A scalable framework for multimedia knowledge management

Yves Raimond, Samer A. Abdallah, Mark Sandler, Mounia Lalmas

Centre for Digital Music, Queen Mary, University of London
{yves.raimond,samer.abdallah,mark.sandler}@elec.qmul.ac.uk
Department of Computer Science, Queen Mary, University of London
mounia@dcs.qmul.ac.uk

Abstract. In this paper, we describe a knowledge management framework that addresses the needs of multimedia analysis projects and provides a basis for information retrieval systems. The framework uses *Semantic Web* technologies to provide a shared knowledge environment, and active *Knowledge Machines*, wrapping multimedia processing tools, to exploit and/or export knowledge to this environment. This framework is able to handle a wide range of use cases, from an *enhanced workspace* for researchers to end-user *information access*. As an illustration of how the proposed framework can be used, we present a case study of music analysis.

1 Introduction

Information management is becoming an increasingly important part of multimedia related technologies, ranging from the management of personal collections through to the construction of large 'semantic' databases intended to support complex queries. One of the key problems is the current gap between the development of stand-alone multimedia processing algorithms (such as feature extraction, or compression) and knowledge management technologies. The aim of our work is to provide a framework that is able to bridge this gap, by integrating the multimedia processing algorithms in an information management system, which can then be usable by different entities in different places. We also want to provide a way to semantically describe these algorithms in order to *automatically* use them. For example, we might want to dynamically compute the segmentation of a sequence of a football match in order to answer a query like 'give me all the sequences corresponding to a corner'.

In order to achieve this goal, we introduce several concepts. A *Knowledge Machine* aims to help people developing, encapsulating or testing multimedia processing algorithms, by designing a *semantic workspace*. Instances of such machines interact with a set of *end-points*, which are entry points to a shared knowledge environment, which is itself based upon *Semantic Web* technologies. This interaction can be either to request knowledge from this environment (in order to test algorithms, or to use external information *in* the algorithm) or to export new knowledge onto it (to make new results publicly available). An

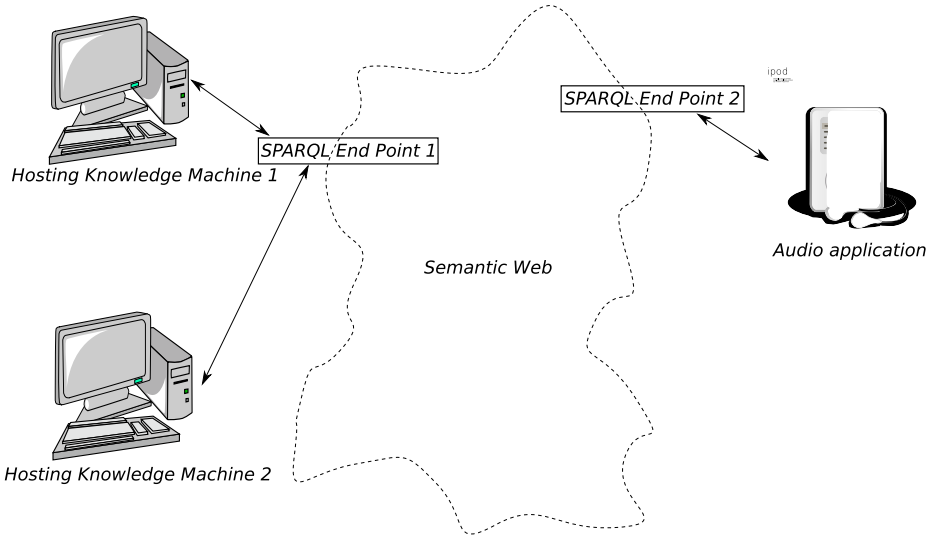

Fig. 1. An overview of the framework

end-point can also have a *planning* [1] role, as part of the knowledge environment describes these *Knowledge Machines*, and the effect of using some of their algorithms. Moreover, these *end-points* can be used by other entities (such as a personal audio player), which may benefit from some information potentially held by this knowledge environment. A simplified overview of the system is given in fig. 1.

