Zentrum für sichere Informationstechnologie – Austria
Secure Information Technology Center – Austria

A-1030 Wien, Seidlgasse 22 / 9
Tel.: (+43 1) 503 19 63–0
Fax: (+43 1) 503 19 63–66

A-8010 Graz, Inffeldgasse 16a
Tel.: (+43 316) 873-5514
Fax: (+43 316) 873-5520

http://www.a-sit.at
E-Mail: office@a-sit.at

DVR: 1035461 ZVR: 948166612 UID: ATU60778947

MULTIDIMENSIONAL SECURITY POLICIES

Technical Report

Version 1.0, 2.2.2016

Bojan Suzic – bojan.suzic@a-sit.at

Executive Summary:

The definition, evaluation and execution of security policies are typically oriented toward a particular organization and its internal infrastructure. In such scenario, the conceptualization of security policies follows organizational processes, being aligned with them both in the terms of capabilities, applied data structures or communication interfaces. The transition to cloud and mobile technologies, which increasingly depend on inter-organizational connectivity and collaboration of heterogeneous environments, introduces the challenges to this approach. In order to be applicable in cross-platform and cross-system scenarios, security policies need to conform to the requirements of interoperability, which especially involves their structure, representation and abstraction level of conceptualization. Hence, the policies need to be understood and applicable beyond central premises and processes, exhibiting the form that supports the collaboration in distributed and heterogeneous environments. The same applies to the entities and processes dealt with these policies, as they need to be understood out of the context as well. The arrangement of these policies additionally needs to demonstrate the advanced expressivity and the capability to support different dimensions of security requirements. These dimensions, depending on a particular scenario, might include the contextual requirements, limitations, data security and legal aspects, as well as the capability to handle security level agreements and contract-based transactions.

The goal of this project is twofold. First, it aims to provide a brief analysis of integration processes and application of security policies in cross-domain environments, reviewing the issues and establishing the requirements for interoperable and multidimensional policies. In the second aim, this project provides an initial groundwork in the form of a framework that can be used to analyze, define, test and integrate security policies in multiple environments. This technical report hence presents both of these results, elaborating on additional aspects and features that enable their application below the definition and exchange of security policies.

Table of Contents

Figures

Tables

1. Introduction

The concepts of reuse and specialization applied in the scope of today's web, driven by the principles of consolidation and efficiency that characterize cloud paradigm, result altogether with the increasing amount of interconnections established between online systems. These connections implemented across the organizations and realized using various technologies or protocols, provide a means to build a new systems and services, introducing interdependencies in an unprecedented manner.

The sharing of data, resources or services taking place along internetworked systems requires a new view of security challenges to be applied. The management of assets and security in such environments necessitate the means that enable organizations to perform effective and efficient control of their resources collocated at various cloud systems, potentially even in different geographical locations or jurisdictions.

In this work we approach two important aspects that concern the sharing and management of data-like resources in the cloud. We examine these dimensions from the perspective of many-to-many based interactions in cross-domain environments, taking place between heterogeneous platforms. In this work, having security in a primary focus, we start by establishing abstracted and interoperable resource management processes that potentially span across different platforms. For this purpose, we introduce common interoperability framework that relies on semantic technologies, for the purpose of achieving a machine-based understanding of services, resources, and their capabilities across the environments. In the second stage, we reuse the architectural and data representation concepts resulting from that work and focus on a subset of management activities that relate to establishing of data security in such environments. Our approach to data security tries to maximize the application effectivity of *less privilege* [1] and *data minimization* [2] principles in connected environments. Furthermore, our proposed model aims to establish consolidated and fine-grained view and control interface on organizational resources available at many outsourced cloud facilities.

The following sections are structured as follows. In the second section, we introduce the concepts Web APIs and integration processes in the cloud. In the third chapter, we examine resource sharing scenarios in the cloud. We first start by providing an overview of the approaches and popular platforms, then shift our attention to two representational use cases. Based on these contributions, in the third section we provide an overview of challenges in cloud-based resource management and sharing. In the fourth chapter, we propose a model that approaches these challenges with the primary purpose to advance the security of interactions in cross-domain environments. We then introduce our system prototype used to implement and evaluate our framework, along with the discussion on results and further work. Finally, in the fifth section, we provide the conclusion of this work.

2. Cross-Domain Integration of Data and Services

2.1. Web API based service integration

Cloud providers, at different service layers [3], deliver various resources and services to their clients. The means to manage these services and capabilities are provided to clients using various approaches. Figure 1 presents management layers found at arbitrary example applications in the cloud. In this sense, the following layers can be identified:

1) *Application* – this part represents the application as whole, in the terms of its scope, capabilities, functionalities and the value it presents to customers. Non-overlapping portions of related surface depict the potential application's service or value for customers that are not realized or exposed in practice.

2) *Proprietary management interfaces* – these interfaces enable users to configure, manage and consume the vast degree of functionalities or resources offered by application. Usually, modern applications provide this facility in the form of web interface. There might be

additional implementations available in the form of application SDKs or other specialized protocols and environments.
The term proprietary here relates mostly to the ability to automate these interactions and perform them in interoperable, heterogeneous and multiplatform environments. By relying on a human operator, or particular programming language and environment, these criteria can be fulfilled only partially.

3) Additional management and consumption layer is provided in the form of Web API interface. The implementations may use different protocols or architectural approaches to Web APIs. Modern applications typically rely on SOAP or RESTful interfaces, whereas the latter tend to be dominant[1]. This interface may expose a subset of actions and resources when compared to a proprietary interface. However, in some specialized cases, it can also provide the access to resources not present or accessible using standard proprietary interface, as shown in the right part of the figure. This case corresponds to the Web API as a dominant way to provide application services.
The overlapping surfaces in the figure depict the functionalities that might be offered by multiple means. For instance, the management of users in applications could be both done using application-specific interface, being further enhanced by relying on Web API for user management or SCIM protocol [4, 5] for standardized cross-domain identity management.

4) The fourth surface of Figure 1 corresponds to standardized or broadly adopted protocols that manage particular aspects of integration and management. Examples of such protocols are SAML and SCIM for cross-domain identity management or OAuth [6] and UMA [7] for web authorization management. These protocols potentially may rely on Web APIs for their flows, which is depicted in the figure as overlapping of two related surfaces. Additionally, the standardized protocols may offer functionality that is not used in or goes beyond the capabilities of the application. This case is depicted in the left example in the figure, as well.

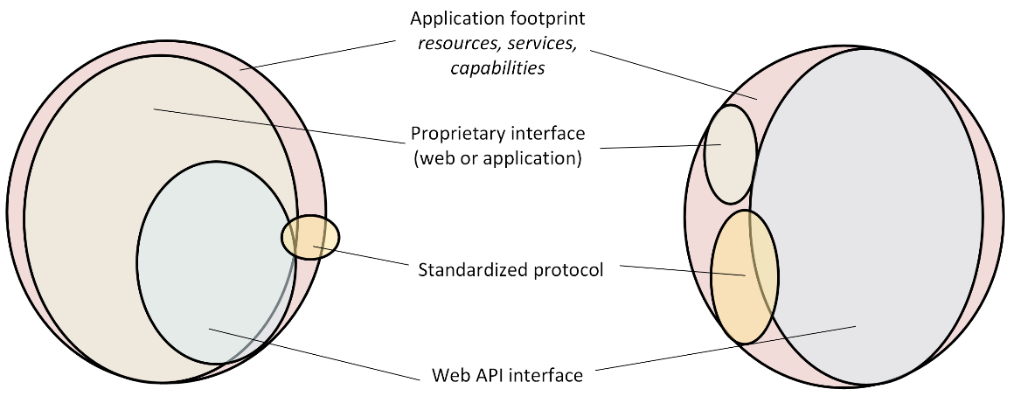
Application footprint
resources, services, capabilities
Proprietary interface (web or application)
Standardized protocol
Web API interface

Figure 1: Services and management interfaces in typical cloud applications

The resources or services delivered and exposed by cloud providers should be consumed in some way by their clients. In a typical application, providers expose Web APIs to their functionalities, often relying on commonly adopted the RESTful architectural approach. However, REST APIs in their typical instantiations do not provide any meaning or expressive semantic layer that could be used by automated agents to perform tasks autonomously. Moreover, although there are different approaches, reference and maturity models for REST APIs [8], the complexity of products and processes, as well as a variety of different understandings of representational models lead to a diverse range of applications that follow contrasting implementation and integration practices.

