Deliverable D2.5

Service Integration Technologies

Grant agreement nr	688382
Project full title	Audio Commons: An Ecosystem for Creative Reuse of Audio Content
Project acronym	AudioCommons
Project duration	36 Months (February 2016 - January 2019)
Work package	WP2
Due date	28 April 2017 (M15)
Submission date	28 April 2017 (M15)
Report availability	Public (X), Confidential ( )
Deliverable type	Report ( ), Demonstrator ( ), Other (X)
Task leader	QMUL/MTG-UPF
Authors	Frederic Font (MTG), Alastair Porter (MTG), Damir Juric (QMUL), George Fazekas (QMUL)
Document status	Draft (), Final (X)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

Table of contents

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

Executive Summary

This deliverables summarises the service integration technologies upon which the Audio Commons Ecosystem is built. These technologies allow the interconnection of services and tools in the ecosystem. The code for this technologies is open source, released under an Apache 2.0 license and available at this public Github repository: https://github.com/AudioCommons/ac-mediator

In the current prototype of the Audio Commons Ecosystem, service integration technologies consist of a web application which mediates between applications (i.e. tools) and the services connected to the ecosystem. This web application is called the Audio Commons Mediator and represents the very core of the Audio Commons Ecosystem.

In this document we describe the main components of the mediator, justify some of the design decisions that were made and describe current plans for further improvements to be made until the end of the AudioCommons project. The actual technologies (which is the objective of this deliverable) are made available as source code provided in the aforementioned repository and as the actual functional web application deployed at https://m.audiocommons.org.

In addition to the description of the current prototype of the Audio Commons Mediator, we also describe the prototype implementation of experimental mediator components which are based on the use of semantic web technologies. Future versions of the main user facing Audio Commons Mediator will iteratively incorporate more semantic technologies based on the experimental mediator outlined here.

This deliverable is complementary to deliverables D2.4 and D2.6, which describe the Audio Commons API specification and present draft guidelines for adding new services to the Audio Commons Ecosystem (respectively).

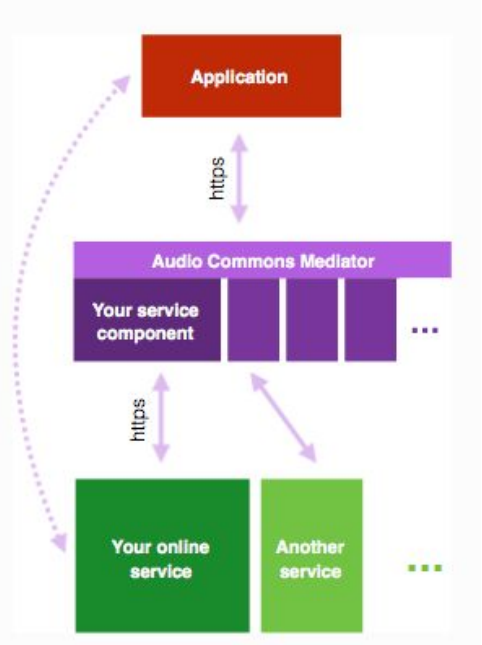
This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

1 The Audio Commons Mediator

1.1 Concept and main functionalities

The Audio Commons Mediator lies in the very core of the Audio Commons Ecosystem and is the component that allows the interoperation between the services and tools (applications) of the ecosystem. All communications between services and tools in the Audio Commons Ecosystem are done via HTTP requests and service requests are negotiated by the Audio Commons Mediator. The HTTP requests must follow the Audio Commons API Specification (see Deliverable D2.4 API Specification). The Audio Commons API specification is implemented by the Audio Commons Mediator.

With very few exceptions, all communications that happen inside the Audio Commons Ecosystem are expected to go through the Audio Commons Mediator (see figure on the right)[1]. In this way, Audio Commons is able to provide a unified and controlled interface to interact with the different services which are part of the Audio Commons Ecosystem. This facilitates the task of developing applications which, for example, consume content from several content providers in the Audio Commons Ecosystem. It also facilitates the management of Audio Commons user accounts and the user authentication with third party services.



The current implementation of the Audio Commons Mediator is already deployed and working online at https://m.audiocommons.org. It features a minimal implementation of the essential services of the Audio Commons Ecosystem, incorporating basic integrations of content providers (Jamendo, Freesound and Europeana), and one license provider (Jamendo Licensing). The code of the mediator is open source and released under an Apache 2.0 license. It can be found at a public Github repository[2].

The main functionalities of the Audio Commons Mediator are the following:

- **Manage user accounts**: Users of all kinds (i.e. application developers, service providers, content creators and content users) need to create an Audio Commons user account in order to, for example, request credentials for developing and application or authenticate in third party applications with their Audio Commons account (i.e. 'login' with Audio Commons). The Audio Commons mediator is in charge of managing these user accounts and allowing users to link their Audio Commons user accounts with third party services' user accounts, which is required in some cases. For example, through the mediator one can link his Audio Commons account with his Freesound account, which will allow to download original quality Freesound content while using the Audio Commons API. Linking Audio Commons user accounts with

[1] This does not necessarily mean that the mediator can't be distributed in the ecosystem, see Section 1.3 for more information.
[2] https://github.com/AudioCommons/ac-mediator

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

third party services' user accounts is done via the mediator's web interface (https://m.audiocommons.org/link_services/).

