ECS7028P Report: Semantic Mapping of Early 2000s Movies and Soundtracks

Denise Azucena

May 12, 2025

1 Introduction and Background

Movies and music are deeply intertwined, raising questions such as: which films share the same composer, when were they released, and which genres recur across soundtracks? This report includes a semantic framework that unifies film and music metadata through a purpose-built ontology centered on five core concepts: Movie, Soundtrack, Track, Composer, and Genre, into a clear class hierarchy and define a rich property structure (for example, 'hasSoundtrack', 'isTrackOf', and 'performedBy') with precisely scoped domains, ranges, and characteristics such as functionality and transitivity. Each modeling decision—whether to declare a property functional, to introduce primitive versus defined classes, or to choose particular inverse relationships, that is justified in terms of its impact on reasoning, query expressivity, and overall performance.

2 Data Collection and Integration

For this project, I relied on two primary data sources: Wikidata and MusicBrainz. These data sources were used to build a comprehensive ontology of film soundtracks. First, I used the Wikidata Query Service to retrieve basic film metadata, constructing SPARQL queries that returned triples for each film's title, composer, release year, and genre. To focus the dataset on English-language films, I applied an additional filter requiring English labels and only obtain movies from the years 2000 to 2010. This produced the initial graph of film entities in the ontology.

Because Wikidata does not include detailed soundtrack information, such as track listings and performing artists, I turned to MusicBrainz for those data. Although MusicBrainz offers a SPARQL endpoint, I discovered it was nonfunctional for my needs, so I instead fetched the API's JSON release-group responses and converted them into RDF triples. In this way, each soundtrack release group became an RDF entity, from which I extracted track titles, music-genre classifications, and artist attributions, even when those artists differed from the composers. By merging individuals drawn from both Wikidata and MusicBrainz,

I populated the ontology with a set of film, soundtrack, track, composer, and artist instances. As a result there were 42 movies, 40 soundtracks, 4 composers and 743 tracks stored in this ontology. The movies "Frost/Nixon" and "How You Look to Me" unfortunately did not have a corresponding MusicBrainz information.

One notable challenge was rate limiting on the MusicBrainz API: requests beyond 300 per second are met with HTTP 503 errors[1]. To work around this, I implemented a simple back-off mechanism in my data-collection script, introducing a one-second sleep interval between requests to ensure reliable retrieval of all required metadata.

3 Ontology Design

The purpose of the design of this ontology is to retrieve and match movie titles with their respective film genre, soundtracks, tracks and music genre. This framework links composers and performing artist to their works, which enables intermediate semantic queries and integration of film-music metadata.

3.1 T-box

3.1.1 Classes

As seen on table 1, there are six main classes with "Genre" having 2 subclasses. These classes are considered primitive classes since they only include necessary conditions.

3.1.2 Object Property and Data Property Assertions

These show which properties are built on each other, their connections and specified constraints. These axioms also show how these properties relate to each other and how the system can infer their relations.

1. Sub-property and Inverse declarations

- $hasSoundtrackComposer \sqsubseteq composedFor \quad and \quad hasMovieComposer \sqsubseteq composedFor$
- $hasSoundtrackComposer \equiv \left(isComposerOfSoundtrack\right)^{-}$
- $hasMovieComposer \equiv \left(isComposerOfMovie\right)^{-}$

2. Domain and Range axioms

- Domain(hasGenreFilm) = Movie; Range(hasGenreFilm) = Genre_Music
- Domain(hasGenreMusic) = Soundtrack; Range(hasGenreMusic)= Genre_Film
- Domain(hasSimilarMusicGenre) = Soundtrack; Range(hasSimilarMusicGenre) = Genre_Music

Table 1: Class axioms: asserted vs. inferred

Class	**Asserted**	**Inferred**
Movie	Every movie has a soundtrack, at least one composer, and at least one film genre. (disjoint with Soundtrack)	Every soundtrack is the soundtrack of at least one Movie.
Composer	Each composer is linked to one or more Movie or soundtrack via a composer role. (disjoint with Artist)	A movie or soundtrack is composed by at least one composer. A composer can be a composer for more than one movie
Genre	Genre splits into two disjoint subclasses: Film Genre and Music Genre.	—
Soundtrack	Every soundtrack contains one or more tracks, has at least one composer, and has at least one music genre. (disjoint with movie)	Every soundtrack is linked (by the inverse of `hasSoundtrack`) to at least one movie.
Track	A or some tracks are part of at least one Soundtrack.	Every track is performed by at least one atist.
Artist	An artist performs one or more tracks (and is disjoint from composer).	–