[1] This can be observed by consulting the catalogs of APIs registered under http://www.programmableweb.com/

2.2. *Integration platforms in cloud*

The approach of cloud-based integration got broader attention recently, as the products focused on integration and management of cloud services started to appear and 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 [9]. Following its emergence in the form of cloud-based technologies, analysts tried to establish and define the field of cloud-based integration services. One of the notable contributions in this direction has been provided by Pezzini et al., who identified IPaaS as *a suite of cloud services enabling development, execution and governance of integration flow connecting any combination of on-premises and cloud-based processes, services, applications and data within an individual, or across multiple organizations* [10].

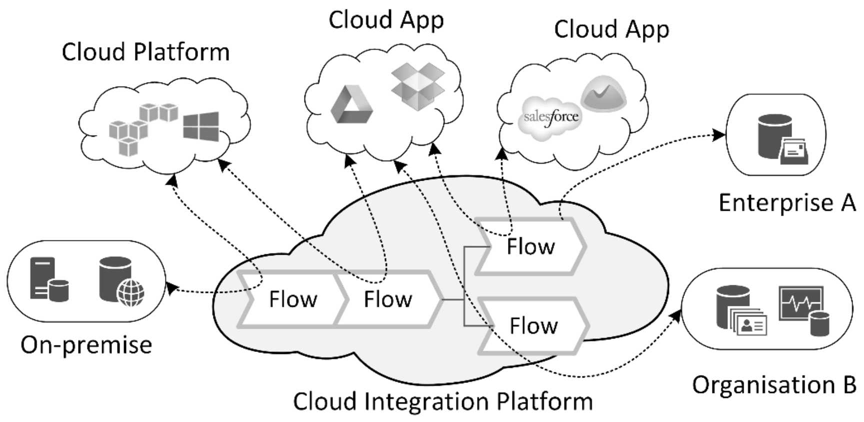

Figure 2: Overview of multi-organizational integration flows in cloud

A typical activity performed by integration platforms in the cloud is shown in Figure 2. It depicts the execution of workflows that consolidate the services and resources across the cloud. In this scenario, the platform connects to on-premise organizational systems, but its processes can also stretch to the systems of other organizations.

One of the scenarios for the interest of this work considers the access to the customer's resources located in other clouds. The example flow can be illustrated with the platform that connects to organizational Gmail account, retrieves and processes the messages and then, according to predefined triggers, consumes the interface on organizational Salesforce account. This scenario exemplifies the cloud-based execution of a business process that consumes customer's resources across different cloud instances.

A typical integration scenario does not differ much from the previous example. In its base form, it encompasses the use of organizational accounts at third party providers, with the goal to execute predefined tasks or workflows. This processing is commonly realized using Web APIs exposed by the service provider, which are secured using widely adopted mechanisms, such as OAuth 2.0 protocol [6]. OAuth 2.0, however, lacks fine-grained, policy-enhanced, assured and auditable data flow control and monitoring, as it will be shown in Section 3, analyzing data flows in two canonical examples.

3. *Management of Organizational Resources*

In the traditional view of enterprise architectures, the organizational information assets were mainly consolidated across organizational premises and infrastructure under its control. In more advanced scenarios, these premises can be cross-connected and distributed across different geographical locations, or even jurisdictions. In a latter case, the geo-distribution might reflect the structure of complex enterprises, consisting of multiple judicial subjects that act under the roof of a common multinational organization.

3.1. Common categories of outsourced services

In a novel models that emerged with the paradigm of distributed and cloud-based services, the organizational information assets can be deployed not only inside of premises or infrastructure of a main organization, but they may be distributed through the premises of third-party organizations that correspond unrelated and independent subjects. An example of such scenario can be found in outsourcing of services or tasks, whereas organization consumes, integrates and reuses third party products in their processes and portfolio offer. The example of such models can be found in some of the following service types:

- *Cloud Storage*
  In this product type, the organization A (principal organization) outsources a part of its data storage facilities by consuming remote storage services offered by company B. In this sense, the data assets of organization A, including potentially customer or company related data of different levels of sensitivity, in encrypted or non-encrypted form.
  In any case, that data is stored on an infrastructure of an external company. Depending on underlying protocols, it can be managed and accessed using either organization A, company B or some other accepted protocol and interface for distributed data access and retrieval.

- *Email Services*
  This scenario assumes that the organization A outsources its email service management to the email provider. In this sense, the receiving, sending, storing, management, searching, checking or any other actions and processes on organizational email are executed on the premises of external email provider.

- *Customer Relationship Management*
  In this scenario the organization uses the software delivered in the form of service of an external company, CRM provider. In this case, the customer and sale related data, such as customer data, contacts, leads, deals or account movements are managed in the scope of CRM provider, its software and infrastructure.

- *Cloud Servers*
  In order to provide more optimal, cost-effective and dynamic deployment of computing-related infrastructures, many organizations reuse the infrastructure of external providers and integrate it with own systems and processes. This infrastructure can be provided in different forms, including virtual server or containers. These resources are then used to deploy organizational internal or general templated applications, relying on them for further transfer and processing of data with different sensitivity levels.

- *Cloud Databases*
  Instead to take infrastructural service, deploy, use and orchestrate required platforms and systems, one of the alternatives for organizations is to sign up for already prepacked and configured infrastructure services. Cloud databases in this sense represent the instances of such services, managed in the premises of external providers. These resources are exposed to organizations to consume those using generalized and compatible interfaces, without the need to perform maintenance-related, scaling or other management activities.

- *Project Management*
  The tools belonging to this category enable users to organize their internal projects in various ways, including activities such as planning, tasking, live collaboration, document exchange, time management and others. These activities are performed using application interface, whereas the application and its data are in a common case maintained and hosted in external, provider's premises.

- *Collaboration*
  Being present in numerous categories of services and characterized by different types of instantiations, the services that belong to this class in common scenario enable live

collaboration between organizational members, as well as with external partners and contributors. This collaboration includes the task management, instant message exchange, integration with social media platforms and others.

3.2. Typical interactions with external systems

In Table 1 and Table 2 we provide an overview of typical resource units, integration types, and example providers both for domains of service and infrastructure outsourcing. For this purpose, we have considered some of the representative service or infrastructure types offered to and consumed by organizations in the cloud. These overviews do not aim for a comprehensive and detailed classifications and taxonomies, but to provide an illustrative summary of characteristics of some categories of products and their typical interactions with external systems.

Considering the trending convergence of services to partially cover different categories, some of the services in Table 1 are joined in a category that corresponds to their primary offered functions.

	Project Management	***Collaboration***	***CRM***
Service category	*Service*	*Service*	*Service*
Resource units	*Tasks* *Documents* *Repositories* *Contacts* *Issues* *Calendars*	*Documents* *Spreadsheets* *Calendars* *Contacts* *Messages*	*Contacts* *Leads* *Sales data* *Messages*
Typical integration actions (incoming)	*Create tasks* *Incoming documents* *Import contacts* *Incoming events*	*Create new document* *Import contacts* *Fetch messages* *Add members*	*Create lead* *Import sale* *Create task* *Import external email*
Typical integration actions (outgoing)	*Transfer calendar event to external party* *Deliver new file* *Act on new time entry* *Export new contact*	*Export new spreadsheet row* *Broadcast new channel message* *Document changed trigger*	*Data transfer to external services on sale* *Sending out messages using external providers* *Externalized data analytics* *Processing messages on external providers*
Example providers	*BaseCamp* *OpenERP* *Redmine* *Trello*	*Google Docs* *Office 365* *Trello* *Slack*	*Zoho* *SalesForce* *SugarCRM* *AgileCRM*

Table 1: Overview of distributed service types for service outsourcing

In the list of typically outsourced services presented in Table 1 the tendency can be noticed for those services to focus on granular categories of data. In this sense, resource units that are stored and processed in these categories correspond to the ranges of particular data records, ranging in granularity from documents and spreadsheets, on one side, to the tasks or contacts, on the other side. From this view, the data provided in a document or spreadsheet contains less structured information than the data presented in the form of contact, or issue. The same corresponds to the various semantic backgrounds of these assets, as the data provided in a sales record bears more concise meaning than, say, a document or spreadsheet.