- **Provide endpoint for the Audio Commons API**: The Audio Commons Mediator implements the Audio Commons API as described in its specification (see Deliverable D2.4). This allows third party applications to make HTTP requests to the mediator (using as base URL https://m.audiocommons.org/api/v1) and interact with the services integrated with the ecosystem.

- **Service orchestration**: The requests that are made to the Audio Commons Mediator need to be interpreted and then the mediator needs to decide to which third party services the requests will be forwarded. For example, if an application requests a licensing URL for a given Audio Commons audio resource (identified by an Audio Commons Resource Identifier, or *acid*), the mediator will guess which services connected to the ecosystem can provide a valid response and then forward the request to them. To do that *forward* step, the Audio Commons Mediator needs to have a knowledge about which are the available services, what functionalities do they provide, and how to communicate with them.

- **Log activity in the ecosystem**: Because all the requests go through the Audio Commons Mediator, the mediator is also in charge of logging all the activity in the ecosystem. Having the activity logged will allow a monitoring of the system that can be used for many purposes such as tracking usage of services and content licenses, provide extra services such as recommendation, identification of behaviour patterns that can allow future improvements in the ecosystem and also identification of possible problems and debugging. The current version of the Audio Commons Mediator does not implement the logging functionality. However, we already performed a number of experiments of such functionality using semantic web technologies (see Section 2 of this deliverable). Logging will facilitate provenance tracking for data management and potentially rights tracking purposes.

1.2 Implementation

The Audio Commons Mediator is implemented using the Python programming language and the popular Django[3] web framework. It features a standard relational database using PostgreSQL[4]. As abovementioned, the source code is open source, released under Apache 2 license and available at https://github.com/AudioCommons/ac-mediator.

In the figure below it is shown a block diagram of the architecture of the mediator in its current implementation. The most relevant parts of it correspond to the implementation of the pair of *RequestDistributor* and *RequestAggregator* (which handle incoming requests and outgoing responses), and the classes of the *acservice* Python package which allow the communication of third party services with the mediator.

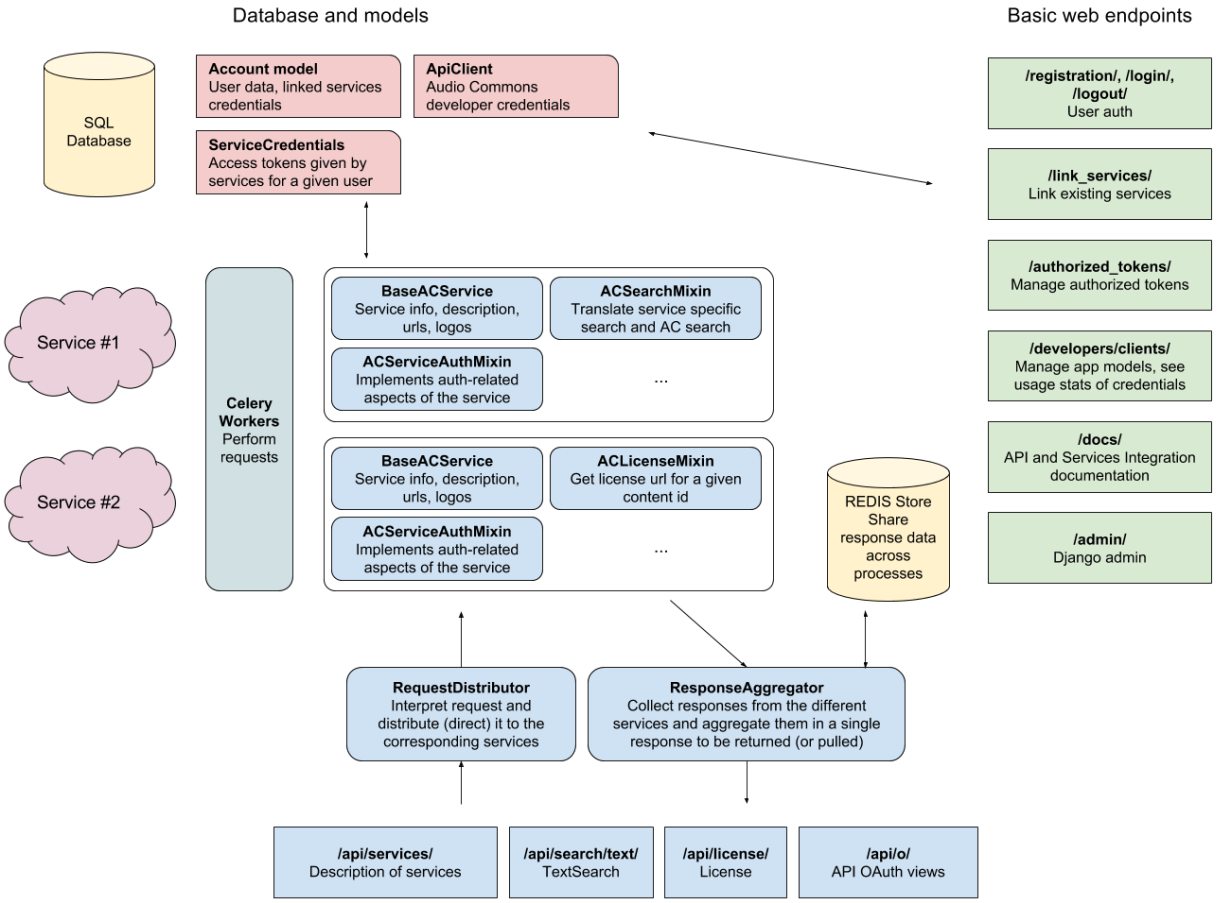
1.2.1 Handling of requests and responses

