationships between entities stating constitutive (hierarchical) relationship between instances
subClassOf	An abstract description of possible inter-class relations
type	Class type of particular entity instance
requestsAccess	Used to relate an access requests with a service
supportsOperation	Denotes operations that can be executed on entity
acceptsOperation	Denotes operations that are accepted by the client
acceptsSubtype	Subtype of resources satisfying a client request

In order to enable heterogeneous systems from different domains to communicate effectively and engage in the process of autonomous, machine-based integrations, in this work we extend the dominant RESTful approach with service description capability. In our proposal, cross-system API modeling is established by using common *semantic interoperability layer*, which introduces the concepts and relations that deal with various use cases. By relying on the concepts defined from the external perspective, provided as a common framework, service providers enable automated integration and management of exposed resources, allowing the interoperability beyond of their local premises or models.

The primary building block of *semantic interoperability layer* introduced in this work are its *core* and *domain specific* vocabularies. Table I presents the main concepts from this framework. It describes three abstract categories of entities, providing a list of related concepts for each entity and their roles in the framework. By using the object properties presented in Table II, providers can describe each of their exposed resources and formalize their hierarchical organizations and relationships. This allows heterogeneous clients in different domains to integrate provided resources and align them with internal processes by reusing existing concepts from known framework.

In the current iteration of our work, available are the *core*

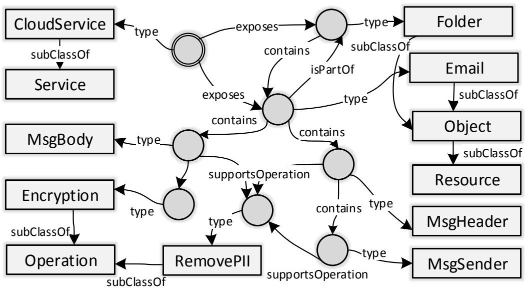

Fig. 3: Service model description

and domain vocabularies for modeling of *storage* and *email* services. In Fig. 3 we reuse their concepts and relationships, as presented in Table I and Table II, and model the typical cloud email service. We rely on graph-based descriptions, distinguishing between concepts, represented through classes, object and data properties, and their instantiations, represented as graph nodes in the figure. Based on that, service providers describe their systems by (1) relying on common classes and properties, and (2) by instantiating the concepts that apply in their particular case. Those models are then further used in other processes, as described in the following sections.

B. Describing security policy capabilities

Security policies define security requirements for a given system [12]. In the context of cross-system collaborations, by applying the term *security policy* as a general concept, we focus on its role in the enforcement of access control in inter-domain collaborations. Existing and broadly adopted web authorization frameworks exhibit different approaches to access control management. For instance, OAuth 2.0 does not explicitly assume the existence of security policies, but relies on user's consent based on *access scope* for access control enforcement. Related UMA profile, however, introduces the notion of a *policy*, defining it as a *set of configuration parameters at authorization server that effect the access management of resources* [10]. UMA, however, does not specify the format, model and application of security policies, leaving most aspects to be provided by particular implementations.

In our proposal, we go one step further by providing the framework for specification and exchange of security policies. Our goal is to enable systems residing in heterogeneous environments to expose their policy models and enable policy interactions to be performed beyond the scope of a platform. For this purpose, we rely on vocabularies provided in *semantic interoperability framework*, extending them with security policy related terms. Table III provides the overview of these concepts established in the scope of our work.

Structure. Our proposed model of data sharing policies partially reuses the terminology defined in [19] and the concepts from XACML language [11]. It considers the *rules* as a basic building block, organized in sets denoting the *policies*. Each rule states its *target* (object), *action*, *subject* and alternatively *context* and *obligation*. The decision over a policy set consisting of multiple rules is done by applying its predefined *combinatorial algorithm*.

Abstract Entity Class: *SecPolicy*	
SecPolicy	The primary point in the definition of security policies. One policy contains one or more rules with combination algorithm associated.

Abstract Entity Class: *SecRule*	
SecRule	Defines a node used to connect policy *subject*, *target*, *action*, *condition* and *obligation*.

Abstract Entity Class: *PolicySubject*	
RegisteredClient, TokenBearer	Includes a range of subjects, allowing the integration with different *access control models* and mechanisms such as *OAuth 2.0*.

Abstract Entity Class: *CombAlg*	
PermitOverride, DenyOverride	Defines a range of algorithms applied in the process of making of *policy decision*.

Abstract Entity Class: *Context*	
TimeCon, SystemCon, RiskCon	Establishes a range of conditions as a multi-level classes whose instances may be evaluated in the process of making of *policy decision*.