In comparison to these resources, the assets presented in Table 2 tend to be positioned in a more coarse resource category. In this sense, the provider that instantiates virtual server or container typically does not provide much more metadata on underlying resources, i.e. that resource itself may contain much more diverse information than a data record of services listed in Table 1.

	Servers	***Databases***	***Email Services***	***Cloud Storage***
Service category	*Infrastructure*	*Infrastructure*	*Infrastructure*	*Infrastructure*
Resource units	*Virtual servers* *Containers*	*Database stores*	*Emails*	*Files* *Folders* *Drives*
Typical integration actions (incoming)	*Monitoring instances* *Resource scaling* *Incoming data*	*Resource scaling* *Incoming data*	*Processing of incoming emails* *Send email* *Create email label* *Classify email*	*Create new file* *Share folder with the contact* *Monitor file status* *Add file task or metadata*
Typical integration actions (outgoing)	*Triggering up new external instances* *Delivering data*	*Delivering data*	*Deliver new email* *Announce email labeling* *Externalized security scanning of emails* *Register new contact*	*File sharing with external customers* *File sharing with external services for the purpose of their processing* *Announce file assignment*
Example Providers	*Amazon* *OVH* *IBM SoftLayer* *Rackspace*	*Amazon RDS* *Azure* *Google Cloud SQL*	*Gmail* *Office 365* *Yandex.Mail* *KolabNow*	*Google Drive* *DropBox* *Box* *OneDrive*

Table 2: Overview of distributed services and resources for infrastructure outsourcing

In both tables we also provide the list of typical integration actions that can be triggered in an incoming and outgoing direction. These paths correspond to the points that originate integration transactions. In this sense, the incoming route is considered for connections that are initiated externally and directed toward a particular resource. This can be, for instance, the import of data to a database, or setting the data sharing parameters by an external party. In the second instance, the outgoing route corresponds to the integrations that originate from a particular subject. The example case for such integration is the triggering and initiation of data transfer to an external system, initiated by the task or message creation event registered on an originating system.

3.3. Resource sharing scenarios

The integration between cloud services can be applied in various ways and using various layers of cloud services. For instance, widely adopted NIST cloud computing reference architecture considers three cloud service layers, involving SaaS, PaaS and IaaS layers. The integration of resources and services crossing these three layers raises various challenges in the terms of sharing, processing and managing of these resources. In this section we focus on SaaS layer, considering its highest level of granularity both in the sense of services and data usage and structuring. For this purpose, we present two example scenarios that involve the integration of resources using third-party cloud integration platforms and their SaaS counterparts. In these scenarios, the data of an organization is both stored, transferred and processed in the cloud, in domains of several subjects. Based on provided scenarios, we illustrate the challenges in sharing and managing resources distributed across various, often unrelated cloud platforms. Although our use cases focus on SaaS integration level, the general principles apply to the integration on other levels as well, considering their similar approaches and more coarse resource representation.

3.3.1. *UC1: Automated integration of calendar appointments in project management system*

In this scenario, cloud integration platform accesses the resources stored on premises of two external services, exposed using web API. The arbitrary integration platform, therefore, checks periodically the status of organizational calendar service, and depending on updated data and new events, triggers processing in the scope of integration platform, eventually creating the tasks in the domain of organizational project management system, hosted externally and delivered as SaaS.

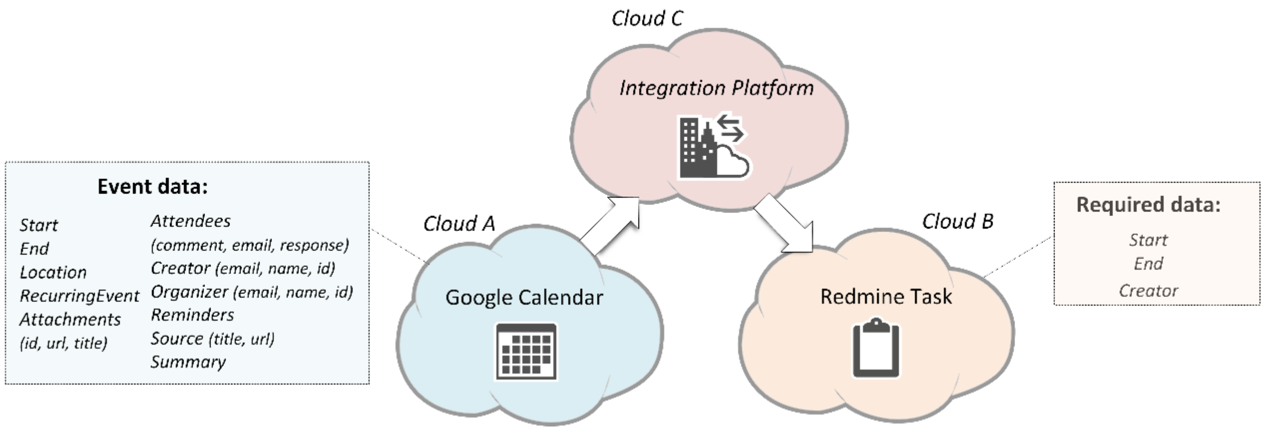

Figure 3: Transformation calendar event into project task

Figure 3 presents the data transfer performed between Cloud A, which hosts a Google Calendar, and Cloud B, which hosts a Redmine project management system. This transformation is done as a part of integration workflow performed in the domain of Cloud C. The related scenario assumes that Cloud C has access to resources and services collocated both on the premises of Cloud A and Cloud B. This access is provided on behalf of the resource or service owner (consumer), using standardized web authorization protocols, such as OAuth 2 or UMA.

The originating data record – which is a calendar event, is characterized by several data categories. Figure 3 presents a simplified and reduced view of resources related to an arbitrary calendar event, as described in Google Calendar API[2]. These resources are provided to authorized clients using Web API, in platform-specific data format.

In a typical scenario, the retrieval of data considers complete data records. In this case, provider API delivers the structured data and its parts in whole to the calling party. The scenario depicted in Figure 3 indicates, however, that the target Redmine system, as well as intermediary processing platform, require only a subset of that data to accomplish the task successfully. The both of them, however, get full access to the source data, directly in the case of the platform and indirectly, in the case of Cloud B appliance.

***Scope*[3]**	***Meaning***
`calendar`	*Read/write access to Calendars*
`calendar.readonly`	*Read-only access to Calendars*

Table 3: Access scopes as designated by Google Calendar

This can be confirmed from the access scopes supported by Google Calendar API service, as presented in Table 3. The presented scopes, therefore, distinguish only between full and read-only

[2] Detailed documentation is available at https://developers.google.com/google-apps/calendar/v3/reference/events

[3] The scopes should be prefixed with `https://www.googleapis.com/auth/`

access to all the resources and endpoints exposed through Web API. The resource owner is not in a position to, say, restrict the access to the resources that satisfy some particular criteria (such as creator, date, place, time range or organizer). It is furthermore not in the position to restrict only particular subsets of resources to be exposed to the client, such as appointed place, date, attendees, attachments or event summary. In the third instance, these scopes are statically defined, excluding the ability to define dynamic authorizations based on contextual parameters

The other side of the integration, the project management platform, relies only on `global` or `full` access scope. This means that the client application having access to Web API of exposed service can perform a broad range of operations and adjustments. The range of this operations does correspond to the degree of exposure of application's resources and related Web API coverage. In this particular case, Redmine system does provide only a subset of its resources and operations to be executed using Web API. There is at the moment no production-ready tool to restrict these operations. In this case, the similar restriction principles apply as to the previous example of Calendar API.