The proposed framework is the first step to bridge the above mentioned gap. Indeed, the algorithm is entirely handled from its implementation process to its impact on a shared knowledge environment, which can either be used by other researchers or by information access systems. Moreover, new knowledge can be dynamically added (either from 'ground truth' sources or from Knowledge Machines plugged onto the knowledge environment), as well as new Knowledge Machines. This *web* approach allows the creation of a scalable knowledge management system.

In § 2 we will describe the structure behind the Knowledge Machines. We will then focus on the shared knowledge environment in § 3, and how it deals with instances of Knowledge Machines. Next, we will focus on two completely different use cases of the system in § 4, in order to give an idea of the wide range of possibilities this framework brings. Finally, § 5 will present a case study on knowledge management for music analysis.

2 Knowledge Machines

In this section, we describe the Knowledge Machine architecture (described in greater detail but in a more specific context in [2]), which is able to design

a workspace for handling and/or developing multimedia processing algorithms. With complex algorithms, there are often many shared steps of computation, multiple computation strategies, and many free parameters that can be varied to tune performance. This can result in a very large amount of final and intermediate results, which needs to be managed in order to be used effectively. fig. 4 gives a global view of the Knowledge Machines architecture, built around a unique knowledge representation framework (see § 2.1), a computational engine (see § 2.2) and *tabling* of results (see § 2.3).

2.1 Knowledge representation for data analysis

In this section, we will describe the main approach for storing the results of the different computations, which can occur while working on a set of multimedia processing algorithms. Its limitation is what led us to using another approach, which will also be described in this section, using *predicate calculus* for knowledge representation.

The dictionary approach The resulting data is often managed as a dictionary of key-value pairs—this may take the form of named variables in a Matlab workspace, files in a directory, or files in a directory tree (in which case the keys would have an hierarchical structure). This can lead to a situation in which, after a Matlab session for example, one is left with a workspace full of objects but no idea how each one was computed, other than, perhaps, cryptic clues in the form of the variable names one has chosen.

The semantic content of these data is intimately tied to knowledge about which function computed which result using what parameters, and so one might attempt to remedy the problem by using increasingly elaborate naming schemes, encoding information about the functions and parameters into the keys. This is a step toward a relational structure where such information can be represented explicitly and in a consistent way.

Relational and logical data models We now focus on a relational data model [3], where different relations are used to model the connections between parameters, source data, intermediate data and results. Each tuple in these relations represents a proposition, such as '*this* spectrogram was computed from *this* signal using *these* parameters' (see fig. 2). From here, it is a small step to go beyond a relational model to a deductive model, where logical predicates constitute the basic representational tool, and information can be represented either as facts or as composite formulæ involving the logical connectives *if*, $\exists$ (*exists*), $\forall$ (*for all*), $\vee$ (*or*), $\wedge$ (*and*), $\neg$ (*not*) and $\equiv$ (*equivalent to*) (see [2] for a short review of predicate calculus for knowledge representation).

For example, in this model, the previous proposition could be expressed using this predicate:

$$\mathsf{spectogram}(DigitalSignal, FrameSize, HopSize, Spectrogram)$$

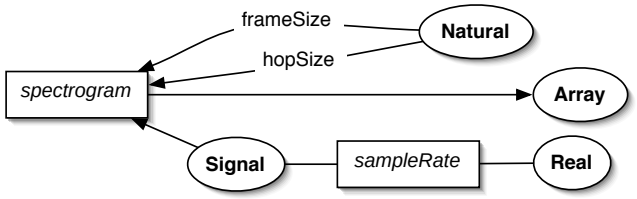

Fig. 2. The relations involved in defining a spectrogram

In addition, we could imagine the following for the digital representation of a signal:

$$\mathit{digitalsignal}(DigitalSignal, SampleRate) \ \mathit{if}$$
$$\exists ContinuousSignal.\ \mathit{sampling}(ContinuousSignal, DigitalSignal, SampleRate)$$

2.2 Evaluation Engine

The computation-management facet of the Knowledge Machines is handled through calls to an external *evaluation engine*. The latter is used to reduce a given expression to some canonical form. For example, a real-valued expression involving mathematical functions and arithmetic operators would be reduced to the floating-point representation of the result. Standard Prolog itself provides such a facility through the `is` operator. By using an interface to an interpreted language processor, such as Matlab , a much richer class of expressions can be evaluated, involving complex numbers, arrays, structures, and the entire library of Matlab functions available in the system.