- Domain(performedBy) = Track; Range(performedBy) = Artist

3. Property characteristics

- $hasSoundtrack \equiv (isSoundtrackOf)^{-}$; Domain($hasSoundtrack$) = $Movie$; Range($hasSoundtrack$) = $Soundtrack$; Functional; Irreflexive
- $isSoundtrackOf \equiv (hasSoundtrack)$; Domain($isSoundtrackOf$) = $Soundtrack$; Range$(isSoundtrackOf)^{-}$ = $Movie$; Inverse-Functional; Irreflexive
- $hasTrack \equiv (isTrackOf)$; Domain($hasTrack$) = $Soundtrack$; Range($hasTrack$) = $Track$; Transitive
- $isTrackOf \equiv (hasTrack)$; Domain($isTrackOf$) = $Track$; Range($isTrackOf$) = $Soundtrack$

4. Data property

- Domain($releaseYear$) $\equiv Movie$;
- Range($releaseYear$) = $xsd : integer$;

3.1.3 T-box level Inference of Axioms

- Any individual that hasGenreFilm must be a movie
  $\exists\, hasGenreFilm.\top \sqsubseteq Movie$
- Each Movie can have at most one hasSoundtrack connection.
  $\top \sqsubseteq \leq 1\, hasSoundtrack.Soundtrack$
- Each Soundtrack can be the soundtrack of at most one Movie.
  $\top \sqsubseteq \leq 1\, (isSoundtrackOf)^{-}.Movie$
- If a Soundtrack has a Track, and that Track has a sub-track, the Soundtrack also has that sub-track
  $hasTrack \circ hasTrack \sqsubseteq hasTrack$
- No Movie can be its own soundtrack (irreflexivity).
  $\top \sqsubseteq \neg\, \exists\, hasSoundtrack.Self$
- The inverse of hasSoundtrackComposer is a sub-property of the inverse of composedFor.
  $(hasSoundtrackComposer)^{-} \sqsubseteq (composedFor)^{-}$

3.2 A-box

The A-Box holds all of the concrete facts about individual movies, soundtracks, genres, and tracks. For instance, it records that the film "Alien vs. Predator" is classified as an Adventure movie and that its official soundtrack, scored by Harald Kloser, which waslabeled with the Classical and Modern Classical music genres. It also captures the soundtrack's track listing, including the piece entitled "Antarctica."

$Movie(\texttt{Alien\_vs\_Predator})$
$Soundtrack(\texttt{Alien\_vs\_Predator\_OST})$
$hasSoundtrack(\texttt{Alien\_vs\_Predator}, \texttt{Alien\_vs\_Predator\_OST})$
$Composer(\texttt{HaraldKloser})$
$hasSoundtrackComposer(\texttt{Alien\_vs\_Predator\_OST}, \texttt{HaraldKloser})$

4 Querying and Reasoning

The system is built around an RDF-based architecture where the python script loads the .owl ontology into the rdflib graph. Queries are executed with g.query() and returns python tuples that were post processed. Basic query includes finding the list of tracks from the movie soundtrack, music genres of a film, and film genre of a particular movie. Intermediate level querying includes finding similar music genres that matches with certain soundtracks, which shows that films of a specific genre have used particular music scores (eg. classical music used in drama and action).

One SWRL rule was included to infer if a soundtrack has a certain music genre, it's inferred that the movie has a certain genre:

$$Soundtrack(?s) \wedge hasSoundtrackComposer(?s, ?c) \wedge isSoundtrackOf(?s, ?m) -> hasMovieComposer(?m, ?c)$$

Pellet was used for reasoning, which allowed inference of subclass relationships, which in turn enforces domain and range constraints, detect inconsistencies (such as unsatisfiable class combinations), and propagate custom inferences, such as assigning a movie the same composer as its soundtrack via SWRL rule.

5 Discussion and Conclusion

During development, a few key challenges were encountered that required targeted solutions. First, by using a global domain and range on the 'performedBy' property, the reasoner began inferring that every class,including 'Movie,was a subclass of the anonymous restriction "performedBy only Artist," effectively treating movies as tracks. This was resolved by moving the universal restriction into a class-level axiom , so only tracks carry that constraint. Second, MusicBrainz often lists multiple "release groups" for the same soundtrack (regional editions, deluxe versions, etc.), leading to duplicate 'Soundtrack' individuals with near-identical labels. This was mitigated by filtering only the earliest "official" release, which cut duplicates by over 75% in the test set. Finally, core concepts were kept ('Movie', 'Soundtrack', 'Track', 'Composer', 'Genre') as primitive classes to preserve flexibility for future extensions, at the cost of requiring explicit ABox typing for full classification.

Further work includes enriching ontology with detailed song-level metadata (durations, instruments, producer credits), composer biographies, temporal properties for recording and composition dates, and deeper genre hierarchies, as well as automate periodic data updates. These enhancements will support more sophisticated queries—such as finding films with violin solos, composers who span multiple genres, or soundtracks recorded before a film's release, and will strengthen the ontology's value as a comprehensive knowledge graph of film music.

6 References

[1] MusicBrainz, "MusicBrainz API / Rate Limiting," MusicBrainz Wiki, Jan. 8, 2012. [Online]. Available: https://musicbrainz.org/doc/MusicBrainz_API/Rate_Limiting. Accessed: May 11, 2025.