In order to illustrate the data flow between these three entities, in Figure 4 we illustrate the workflow considering this integration scenario.

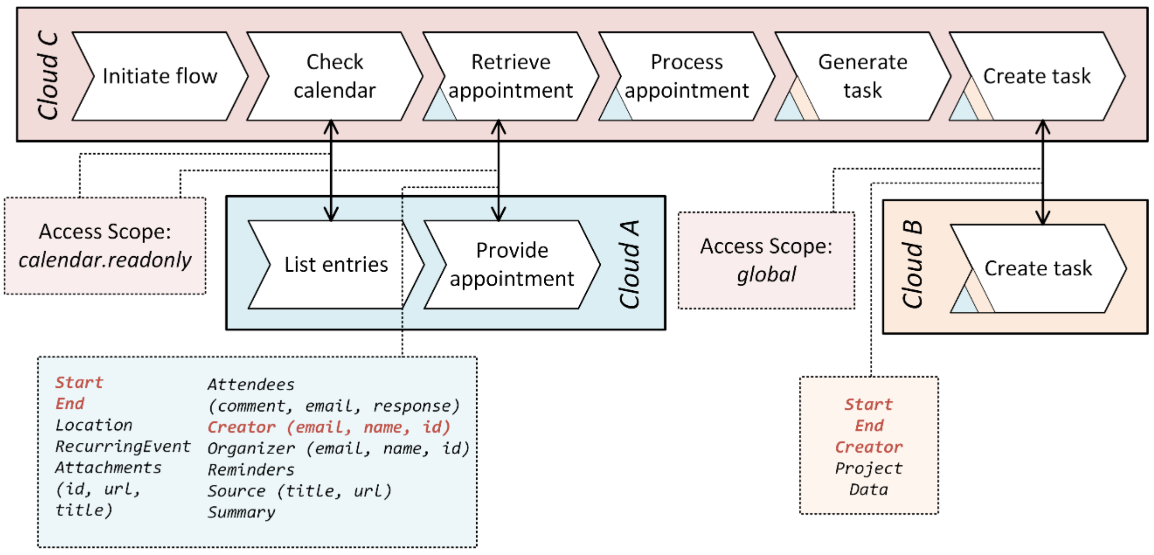

Figure 4: Integration workflow involving calendar and project management system

Primary workflow, in this case, is performed in the scope of Cloud C, which hosts integration platform. This service has access to the resources of data owner (organization) distributed across the different cloud premises, organizations, accessible using various technologies. In this current case, the particular task executed by the platform considers periodical checking of organizational calendar account. According to the requirements defined in the process, the platform checks calendar events, processes them, generates new resources in the form of tasks and instantiates them in the project management system hosted at Cloud B premises.

In this scenario, Cloud A with its `calendar.readonly` access scope provides read-only access to Cloud C service, including the access to all resources, their complete data and meta-data sets. The restricted set of data that is provided is depicted in the lower left part of the figure. The data records that are actually necessary to execute the tasks are marked in red. From this view, all the other data that are provided to the external entity are redundant for a particular view and might leak to information leakage and other security and privacy related issues. The data that is in a final instance, processed and transformed, provided to the Cloud B application, is shown in the lower right part of the figure. Again, this representation distinguishes between information that originates from the Cloud A system (marked in red), and other information that is produced in the processing steps both within the premises of Cloud B and Cloud C.

3.3.2. UC2: Automated processing of emails involving the creation of CRM contacts and leads

The second scenario presented in this section considers an automated processing of emails and update of remote CRM system based on a several interaction steps, executed across three cross-organizational cloud systems. In its basic form, third entity executes the processes on the behalf of data and resource owner. This entity, collocated in Cloud C, periodically examines the status of email account hosted at Cloud A premises and fetches new emails. In the next step, message processing (Figure 5), the platform examines the messages, contacts external public and subscription-based data services to augment the provided data. Based on the results of this step, the platform fetches the data from external CRM system and, if necessary, creates new contacts and sales leads. The detailed workflow and the data provided across different subjects are illustrated in Figure 5.

Scope[4]	***Meaning***
`gmail.readonly`	*Read all resources and their meta-data*
`gmail.compose`	*Create, read, update and delete drafts, including sending of messages and drafts*
`gmail.send`	*Sending of messages, without read or update privileges*
`gmail.insert`	*Inserting and importing messages*
`gmail.labels`	*Creating, reading, updating and deleting of labels*
`gmail.modify`	*All read/write operations except immediate, permanent deletion of threads and messages*
`https://mail.google.com/`	*Full access to the account*

Table 4: Access scopes defined for Gmail API service

In Table 4 we provide a list of currently supported access scopes on Gmail API platform, which serves as a data source for the process executed in this scenario. Similarly as in the previous case, the primary distinction is based on functionalities and access types. In the first case, the access to messages, labels, or activities such as inserting or sending of messages requires different scopes. In the second case, the distinction is established on whether the access requires reading or creating and modification of resources. Similarly as in the previous example, the resource owner is not enabled to introduce dynamic, context-based restrictions, nor to impose restrictions on particular instances of the resources that satisfy a range of criteria.

The access scopes supported by the other service, Salesforce CRM system, are presented in Table 5. Salesforce in this sense provides two layers of authorization. The first layer considers the user logged into the Salesforce system. That user may have a range of privileges over the resources and services provided using an organizational or common account. By issuing access token for REST API based access, this user is able to provide only a full set or a subset of permissions to access resources on behalf of its account.

Scope	***Meaning***
`api`	*Access to user's account using API. Additionally encompasses access provided by `chatter_api` scope*
`chatter_api`	*Access to Chatter API resources*
`custom_permissions`	*Allows access to the custom permissions in an organization associated with the client*
`full`	*Provides access to the full data available to the user. Encompasses all other scopes as well.*

[4] All scopes excluding the last one should be prefixed with `https://www.googleapis.com/auth/`

`id`	*Access to the identity URL service*
`openid`	*Enables access to unique identifier of user in OpenID Connected applications. Can be used to retrieve signed token conforming to OpenID Connect specifications*
`refresh_token`	*Allows provision of refresh tokens to eligible users, enabling the client to interact with the resource in offline mode.*
`visualforce`	*Provides access to Visualforce*
`web`	*Allows the ability to use the access token on the web, including access to Visualforce pages*

Table 5: Overview of scopes defined under Salesforce API

From that point, the primary distinction for access scopes is based on functionalities and lateral services provided by the application, such as management of user's account, checking of permission or allowance to consume other APIs. Practically, the access to main application parts is not customizable or segregated in different categories. This leaves the owners with fewer possibilities to restrict the access to the functionality of an external application. There is, for instance, no distinction between read, write, modify or delete accesses in the API. The restricted set of limitations can be, however, managed on the level of user accounts that are linked to web API instance, using proprietary Salesforce web interface.