As mentioned above, the Audio Commons Mediator handles requests made by applications (i.e. at API endpoints) and decides to which services this requests should be forwarded. This is an important part of service orchestration as it distributes load to several components of the ecosystem. The way in which this part of service orchestration is implemented in the current mediator fulfils the most

[3] https://www.djangoproject.com/
[4] https://www.postgresql.org/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

basic needs. In essence, services declare which *API components* they implement (see Section 1.3.2) and the mediator forwards incoming requests to all the third party services that support the requested component. For example, a text search request will be forwarded to all the services that support the text search component. This happens in the RequestDistributor[5] of the Audio Commons Mediator. In the future we envision a more advanced solution for distributing requests based on declarative mappings and knowledge about service capabilities represented in OWL ontologies.



Example API endpoints

A full resolution version of this image here

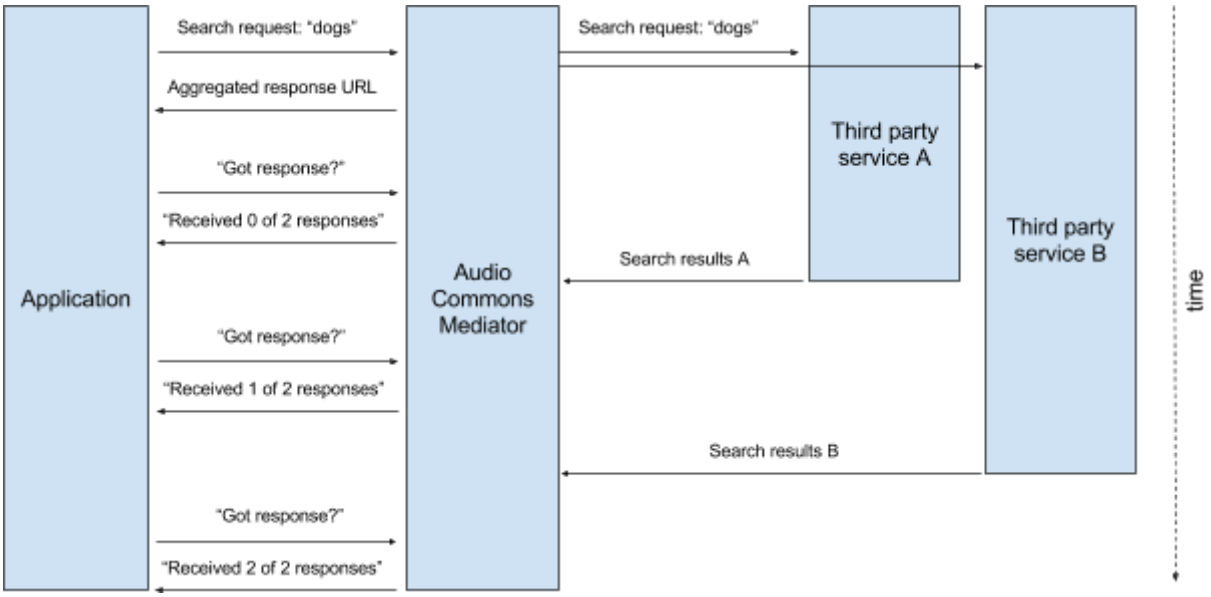
Service orchestration

The RequestDistributor forwards a single incoming request from an application to N third party services. The third party services might behave in completely different ways and take different time to respond, yet we want to optimize the time passed until the mediator can provide a response to the application. In order to solve that problem all the requests to third party services are forwarded in parallel (i.e. if the RequestDistributor decides to forward a request to 3 services, the mediator forwards the 3 requests in parallel). Immediately after forwarding the requests, a response is given to the original application with a URL that that application can use to retrieve the "real" results (as explained in Deliverable D2.4, API specification, we call this an aggregated response). As soon as a

[5] https://github.com/AudioCommons/ac-mediator/blob/master/api/request_distributor.py

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

response is received from a third party service, its contents are processed by the mediator and stored in a shared memory store (see *REDIS store* in diagram). The application that made the original request is expected to pull the aggregated response from the mediator in an iterative way until all expected responses from third party services have been received. The task of aggregating the received responses and storing them in the common store is carried out by the RequestAggregator[6] of the mediator. The following diagram exemplifies the described request/response flow:



1.2.2 Integration of third party services

Third party services can be integrated and "made available" through the Audio Commons Mediatorwithout significant architectural change at any provider. The task can be accomplished by providing a Python object which is known by the mediator and which knows how to "talk" to the third party service. This object must extend the *BaseACService* class defined in the *acservice*[7] Python package of the Audio Commons Mediator. The Python object representing a service is in charge of translating incoming application requests to the format of the third party service (i.e. translating from Audio Commons API specification to individual services' own API specification), and of interpreting the responses of the third party services and transforming them to a unified format across services (i.e. translating from individual services' own API specification to Audio Commons API specification).

