TROMPA

TROMPA: Towards Richer Online Music Public-domain Archives

Deliverable 5.1

Data Infrastructure - V1

Grant Agreement nr	770376
Project runtime	May 2018 - April 2021
Document Reference	TR-D5.1-Data Infrastructure v1
Work Package	WP5 - TROMPA Contributor Environment
Deliverable Type	Report
Dissemination Level	PU- Public
Document due date	31-10-2018
Date of submission	31-10-2018
Leader	VD
Contact Person	Wim Klerkx (wim@videodock.com)
Authors	Wim Klerkx (VD), David Weigl (MDW), Álvaro Sarasúa (VL), Werner Goebl (MDW), Tim Crawford (GOLD), Aggelos Gkiokas (UPF),

	Ioannis Petros Samiotis (TUD), Vladimir Viro (PN), Nicolás Felipe Gutiérrez Páez (UPF)
Reviewers	Ioannis Petros Samiotis (TUD), David Weigl (MDW)

Executive Summary

This document describes the Data Infrastructure, which plays a central role in the TROMPA project. It will be a web-based platform where the efforts of all participants, partners and contributors come together, and where their results are to be made accessible to the public to engage with.

The Data Infrastructure will consist of two parts; there is the Contributor Environment, which is essentially a web API that will expose the entirety of TROMPA content and functionalities. And there is a collection of four components, each offering a coherent cluster of predefined functionalities to consume and enrich the TROMPA dataset. Examples of these components are a semantic search interface and an annotation tool.

Section 2.1 offers a detailed clarification of the Data Infrastructure as described throughout the project proposal, alongside the functionalities and requirements that can be inferred from that. These requirements are further elaborated in section 2.2, which is a consortium-wide best effort to define what each part of the TROMPA project will need or expect from the Contributor Environment to achieve its goals.

In short, the Contributor Environment is expected to be the web platform where public classical music content is enriched and expanded through the work of the TROMPA participants. This will mainly be achieved by engaging professionals and the general public to interact with classical music content in novel ways. In this context, the components can be seen as interfaces to coherent bundles of Contributor Environment functionalities, that are ready to be applied in user interfaces to build engaging web applications.

Section 3 summarises the requirements for the Data Infrastructure that can be distilled from section 2. Taken together, these provisional requirements allow us to confidently determine what needs to be implemented to fulfil TROMPA project requirements.

The first task of the Contributor Environment is to offer a solution for an internal TROMPA data model. On one hand, there are many sophisticated but often incompatible data models used by various public repositories, TROMPA participants and partners. And then there is the need to interlink the data based on these divergent models, to be consumed and enriched as a whole. The internal data model of the Contributor Environment needs to provide common ground that allows unified interactions with the content as if it were stored in a single repository, and provide access to this content at a granular level.

Another requirement for the Contributor Environment is to be highly performant. The dataset will contain objects of many different types that can be related to one another through a web of intermediate objects and relations. When we want to provide a useful experience to the end users, we should support performant queries across those many objects and relations. Often, the query results will have to be combined with content gleaned from public repositories, or processed by algorithms. To maintain acceptable response times, the API interface should be flexible enough for clients to express exactly what they need from the Contributor Environment, and no more.

The components play an important role in the development of the Contributor Environment and both should be developed in close collaboration. As the components provide public access to aggregated content and functionalities from the Contributor Environment, they are early and

reliable indicators that guide the optimisation of the performance of the TROMPA Data Infrastructure as a whole.

In section 4, the provisional requirements from section 3 are translated into a system architecture.

A minimalistic data model is proposed that is based on existing linked open data vocabularies. This will provide the common ground needed to interrelate disparate data. Using open data vocabularies for this purpose has the additional advantage of creating a dataset that is well adapted to usage on the web. By providing a way to label objects with additional classes and add custom properties, the various clients can maintain essential parts of their own data models when interacting with the Contributor Environment. To enhance performance and to prevent potential rights issues, the policy for public content, but also for content produced by contributors and participants, is to be left at its original location or in a data store. The objects in the database will contain only metadata and references to files and large bodies of data stored elsewhere.

With a dataset emerging that will consist mainly of simple objects interlinked through a rich web of relations of many kinds, the choice was made for a Graph database approach. Graph databases allow for schema-less datasets with rich relations, and are designed to perform well when queries need to traverse many of these relations.

An API interface that allows clients of the Contributor Environment to express their needs in a precise and flexible way, was found in the GraphQL query language. This well documented open specification is well suited to deal with a Graph database backend and is intuitive to use. Implementing a GraphQL API interface on a Graph database gives us a clear path to deliver on the performance and functional requirements for the Contributor Environment.

For the development of the components, requirements pointed to the adoption of React as the user interface library of choice. React is a solid and popular library for creating advanced user interfaces in web applications. Using React for the development of the components forges a solid base on which to build the pilot applications, and allows third party developers to easily take up those components to use in their own applications.

With these choices, the system design is versatile and able to absorb, to some extent, evolving requirements during the TROMPA project.

Section 5 offers a provisional planning for the development of the Data Infrastructure. With the specifications narrowed down, building of the Data Infrastructure can start immediately in M7. By M12, the first integrated versions of Contributor Environment and components can be expected. In M24 the Data Infrastructure will be further integrated with the WP3 (automated processes) and WP6 (pilots) tasks. The next six months, up to M30, will be about integration with WP4 (crowd annotation) and will be the point where the Data Infrastructure is fully functional. The last six months up to M36 will be devoted to consolidating the full integration and to support dissemination of the TROMPA project to the public.

In section 6 we conclude that the consortium-wide process that led to this document, in conjunction with D2.1 and D8.4, has brought more clarity and oversight among the TROMPA participants, on how the various parts of the project will play their part and fit together.

The result is a set of specifications that include relatively young but mature technologies. These specifications are ready to act on and promise an innovative period ahead for the developers.

Version Log		
#	Date	Description
v0.1	3-10-2018	Draft version submitted for internal review
v0.2	26-10-2018	Revised version after contributions
v0.3	29-10-2018	Revised version after informal reviews
v0.4	30-10-2018	Version ready for final review
v1.0	31-10-2018	Final version submitted to EU

Table of Contents

1. Introduction

This document, D5.1 Data Infrastructure, contains the first iteration of the TROMPA online Data Infrastructure jointly built by participants in T5.1

An overview will be given of the responsibilities this infrastructure will have within the TROMPA project, as the online intermediary between the work of participants and the audience using the pilot applications. From this overview, the overall requirements for the infrastructure are defined. These requirements are translated into a detailed data-modelling plan and system-architecture for what is to be built during subsequent iterations of T5.1, including planning.

From project proposal WP5 objectives:

This work package lays the fundament for the pilots developed in WP6. It delivers an environment for mid- level integration of components that will be further exploited in WP6 pilots. In order to do so, the data produced in WP3 (musical repertoire, automatic descriptions and generated audio) and annotations delivered through WP4 need to be made accessible and usable in reusable components, meeting common standards

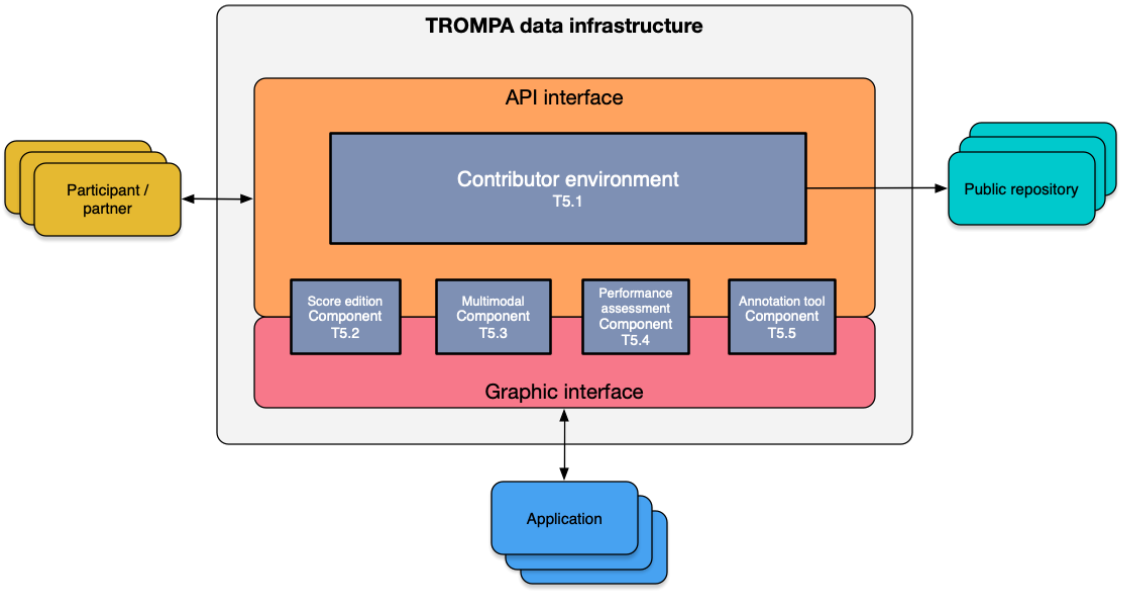

Figure 1.1. TROMPA Data Infrastructure

This online Data Infrastructure consists of two parts; The Contributor Environment and the components *(Figure 1.1)*.

At one end the infrastructure will enable the interlinking of musical data available in a selection of public repositories, enriched with data produced by various participants in WP3 and WP4, like annotations, crowd curations and algorithmically-generated data. This is the Contributor Environment.

At the other end, by means of at least 4 ready-to-use packages of functionalities, the infrastructure will enable standardized access to this enriched data, aggregated for the 5 pilot applications but also to be used by 3rd party applications. This is the components part.

The structure of this deliverable is as follows. Firstly, Section 2 is an overview of what is prospected in the project proposal, further elaborated on the basis of the expectations of each of the various clients that will interact with the Contributor Environment. These expectations are based on the current, initial requirements, under the assumption that these requirements will evolve during the project. We will clarify the tasks of the various client components that need to be performed in conjunction with the CE. Each task can influence the Data Infrastructure requirements as a whole.

Based on this overview we will group and summarise all the requirements of the Data Infrastructure in Section 3.

Section 4 is dedicated to the initial specification details following from the Data Infrastructure requirements as summarised in section 3. These specifications are the basis on which development of the Data Infrastructure starts in M7.

Section 5 provides a provisional planning of the development of the Data Infrastructure from M7 onward.

2. Overview of prospected functionalities

This section will detail the functionalities that the Data Infrastructure is expected to provide. It is the basis from which the total package of requirements is derived, as detailed in section 3.

The project proposal describes the role and functions of the Data Infrastructure and its constituent Contributor Environment and components in general terms. Section 2.1 translates this role and these functions to more specific functionalities.