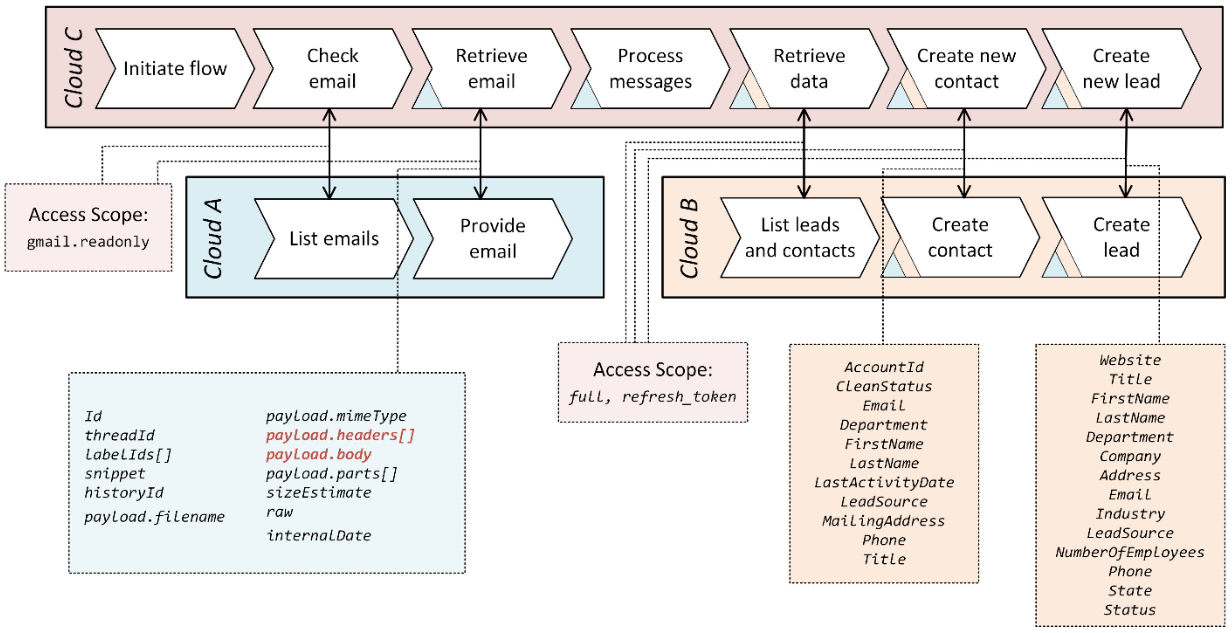

Figure 5: Integration workflow involving Gmail and Salesforce

Figure 5 presents the workflow that connects Gmail accounts with Salesforce CRM using intermediary integration platform. Similarly as in the previous scenario, the Gmail provides full data sets, whereas the data marked in red is necessary for the execution of tasks. In this case, the application requires access to some of the message headers (such as `from` and `date`) and message `body`. Gmail, however, using `gmail.readonly` scope provides access to all messages and their parts on user's account, whether they might be relevant for the target system or not.

The target Salesforce system requires full privileges and refresh token (for offline use) in order to allow periodic activities on the platform. These privileges allow in principle the retrieval and modifications of all resources available under account's web API, providing the client system with more privileges than needed. In the both cases of partially-restricted privileges, the information leakage and service compromise are possible.

3.4. Challenges in sharing distributed resources

As presented using previous two use case scenarios, there are several challenges for data sharing, especially in the light of cross-organizational interactions, privacy and security requirements. Considering that the vast majority of Web API service based interactions are protected using OAuth 2, UMA or a similar approach, we primarily focus on the challenges emerging from data sharing processes based on such flows. In this section, we identify and discuss these challenges.

Arbitrary definition of access scopes

The access scopes on resources are established by the service provider using its arbitrary assumption. This approach partially neglects the perspectives of two other parties – resource owner (or subscriber) and accessing client. Hence, other parties are forced to rely on access scopes in the range and capabilities predefined by the service provider, adjusting their internal processes and practices around these definitions.

Non-standardized approaches to scope definition

OAuth or UMA standards do not tackle the granularity, structure or suggest any ways how the access scopes should be defined. By lacking broadly adopted guidelines on how to classify and establish access scopes, the resulting implementations differ from each other in the approaches and levels of granularity. Many dimensions of access control and security are therefore either omitted or nor properly dealt. This additionally may introduce confusion among developers both when designing server APIs or client implementations, potentially leading to the sub-optimal results in the terms of security.

Coarse-grained permissions

The permissions that result from access scopes, as shown in the previous scenarios, may be too coarse-grained, preventing the optimal data segregation for cross-organizational connections.
For example, the typical API does not enable limitations based on resource instances, including their inherent characteristics, service or user-defined partitions. Instead, the permissions are assigned on the abstract level, representing the category of resources, type of accesses or API endpoints.
In the second instance, although the RESTful paradigm assumes the application of different HTTP methods, such as `PUT`, `POST`, `GET`, `PATCH`, `DELETE`, `OPTIONS`, many of the popular API implementations does not consider this dimension at all. Instead, they define out-of-the band sets that represent abstract groupings of these methods.

Requesting the permissions

Clients are typically limited to request the permissions that are supported by the service. These permissions are designated in out-of-the band process, using abstract, service-specific terms. Although there can be requested several access scopes in one client transaction, these scopes can overlap, resulting possibly in excessive or insufficient permissions to accomplish the operation. The clients are therefore not in the position to structure authorization request so that it optimally fits their access and workflow requirements. Due to the availability of access scopes that fit the use-case only particularly, many applications request maximal authorization scope. This results with potential security issues, as the vulnerabilities or improper maintenance of their systems might enable third parties to access or alter the data of their owners on a mass scale. Furthermore, the clients itself are allowed to access or alter the remote data of resource owners using broadly stated authorizations. In this sense, the only guard in exploiting these possibilities are trust and organizational measures executed between resource owner and external client system.

Detached scopes

Considering that the definition of the access scopes is non-structured, the scopes and their meaning need to be hardcoded in applications. The automated agents are thus prevented from the learning about their grants and permissions using automated means. The iterative changes of APIs and altering of scopes are hard to communicate and automatically infer, as there are no broadly adopted means to derive their underlying semantics. For instance, each API version change, scope change or introduction of new scope have to be communicated out-of-the band and manually hardwired into agent application. Such approaches introduce additional maintenance overhead, raising the concerns about long-term development and security of such practices adopted at large scale.

3.5. Challenges in managing distributed resources

In the current landscape characterized through the resources distributed across different systems and domains, there is an increasing need to employ the means to manage these resources in a standardized and automated manner. Such mechanism would enable organizations to execute information governance and compliance processes that span across the systems and platforms.
In this section, we elaborate the challenging perspectives hindering the optimal management of those resources.

Discovery of resources across services, clouds and data centers
Organizations that consume cloud services on their different layers lack the standardized and efficient approaches to discover their resources distributed across various cloud applications. These resources may include sensitive data that originate from organizational premises, or from external subjects using subscribed cloud service. In the latter case, the data can be created, imported or inferred both by the means of transferring it from external premises or producing it in the domain of subscribed cloud services. By using cloud services, employees of organizations can also generate new data that is not trivial for organizations to discover and manage properly.
The discovery does not include only identification of resources – in its advanced level, the discovered resources need to be properly characterized using meta-data and provenance descriptions. This descriptive data is crucial in the execution of processes that enable cross-organizational governance of resources and conformance checks.

Management of distributed resources
After they are identified, located and described, the organizational assets located at third party premises need to be subjected to management processes that cross organizational and service boundaries. These means need to enable organizations to orchestrate the data, define the restrictions, authorization and authentication parameters, as well as to establish and derive integrated SLAs that cover the usage and integration of these resources across various and unrelated third parties. In the current stage, many of these activities are performed manually and/or using proprietary, service-specific interfaces, introducing additional overhead especially in the cases of repetitive processes or heterogeneous platforms and services.

Integrated security policy management
One subset of resource management activities is represented by policy management. Currently, there are no standardized and automated approaches to policy management, and especially security policies that govern the usage of organizational accounts or processing and sharing of their resources in cross-organizational and heterogeneous environments.
In a typical cloud service, the policies over resources and accounts are managed using proprietary web interfaces, processes and terminologies, requesting manual adaptations and activities to be performed for each service separately. The organizations, therefore, lack the means to manage the security aspects of their resources in an automated, service and organization independent way. Currently, the authentication at some service providers can be managed by integrating organizational authentication services into cloud service provider infrastructure. This integration, however, still requires that organization define security policies using proprietary interfaces of cloud providers.

Portability of policies
The portability and interoperability of data and services between different providers are still an issue, especially considering the interoperability issues arising from granular and rich offer characterizing many SaaS services. By relying on provider-specific security enforcement and processes, the portability of security policies can be considered as related issue as well. Even when the data portability is achieved and resources seamlessly transferred to the other service, the security policies over these resources still need to be defined. Without security policies and infrastructures that are detached from particular providers and service types, this transition needs to be performed manually, introducing additional overheads and potentially leading to errors.