Abstract Entity Class: *Obligation*	
LogPre, LogPost, CustView	This category defines a range of obligations that may be applied in the process of *policy enforcement*, both in *pre* and *post* resource delivery stages. This may include resource transformation using *operations* provided in the scope of *resource sharing model*.

Abstract Entity Class: *SecAction*	
ActionDelete, ActionRead, ActionUpdate, ActionCreate	This class includes different actions that may be defined on a resource.

TABLE III: Information model concepts for security policy

Expressivity and granularity. Considering the potential complexity of representations of granular resources, we restrict the modeling of policies to each separate resource and entity. In our opinion, this is especially necessary when considering a range of possible relations between resources, subresources and applicable actions, as well as limitations of representation formats, underlying languages and supporting tools. We, therefore, model security policies available at the service provider and applicable to a particular resource in a similar way as the services are modeled, as provided in the previous section. On Fig. 4 we visually depict the excerpt of a model of security policy applicable to *Email* resource, where the nodes represent entity instances, arrows relationships, and rectangles related and generalized class concepts.

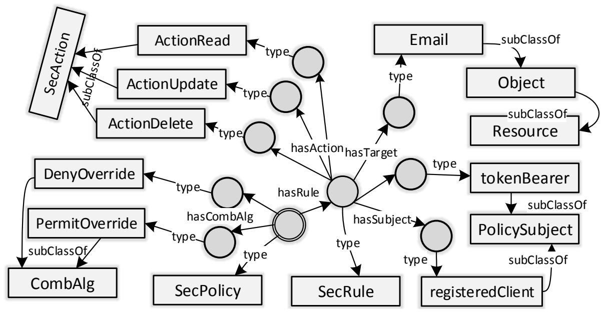

Fig. 4: Model of security policy, excerpt for Email

C. Defining security policies

Once the system provides a structured model of security policies applicable to its resources, resource owners and their automated clients are allowed to separately create, retrieve and update these policies on a remote system. This process is performed by applying following steps:

1. *Discovering policy model.* Clients retrieve models of policies applicable to particular resources. These representations state a range of supported actions, concepts of subjects, as well as obligations and contextual restrictions. They can be delivered in the following ways:
   a) *Embedded with resources.* Models are retrieved together with resources by using web API endpoints and providing the models as separate descriptions that reference each related resource.
   b) *Separate policy endpoints.* Additional, dedicated API endpoint is provided for policy management.
2. *Defining policy.* Based on the resource-specific model, the client determines the policies and apply them.

D. Requesting resources

By relying on previously discovered service model descriptions, third party clients request data and service access from cloud providers, on behalf of data owners and users. The scope of their requests depends on *expressivity* of the provided model, *exposed capabilities* and *client's processing requirements*. Using provider-specific service model, clients are able to: (1) request only a subset of exposed resources that conforms to their use case, and (2) express the transformations that do not impact their processing workflow significantly.

Based on the inherent semantics of common framework models, clients are able to understand the type of resources, as well as the restrictions and operations applied to them.

The extent of supported possibilities can be illustrated by looking at request shown in Fig. 5. In this example, the client reuses common *interoperability framework* and prepares a request that conforms to its particular use case. Accordingly, it requests access to an *email resource*, stating that this resource can be reduced to the form of its constituent, an *email message header*. It furthermore declares that the resource can be subjected to additional processing in the form of *removal of personally identifiable information* (PII). This way, the clients can request data views that conform to the *least privilege principle* [15] and requirements for *data minimisation* [17].

```
{ "@context": {
  "dasp-email": "http://www.daspsec.org/on/dasp-email#",
  "owl": "http://www.w3.org/2002/07/owl#",
  "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
  "xsd": "http://www.w3.org/2001/XMLSchema#",
  "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
  "dasp-general": "http://www.daspsec.org/on/dasp-general#"
}, "@graph": [
  { "@id": "http://www.daspsec.org/on/dasp-general",
    "@type": "owl:Ontology", "owl:imports": [
      { "@id": "http://www.daspsec.org/on/dasp-general" },
      { "@id": "http://www.daspsec.org/on/dasp-email" } ] },
  { "@id": "dasp-general:AccessRequest",
    "@type": [ "dasp-general:Request", "owl:NamedIndividual" ],
    "dasp-general:acceptsOperation": { "@id": "dasp-general:RemovePII" },
    "dasp-general:acceptsSubtype": { "@id": "dasp-email:MsgHeader" },
    "dasp-general:requestsAccess": { "@id": "dasp-email:Email" } } ] }
```

Fig. 5: Sharing request issued to service provider

IV. Prototype Implementation

In order to validate the feasibility of our approach, we have first defined ontologies for *interoperability framework*, establishing both core concepts, as well as storage and email service vocabularies and relationships. Then, we have developed a proof-of-concept prototype implementation of supporting components for our framework. Our implementation relies on *Java* and *PlayFramework* for RESTful API and proxy support, using *JSON-LD Tools* for data representation transformation. It furthermore relies on *Jena* library for semantic modeling and reasoning using reduced OWL [20] capability set. In our prototype we also reuse *Google API Client Library* and *Box's developer SDKs* for the purpose of email and storage integration in our sample application.