Roughly following the sequence of work packages, section 2.2 offers the detailed needs and preferences described from the perspective of the various TROMPA subtasks that are interdependent on the Data Infrastructure.

Figure 2.1 shows a conceptual overview of the Data Infrastructure and its various types of interdependencies.

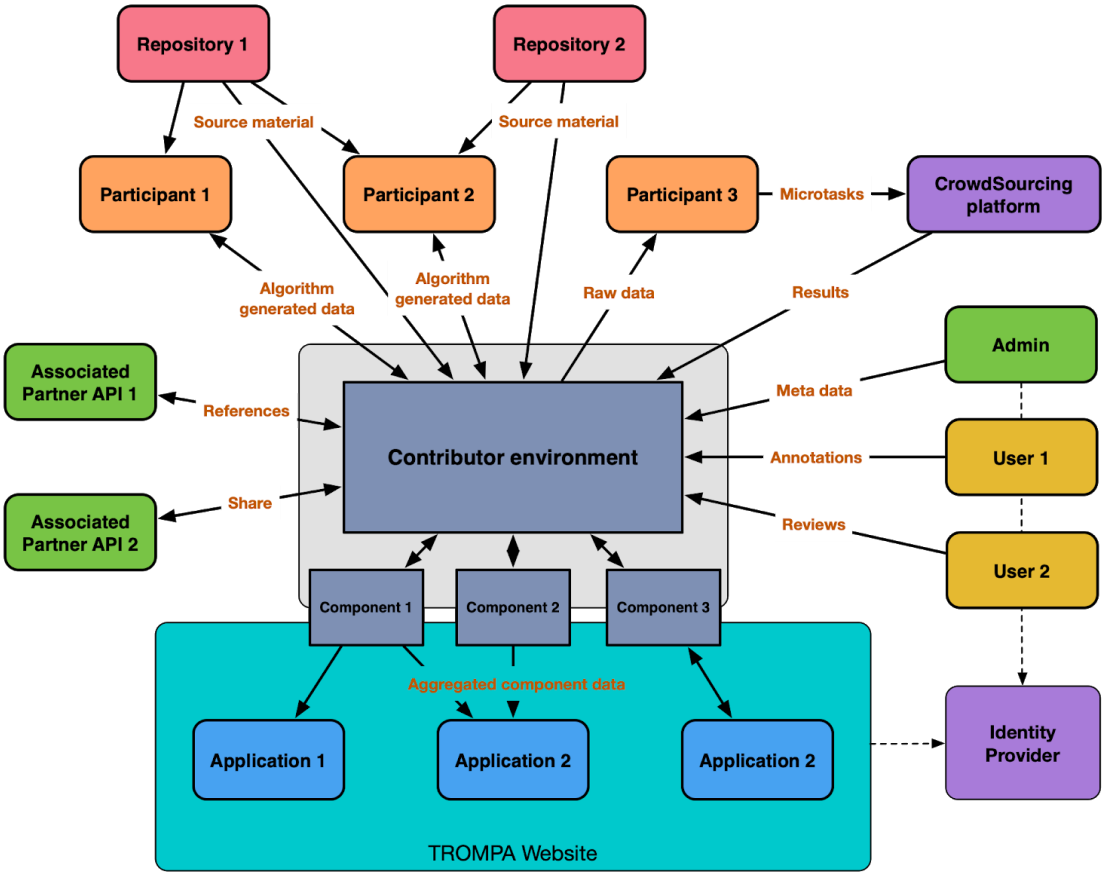

Figure 2.1. Schematic overview Data Infrastructure interdependencies

2.1 Data Infrastructure

From project proposal Task 5.1:

[...] a sustainable online data infrastructure available online that turns musical data processing results from WP3 [...] and WP4 [...] into reusable components. Common API access conventions will be implemented. For storage, we will rely on existing online repositories developed by partners and Associated Partners [...]

2.1.1 Contributor Environment

From project proposal WP5 objectives:

[...] the fundament [...] an environment for mid-level integration of components that will be further exploited in WP6 pilots.

The core responsibilities of the Contributor Environment can be summarised as supporting multiple clients (contributors, partners, application components) to interlink and retrieve musical data from multiple public resources, to support the creation and interlinking of additional musical data and to support responsive and fine grained access to the resulting dataset.

An additional, equally important responsibility of the Contributor Environment is to deliver on those responsibilities in a timely and manageable fashion. The development of the Contributor Environment should allow for on-time provision of functionalities needed by the clients, but also for keeping up with the evolving needs of those clients.

To make this attainable, the inputs and outputs of the system need to be controlled. Standards need to be implemented as for how contributors and components will interact with the Contributor Environment. As the project proposal defines the need to have 'common standards' for access, it is obvious that open standards are considered first for this purpose. At one side there is a standard type of API interface to adopt, and at the other there is the data model used to populate the requests and responses flowing through this interface. This interface data model is closely related to, if not the same as the internal data model, and would best be based on open standard ontologies.

Such interface standards will serve as a contract between client and Contributor Environment and will be essential in controlling the building phase of the infrastructure and for communications between contributing developers.

One of the main challenges for the Contributor Environment will be to offer interoperable access to the many and varied data models in use by the different clients. This solution needs to ensure that changes in a client data model will not create a ripple effect for Contributor Environment development, upsetting its functionality and planning. This puts strict requirements on the interface standards for the interaction between clients and Contributor Environment, as well as on the internal data model of the Contributor Environment.

Part of the functionalities offered by the Contributor Environment will enable clients to alter data. This requires robust access control to ensure that potential harmful functionalities are exposed to trusted users only.

2.1.1.1 Interface standards

The two access points to the Data Infrastructure are the Contributor Environment and the components. Users of the Contributor Environment will have diverse needs, but would generally be

expecting API-like access to functionalities. Users of the components will be public audiences. They will expect a graphic user interface that will translate their actions to frontend specific calls to the Contributor Environment.

To allow for different participants and 3rd party developers to create rich user applications, the preference is for an open standard frontend framework that is a solid base for graphic user interface needs. This would also standardize application interaction with the Contributor Environment backend (API). Preference is for the same frontend framework to be implemented in all components. This will make it easier for developers to use multiple components for the same application and to share pieces of code between applications. Further preference is for a widely used, well maintained and powerful framework that allows developers to create engaging applications that can be extended and maintained for years to come.

For the Contributor Environment, API conventions are needed to give standardised access to the various clients, including the frontend framework mentioned above. As the needs of clients vary widely, there might not be one API interface standard that can satisfy all. The project proposal mentions that at least a REST interface is required for the Contributor Environment. As the Contributor Environment is to be built within a strict time frame and with limited resources, there is the need to keep the number of supported API interface standards to a minimum, and streamline the available functionalities on the basis of what is actually needed by clients for each supported API interface standard.

Some contributor processes applying algorithms to (sub-)sets of data will need to run inside, or be controlled from within the Contributor Environment. When these processes are asynchronous, the interface(s) will have to provide for a mechanism to report results back to the requester.

2.1.1.2 Access control

The Data Infrastructure needs to expose many different functionalities to users. Some of those functionalities will be potentially harmful to the data, or might be rights restricted. For parts of the data (e.g. annotations), it is necessary to track the creator. Some functionalities (e.g. semantic search) might be completely open to the public. Therefore it is necessary to have a robust and manageable way to add and edit users and manage their access rights to the Data Infrastructure.

In the TROMPA project, users can approach the Data Infrastructure in a number of different ways; a user should be able to access functionalities through one of the components, but also directly through one of the Contributor Environment interfaces. This might be the same user, and it would be very useful if this user can use the same credentials for both access points. For example, a user identified as an expert could be adding metadata directly into the Contributor Environment, but could also use the annotation tool component to add further comments. Even when contributed through different access points, it would be valuable if both annotations could be traced back to the same user.

As described in section 2.5.1, there should be different interfaces for users to interact with the Data Infrastructure. Access control should ensure that a users' rights to functionalities are the same for whichever interface is used to access this functionality.

2.1.1.3 Data formats

Various clients (e.g. participants) of the Contributor Environment will prefer different data formats to interact with the Contributor Environment. Trying to accommodate all these formats will derail

the main responsibility of the Contributor Environment: timely delivery of functionalities needed by the various participants.

The need is to limit supported data formats to a few standards which, taken together, can satisfy the needs of all participants even if some participant's preferred formats are not included. This prevents burdening the development of the Contributor Environment with exotic serialisation cases.

The selection of supported input formats might differ from supported output formats. A feature request procedure would help a participant when reaching the limits of currently supported data formats.

An important Contributor Environment policy is to keep data, especially bulky data like audiovisual recordings, as much as possible at its original location (URL). Of course, this data can remain there in its native format.

2.1.1.4 Data models

Various clients (e.g. participants) of the Contributor Environment will be working with very dissimilar client specific data-models, which are expected to evolve throughout the TROMPA project. As is the case with data formats, trying to accommodate all of these models will derail the main responsibility of the Contributor Environment: timely delivery of functionalities needed by the various participants.

Even more so than with data formats, the need is to come up with preferably one way to model incoming and outgoing data, in such a way that the needs of all participants are met. Participants using the Contributor Environment would have to convert their data to this internal model, so that it can be stored without losing essential expression from the original model. A similar conversion should take place when retrieving data, which could be enriched by other participants. This practice is also in the interest of project-wide data management; specifying metadata standards for the internal data model will be essential in ensuring all contributed content will adhere to those standards.

Support should be available for participants who have conversion problems. A feature request procedure would be helpful for when a participant reaches the limits of the current internal data model of the Contributor Environment.

2.1.1.5 Data storage

To ensure a maintainable Data Infrastructure, Contributor Environment clients should strive to not duplicate content from public music repositories. A clear policy should be defined at where the boundary is between data that can be stored in the Contributor Environment, and data that should not. The internal data model should accommodate storage of rich references to remote content, covering the means for retrieving it from its original location.

It cannot be avoided that some content will need to be stored by the CE. For instance, privacy sensitive user data, or content generated by processes that run from within the CE.

The storage and availability of data produced during the TROMPA project will need to comply with open data and privacy guidelines set out in the project proposal.

2.1.1.6 Interlinking data

One of the main responsibilities of the Contributor Environment is to enable the interlinking of musical data objects, and to enable retrieval of this interlinked data. The internal data model should

provide sufficient expression to enable linking all types of (partial) pieces of data to other (partial) pieces of data, whether these pieces of data are stored in the Contributor Environment or only exist there as references to external locations.

2.1.1.7 Annotation data

One of the components is dedicated to annotations. Users with different levels of expertise should be able to create multiple types of annotations. The internal data model should allow to relate those annotations to their target (partial) piece of data, whether this piece of data is stored in the Contributor Environment or left at its original location. The internal data model should provide for annotations to be linked to their creators and have metadata like creation date.