In order to carry out these tasks, the Python object can implement a number of different *API components* which correspond to the different potential services defined in the Audio Commons API specification. Technically, the way in which a service declares that it implements a number of components is by inheriting from a number of "ACServiceMixins" defined in the acservice package. For example, the current implementation of the Freesound service inherits from BaseACService, from ACServiceAuthMixin, from ACServiceTextSearchMixin and from ACDownloadMixin. By inheriting from these classes and implementing a number of required methods (see Deliverable 2.6), the Freesound service is declaring that it supports Search component and Download component (besides the mandatory "base" and authentication components). Using this information, the RequestDistributor can understand what each service is capable of and make decisions about how to distribute requests.

[6] https://github.com/AudioCommons/ac-mediator/blob/master/api/response_aggregator.py
[7] https://github.com/AudioCommons/ac-mediator/tree/master/services/acservice

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

Future evolutions of the RequestDistributor and the acservice package will explore the use of semantic technologies to perform service orchestration in a richer way which also allows new services to be deployed to the ecosystem in an even easier way.

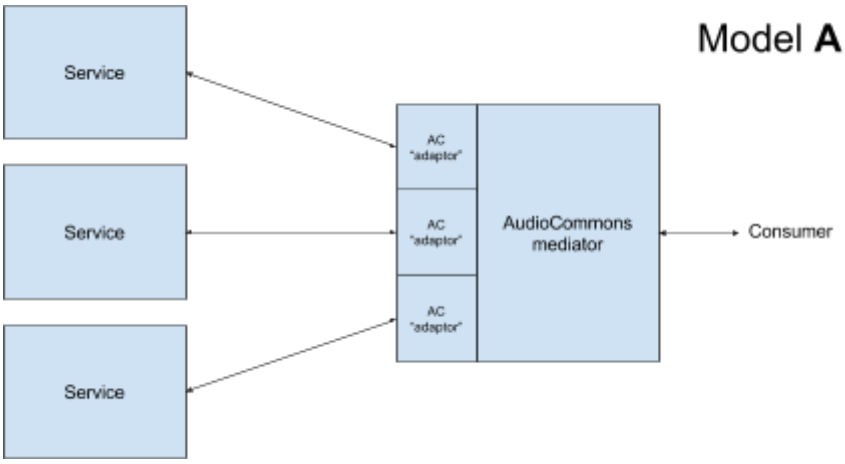
More information about how the acservice package should be used to declare new third party services can be found in Deliverable 2.6, Service Integration Guidelines. Full examples of already implemented acservice objects for Freesound, Jamendo and Europeana can be found in the mediator source code repository[8].

1.3 Models for the Audio Commons Ecosystem architecture

In the above sections we have given some details about the architecture of the Audio Commons Mediator and how it interacts with third party services. In the current implementation and therefore in the current prototype of the Audio Commons Ecosystem, the Audio Commons Mediator is implemented as a central service that operates at the core of the ecosystem. However, during the design phases of the mediator we envisioned an alternative architecture in which the Audio Commons Mediator component is replicated in each of the third party services and does not exist in a "single" central service.

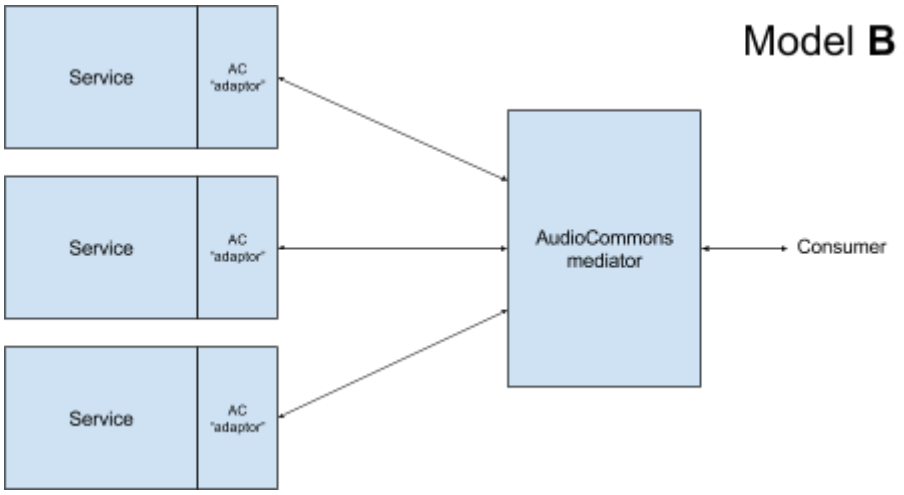
With this idea in mind, we proposed three ecosystem architecture models (Model A, B and C) that are illustrated in the pictures below. Model A is the most centralised one and the one currently implemented and deployed, while Model C is the most distributed. In any case, developers should be able to access any instance of the Audio Commons Mediator indistinctly and obtain the same results.