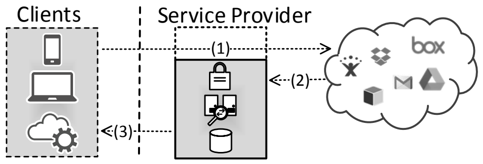

Fig. 6: Layered policy evaluation

Scope. Focusing on a prototype with restricted functionality, we explored the possibility to integrate our proposed approach with existing OAuth 2.0 deployments. In this scenario, we place a transparent proxy between clients and cloud service, with the purpose to serve as a second policy enforcement layer. It, therefore, applies security policies that (1) reduce the provided resource set, and (2) enforce client-specific resource-view by performing dynamic operations.

Workflow. Our prototype implements a *layered integration* by acting as a *model proxy* to OAuth 2.0 [9] sample application. It furthermore implements endpoints to describe the service model of the cloud provider and applicable policies to clients. The related flow is shown in Fig. 6. In the first step, the system receives requests from the clients with authorization tokens that correspond to standard *access scopes* from *Gmail*, *Drive* and *Box* services. These requests are proxied to the main application running on the service provider. In the step (2) the main application provides the resource located at cloud service and enhances the response with concepts defined in our framework, using JSON-LD [13] entities in a response header. By following these descriptions, *model proxy* transparently reduces the provided data set and performs basic operations supported by our framework (*PII removal* and *encryption*).

Cross-system messaging. The model descriptions in the prototype are exchanged in JSON-LD format [22], enabling external parties to explore system capabilities and implement advanced integrations with their workflows. Using tools provided in JSON-LD framework, these representations can be converted to ontologies and resource instantiations, processable by different systems. In our case, this approach is used to transform representations, such as ones provided in Fig. 3 and 4, to their textual, machine-readable counterpart.

In the example in Fig. 7, an *excerpt* of a security policy is formated as a JSON-LD record, defining the policy with one rule that applies to *Email resource* and allows *read operation* to designated token bearer (authorization token as provided in [9], [10]). This record is sent as *HTTP PUT request* to *RESTful API* endpoint that corresponds to the instance of the resource.

```
{ "@context": {
  "dasp-email": "http://www.daspsec.org/on/dasp-email#",
  "owl": "http://www.w3.org/2002/07/owl#",
  "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
  "xsd": "http://www.w3.org/2001/XMLSchema#",
  "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
  "dasp-general": "http://www.daspsec.org/on/dasp-general#"
}, "@graph": [
  { "@id": "http://www.daspsec.org/on/dasp-general",
    "@type": "owl:Ontology", "owl:imports": [
      { "@id": "http://www.daspsec.org/on/dasp-general" },
      { "@id": "http://www.daspsec.org/on/dasp-email" } ] },
  { "@id": "dasp-general:SecPolicy",
    "@type": [ "dasp-general:SecPolicy", "owl:NamedIndividual" ],
    "dasp-general:hasCombAlg": { "@id": "dasp-general:PermitOverride" },
    "dasp-general:hasRule": { "@id": "dasp-general:SecRule" } },
  { "@id": "dasp-general:SecRule",
    "@type": [ "dasp-general:SecRule", "owl:NamedIndividual" ],
    "dasp-general:hasAction": { "@id": "dasp-general:ActionRead" },
    "dasp-general:hasSubject": { "@id": "dasp-general:TokenBearer" },
    "dasp-general:hasTarget": { "@id": "dasp-email:Email"} }  ] }
```

Fig. 7: Security policy model for email target

V. Discussion

Our proposal for securing cross-service collaborations in the first line aims to establish inter-service *model awareness*, by providing the means for hosts to *describe* their resources, the clients to *structure* their sharing requirements, and the resource owners to *define* security policies independently of proprietary environments.

Compared to OAuth 2.0 [9] and UMA [10], our approach contributes with discovery and modeling of both services and policies applicable to resources. Contrary to OAuth 2.0, UMA defines these steps in the main flow, but their structure and flows are left to implementations to decide, causing the interoperability issues across different services. These steps are shown in Fig. 1 as *manage resources* and *define policies*.

We furthermore structure the definition and execution of *obligations* in web API services. This enables not only the implementation of traditional *access control* capabilities, but also their enhancement with *dynamic data transformation*, allowing context-specific reduction of information footprint and its alteration by enforcing *removal*, *encryption* or *masking* of sensitive information.