The Contributor Environment interfaces (API) should provide a way to retrieve annotations along with the content, employing fragment identifiers to allow partial targeting of data resources.

2.1.1.8 Curation data

A central aim of the TROMPA project is to involve the crowd in curating data located in the Contributor Environment or in public repositories. The Contributor Environment access interface (API) should enable retrieving candidate pieces of data, or relations between pieces of data, for these crowd curation jobs. The internal data model should provide for enough expression to write the crowdsourcing results back on the relevant curated (partial) pieces of data, or on the curated (partial) relationships.

The Contributor Environment should have a mechanism that can evaluate curations at a granular level, and factor them in at searches and retrievals. For example, Crowd curation of a piece of data has determined that it is of low quality. From then on, the results of a search for this type of data will no longer include this piece.

The Contributor Environment should also offer a way to retrieve anonymised curations wholesale, to enable the refinement of participant algorithms while respecting user privacy.

2.1.1.9 Referred data integrity

Much (public) content will be left at its original (TROMPA-external) location, with only a reference stored in the Contributor Environment. As one of goals of TROMPA is to create a sustainable platform for classical music enrichment and sharing, the issue of integrity of this public content becomes important. A solution or best practice is needed for scenario's like '404 on referred location', 'content has changed', 'no longer aligns with contributed content', 'public rights no longer granted'. The issue can be divided into 'detection' and 'handling'.

2.1.2 Components

From project proposal WP5 objectives:

[...] the data produced in WP3 (musical repertoire, automatic descriptions and generated audio) and annotations delivered through WP4 need to be made accessible and usable in reusable components, meeting common standards [...]

Components are clients of the Contributor Environment, just like participant and partner applications. As described in the preceding section, the components use shared access interfaces and controls when interacting with the Contributor Environment, to get access to interlinked and

enriched musical data. Components might have special needs affecting the requirements for the internal data model and the access interface standards of the Contributor Environment.

Besides being Contributor Environment clients, components deliver functionalities to end-users, typically via a web browser or mobile applications. Five pilot applications are planned, each using one or more components. As stated in the project proposal, the requirement here is to follow common interface standards between components and end-user applications.

As end-users will interact directly with the components, these components will be an important factor in the performance requirements of the Contributor Environment.

2.1.2.1 Digital Score Edition

Digital Score Edition components present authored views of musical score interlinked with multimodal information resources using semantically appropriate terms. Digital rendering and interaction mechanisms reveal and clarify these interconnections, making them readily explorable by the reader. The multimodal resources themselves are incorporated by reference (see 2.5.9), providing a clean separation between source content and scholarly enrichment, and enabling multiple views of the same source material to be expressed by different authors. Provenance information regarding authorship is tracked and exposed (e.g. using PROV-O), supporting scholarly communication.

At a deeper level, the use of MEI as score format wherever possible allows the encoding of alternative versions or passages, as well as details of editorial or performer's decisions to be recorded (at any point) with full provenance. For example, an annotation exercise might be envisaged whereby users are asked to suggest which of several versions of a musical work was used for a given recording.

2.1.2.2 Multimodal presentation and visualization library

From the perspective of the Contributor Environment, the multimodal component can be understood as a rich search interface, allowing users and other components to retrieve interlinked data. Through query calls to the Contributor Environment API, this interface will provide users with a way to search through different types of data with a single search term, and with the possibility to filter on types of data. This requires unified metadata of high quality across all TROMPA data that needs to be available for querying, and an infrastructure that is able to handle complicated and deep queries fast enough to keep end users engaged.

Envisioned query results may go well beyond standard metadata matches. Where alignment data is available between different, potentially multimodal representations of the same musical object, say scores and recordings of a queried composition, algorithm and user input on either of these related data types can be aggregated and included in the response. This offers the user a much richer search result. A user query may not only consist of flat text, but may also include, for example, a selection of notes from a digitised score. Being able to handle non-textual queries like this would bring an extra dimension to multimodal representation of a musical work by, for example, returning musical works containing similar sequences of notes. To make these kinds of searches possible, the Contributor Environment API interface should be able to handle semantic and custom queries and the internal data model should not stand in the way of automatic indexes on cross-file alignments or algorithm created metadata files.

2.1.2.3 Annotation tool

The annotation tool provides an interface for contributors to TROMPA to listen to audio recordings and respond to tasks to provide time-aligned annotations related to the audio. These annotations could represent any required information about what is happening in the recording at a particular time, including instrumentation, tempo, mood, or section segmentation. This information can be further used to build an improve automated algorithms.

Annotation clients will be web-based applications and can access information about the TROMPA data through the CE API interface. This interface can be used to provide a list of metadata about recordings that are to be annotated. The annotator tool will download the audio related to an annotation task and present it to the contributor to perform the relevant annotation task. Once the annotation task has been completed, the annotation tool will push the completed annotations to the CE for storage and to update the data model to include information about this annotation. The annotations will be stored in a time-series annotation format such as JAMS. The annotation tools will be hosted external to the CE infrastructure and will need to allow contributors to authenticate to track their work.

2.1.2.4 Performance assessment engine

The performance assessment engine component provides automated characterisations of performance streams – retrospectively or in real-time – enabling the precise description and comparison of performance characteristics. Performance descriptions are produced in standardised formats and are associated with provenance information regarding the feature extraction process (software libraries / plugins and versions) to support fair comparisons between performances from potentially different data sources, as well as future reuse and re-interpretation. Provenance of the performance itself is tracked, providing performer attribution as well as a mechanism for licensing and permissions tracking.

Within the scope of this component, the Contributor Environment provides performance-derived features (performance characteristics and performance-score alignments) to client applications operating within a performance context (e.g., on stage or next to a practicing musician). In return, feature data derived from performance audio, MIDI, and/or other sensor data (e.g., key position trajectories) are provided back into the Contributor Environment by client applications. With performer permission, audio / MIDI streams (or, references to external repositories storing this data) are also contributed. Aggregation/comparison metadata providing summarised assessments of performance characteristics and quantified comparisons of such characterizations across performances are also determined and stored.

Metadata are generated by processes including automated alignment between audio- or MIDI-streams and score positions (Audio/Score Alignment); Audio-, MIDI-, and sensor data feature extraction; and aggregations / comparisons of derived features (across performances). These processing tasks need to be shared between client machines within the local performance context (on stage, or in the practice room) and the CE. The precise balance of load betweens these processing locations remains to be determined. From a usability perspective, moving processing responsibilities to the CE as much as possible reduces complexities (set-up and timing constraints, need for performative hardware) in the local performance context; but the need for real-time processing of performance data may preclude remote processing due to network / bandwidth constraints.

We expect metadata interconnections with external knowledge bases (e.g. MusicBrainz, wikidata) for work-level descriptions (e.g., interlinking to a conceptual representation of a particular Beethoven sonata), or person descriptions (of a particular performer or composer). We further expect interlinking with external media repositories for scores (e.g. MEI files) or multimedia recordings (e.g. Youtube), where such resources are not hosted in the CE directly.

2.2 Interdependencies

With its central role in the TROMPA project, as the platform from which all TROMPA data and functionalities are disseminated, the Data Infrastructure relies on many externals to do its job. At the other side, most TROMPA participant tasks rely on the CE or on the components to either fetch data or content from other participants or repositories, or to post generated metadata to be interlinked and disseminated to end users.

This section is an attempt to provide an overview, with short descriptions of, and requirements following from the interdependencies between the Data Infrastructure and those externals.

2.2.1 WP2 interdependencies

From project proposal WP2 objectives:

This work package elicits and prioritises the requirements for [...] target groups and [...] other stakeholders involved (e.g. musical partners, academic partners, and integrators). [...] the process is setup to align [...] technological ambitions with the needs from the target groups.

Users of the target group applications will expect responsiveness when interacting with the data. This data will be fetched from partner repositories and interlinked, enriched, annotated, crowd-curated and made available by means of the components within the Contributor Environment infrastructure. This way, the requirements for these applications also determine the lower limits of the performance requirements of the Contributor Environment.

The components have much in common with the other users of the Contributor Environment: contributors and partners. The users of the Contributor Environment are numerous, diverse and, at least during the project, in constant development. This determines requirements on the versatility and standardization of the interface used to access the data and functionalities of the Contributor Environment.

2.2.1.1 Application scenarios

Sufficient compute capacity should be available for computationally expensive operations, such as audio analysis and other participant algorithms. For example, timely processing in the performers and choir application scenarios needs to be ensured.

Once relevant algorithms become operational and demand patterns can be analysed in more detail, compute capacity might need to be reconsidered and setups adapted.

2.2.1.2 Functional requirements for integrated components

The CE should support internationalization at the UI level and for handling metadata in multiple languages due to localization requirements of 4.1.3 from D2.1.

Care should be taken with regards to data management in according to the GDPR regulations. All PII connected to a user account should be referenced across the data stores in the database. It should be possible to delete or sufficiently anonymize any data that is related to a user in an automated fashion.

Data anonymization and deletion policies should be defined and implemented across all relevant data stores for all collected PII. The actions of data deletion and anonymization, as well as of user's consent to the use of certain PII, should be logged and the logs persisted in a tamper-safe way.

Once the specific data flows are defined, a further internal GDPR compliance review will determine the technical measures that will ensure the compliance of the CE with the GDPR.

2.2.1.3 Technical integration

At M18 and M35, guidelines need to be delivered on a integration strategy for the data generated during the project. Although an integration strategy is a prerequisite for the Data Infrastructure, this task will inform requirements on the scalability of the Contributor Environment.

2.2.2 WP3 interdependencies

From project proposal WP3 objectives:

This WP provides Music Information Retrieval […] technologies to process digital music resources […] The resources of interest will be determined by the use-cases as posed in WP6. TROMPA focus is [on] the automatic description of audio-visually recorded performances and symbolically encoded

Which data the Contributor Environment has to handle is largely defined by the tasks in work package 3. At their basis is the content and metadata from a selection of public classical music repositories. On top of this, there will be the data produced by the various contributors which is related to the contents of those public repositories. The data models of the data from public repositories and those of the data produced by the contributors, inform the requirements for the internal data model that is to be used within the Contributor Environment.

These details also affect choices on what content from public repositories should be left at the original, externally hosted location (referenced by URL), and what data should be duplicated within the Contributor Environment to allow fast access to data interlinked from different public repositories or enriched by one of the contributors. These choices are closely related to performance requirements for the Contributor Environment.

Another concern is the automated processes (to be) created by contributors to generate new data. These processes will apply algorithms to data retrieved from public resources, from the Contributor Environment or from a pilot application user. Where these processes are initiated, where these processes will run (Contributor server, Contributor Environment or in component) and whether they are synchronous or asynchronous, will create additional requirements for the internal data model, for the access interfaces and for Data Infrastructure performance.