4. Proposed Approach

In this section, we present two concepts of discovery and management of organizational resources and policies. The concept establishes the service and resource description for heterogeneous environments. The second concept, building on the service description model, establishes the definition and management of security policies for cross-domain environments.

4.1. Service and resource description

In a common Web API (RESTful) approach, clients access API endpoints, which expose resources in some abstract model. These resources are accessed using different `HTTP` methods, i.e. `GET` for retrieval, `PUT` for modification or `POST` for the creation of resources. In a general case, these endpoints, as well as the resources they represent, are designed in the scope of provider's environment and using its internal practices. Client developers or other systems facing this API in principle have to follow the documentation of REST API and guidelines on how to use the services and implement particular functionalities. From this point, the conceptualization of the API is performed in out-of-the band process, which is described in the human-readable form using API documentation.

Although there are approaches, such as Swagger[5], to provide more coherent and harmonized API descriptions, they still lack the capability to provide machine-readable and understandable specifications aimed at automated agents and systems. Due to this, the agents accessing exposed APIs need to be hard-wired for each resource and functionality getting published, both in the dimension that relates to specification and in the temporal dimension, considering the iterative enhancements and developments of API over the time. The particular implementations are therefore hard to be reused, say, for the service of the same type provided by different providers. The example of this may be the email API services of Google and Microsoft: although they expose a basically similar resource, their APIs, methods and processes are different.

Each application that needs to fetch the email from both of these services using web APIs has to implement clients conforming to specifics of each provider. In the above-mentioned case of email, the client needs to follow different endpoints, ingest different formats and use the same methods on a different way for both platforms.

Furthermore, the problem of representing API endpoints and their provided resources or services can be analogously replicated to the representation of these resources, at the level of particular entities. In the previously mentioned case of email service, we may observe how Google and Microsoft deliver email to clients (via API) using different formats of that data. Therefore, the clients need to take care of the formatting and semantics for each case separately, as providers expose the resources in various ways.

In our proposal, we introduce the model relying on a semantic interoperability framework for the purpose of harmonizing of descriptions between heterogeneous agents. In our initial experiments, we have developed independent vocabularies for general service description and the ones that cover two domains of storage and email. These vocabularies are provided in the scope of semantic interoperability framework, which is meant to be extendable and reusable by various systems.

The first vocabulary provides general entities and relations that may be applied to an abstract service to present its model using common, reusable entities and relations.

In the latter two models, we introduce domain-specific descriptions, entities and relations that can be reused for similar services.

---

[5] http://swagger.io/

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 particular context.*

Table 6: Entities of general and domain specific vocabularies, excerpt

Table 6 presents an overview of entities designated in our three vocabularies provided under semantic interoperability framework. They contain the entities on different levels and layers of abstraction, enabling clients and other interacting cloud providers to gain an understating on underlying semantic and type of entities offered by REST APIs, on a machine-to-machine interaction level. For this purpose, `Storage Service` from Table 6 is considered as a subordinated, specialized entity of more abstract class of `Cloud Service`. In a similar manner, particular objects, such as `Drive` – `Directory` – `File` – `Document` maintain the similar relation of specialization. Furthermore, the specialization can encompass the entities of sub-entity level as well, such as message body and headers that are considered as integral parts of an email (`MsgBody`, `MsgHeader`, `Media` as parts of `Email`).

The relationships that can be established between entities are provided in Table 7. These include, for instance, the specializations mentioned above, additionally enabling the cloud services to describe supported operations or consisting parts (`supportsOperation`, `contains` and `isPartOf`) or clients to request particular entity or its form (`requestsAccess`).

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

Table 7: Object properties defined by general vocabulary, excerpt

4.2. Framework integration, service discovery and management

Canonical flows that consider the integration and reuse of semantic interoperability framework (common vocabularies) among various actors are depicted in Figure 6. In this overview, the clients, such as user's device, autonomous client or cloud integration platform connect to other cloud-based services, accessing the data and service endpoints on the behalf of the resource owner.

In this flow, actors first need to be aware of services and entities presented using common vocabularies. In the first step on the side of cloud services, this is done by reusing the common concepts and exposing them in the scope of provided APIs. By interacting with cloud services, diverse clients are enabled to learn the capabilities and types of exposed resources. By relying on external and generalized vocabularies, clients can learn that different cloud services expose the resources of same or similar types, with differing descriptions that are machine-understandable and, therefore, can be appropriately classified and integrated in internal flows.

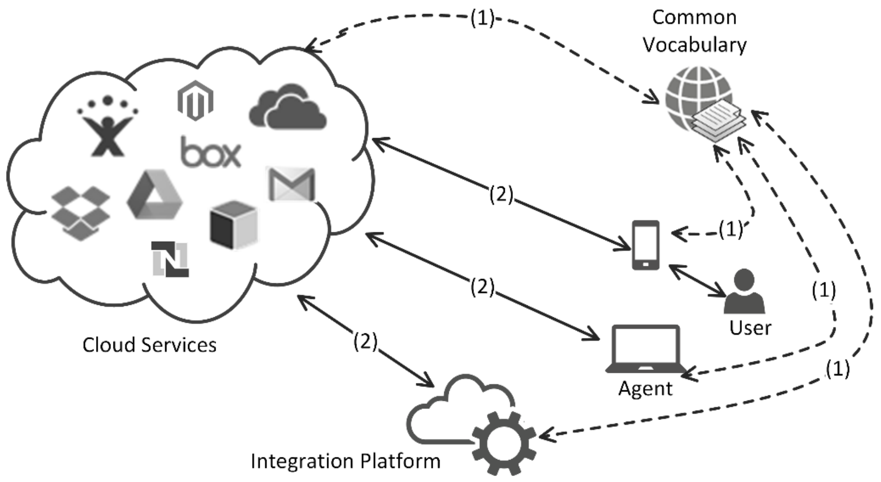

Figure 6: Interaction model using common vocabulary

Similarly, the discovery, and consequently the management of user's resources distributed across diverse clouds can be facilitated by relying on a common framework. Figure 7 shows the interactions that enable consolidated and harmonized discovery and management of cloud resources.

In the first step, cloud instances and agents need to be aware of resource representations. By reusing common vocabulary, these entities then expose their resources and interfaces, along with the descriptions relying on a common framework. The user's agent, on the other hand, browses the resources on different clouds, discovering the exposed services and resources.

The integration of representations based on a common framework and this flow enables the agents to bridge the differences among various web APIs, their endpoints, methods and representations, and to derive the understanding on the exposed entities, as well as to gather their meta-data and semantic descriptions in a structured way.

In other cases, user's agent may periodically inspect organizational accounts at different clouds and determine and discover the changes on data and services, enabling more advanced cross-system governance.

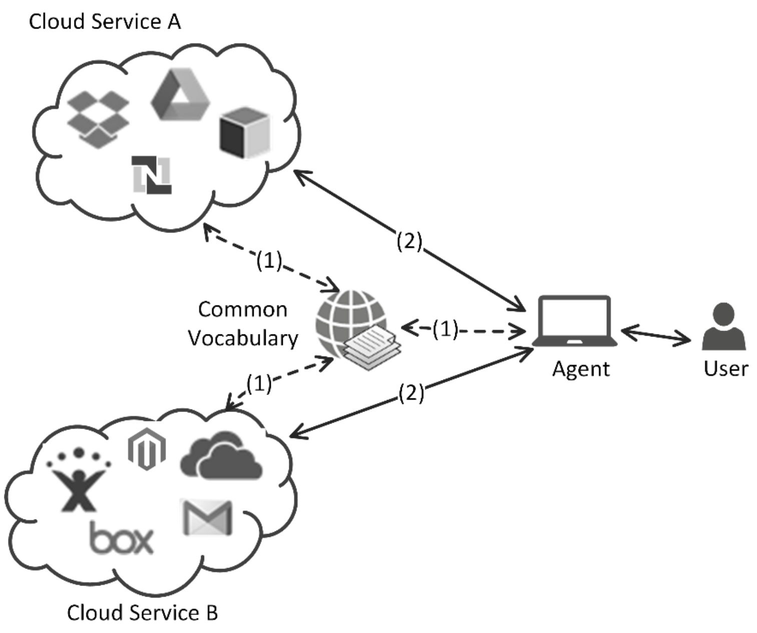

Figure 7: Resource discovery flows

By relying on a harmonized framework, automated agents may be used to demonstrate compliance with various legal and organizational requirements in an automated, fine-grained manner.