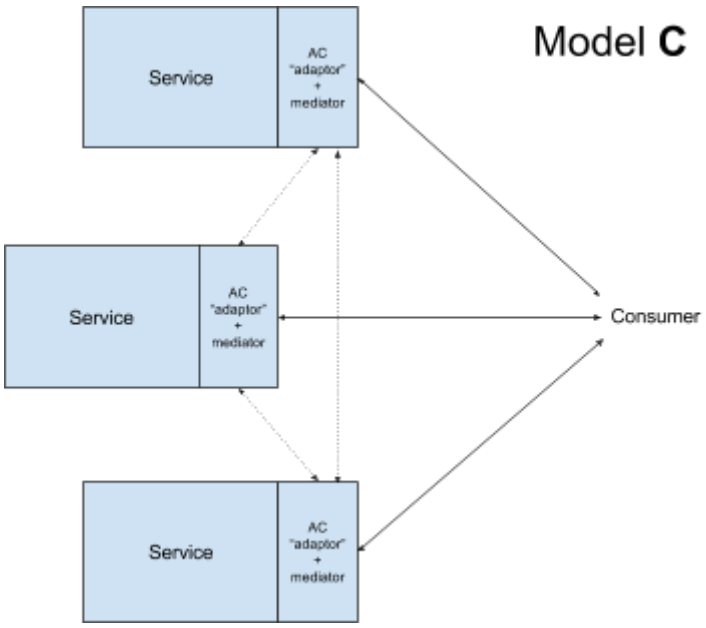
The three models could be seen as evolution steps that we can further investigate as the project evolves. The full implementation of Models B and C is subject to the feasibility and resulting complexity of them as compared to the complexity of developing and maintaining a single central service (e.g. in a distributed system we would still need to share things like user accounts and the logging backend). Models B and C could also raise other issues like third party services not willing to run Audio Commons Mediator code in their infrastructure. Nevertheless, future plans for the Audio Commons Mediator include the exploration of such alternative ecosystem architectures.



[8] https://github.com/AudioCommons/ac-mediator/tree/master/services/3rd_party

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382





This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

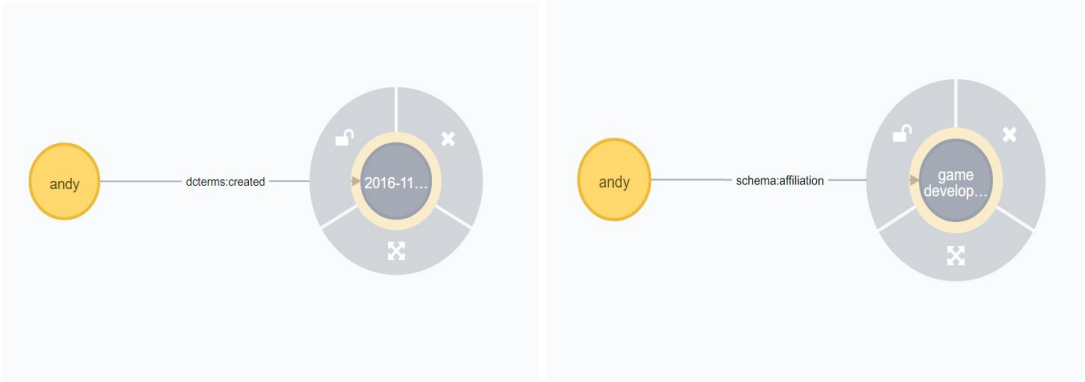
2 Experimental Audio Commons Mediator components using semantic web technologies

The Audio Commons Mediator as described in the above sections and as currently implemented and deployed for the first prototype of the Audio Commons Ecosystem makes only limited use of of semantic web technologies to perform service orchestration and communication with third party services. Because the use of such technologies in production-ready systems is not as widespread as the standard approaches followed in the current implementation, we decided to first implement a functional user facing deployment version of the mediator which implements the main infrastructure and components (such as the requests and responses flow). In parallel to that development, we worked with an alternative prototype Audio Commons Mediator fully integrated with the Audio Commons Ontology and that allowed us to start experimenting with service orchestration based on semantic web technologies. What follows is a description of the design decisions behind the semantic-web based Audio Commons Mediator. The code of this mediator can be found in the following Github public repository: https://github.com/damodamr/ac-webServies.

2.1 Graph database

The Semantic Audio Commons Mediator is implemented using the Python Flask framework[9] and the Neo4J graph database[10] (as opposed to the PostgreSQL database of the production mediator). The Semantic Audio Commons Mediator is fully tied to the Audio Commons Ontology and all objects and instances that interact in it are represented using ontological concepts in the graph database.