2.2.2.1 Selection of public Musical Repertoire Repositories

From project proposal 'Relation to the work program':

TROMPA combines the wealth of hybrid content […] from public repositories and partner-contributed collections. The TROMPA project will generate derived knowledge and meaning

from these collections, as well as links between related resources in different modalities. This information will be contributed back as open data to these collections to foster as much enrichment as possible.

Public repositories the Contributor Environment will interact with:

- TROMPA Partners
  - CDR Muziekweb catalogue
- TROMPA Associated Partners
  - IMSLP Petrucci Music Library
  - British Library
  - Escola Superior de Música de Catalunya - ESMUC
  - Barcelona Town Hall / Institute of Culture - ICUB
  - European Choral Association – Europa Cantat
  - Wikidata
- External Repositories
  - Europeana Music
  - Choral Public Domain Library (CPDL)
  - MuseScore
  - Répertoire International des Sources Musicales (RISM)
  - ECOLM - An electronic corpus of Lute music
  - EMO - Early Music Online
  - AcousticBrainz
  - MusicBrainz
  - YouTube
  - Kunst der Fuge

Some repositories provide public APIs that will allow us to retrieve data on demand when needed for a task. The Contributor Environment will store a reference to objects available in these repositories and provide a URI to allow clients to access this content directly.

Some repositories have a public API but have expressed a wish that we keep a local cache of that data. Some repositories have no public interface to the data, and we expect to receive an archive of the content or a subset of the content that they hold. In these cases the Contributor Environment will have to store these objects in an internal storage system and provide clients with a link to the cached version of these objects.

The specific manner in which we interact with each repository will be clarified at the moment of the integration of each repository in the TROMPA project.

2.2.2.2 Automatic transcription

Automatic transcription is a process using software to extract symbolic or semantic information from video and sound recordings of music. This process will analyse input data and generate a symbolic output representing the input. Such a system would need to retrieve input content from the Contributor Environment, and store the output of the algorithms and the relationships of this output to the original input in the Contributor Environment. The input can include audio, video, scores and symbolic representations. The system will also use manually created information, for example, annotations made by crowdsourcing processes.

These systems will be based on the Essentia signal processing library, developed in the Music Technology Group, and will run on UPF computing infrastructure.

2.2.2.3 Optical to symbolic score conversion

Commercial Optical Music Recognition (OMR) programs aim to provide completed score-representations as output (whether in MIDI, MusicXML or proprietary formats) and do not expose intermediate data. For TROMPA we need to be able to address recognised elements within an OMR system before they are passed as input to an interpretative process such as score assembly. This is best achieved by the use of a persistent unique ID provided for each recognised symbol. This ID can then be passed unchanged through any ensuing process, thus ensuring that at any point the musical data-element in the final output can be associated directly with the recognised symbol.

In the specialist early-music OMR program Aruspix, this is achieved by providing the MEI output with unique xml:id attributes for every recognised symbol. While this program does not attempt score-assembly from individual voice-parts, the method allows for continued linkage back to the original image for any such process. This is particularly important for parallel or alternative visualisations of image and rendered music; it will also be essential in audio/score/image alignment, which would be desirable within the context of music-scholars' use-cases.

2.2.2.4 Score performance alignment

Score performance alignment necessitates Music Information Retrieval (MIR) components capable of aligning audio and symbolic music information (audio/score alignment); optionally, this could be done by synthesising the symbolic representation into an audio signal, and then performing audio/audio alignment, leveraging techniques such as Dynamic Time Warping (DTW). The decision of a precise means of producing such alignments may be left open at this stage, and indeed different approaches may be trialed or combined, as long as each alignment algorithm generates suitable provenance information (software agent and version).

Automated alignment outcomes are excellent candidates for crowdsourced evaluations. These evaluations can then be used to correct an outcome and also as feedback on the automated alignment algorithm.

2.2.3 WP4 interdependencies

From project proposal WP4 objectives:

This WP focuses on involving the crowd [...] in unlocking knowledge and expressing own perspectives on the music material [...] It exploits the added value of human annotations for the description of multimodal music information and its use in the improvement of automatic algorithms.

Regarding this work package, the responsibilities of the Contributor Environment are two-sided; On one end, WP4 contributors must be able to find and retrieve data to be crowd-annotated from the Contributor Environment. On the other end, WP4 contributors must then be able to store crowdsourced results or at least have a way to relate these results to the original pieces of data. This adds internal data model requirements for the Contributor Environment as well as for the versatility of the interface(s) to access the data.

WP4 also provides for evaluating annotators and allow this to be a factor in the crowdsourcing of data. Furthermore, crowdsourced results are also to be used to improve the algorithms from WP3. Details on how annotator evaluation will be factored in, and on how crowdsourced results are to be retrieved to allow algorithm refinement, affect the internal data model requirements for the Contributor Environment.

2.2.3.1 Data evaluation and improvement through Crowdsourcing

Users of TROMPA will be able to evaluate Contributor Environment data through a variety of crowdsourcing tasks. Their contributions in those tasks will help the sustainability of CE's databases and improve existing data and links. These crowdsourcing tasks will take place using platforms which accommodate content annotation tasks. These platforms include crowdsourcing platforms like Amazon Mechanical Turk and FigureEight, as well as online Social Networks such as Twitter and Facebook. The produced annotations will also be made available for evaluation and improvement of algorithms applied within the Contributor Environment. Each human- or automatically-generated annotation will be assigned a confidence measure, which will describe the level of uncertainty of the annotation according to user reputation score or algorithmic confidence measures. The system will be able to automatically recognise low-confidence annotations and propose them for evaluation.

2.2.3.2 Annotator evaluation

The annotators of TROMPA will be assigned profiles which contain metadata from their activity in content annotation platforms (e.g. URL of a profile, the mastery level in Amazon Mechanical Turk) as well as of their competence. Their competence profiles will contain information about each user with their knowledge in TROMPA related fields (e.g. how proficient they are on playing piano or a relevant music degree). Through those profiles, we can algorithmically infer which users will be more capable for specific crowdsourcing tasks as well as adjusting their profiles based on their performance on those tasks.

2.2.4 Applications

From project proposal 1.3.2 Methodology

TROMPA methodology is based on the principles of co-creation [...], citizen science [...], and crowdsourcing [...] TROMPA will integrate these principles by targeting five use cases, which require research into similar music data processing methods and types of user feedback, but will lead to tools, infrastructure and applications targeting different dedicated audiences

During the TROMPA project, the applications targeting our five use cases will be the most important clients of the Data Infrastructure. They will make the raw material of partners and the work of all participants come to life. These applications will be the part of the TROMPA project facing the public. This public audience will not only be the consumers of the TROMPA enriched musical data, they will also be important contributors.

Ultimately, it will be the designs and the user-stories for these applications that will point out where the efforts of the participants should be concentrated, and to what specifications the Data Infrastructure needs to be built.

2.2.4.1 Music scholars

The basic requirement will be to provide music scholars with the means to interact with large quantities of musical data in ways that are not possible using 'traditional' methods. These mainly involve the use of music information retrieval (MIR) techniques (on collections built from score or audio) plus the semantic affordances of methods such as Linked Open Data, which can link both within the music domain (to multiple locations in a given score/audio, or to other manifestations or even to other works), or outside (to musical literature, or to artistic, historical, geographical, biographical, and other data as appropriate). In some respects (e.g. with standardised personal names or geographical locations) this can be done to a large extent automatically (e.g. using MELD); but it is necessary to allow for such links to be established and verified by human annotation.

Music scholars need to be able to find musical parallels at several levels, often simultaneously. These might be 'musical similarities' between works, or passages within works, or they might be the common use of melodies or harmonic sequences, down to motifs of just a few notes, which sometimes carry external semantic significance which might also be the subject of annotation.

A user should therefore be able to select sections of a displayed score, including arbitrary sets of separate polyphonic lines/instruments, which can then be submitted as queries to a search engine. The actual search method will be dependent on the nature of the query (and may involve further specification by the user): a short melodic fragment will in any case require a different search strategy than a page of full score. A similar selection method for audio search/comparison would be desirable, preferably linked to score visualisations where available.

In order to allow these types of scholarly searches to be supported, the Contributor Environment access interface should support MEI formatted segments to be part of a query. Within the Contributor Environment, this query should be processed against a service containing an index of relevant patterns within known MEI documents or even public resources like audio recordings. Whether this service, when built, is accessible as a remote API, or to be hosted as an available process from within remains to be decided at this point. In any case, the internal data model should allow the interlinking of the results to additional public data through standardised metadata.

2.2.4.2 Content owners

The goal of this Pilot application is to make the scores of all Mahler's work available for usage not only by orchestras, but also by music enthusiasts and the general public.

These users should be able to use the scores for performances (orchestras), to annotate these scores at the granular level and to use them as an entry point to discover more about Mahler, performances of his work and about classical music in general.

Phase one will have experts transcode the scores of 3 Mahler symphonies to MEI format, to be hosted or served by the Contributor Environment. The Contributor Environment should be able to handle listings of and searches for scores, and to serve scores in MEI format. The score edition component should be able to translate a MEI document to a interactive visualisation layer displaying this score in various types of user devices.

The next step will be to have this transcoding process automated using crowdsourcing. Via the annotation tool component, and maybe also via other intermediaries, users should be able to receive and process microtasks based on a (potentially incomplete) MEI document. The Contributor Environment should be able to maintain this MEI document and process incoming crowd corrections and additions at a granular level.

The last step will be to interlink these scores with additional public data about Mahler, his work and recordings. The multimodal component should provide for a search interface for users to discover more about Mahler and his work, recordings and the classical music domain. The Contributor Environment should support the multimodal component with an interface for deep searches through interlinked content including filtering on metadata properties.

2.2.4.3 Instrument players

Instrument players will benefit from a "Performance Companion" capable of tracking, characterising, and analysing performances, post-hoc or in real time. To realise this, the Contributor Environment must provide a data model integrating musical score (encoded as a web-addressable musical structure, using MEI) and timed multimedia streams (e.g., audio/video, performance metadata feeds), exposing each media entity for semantic description and annotation (using Web Annotations).

The Performance Companion will expose a score-following (audio/score alignment) system matching performances to identified positions in the score. Recorded performance-characteristic metadata must be aligned with both the individual performance timeline and score position. MIR algorithms will be used to provide quantified measures of performance characteristics, and to compare these to other performances in order to compute inter-performance similarity measures, or to visualise the evolution of performance characteristics over time (e.g. for pedagogical reasons).

2.2.4.4 Choirs

The goal of the Choir Singers Pilot (CSP) is to assist amateur choir singers during individual performance. Users of the pilot should be able to synthesize existing scores, to sing-along with the synthesized voices, and to receive feedback on their performance.