4.3. Policy description

The general interoperability framework and management approach introduced in previous sections establish the ground for the definition and integration of policy descriptions. Security policies in our approach reuse the concept of common interoperability framework, which we extend with appropriate entities, properties and relationships.

The overview of entities that encompass security policy vocabulary is provided in Table 8.

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 8: Entities establishing security policy representation, excerpt

The approach used to define security policies in this framework partially relies on the concepts from XACML [11]. The definition of policies and rules in this framework enable the implementation of attribute-based access control [12], as well as other concepts such as role-based access control [13], discretionary and mandatory access control models. In the scope of this work we stick to attribute based access control, whereas another approach will be considered in future iterations of this work.

Table 9 presents the properties defined under the security policy vocabulary. These properties are used to establish the relationships between classes and instances in ontology provided with this framework.

Property	***Description***
type	*Reuses standard rdf:type property to denote the class type of relating entities*
hasRule	*Determines the rule, a consisting part(s) of a policy*
hasAction	*Actions are related to the rules, which state allowed actions for entities under particular context*
hasTarget	*Denotes target resource of security rule. This target is used to relate with service types or resources, as defined in the general and domain-specific parts of the framework.*
hasCombAlg	*Determines combinatorial algorithm applied over the set of rules in particular policy. Combinatorial algorithms are used to define the decision approach used when combining multiple rules present in particular security policy.*
hasSubject	*Establishes the subject or subjects that are considered under particular security rule*

Table 9: Properties introduced for domains of security policies, excerpt

There are two ways how these vocabularies can be used in practical application.

In the first approach, services expose their objects and the policies applicable to them. The description of these policies is performed by reusing the concepts of security vocabularies, for security policies, and general and domain-specific vocabularies, for implementation of service and resource descriptions in the scope of the general framework and each domain. Overall, the description of security policy capabilities of these services is performed by reusing the concepts from interoperability framework and providing them in the form of ontologies i.e. ontology descriptions, on the server (service side).

Considering the second case, the vocabularies are used to explore the concepts both on the client and server sides. Servers, therefore, reuse these vocabularies to describe actual policies over particular resources. Clients, on the other hand, reuse the same vocabularies and, by relying on security capabilities provided in the model description exposed by servers, define the policies that have to be implemented on the server for particular resources or their range. These policies can be uploaded in order to create new or update existing policies.

The practical difference in exposing policy models, and defining policies, is characterized by the following points:

1) The policy models are described as a subset of descriptions from the common vocabulary. Each service provides policy model description targeting particular resource and referencing the common interoperability framework. This description, therefore, provides a model of capabilities that can be reused for the definition of policies.

2) The particular policies reuse the resources presented in the policy model and define exact rules and restrictions.

3) The model descriptions are, therefore, implemented as ontology representations. The particular policies reuse these representations and instantiate the range of classes and relationships that are presented as available in the model description. The difference is thus observable through conceptualization and practical instantiation of concepts, in a latter case.

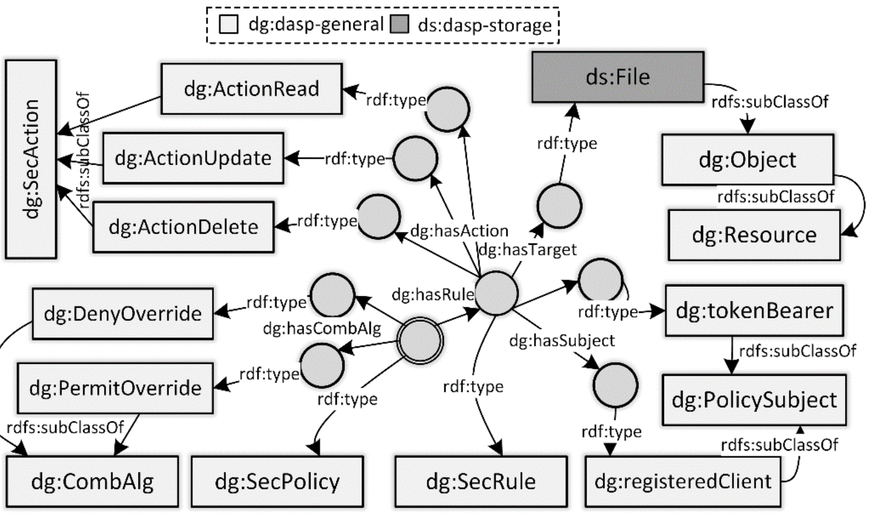

Figure 8: Example model of security policy, exposed by hosting server

The description of a security policy model is presented in Figure 8. This model reuses available common concepts, restrict them to the entities and relations that are available for the current resource. In a practical implementation, the policy is built by instantiating the entities and relationships available in this model.

Considering the integration of semantic interoperability framework, which provides reusable and common elements, the policies can be understood by different systems and platforms. This facilitates the cross-system machine-based understanding of security policies across heterogeneous infrastructures and platforms.

The organizations or resource owners are, therefore, enabled to enquire all the systems they use, including the ones in third-party domains and premises, and 1) discover the resources presented there, as well as 2) check and update security policies related to these resources. With this approach, the cross-system governance of resources can be implemented in a way that lowers integration and maintenance overhead traditionally present in the integration of heterogeneous infrastructures.

4.4. Interaction model

In the previous sections of this report presented are the concepts of common semantic interoperability framework and the methods of its integration in the processes of discovering and managing of resources and services, including the security management in cross-domain interactions. This section more closely explains how this is done in practice by elaborating interactions between different actors.

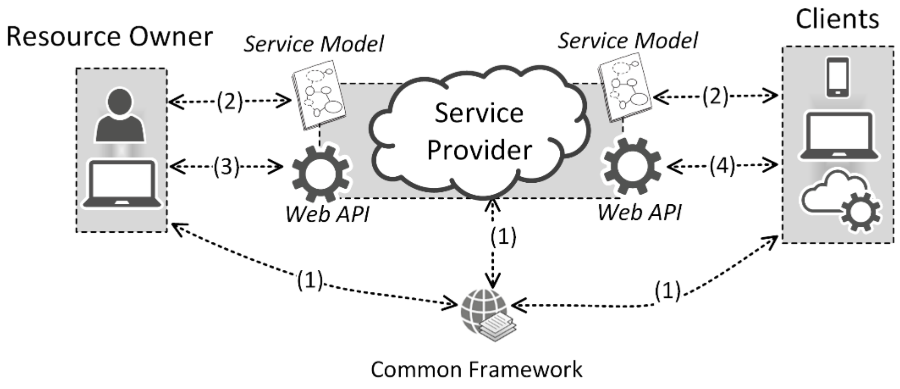

Figure 9: Interaction model for security policy definition

For this purpose Figure 9 presents the primary actors in this model, including resource owner, service provider and clients. Although presented as single instances in the figure, for a sake of clarity, all depicted actors should be considered as multiple instantiations of different and unrelated entities. Their common denominator is that they reuse the vocabularies provided in the common semantic framework and follow the implementations and processes presented in this approach, in order to achieve cross-organizational understanding and interchange of resources and security policies. The steps presented in Figure 9 are described as follows:

Step 1: Integration of common framework in internal systems

All actors need to load and integrate elements of the common framework.
In the case of service provider, these elements are used to present: a) service and resource models, b) policy models that relate to resources, and c) policies – their implementations that relate to particular resource instances.