For example, in accordance with Audio Commons ontology user node created in the AC database is labelled “OnlineMusicAccount”. Every action carried inside the AC system, by this account, will be tied to this node. Actions in this case represent various services that are and will be implemented in AC (like search, upload, sound analysis, etc.). The figures below show how edges between user node and the nodes carrying data about the user are connected with edges that are labelled in accordance with AC ontology.



[9] http://flask.pocoo.org/
[10] https://neo4j.com/product/

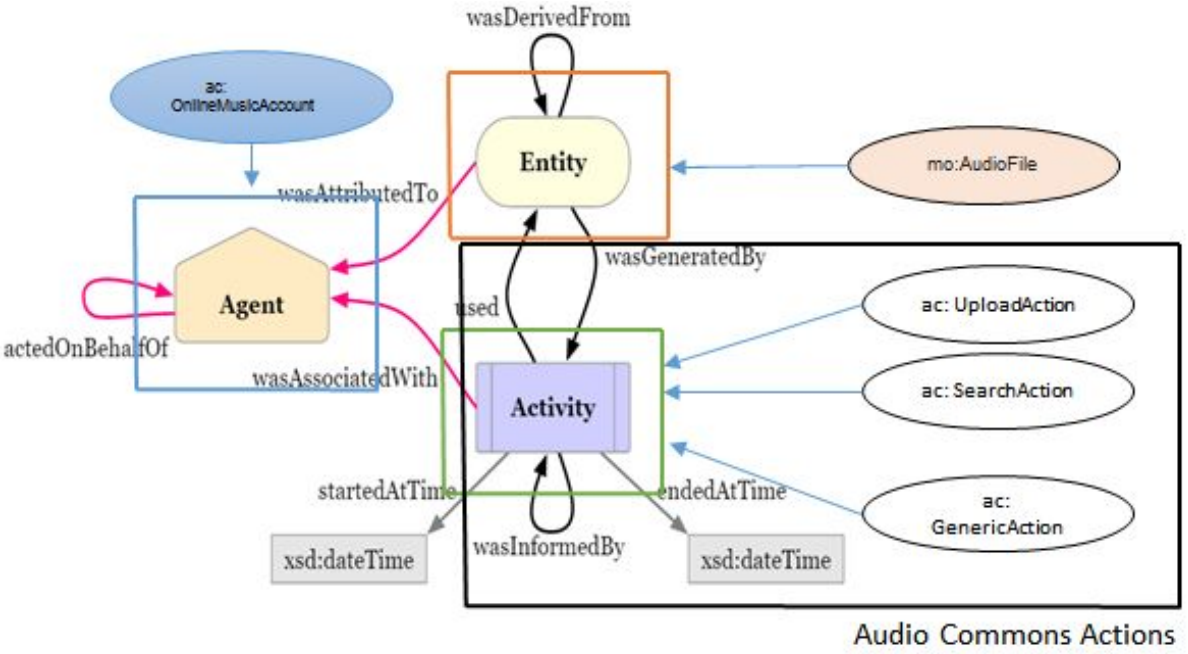
This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement N° 688382

2.2 Definition of third party services

The Audio Commons mediator contains a list of Audio Commons services. For a service to become an Audio Commons service it is necessary to:

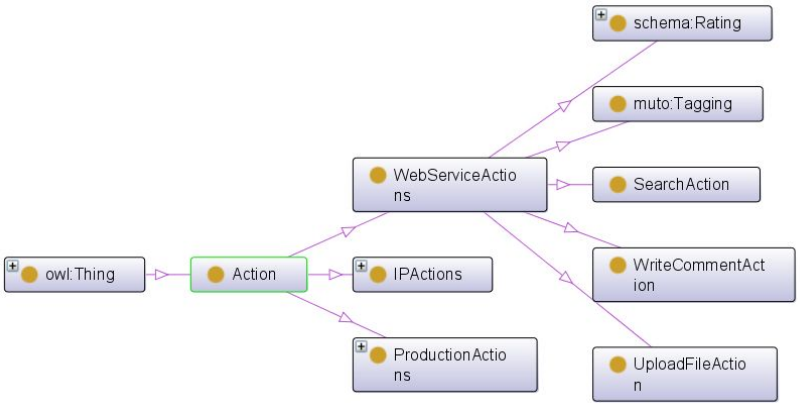
- Add the service to the AC mediator repository
- Describe the service inputs and outputs
- Use labels described in Audio Commons ontology whenever it is possible

Adding the new service to Audio Commons repository requires a knowledge of Audio Commons ontology. It is important that every new service in AC mediator repository is properly classified. As shown in the figure below, services like search or upload are defined as subclasses of an Audio Commons ontology *action* class.