Following this, the Contributor Environment should be able to handle searches for choral scores and to provide them to the CSP. The synthesis of the scores will take place in the Voiceful Cloud API provided by Voctro Labs, and the results will also be stored there, accessible through a public URL. The same applies for performance analysis input and results (computed and stored in the cloud and accessible through public URLs).

In order to keep track of the generated scores, the Contributor Environment and the CSP should interchange metadata with session information. E.g. if the user that arrives to the CSP through the Contribution Environment is logged in, the CSP should receive the ID of the user (and other possible relevant metadata associated to her). Similarly, if the user generates new material (synthesized voices, analyzed performances), the CSP should communicate the Contributor Environment about this new material so that this new generated data can be associated to the original score and/or the user who generated it.

Finally, in order to use input from the community to improve the voice synthesis algorithms, the Contributor Environment should allow to provide general scores for the synthesis (e.g. by rating the overall quality of the synthesis) and to make timestamped annotations for the generated material, i.e., allowing the user to input free text comments to inform about specific problems (e.g. "this phoneme sounds weird at this point in time in the soprano voice").

2.2.4.5 Music enthusiasts

The goal of this Pilot is to provide novel and playful interaction mechanisms for musical cultural heritage content aimed at people without formal musical knowledge, but with interest in learning more about music. These interactions will be mediated in a first approach through a MOOC in which participants will acquire basic knowledge and skills related to emotion analysis in music, and then they will be asked to create new learning material and find new links between the resources in repositories using the public-domain content and analysis and annotation tools provided by TROMPA.

Thus, the users (MOOC participants) should be able to interact with the data, through recommendation systems and semantic search techniques, accessing to the tools from a multiplatform interface. They will also provide ground truth data for these recommendation systems, as well as to evaluate the automated processes used for emotion analysis within the context of TROMPA. Therefore users should be able to access sections of specific pieces suggested by experts, in order to explore and link their own experienced emotions with performance and contextual features. Then, according to the suggestions and the analysis, users should be able to explore other similar/related pieces.

Likewise, for the Music Enthusiasts Pilot, The Contributor Environment interfaces (API) should provide a way to retrieve annotations along with the content and related information (user, timestamp, etc.).

2.2.4.6 Third party applications

To encourage third party developers to use the components in existing or new applications, these components need to live up to industry standards in use with professional application builders today and for some time in the future. The aim would be for a widely used and respected open source frontend framework, compatible with the Contributor Environment API interface of choice.

As the goal is to allow applications to combine multiple components, choosing the same framework for all components would be a strong preference .

In light of a potential requirement on (algorithmic) asynchronous processes to be requested from the application (e.g. score alignment, audio synthesis), or live feedback from other users, it is advisable to choose a framework that supports real time data transfers from and to the server, for example, web socket connections.

3 Data Infrastructure requirements

This section will group and summarize the requirements for the Data Infrastructure from section 2.

In section 4, the resulting set of requirements is then translated into a coherent collection of specifications that are expected to satisfy all those requirements.

3.1 Performance

In the context of Data Infrastructure requirements, performance has to be understood in terms of how well the system as a whole behaves under workload. From section 2, it follows that there are two main performance aspects to consider: responsiveness and quality. Responsiveness is about how fast the system will respond to user requests like a search query, the posting of an annotation on a score, or the analyses of a voice recording after it is uploaded. Quality is about the completeness and accuracy of the responses.

3.1.1 Responsiveness

Responsiveness requirements originate mainly from pilot applications that will serve users who expect multimodal data, often aggregated from several sources and/or algorithms, to be available immediately. Even simple user queries and mutation requests will have a lifecycle that touches multiple parts within the Data Infrastructure, through the interface to the database and back, directly or indirectly via components. More complicated requests like the analyses of an uploaded voice recording, or the generation of a PDF file from a richly annotated MEI document, will additionally depend on process(es) that will be triggered by this request.

The priority will be to ensure responsiveness for the parts that are used for every request: the CE API interface and database. For all other parts, like public repositories and participant systems that are hosted either within the CE or outside, there should be continuous focus on minimising latencies. Where dependency processes are slow, efforts should be made to facilitate feedback to users on progress and completion of tasks. Furthermore, all parts will need a scaling strategy to maintain responsiveness when user numbers increase.

Components will need to be build on a framework that is well versed in rich interactions with an API backend, with a minimum of requests and data conversions. Another preference is for a component framework that natively supports socket connections to allow user feedback on slow or asynchronous requests.

3.1.2 Quality

To be of use for professional or amateur end users within the classical music realm, the various CE clients require the data stored, managed and returned by the CE to be accurate and richly interlinked.

To maintain accuracy and interlinkedness across the data provided and enriched by multiple participants and public resources, the CE needs to be built on a data model and API that is clear and straightforward to use. The API interface through which CE clients will access TROMPA data should leave little room for uncertainty or error when fetching or manipulating data.

Requiring to aim for simplicity in the internal data model will go a long way in allowing the various participants to access the CE with little ambiguity, and maintain a high degree of 'hygiene' in the resulting data sets. To this end, documentation on CE API functionalities and conventions should be available and maintained. Additionally, API endpoints providing self-documentation and including discoverability features in the API output is highly recommended.

For developers working on the CE, best practices in maintaining API backward compatibility, versioning and communicating version changes need to be encouraged and where possible enforced.

Like all CE clients, the components will benefit from aforementioned requirements and recommendations. Further gains can be attained by adopting a single framework for the development of all components. Besides less variance in practices and better communication between component developers, code written for CE interactions can then be easily shared between components.

As the CE will follow a policy of leaving public resources at their original location instead of creating a duplicate, this referenced data is prone to quality problems that are outside the control of CE or even TROMPA participants. A best practice approach is to be defined of how to continuously audit those references and how to handle any discrepancies, or fail gracefully where these cannot be accommodated.

3.2 API Interfaces

The project proposal requirements call for at least a RESTful API interface.

The anticipated CE clients generally need to be able to do deep and flexible semantic queries, as well as postings. To natively support deep and semantic querying, planning and dissemination needs suggest the need for a secondary API interface specialised for graph querying, following open standards. Planning and dissemination needs suggest an API interface that follows open standards. This leaves the way open to a secondary API interface, better suited to semantic searches, as long as it follows open standards.

The CE will consist of a central application tasked with managing the main database, in conjunction with a number of participant applications that provide specialized functionalities ranging from crowd sourcing management to the analyses of user recordings. From a CE client perspective, the API interface should be able to be opaque to this setup and completely hide the partial rerouting, subtasking and aggregation of results that a request might involve.

The preliminary requirements do not mention the need for customised API input or output data formats. Therefore, the preference is for an open data format standard, acknowledging that data hosted externally and referenced from the CE by URL may be represented in any given format. In order to handle internationalised input and output, the API interface should provide for a method to indicate the language in which metadata is posted or in which the result is expected to be translated.

As there is the potential of secondary processes triggered by an API request, responses could take too long for synchronous API responses. An API interface that is able to handle an asynchronous request/response cycle has a strong preference.

For scholarly research and to support participants to encode and decode data stored in the CE, support for RDF output is highly recommended. Such a feature would also add a valuable advantage to applications using RDF statements to optimise for search engines.

Adopting an open standard API interface has the advantage of existing documentation and online community support. This will make participants more self-reliant when adapting to the CE API and lower support pressure on CE development. An API interface that supports dynamic online API Docs or some other means of auto discovery through the API itself, would have a strong preference. Where the standard documentation and dynamic discovery features of the API interface does not suffice, CE developers will maintain API documentation aimed at the users of the CE API, including examples.

3.3 Components

To ensure a consistent user experience across all components, and promote reuse of components and code across pilot applications, there is a strong argument to use a single frontend framework as the basis for all components. Besides providing frontend functionalities, this framework will in effect be the interface between the application and the CE API, so it needs to comply well to the API interface of choice.

Choosing a popular framework has many advantages that are in the interest of the TROMPA project. As several different participants will be developing the components, it will be easier to find developers with experience. A popular framework also ensures that the threshold for other developers to use a component to enhance their own application remains low, increasing the chances of uptake of the components.

Pilot applications need to be able to be used on mobile devices. A framework that is primarily build for, or at least supports mobile devices has preference.

3.4 Internal data model

For the internal data model, two central needs emerge; On one side the data model should be designed for the interlinking of diverse types of data originating from different sources, which calls for a model that unifies the data to common ground. On the other side the data model should enable the various CE clients to store and retrieve data on their own terms.

Most data will remain at its original location, which necessitates referencing with sufficient detail for retrieval, potentially going beyond just storing a URL. Some referenced data will be behind a protocol or will be prone to change, so the relevant information to circumvent these situations should be stored along with the reference. The model should provide for properties to track the provenance of, for example, aggregated data.

Related to these main needs is that the internal data model has to provide for features that allow annotations and crowd curations to be targeted at any musical data object known within the CE. These features should include the possibility to target a segment (resource fragment) of an object, for example, to target a note within a score, or to target a time range within a recording.

The output of the CE API will be multilingual. The internal data model should support internationalisation on all data classes and properties that can contain publicly available data, including the references to remote data.

The internal data model should further provide rich user features, in order to serve users with features like recommendations and bookmarks, but also to control users' access and track expertise. A public profile should be part of the user model.

A number of participants have expressed a potential need for RDF output. Especially for scholarly research of the TROMPA data set, RDF output would be of high value. Basing classes and properties on well known RDF schemas has the additional benefit of creating a dataset that is prepared for semantic web usage.

Taken together, these needs should not lead to an overly complex data model. Most participants will not only query, but also contribute data to the TROMPA data set. A complex internal data model to comply with will be a source of ambiguity and of increased communications with CE developers, that might emperil the data quality and also the planning of the project as a whole.

3.5 Data storage

The Data Infrastructure will need to handle data from many different sources and in many different formats. Its most important function is to maintain cross references between this diverse data, to enable performant queries on these cross references and allow prompt aggregation of results.

A balance needs to be found as to what data is stored in a database, allowing fast queries, and what data is to be kept at data stores and public repositories, supporting large bodies of data in diverse formats, and avoiding excessive data transfers and rights issues.

TROMPA's Data Infrastructure will conform to the FAIR guidelines as outlined in D8.4. As this is a living document, this is an ongoing process during the TROMPA project.

3.5.1 Database

The main requirements for the database fuelling the CE are to support rich relations between objects and to allow fast flexible queries for objects that might be connected to other objects over a large number of relation-object 'hops'. Impact of a growing user base during and after the project must not have adverse effect on query performance, which means horizontal scaling options (more read copies) of the database should be a possibility.

To maintain a cost effective setup and fast queries, the aim is to only store relevant (meta)data, references, curation data and annotations, needed to fulfill queries on the dataset, and to leave large bodies of data or excessive amounts of unused objects out of the database. This way, the anticipated volume of the database will remain average at most, and the necessity of sharding is not foreseen.