For example, if we define the operator `===` as evaluating terms representing Matlab expressions, we can define (in terms of predicate calculus) a matrix multiplication like this:

$$\mathit{mtimes}(A, B, C) \ \mathit{if}\ C\texttt{===}A * B$$

We can now build composite formulæ involving the predicate `mtimes`.

Interpreters for different expression languages could be added, provided that a Prolog representation of the target language can be designed.

2.3 Function tabling

To keep track of computed data, we consider tabling of such logical predicates. Some predicates, when used in a certain way (in a particular mode), can be considered as 'functional'—one possible resulting tuple given a set of inputs (such as `mtimes`, when used with the first and the second argument bound, and the third one unbound). If we store the tuples generated by the functional predicates, then we can save ourselves some evaluations, because of this functional

mode. Moreover, the function tabling mechanism allows us to access functional predicates in *all* modes. Given the produced data, we can obtain back to the inputs and parameters that were used to create them.

For example, if we declare the predicate `mtimes` (declared as in § 2.2) to be tabled, and we have two matrix `a` and b, the first time `mtimes`(`a`,`b`,C) will be queried the Matlab engine will be called. Once the computation done, and the queried predicate has successfully been unified with `mtimes`(`a`,`b`,`c`), where `c` is actually a term representing the product of `a` and `b`, the corresponding tuple will be stored. When querying again `mtimes`(`a`,`b`,C), the computation will not be done, but the stored result will be returned instead. It also means that, given `c`, we can get back to `a` and `b`, using the query `mtimes`(A,B,c).

2.4 Implementation

Knowledge Machines are built on top of SWI Prolog[1] (which is one of the most *user-friendly* Prolog) and PostgreSQL[2]. The *workspace* they implement is accessible through a Prolog command-line, and new facts, composite formulæ or new evaluation engines can be directly asserted through it, or through external source files.

Each Knowledge Machine also wraps a component able to make it usable remotely. This can be seen as a simple Servlet, able to handle remote queries to local predicates, through simple HTTP GET requests. This will be useful when other components of the framework (such as the planner described in § 3.4) have a global view of the system and need to dynamically organise a set of Knowledge Machines.

3 A Semantic Web Knowledge Environment

In this section, we describe how we provide a shared and distributed knowledge environment, using Semantic Web technologies (see § 3.1) and a set of domain ontologies (see § 3.3). We will also explain how Knowledge Machines can interact with this environment in § 3.2, using entry doors generated using the tool described in § 3.4.

We will refer to several technologies, all part of the Semantic Web effort, which we will briefly overview here. RDF (Resource Description Framework[3]) defines how to describe resources (located by an Universal Resource Identifier [4]), and how to link them, using *triples* (sets of subject/predicate/object). For example, using RDF, I can express that the resource representing a given artist has produced several albums. An OWL (Ontology Web Language[5]) ontology is able to express knowledge about one particular domain by identifying its

[1] see `http://www.swi-prolog.org/`
[2] see `http://www.postgresql.org/`
[3] see `http://www.w3.org/RDF/`
[4] see `http://www.gbiv.com/protocols/uri/rfc/rfc3986.html`
[5] see `http://www.w3.org/2004/OWL/`

important concepts and relations, in RDF. SPARQL (Simple Protocol And RDF query language[6]) defines a way to query RDF data. Finally, a SPARQL *end-point* can be seen as a public entry door to a set of RDF statements.