Finally, by relying on provided models, we enable third-party security agents and automated tools to access, discover, manage and control organizational resources hosted at various third-party premises. These tools can be used to control, audit, analyze and orchestrate security policies applicable on these distributed resources, establishing the broader picture of risks and processes that cross organizational boundaries.

VI. Conclusion and Future Work

In this paper, we have presented our ongoing work on interoperability framework, proposed with the goal to advance security and interoperability of distributed services integrated using web-based APIs. Our framework introduces two layers that contribute with domain specific ontologies and provide the tools for seamless integration of existing APIs and services. In this work, we have introduced the modeling layer of proposed framework, its building blocks and processes, providing the excerpts from applied ontologies and service descriptions. We also introduced the initial prototype used to examine the applicability of our model within existing deployments. This application is implemented as a proxy deployed between clients and cloud service provider. It integrates basic vocabularies for domains of *email* and *storage* services and provides basic components for policy management and enforcement. In its current iteration, the prototype supports enforcement of *selective restrictions* in a form of *transforming obligations* on shared data. It also integrates with existing OAuth 2.0 environments, acting as a policy enforcement and service model description point, deployed in a second layer.

In our future work, we plan to extend our general framework by (1) defining models of supported service and policy vocabularies for additional domains, (2) providing additional tools that support automated integration with APIs and (3) providing a web-based interface for semi-automated generation of semantic mappings between vocabularies and APIs. We furthermore intend to explore and demonstrate the applicability of our work beyond the cloud domain, focusing additionally on the security of both IoT, mobile and fog computing applications.

Acknowledgment

This work has been supported by the EU H2020 Programme under the SUNFISH project, grant agreement N.644666.

References

[1] Xu, L. D. "Enterprise systems: state-of-the-art and future trends". Industrial Informatics, IEEE Transactions on, vol. 7, no. 4, 2011.

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

[3] Raj, P, "Enriching the integration as a service paradigm for the cloud era." Cloud computingprinciples and paradigms. Wiley, Hoboken, 2011.

[4] Kleeberg, M. et al., "Information systems integration in the cloud: Scenarios, challenges and technology trends." Springer, 2014.

[5] Baude, F. et al. "ESB federation for large-scale SOA." Proceedings of the 2010 ACM Symposium on Applied Computing. ACM, 2010.

[6] Programmable Web. http://www.programmableweb.com (last accessed Dec. 2015)

[7] M. P. Machulak and Van Moorsel, A. "Architecture and protocol for user-controlled access management in web 2.0 applications," in *Distributed Computing Systems Workshops, IEEE 30th Int. Conf. on*. IEEE, 2010.

[8] Maler, Eve, et al. "OAuth 2.0 Resource Set Registration." (2015).

[9] Hardt, D. "The OAuth 2.0 authorization framework". 2012.

[10] Maler, E., et al. "User-Managed Access Profile of OAuth 2.0". 2015.

[11] Rissanen, E., et al. "Extensible access control markup language (XACML) version 3.0". 2013.

[12] Goguen, J. A., and Meseguer, J. "Security policies and security models." IEEE, 1982.

[13] Lanthaler, M., and Gtl, C. "On using JSON-LD to create evolvable RESTful services." Proceedings of the Third International Workshop on RESTful Design. ACM, 2012.

[14] Suzic, B. "Securing Integration of Cloud Services in Cross-Domain Distributed Environments". ACM, 2016 (in publishing).

[15] Schneider, FB. "Least privilege and more." Springer, 2004.

[16] Kertesz, A. et al. "Legal aspects of data protection in cloud federations." Security, Privacy and Trust in Cloud Systems. Springer, 2014.

[17] Art. 29 Data Protection Working Party. "Opinion 5/2012 on Cloud Computing". European Commission. 2012.

[18] Art. 29 Data Protection Working Party. "Opinion 8/2014 on the Recent Developments on the Internet of Things". European Commision. 2014.

[19] Westerinen, A. "Terminology for policy-based management". 2001.

[20] Bock, C., et al. "OWL 2 web ontology language structural specification and functional-style syntax". W3C Recommendation. 2012.

[21] Brickley, D. and Guha, R. V. "RDF Schema 1.1. W3C Recommendation". World Wide Web Consortium, 2014.

[22] Sporny, M., et al. "JSON-LD 1.0-A JSON-based Serialization for Linked Data". W3C Recommendation. 2014.

[23] Beimel, D. and Peleg, M. "Using owl and swrl to represent and reason with situation-based access control policies, Data & Knowledge Engineering, vol. 70, no. 6, 2011.