3.5.2 Data stores

With the exception of metadata, the data from public repositories will remain at its original (externally-hosted) location if possible.

The same applies for large bodies of data produced by TROMPA participants or uploaded by users. This data will either be stored on a location managed by the participant, or in a data store managed by the CE. The main considerations here are user privacy and latency from the end user perspective.

All data that can be associated with users will be stored in the CE handled data store. This way, privacy standards and control over user data can be maintained and allows for straightforward removal of a users' data on request. Moreover, the CE handled data store will be for TROMPA generated data that needs to be stored secure, redundant and with superior performance standards.

3.6 Participant services

The Contributor Environment part of the Data Infrastructure will consist mainly of a CE primary application, handling incoming requests and aggregating responses. This primary application has dependencies on services provided by participants, like processes running algorithms on demand. For example, an alignment file needs to be interpreted to determine the place of a score annotation on the timeline of a recording, or a MEI document needs to be generated on the basis of a base MEI document and a layer of MEI snippet annotations.

These services can run on participant infrastructure and be made available to the CE by an API interface. These external participant services will need clear documentation and maintain a strict versioning or backward compatibility policies that will assure the CE can continue including those external functionalities.

Some participants have indicated that they prefer to run those dependency services within the CE infrastructure as secondary CE applications. This is because these dependency services will be set up exclusively for the TROMPA project and also to mitigate latencies caused by additional network overhead or reliance on less performant web infrastructure. These internal dependencies, or secondary CE applications, will then share hardware and deployment schemes with the primary CE application. The CE system architecture will need to provide for a way to separate the concerns of primary and secondary applications to minimise interference of development cycles, as this will add complications and affect planning.

3.7 Access control

Some of the functionalities of the CE should be restricted to certain users only, as they are potentially destructive or can compromise user privacy. As there will be multiple ways to access the CE, for example, through the API interface or through one of the components, there should be an Access Control (AC) layer that is able to authorise an authenticated user to only access a part of the data or functionalities.

This authentication part of the AC should be coupled with user management to enhance the user experience. Independent of which way a user enters, she will encounter similar access rights. This authenticated identity should be used to add additional functionalities to a user account, like coupling a public profile or the storage of bookmarks on content.

It should also be possible to make users part of one or more groups. This functionality can be used to easily grant CE rights to whole groups, but also, for example, to allow a group of musicians annotating a certain score, to create a PDF edition of that score including only the group’s notations.

4 Specifications

This section will present the provisional specifications for the TROMPA Data Infrastructure. Taken together, the specifications presented in this section should meet most requirements listed in section 3, and prepare for meeting the remainder of the requirements during the project.

The requirements these specifications are based on are provisional and based on a consortium-wide best effort to think through and define the various subtasks of the TROMPA project, from the perspective of how these tasks will be dependent on, or are to be a dependency for the TROMPA Data Infrastructure.

It is not possible to fully foresee how these interdependencies and requirements will evolve once the participants start realising the Data Infrastructure and interdependent tasks. In this regard, and as per requirement, a sensible amount of flexibility will be achieved with these specifications. This flexibility provides capacity to handle those additional requirements without the need to respecify and rebuild parts of the Data Infrastructure at a fundamental level.

4.1 Internal data model

The main requirements for the internal data model of the Contributor Environment (CE) are twofold and may seem contradictory at first; Participants need to be able to store and retrieve data that conforms to their own data model, in order to pre- or post process it. On the other side, the data of different participants needs to be interlinked, not only to data of other participants but also to data from partners, public repositories or the web in general. This is necessary to achieve the multimodal enrichment and Linked Open Data goals of TROMPA.

A solution that meets both requirements can be found in an internal model that CE clients, like participants, have to comply with when interacting with the CE. If this internal model provides for a mechanism for clients to preserve (part of) the structure and nomenclature of their preferred data model, it remains possible to convert CE responses back to the client data model if needed. Such a data model creates the common ground on which a quality performant Contributor Environment and components can be built. Requiring participants to translate data to a common internal model has other benefits as well: by grounding this internal data model in metadata and open web structure standards, we can ensure that CE content conforms to conventions that will allow better sharing and discovery on the wider web. Furthermore, if those open web standards are derived from stable RDF ontologies, a path to scholarly research on RDF output, or on a full TROMPA dataset RDF triple dump, remains open.

Further requirements call for allowing participant data models to evolve. When participants have to conform to an internal data model as mentioned above, the responsibility for CE compliance remains with participants; a change in their data model may lead to a change in the conversion of data to and from the CE, but not to a change in the CE internal data model. The internal data model should provide ample room though, for client data models to be expressed in CE internal data model terms. It is impossible to anticipate all future participant needs. Therefore the feature request procedure should allow for internal data model discussions and proposals to facilitate a manageable path to changes in the internal model.

4.1.1 Classes

Base classes of the internal data model are adopted from the 8 top level classes of schema.org. Schema.org provides an open standard for structured data on the internet and is maintained in collaboration with the W3 community. These base classes create a thorough yet clear data model that is easy to comprehend and adapt to by the various CE clients. It also ensures broad applicability and interlinking of the CE data throughout the web.

Additionally, several classes are added that will take care of TROMPA specific needs for storing references, annotations, curations and users. Like the base classes, these additional classes are adopted from stable and wide known ontologies. In theory, further classes could be added to allow unforeseen functionalities that cannot be accommodated with the initial set of classes.

Appendix A has an overview of the base classes on which the internal data model will initially be based, including their origin and RDF URI.

4.1.2 Properties

Apart from default class properties, additional properties are added on a per-class basis. These additional properties are adopted from other stable and well known ontologies, to allow generic metadata, quality assertions, provenance tracking, relations to segments of objects and storage of public repository id's.

4.1.3 Additional classes and properties

Instances of each class (objects) are allowed to be labelled as one or more additional classes. This way, a CE client can store objects while retaining the entity class(es) corresponding to their own data model.

Where the default properties of a class are not sufficient, each instance can have additional properties added. These properties can have a scalar value or be a relation to another instance or collection of instances. These additional properties can be classed and labelled according to the clients data model, enhancing expression of the original client model in the internal data model. The additional property and class information can be output along with the values, which will enable clients to convert data back to their own data model and to RDF.

4.1.4 Internationalisation

To support internationalization, textual translations for a fixed number of supported languages are stored in objects of the class matching the translated object. A translation object is labeled as such, and only its scalar properties, excluding the ID, are relevant and interpreted. A translation object has no relevant relations other than to the translated object. Translation objects will not be directly exposed to querying, but will only be interpreted for set properties, when matching the language set in the ACCEPT-LANGUAGE header in the request. By default, base class objects will contain properties in English. If for a given property no translation is set, the output will default to English, even if this does not match the language in the ACCEPT-LANGUAGE header.

The same mechanism will be used for references. If for a specific language another, different file needs to be referenced, the reference translation instance will contain the URL matching the specific language.

4.2 API Interfaces

In addition to the RESTful API required by the project proposal, the CE will provide a secondary graph querying interface specialised to handle deep semantic queries.

An open API interface standard that fits these requirements is GraphQL. It is a query language specification that was developed for mobile app development. Since the GraphQL specification became available under an Open Web Foundation license, it is growing in popularity as an alternative to REST API's for serving web applications.

A GraphQL implementation allows clients to query an API through a single endpoint by means of a request body containing the query. This query body can be dynamically composed by the client, and permits the client to define the classes, relations and properties it needs returned, which is effectively an enriched filter mechanism. This query body approach fits well with the requirement of an API interface that is able to hide from the client the complexities of a response that is aggregated from the subresults of different applications run or managed from within the CE. The default output data format for a GraphQL endpoint is JSON, which is a widely used standard for web applications.

Subscriptions to server-side events like process results becoming available, or the occurrence of object mutations, are part of the GraphQL specification. Implementing the handling of these subscriptions over socket connections are a web application best practice when handling asynchronous requests, and will meet the requirement. Appendix B contains an example of a request and a response in compliance with GraphQL specifications.

The GraphQL specification is well documented and supported by an active community. The specification also includes introspection features that allow users to discover API features and underlying data models.

By fully implementing a GraphQL endpoint, the CE redeems all but two of the API interface requirements: RDF output and internationalisation.

No RDF support is specified by default, but by extending the JSON output with customized contexts according to the W3C JSON-LD standard, this requirement can be met.

The specification does not describe how to handle internationalisation. A straightforward solution is to use the ACCEPT-LANGUAGE and CONTENT-LANGUAGE headers for the HTTP requests. If the GraphQL implementation honors those headers, with a fallback to a default language (English), this requirement is met.

A GraphQL implementation is not ideal for all requested functionalities. Although it is possible to create custom request/response cycles to handle, for example, file uploads or 'delete all user data' requests, it is easier to expose these functionalities through the REST interface.

Appendix B contains an example of a request and a response

4.3 Technology stack

To build the Data Infrastructure according to requirements, a number of technologies need to be used that fit together coherently. These technologies range from the database type, the servers, the

software in which Contributor Environment applications and services are written, the software versioning, to the library used for the components. Whereas each of these technologies could be one out of many available, some fit better together than others, certainly when considered within the context of a set of project requirements.

The technologies proposed in this chapter are all mature and widely used technologies by themselves. Together, they can provide a consistent basis that allows the Data Infrastructure to be developed to requirements. This can be done for and by the various participants, with little friction between technologies and in a manageable way. These proposed technologies also ensure that once the Data Infrastructure gets deployed for production, it is ready to scale according to growing user numbers and load.

4.3.1 Database type

Although multiple database types are eligible, a strong preference for the database type is the Neo4j labeled graph database. Graph databases in general are well suited to store data where there is emphasis on rich relations, as is the case in the CE. The Neo4j database is a production-worthy and widely used native graph database, performing exceptionally well for deep queries. Neo4j is horizontally scalable in a cluster setup. Also, it fits well with the GraphQL API interface of choice. It meets all requirements.

As a graph database is essentially schema-less, changes in the internal data model are less consequential than when using relational database types. This is convenient during the development phase, as this will lower the need for database migrations. This lack of database migrations after data model updates simplifies automatic deployment and continuous integration schemes.

Neo4j is proprietary software. It has a community edition with limited performance specifications that will suffice during the initial development phase of TROMPA. Once performance needs or user numbers grow significantly, an evaluation license and eventually a paid enterprise edition license will be needed.

4.3.2 Containerization

The Contributor Environment part of the Data Infrastructure will consist of multiple service applications as illustrated in Figure 4.1. The primary application accepts incoming requests and aggregates responses, and practically consists of the implementation of the GraphQL and REST interfaces. Depending on the request body or on the outcome of the database query, additional services need to be called upon to complete the response. These participant services can be hosted on participant infrastructure and be available through an API.