3.1 Why use Semantic Web technologies?

The 'metadata' mistake In this shared knowledge environment, we may want to state circumstances surrounding the creation of a particular raw multimedia data. One option is to 'tag' each piece of primary data with further data, commonly termed 'metadata', pertaining to its creation. For example, CDDB[7] associates textual data with a CD, while ID3[8] tags allow information to be attached to an MP3 file. The difficulty with this approach is the implicit hierarchy of data and metadata. The problem becomes acute if the metadata (eg the artist) has its own 'meta-metadata' (such as a date of birth); if two songs are by the same artist, a purely hierarchical data structure cannot ensure that the 'meta-metadata' for each instance of an artist agree. The obvious solution is to keep a separate list of artists and their details, to which the song metadata now *refers*. The further we go in this direction, i.e. creating new first-class entities for people, songs, albums, record labels etc., the more we approach a fully *relational* data structure.

Towards a scalable solution We also want this data structure to be *distributed*. Any entities contributing to the knowledge environment may want to publish new assertions, eventually concerning objects defined in an other place. This is why we are using RDF. We also want to be able to specify what types of objects are going to be in the domain of discourse and what predicates are going to be relevant. Designing an *ontology* [4] of a domain involves identifying the important concepts and relations, and as such can help to bring some order to the potentially chaotic collection of predicates that could be defined. We may also want to dynamically introduce new domains in the knowledge environment. This is why we are using OWL.

3.2 Integrating Knowledge Machines and the Knowledge Environment

Querying the Semantic Web within Knowledge Machines Another characteristic of the Knowledge Machines is the ability to query the Semantic Web (through a set of *end-points*, as we will see in § 3.4) using SPARQL. Someone working on a Knowledge Machine is able to create an *interpretation* of the *theory* (OWL ontologies) in the form of a collection of predicates. It is then possible to use these predicates when building composite formulæ in the language of predicate calculus.

[6] see `http://www.w3.org/TR/rdf-sparql-query/`
[7] see `http://www.gracenote.com/`
[8] see `http://www.id3.org/`

For example, we can imagine the following SPARQL query that associates an audio file and the sampling rate of the corresponding digital signal:

```
PREFIX mu: <http://purl.org/NET/c4dm/music.owl#>
SELECT ?a ?r WHERE {
?a rdf:type mu:AudioFile. ?a mu:encodes ?dts.
?dts rdf:type mu:DigitalSignal.
?dts mu:samplingRate ?r }
```

We can associate to this query the following predicate, whose first argument will be bound to the audio file, and whose second argument will be bound to the corresponding sampling rate:

$$audiofile\_samplingrate(AudioFile, SamplingRate)$$

Now we can use this predicate in composite formulæ, perhaps to use this sampling rate information as one of the inputs of an algorithm wrapped in another predicate.

Exporting knowledge to the Semantic Web Someone working on a particular Knowledge Machine may want, at some point, to state that a particular predicate is relevant, according to the domain ontologies hold by the knowledge environment. This is equivalent to stating that this predicate has a particular *meaning* which can be expressed using one of the *vocabularies* we have access to. Thus, we want to be able to state a match between a predicate and a set of RDF triples. Moreover, we want to express this match either in the language of predicate calculus (in order to export new information when this predicate holds new knowledge) and in terms of OWL/RDF to allow automatic reasoning (as described in § 3.4) in the Semantic Web layer. We developed a simple ontology of *semantic matching* between a particular predicate and a conceptual graph. This ontology uses the RDF *reification*[9] mechanism in order to express things like '*this* predicate in *this* Knowledge Machine is able to create *these* RDF triples'. This can be seen as a limited subset of OWL-S[10], where the *effects* of a particular *process* can only consist in creating new RDF triples. For example, the predicate `soxsr`, able to change the sample rate of an audio file, can create some RDF triples, as represented in fig. 3. Thus, a Knowledge Machine can be represented as in fig. 4.