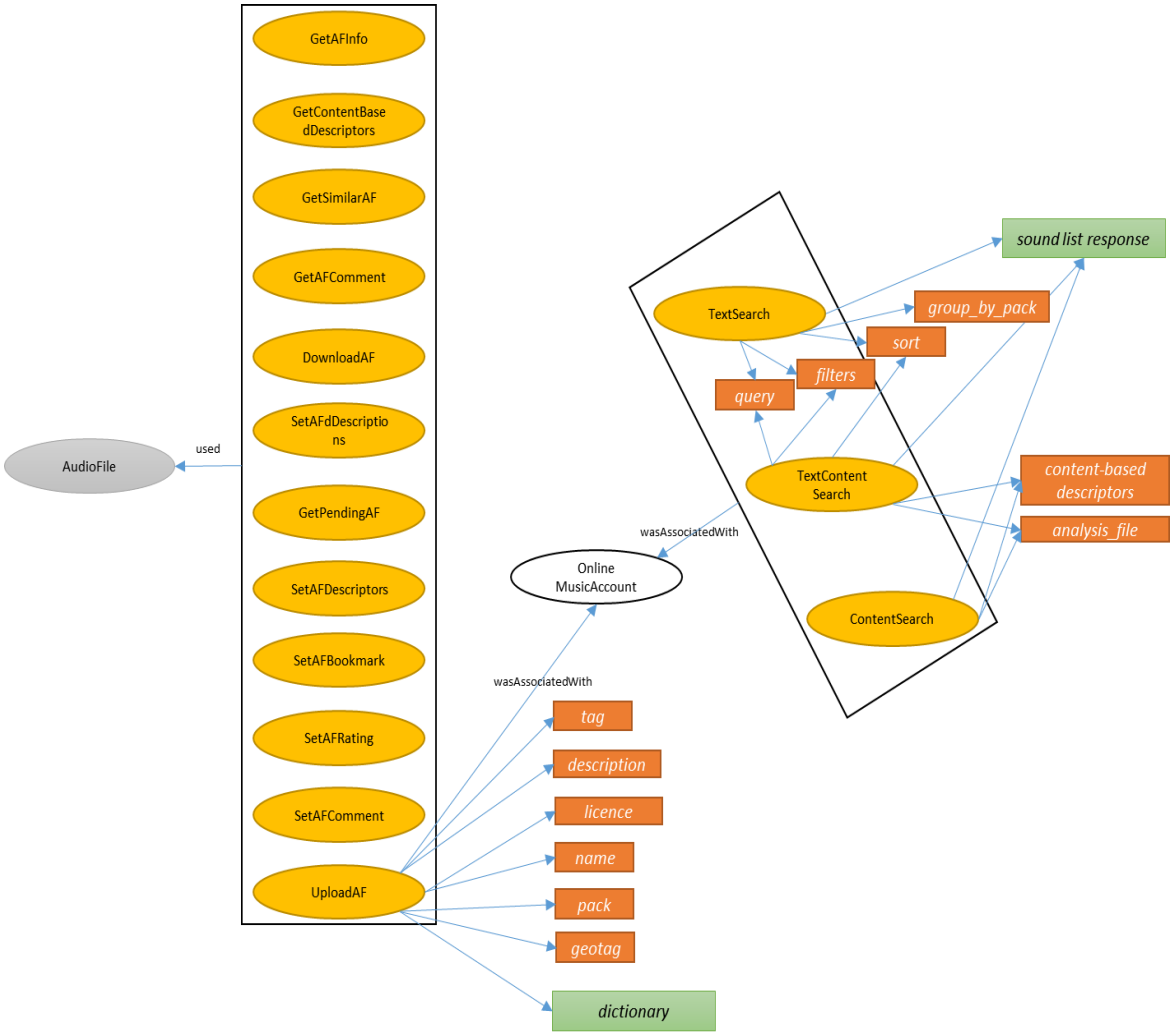


More specifically, actions that are implemented in this prototype (search, upload, etc.) are web service related actions, so that is the reason why they are defined as subclasses of a *WebServiceAction* class in Audio Commons ontology (see figure below). This definition makes these actions fundamentally different from other actions that can be carried out on an audio source (like production or licensing actions).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382



A new service provider who wants to add services to Audio Commons mediator repository will have to carry on an analysis of its existing services and define how those services will communicate with other Audio Commons services. This guideline is focused on inputs and outputs of web services and how these messages are represented in AC mediator. The following figure shows potential services that could be implemented in AC mediator and their ontological classes.



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

Services are grouped depending on what object (Web resource) they use/manipulate. In the figure above we see two groups of services where the group on the left side of the figure represents services that are manipulating an AudioFile (again concept from AC ontology) and the group of services on the right that are providing the search functionality. All of these services are always associated with a certain *OnlineMusicAccount*. Also, services differ in an output that they produce (services are usually producing different kind of documents like list, dictionaries, etc.).

This kind of analysis is important for the mediator in case of service orchestration involving a large number of services. If services are manipulating the same type of resource and are producing the same type of document / response, that means that they can be mutually interchanged and still produce a required goal.

2.2.1 Experimental search service

One of the most important and most used service in Audio Commons ecosystem is service of searching audio repositories. To simulate the actions that the potential users will carry inside Audio Commons ecosystem a prototype interface is developed where user can type a query and such query will be used as an input for services defined in AC repository.