Some of those services are required to run from within the CE as secondary applications and will share hardware and deployment schemes with the primary CE application.

In order to maintain some separation of concerns and not have development cycles interfere, CE applications will be built within Docker containers. From the perspective of the primary or a secondary application, any other CE application should be manifested as an isolated Docker container. This grants each CE application its own context and development cycle, but requires these applications to implement some kind of API to interact with each other. A condition for running a secondary application within CE is that its functionalities are only exposed to the primary application, and not directly to CE clients.

Like external services, clear documentation of the CE applications and maintaining a strict versioning policy or backward compatibility is paramount.

Using Docker containers has an additional advantage. Kubernetes is an open source solution to manage a microservice system architecture based on (Docker) containers. In conjunction with AWS services, it is possible to configure and run a constellation of Docker containers and run them on an arbitrary number of virtual servers. With CE applications cleanly designed to run in separate Docker containers, and with a well configured Kubernetes setup, it becomes trivial to scale the Contributor Environment up or down depending on traffic. Another advantage of a well configured Kubernetes - Docker combination is that this setup can function as an advanced auto deployment system; a new version of any of the CE applications can be deployed within an environment with preconfigured parameters and versions of the other applications. This simplifies the technical aspects of maintaining different CE environments for development, staging and production.

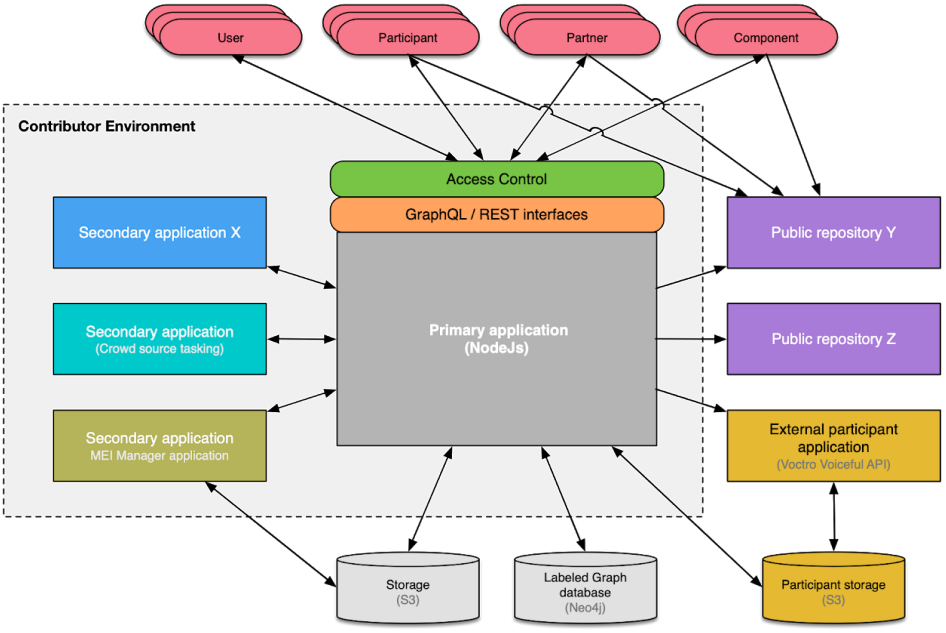

Figure 4.1. Contributor Environment service application architecture

4.4 Software

The Data Infrastructure requires software to be written for CE applications and components.

The Contributor Environment includes a number of applications (Figure 4.1), of which only the primary application needs to be created in the context of this deliverable. The remainder of (secondary) applications are to be written by TROMPA participants as tasks and deliverables of other work packages.

The five components are all to be created within the context of this deliverable.

4.4.1 Contributor Environment application

Our preference is to create the CE application in Node.js, which is a JavaScript based platform to write server side applications. Node.js is a popular and widely used platform designed for web application development.

There are several reasons to prefer Node.js for the development of the primary CE application;

- Node is written in Javascript, a generic, broadly used software language with a large supportive open source community
- Node.js natively supports websocket connections, which solve the need for backend support for a GraphQL subscription implementation
- Node.js is fast when compared to other server-side languages or platforms
- Node.js integrates well with the GraphQL specification
- Node.js integrates well with a Neo4j labelled graph database
- Javascript is also the software language used for React, the library proposed for component development

4.4.2 Components

Our preference is to create the components on the basis of React, which is a JavaScript open source library to build user interfaces. React is a popular and widely used library, mainly used for frontend web application development.

The reasons to prefer React for the development of all 5 components are:

- JavaScript is the same language as for Node.js, the platform on which the CE primary application is to be built
- React fits with the requirements for all 5 components
- Using the same React framework for all components allows and encourages code portability
- React integrates well with the GraphQL API interface of the CE
- React is created to perform well in a large collection of browsers
- React-native allows developers to re-use React components for mobile app development

4.5 Data store

Although content typically is stored outside of the TROMPA Data Infrastructure, there will be cases where it is necessary or convenient to have a data store controlled by the CE. Private data of users like recordings or uploaded images, has to be either anonymised or stored in a data store controlled by the CE. This way, this data can be removed at the user's request without reliance on external services. Some participant services will run applications within the CE for performance reasons, and might need performant storage as well.

For these cases the CE will include access to AWS cloud storage. This ensures a performant, cost-effective and versatile store for data that is uploaded by users, and for TROMPA-produced data that cannot be hosted by the participant.

4.6 Access control

The best way to guarantee uniform Access Control (AC) for users approaching the CE via different paths, is to make use of an external identity provider.

The CE can only be accessed through a Access Control layer and will not allow anyone access unless a valid token is presented. This token can only be obtained from the identity provider (Figure 4.2). Before granting access, the CE will use the token to inquire at the same identity provider and provide access (or not) based on the rights as set for that user. This way a CE user could seek access to the CE through different paths, but would not get access unless the identity provider authenticates the user.

We prefer to use the AWS Cognito service for this purpose. Cognito is a performant and cost effective service that provides user management, but also allows identity federation to 3rd party identity providers like Facebook and Google. Cognito provides also some additional services, like storage of user preferences across multiple devices, that could become helpful at a later stage.

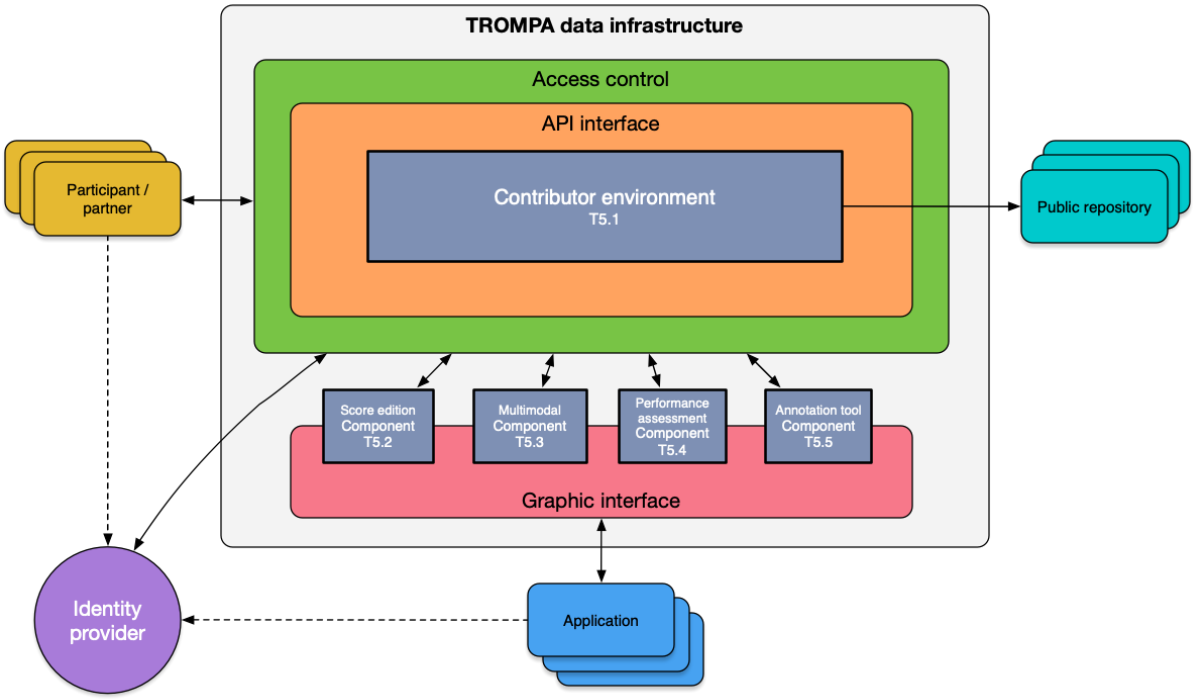

Figure 4.2. Contributor Environment extended with a Access Control layer

5 Planning

In this section we will provide an approximate implementation plan of the Data Infrastructure along with its dependencies with tasks from other WPs. Following the DoA and taking into account the current status of the project, the dependencies of the Data Infrastructure for the next period can be separated in the following categories:

- **Individual development of the CE and the other components of WP5**: Apart from the CE, there are four components to be developed under WP5, as illustrated in Figure 4.2
- **How the CE interlinks with the technologies in WP3**. CE will interlink with WP3 in two ways: a) by incorporating data created by WP3 modules in the CE and b) by potentially using APIs of WP3 modules.
- **How the CE will handle the crowd-contributions in WP4**: The CE will potentially interlink with WP4 by using APIs of WP4 modules, whenever an event is triggered
- **How the CE will integrate the other component of WP5**: Apart the development of the individual components of WP5, these should be integrated with the Data Infrastructure
- **How the components of WP5 will integrate with the use cases of WP6:** The components will have to be integrated with the use case clients.

An overview of the interdependencies across WPs and Tasks are presented in Figure 5.1.