Step 2: Service model discovery

Both clients and resource owner (or their agents) fetch service and resource models exposed by service providers in order to learn what services and entities are offered by a provider. Considering their understanding of the common framework, these representations can be aligned with internal resources and processes, providing a higher degree of interoperability and machine-based process and resource awareness.

Step 3: Discovery and resource management

In the first part of this step, resource owner (or its agents) connects to the service provider and discover their resources and related capabilities exposed using the common framework. This way, agents are able to learn which kind of resources are present on third party premises and how they can be managed by using unified interface.

In the second part of this step, clients discover a model of security policies, and security policies in force, for each of their resources present on a particular premise. Resource owners can also update these policies, or define new policies that should be applied to resources.

Step 4: Requesting and retrieving resources

Third-party agents and clients can request the resources exposed by the *service provider* and owned by the *resource owner*. One way to retrieve these resources is to take a part in interactions based on *web authorization protocols*, such as OAuth 2 or UMA. These protocols are applied to request resource owner consent to reuse its resources as a part of client's workflows. By relying on a common framework and service models provided in step 1, clients are able to coin the resource requests targeted to their particular use case and information needs. This way, the conformance to the *least privilege principle*, present as a technical and security requirement [1] or *data minimization principle* [2], introduced by European Commission as a legal requirement, can be ensured.

4.5. Implementation

In order to validate the feasibility of our approach, we have first defined ontologies for interoperability framework, establishing core concepts and security vocabulary, as well as storage and email service vocabularies and relationships. These vocabularies, as well as models of their application and integration into cloud interactions, are described in the previous sections.

As the second, we have developed components that support the integration of our framework. The implementation relies on Java and PlayFramework for RESTful API and proxy support, using JSON-LD Tools for the transformation of data representations from internal models to JSON-LD [14]. The implementation furthermore integrates Jena library for semantic modeling and reasoning using reduced OWL [15] capability set.

Scope

Focusing on a prototype with restricted functionality, we explored the possibility to integrate proposed model with existing OAuth 2.0 deployments. In this scenario, a transparent proxy is placed between clients and cloud service, with the purpose to serve as second policy enforcement layer. It, therefore, applies security policies that reduce provided resource set by transforming it to client-specific resource-view using dynamic operations. The implemented use case explored the possibility to reduce and transform email provided using Gmail API. In the first instance, the exposed email messages and their consisting parts are identified and structured using the ontology for the domain of email services. This way, the parts of the messages can be restricted or removed from the data set delivered to clients. In the second case, the transformation of messages can be performed by executing transformation functions over data structures.

Workflow

Our prototype implements a layered integration by acting as a model proxy to OAuth 2.0 application. It furthermore implements endpoints to describe the service model of the cloud provider and applicable policies to clients.

In the first interaction step, our prototype receives requests from the clients with authorization tokens that correspond to standard access scopes from Gmail services. These requests are proxied to the main application running on the service provider. In the second step, this application provides the resources located at cloud service, directing them back to the originating client through the proxy layer. This response is enhanced with the concepts defined in our framework, using JSON-LD entities in a response header. By following these descriptions, the model proxy is able to understand the message structure and transparently transform the provided data by performing basic operations supported by our framework (PII removal and data masking).

Cross-system messaging

The model descriptions in our prototype are exchanged in JSON-LD format, 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 various systems.

Figure 10 represents the simplified security policy for an REST-API exposed resource communicated in that way. In this policy excerpt included is one rule that targets *Email* resource, designating the allowed *read action* for the clients that provide authorization token of the *Bearer* type conforming to particular properties (not shown in image).

```
{
"@context": {
  "dasp-email": "http://www.daspsec.org/ontologies/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/ontologies/dasp-general#"
},
"@graph": [
  { "@id": "http://www.daspsec.org/ontologies/dasp-general",
    "@type": "owl:Ontology",
    "owl:imports": [
      { "@id": "http://www.daspsec.org/ontologies/dasp-general" },
      { "@id": "http://www.daspsec.org/ontologies/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"} }  ]
}
```

Figure 10: Security policy for service provided in JSON-LD format, excerpt

Delivered using PUT request to RESTful API endpoint that corresponds to the instance of the resource, this token updates security policy that has to be applied to the resource.

4.6. Discussion

The approach presented in this work can be considered as a semantic enrichment of standard REST API interfaces, in the terms that it enables providers and clients to communicate service and policy meta-data in several layers, along with API resources and calls. In the first layer dedicated to client – provider interaction, these descriptions are delivered to clients to present the model of offered service. In the second layer, the clients analyze that model and request the sharing of resources in a customized scope. The third layer that considers resource owner – provider interaction, additionally supports machine-to-machine discovery and management of resources and related security policies.

By integrating the concepts introduced in this work, the challenges presented in Section 3.4 can be approached from a more generalized perspective. The definition of access scopes, an issue that excludes resource owners and clients from the specification of authorization capabilities, in our model is dealt with by introducing common entities and relationships that are applied by involved parties to structure their requests and requirements. Following that, the requesting of the permissions is performed in a structured, standardized way, enabling a heterogeneous system to gain or communicate its understanding of shared resources. Similarly, the requirements and view of requested resources can be communicated using multiple dimensions, enabling the conformance to least privilege and data minimization principles in sharing processes that span across different systems.

The second category of challenges, presented in Section 3.5, is dealt in this framework by shifting the resource discovery and management from service-provider proprietary implementations to interoperable, decoupled interfaces. The integration of policy modeling and execution, as presented in this framework, enables resource owners to obtain interoperable, harmonized and integrated view of their resources distributed across the cloud. It is not only that these resources can be identified, but their characteristics can be also derived at management level, potentially allowing the users to perform more demanding regulatory and compliance related tasks with a fraction of previously imposed overhead. The same applies to the management of security policies as well, which can be communicated, and even exchanged across the environments in an abstract, description-rich and machine-understandable manner.

The prototype developed in the course of this project implements a subset of features necessary to integrate the model within premises of all involved actors, including resource owners, service providers, and accessing clients. This first iteration, therefore, delivers an integrated proxy that enables online processing of data sharing flows, applying the policies defined in the scope of a proxy and data flow directed from servers to clients. Its current limitation is reflected in a limited scope of supported service and application types, including the limited set of entities and functionalities integrated with security policy vocabulary and management engine.

In the future work, we aim to develop a separate web interface of proxy management engine, enabling automated inferring and definition of security policies. The toolkit to support the specification of resource sharing requests is also planned to be implemented. Finally, the performance impacts of the overall solution are envisaged to be examined in detail.

5. Conclusion

This technical report presents the initial 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. The report first introduces the approaches on resource sharing and API-based collaborative integration of systems. We analyzed different cloud sharing and integration scenarios, deriving the issues related to data security and privacy in processes that span across the environments.

By establishing the challenges in resource sharing and management using broadly adopted web APIs, in this report, we introduced our approach that practically extends existing web API interfaces with capabilities that enable rich and interoperable descriptions of exposed and requested resources. In this manner, the automated agents are enabled to derive, correlate and align data representations, gaining a clearer understanding of what kind of data and data exchange are taking place. Furthermore, the proposed approach enables resource owners to manage their data and resources collocated at various systems and environments, allowing them to gain a more holistic picture of their resources, as well as to engage in their management by using non-proprietary and interoperable means. These means enable users to define security policies applicable to their resources, allowing them to manage the security of their assets located in different clouds. The proxy-based system developed in the scope of this work takes the role of management point that analyzes ongoing resource sharing transactions and enforces security policies, not only for the purpose of traditional access control but enabling additional run-time based resource transformations that provide a context-based, dynamic and restricted view of shared resources.

In the future work, we intend to extend our framework with additional vocabularies and to evaluate its practical application in additional scenarios.

6. References

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

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

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

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

[5] B. Suzic, *E-Id in the Cloud with SCIM,* ASIT, 2015.

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

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

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

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

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

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

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

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

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

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