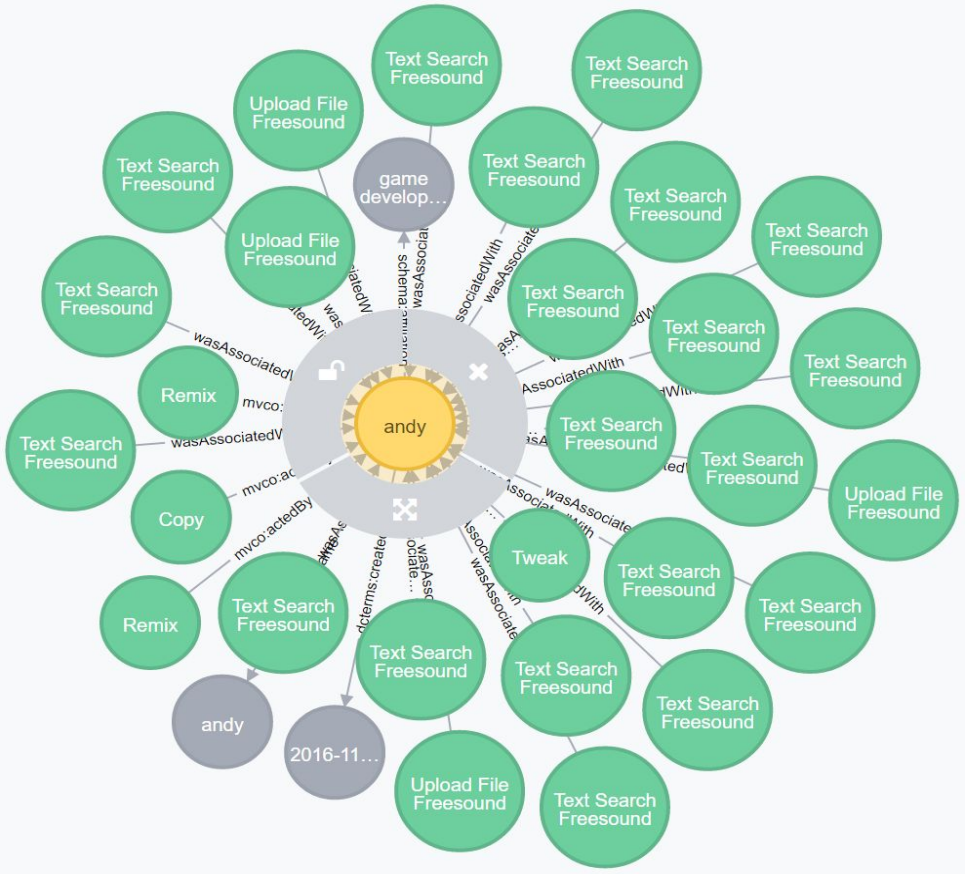
When the search action is triggered the mediator will create data in the graph database that will describe the provenance of the action. Created data (a node in the graph) will contain all the relevant information about the action including its description, agents involved and objects produced. See the following code examples for the definition of a node that will be created when service that provides the searching of Freesound or Jamendo repository has been called:

```
search = Node(
    "Action",
    id=str(uuid.uuid4()),
    timestamp=timestamp(),
    date=date(),
    description="Text Search Freesound",
    provider = "Freesound",
    actionDesc="SearchAction",
    hasInputMessage=query,
    hasMethod="GET",
    hasAddress="URITemplate",
    hasOutputMessage="mo:MusicalBuildingBlock",
    query=query,
)
```

```
search = Node(
    "Action",
    id=str(uuid.uuid4()),
    timestamp=timestamp(),
    date=date(),
    description="Search Jamendo",
    actionDesc="SearchAction",
    hasInputMessage=stateO,
    hasMethod="GET",
    hasAddress="URITemplate",
    hasOutputMessage="mo:MusicalWork",
    query=stateO,
)
```

Search actions (as well as other actions) are related to a particular user account. The figure below shows an excerpt of a graph surrounding a particular user node and displaying his interactions with the Audio Commons Mediator.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382



Types of actions carried by a test user (andy) in AC mediator

Because the data about the actions that are being carried out inside the AC ecosystem are being saved into a graph it is possible to get some of the interesting insights into services usage and use this information to build, for example, recommendation services.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

3 Conclusions

In this deliverable we have summarised the main concepts and implementation aspects behind the Audio Commons Mediator that is at the core of the Audio Commons Ecosystem. A working version of the Audio Commons Mediator is deployed at https://m.audiocommons.org, and can effectively be used to build prototype applications that access the current Audio Commons Ecosystem prototype and that interact with services like Freesound and Jamendo.

Besides the functional and "production-ready" implementation that we have developed and deployed, we have also shown experimental work that we have done with a fully semantic-web based prototype mediator. The Audio Commons Mediator will continue evolving during the next phases of the project together with the Audio Commons API. During this evolution our plan is to progressively incorporate relevant semantic technologies into the mediator (drawn from the experiments with a semantic mediator) and deeply integrate the Audio Commons Mediator with the Audio Commons Ontology. This will include the use of an API response format that facilitates an advanced way of consuming data from the AC end-ponts tied with the AC Ontology. For more specific details on other plans for future improvements we refer the reader to the conclusions section of Deliverable D2.6, Service integration draft guidelines.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382