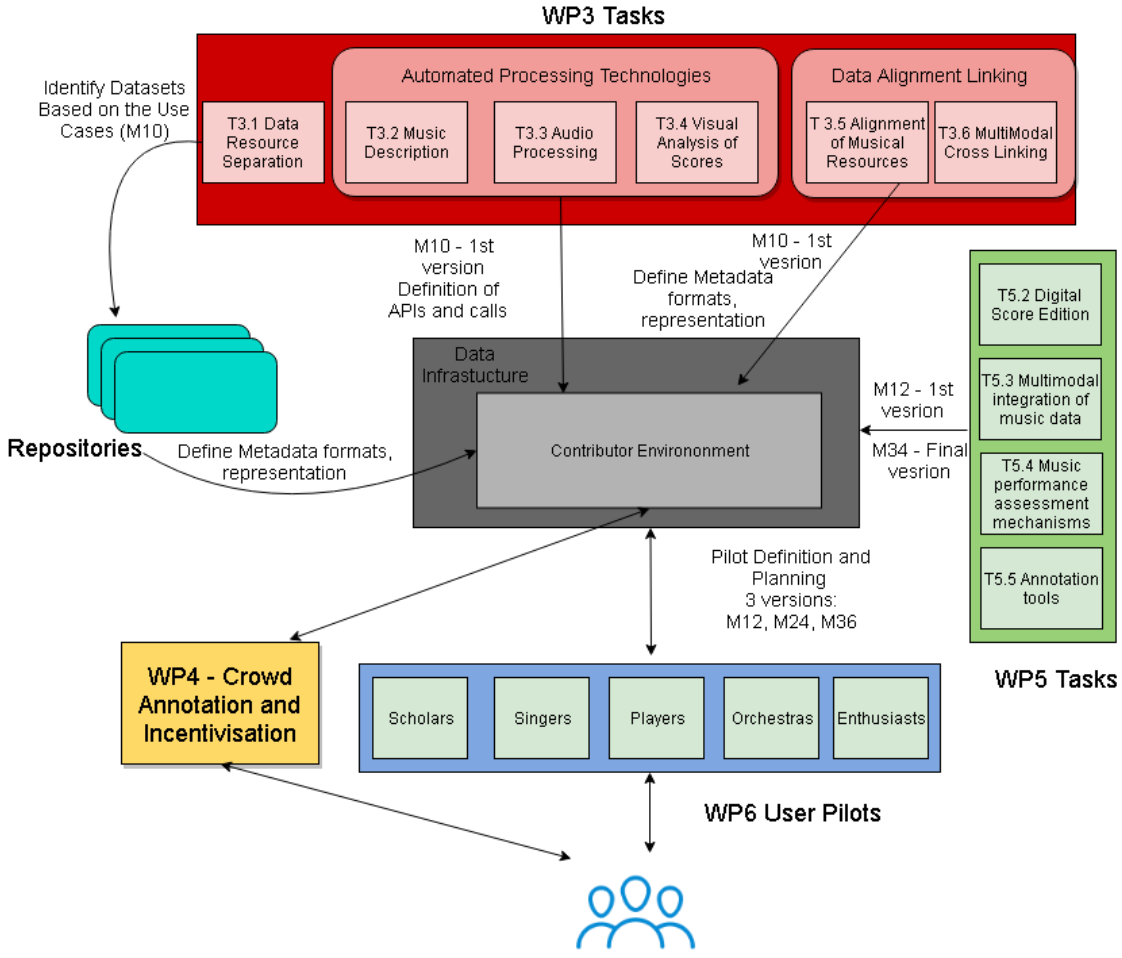

Figure 5.1. Data Infrastructure and interdependencies planning

From the Description of Action of TROMPA, we can derive that the development of the Data Infrastructure with respect to its progress, can be divided in three periods that are described below:

- **Period 1: M7-M12:** In this period we will have the following contributions from other WPs and Tasks, and especially from WP3. There are several components and deliverables delivered on M12, that have to be synchronized with the Data Infrastructure::
  - **WP3 - Automated Music Data Processing and Linking**
    - **Task 3.1 Data resource preparation, M10:** This is a very important task, since it will the define the music repertoire based on the use cases. The Data Infrastructure should be adapted to support communication with these repertoires.
    - **Task 3.2 Music Description, M10:** This is the 1st version of the automated music description. There should be defined how the Data Infrastructure will communicated with these technologies, which data needed will host etc.
    - **Task 3.3 Audio Processing, M12:** Similar to Task 3.2. It is related to singing voice analysis and synthesis.
    - **Task 3.5: Alignment of Musical Resources, M12.** Related to connect performances to scores.
  - **WP4 - Crowd Annotation and Incentivisation.**
    - Task 4.3, Crowd incentivisation techniques, M12: This is the 1st version of crowd incentivisation mechanisms and it will include draft user interfaces mock-ups.
    - Task 4.4, Hybrid annotation workflows, M12: This task will include the first hybrid music description workflow.
  - **WP5 - TROMPA Contributor Environment**
    - The Data Infrastructure will have to integrate with the Individual Components of the Contributor Environment (Tasks 5.2-5.5). The first versions of these components are to be delivered on M12.
  - **WP6 - End User Pilots**
    - The first version for all pilots will be on M12.
- **Period 2: M13-M24:** In this period we will have final versions for most of the technologies from WP3, and the 2nd version of the end user pilots. On M24, we will have a version of the Data Infrastructure that will integrate most of its functionalities, and will be close to its final form.
  - **WP3 - Automated Music Data Processing and Linking**
    - **Final version for Task 3.1.** All TROMPA resource data should be indexed in the CE.
    - **Final version for Tasks 3.2, 3.3 and 3.5.** The corresponding technologies should be fully integrated with the Data Infrastructure.
  - **WP4 - Crowd Annotation and Incentivisation.**

- **Final version of Task 4.1:** The technologies of this task will be integrated with WP3 technologies to provide evaluation methodologies and non-obvious music descriptors, relevant to different TROMPA-audiences.
- **Final version for Task 4.2, 4.3:** These will include the final implementations of user models and Incentivisation mechanisms.
- **Final version for Task 4.4:** This final form of hybrid annotation workflows, with a system which assigns crowdsourcing tasks to the suitable set of contributors.

➢ **WP5 - TROMPA Contributor Environment**

- Working version of the Contributor Environment, supporting WP3 tasks and including continuous integration with the components T5.2, T5.3, T5.4, T5.5 and WP4 tasks.

➢ **WP6 - End User Pilots: The 2nd version for all pilots will be on M24.**

❖ **Period 3: M25-M30:** It is the final version of the Data Infrastructure.

➢ **WP3 - Automated Music Data Processing and Linking:** The Tasks 3.4 - Visual Analysis of Scores and Task 3.6 - Multimodal cross-linking will finished one month prior to the Data Infrastructure (M29)

➢ **WP5 - TROMPA Contributor Environment**

- Final version of the Contributor Environment, with full support for WP3 tasks and fully integrated with the components T5.2, T5.3, T5.4, T5.5 and WP4 tasks.

➢ **WP6 - End User Pilots:** Data Infrastructure should be ready before the completion of the End User Pilots (M36) in order to fully support them.

6 Conclusion

This document is the outcome of a process that involved all TROMPA participants. In conjunction with D2.1 and D8.4 (the Early Requirements and Data Management Plan deliverables), each participant had to think through each of the TROMPA tasks they are involved in, and the corresponding Data Infrastructure requirements within the larger context of the TROMPA project.

The specification described in this document provides a comprehensive and detailed overview of how each of the parts will need to play their role and fit together as a whole. This is the basis on which a list of provisional requirements could be made, which enabled us to narrow down the specifications as presented in this document. By design, these specifications grant a generous amount of flexibility that allow the various interdependent parts to evolve during the project, and leave ample room for the Data Infrastructure itself to evolve and grow with them. The specified technologies to be used are mature and widely used, yet most are also relatively young and have momentum. This way, these specifications lean forward and all but guarantee that creating the Data Infrastructure will be an innovative experience for the development teams. Moreover, it ensures that the Data Infrastructure can have a life beyond the TROMPA project as a platform that can continue to play an innovative role for classical music communities.

Most importantly for the TROMPA project and its planning, though, is that the specifications provide sufficient detail to start the development of the Data Infrastructure - the Contributor Environment and the four components - right away.

7 References

7.1 List of abbreviations

Abbreviation	Description
AC	Access Control
API	Application Program Interface
CE	Contributor Environment
GDPR	EU General Data Protection Regulation
MEI	Music Encoding Initiative
MOOC	Massive Open Online Course
PII	Personally Identifiable Information
RDF	Resource Description Framework
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
Participants	
UPF	Universitat Pompeu Fabra
TUD	Technische Universiteit Delft
GOLD	Goldsmiths' College
MDW	Universität für Musik und darstellende Kunst Wien
VD	Videodock BV
PN	Peachnote GmbH
VL	Voctro Labs SL
RCO	Stichting Koninklijk Concertgebouworkest
CDR	Stichting Centrale Discotheek

Table 7.1. List of abbreviations

Appendix A

Internal data model classes

Internal model classes / types		
type	*origin*	*URI*
Base types		
Action	schema.org	https://schema.org/Action.rdf
CreativeWork	schema.org	https://schema.org/CreativeWork.rdf
Event	schema.org	https://schema.org/Event.rdf
Intangible	schema.org	https://schema.org/Intangible.rdf
Organization	schema.org	https://schema.org/Organization.rdf
Person	schema.org	https://schema.org/Person.rdf
Place	schema.org	https://schema.org/Place.rdf
Product	schema.org	https://schema.org/Product.rdf
Extra types		
Property	schema.org	https://schema.org/Property.rdf
PropertyValue	schema.org	https://schema.org/PropertyValue.rdf
Annotation	www.w3.org/ns/oa	http://www.w3.org/ns/oa#Annotation

Table A.1. Overview of base classes for the internal model

Appendix B

GraphQL example

```
query {
  creativeWork (id: "8540a212-38c3-4e8d-90b2-2cd3d6baceba", type:
"mo_composition") {
    additionalType
    identifier
    title
    author {
      identifier
      title
      additionalType
      familyName
      givenName
      birthDate
    }
    hasPart @propertyValueCondition (type: "mo_musicalWork") {
      hasPart @propertyValueCondition (type: "mo_arrangement") {
            hasPart @propertyValueCondition (type: "mo_score") {
        additionalProperty @propertyCondition (property:
"mo_producedScore") @propertyValueCondition (type: "mo_publishedScore")
{
          identifier
                type
            url
        }
        }
      }
    }
  }
}
results in:
{
  "data": {
    "creativeWork": {
      "type": "mo_composition",
      "additionalType": null,
      "identifier": "8540a212-38c3-4e8d-90b2-2cd3d6baceba",
      "title": "Piano Sonatas",
      "author": [
        {
```

```
          "identifier": null,
          "title": "Ludwig van Beethoven",
          "additionalType": null,
          "familyName": "van Beethoven",
          "givenName": "Ludwig",
          "birthDate": "1770-12-17"
        }
      ],
      "hasPart": [
        {
          "hasPart": [
            {
              "hasPart": [
                {
                  "mo_producedScore": [
                    {
                      "identifier":
"86a905f5-9255-4cf8-ae54-aaff77353a75",
                      "type": "mo_publishedScore",
                      "url": "... score location..."
                    },
                    {
                      "identifier":
"610ab5c2-28e9-4920-ab25-5118eea68aa1",
                      "type": "mo_publishedScore",
                      "url": "... score location..."
                    }
                  ]
                }
              ]
            }
          ]
        }
      ]
    }
  }
}
```


Figure B.1. Example GraphQL query and response