3.3 Domain specific ontologies

In order to make this knowledge environment understandable by all its actors (Knowledge Machines or any entities querying this environment), it needs to be designed according to a shared understanding of the specific domains we want to work on. An ontology can provide this common way of expressing statements

[9] see `http://www.w3.org/TR/rdf-mt/#Reif`
[10] `http://www.daml.org/services/owl-s/`

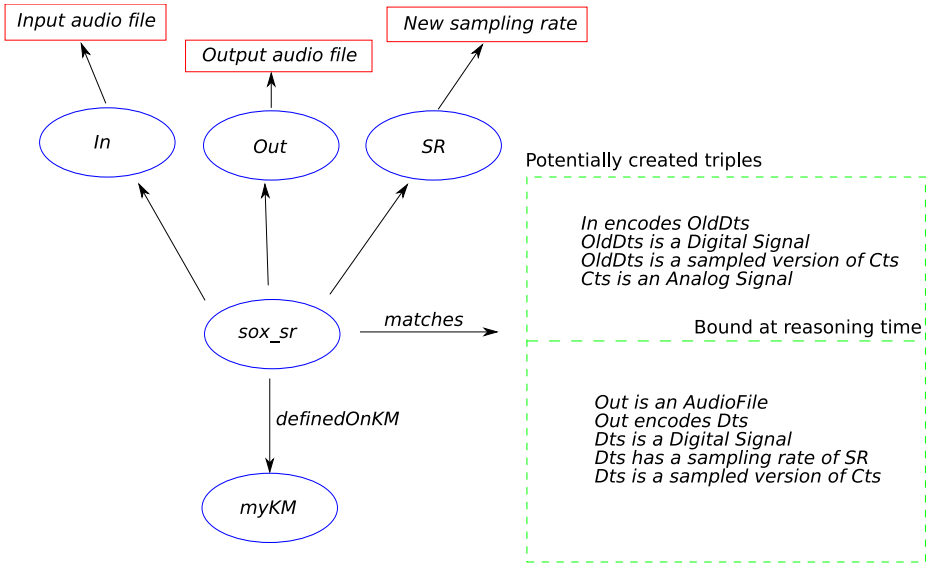

Fig. 3. Expressing a match between the predicate `soxsr` and the fact that it is able to change the sample rate of an audio file

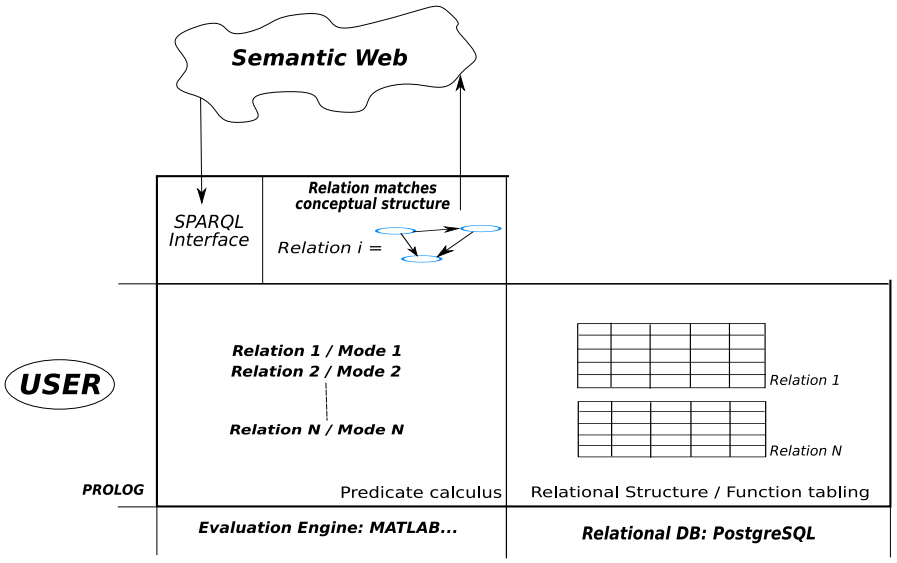

Fig. 4. Overall architecture of a Knowledge Machine

in a particular domain. Such ontologies will be developed in the context of music in § 5.1. Moreover, the expressiveness of the different ontologies specifying this environment will implicitly state how dynamic the overall framework can be. Indeed, the *semantic matching* ontology defined in the previous section has an expressiveness that directly depends on the different domain ontologies that are known.

For example, if we write an ontology that is expressive enough in the domain of football games and time sequences, and we have an algorithm which is able to segment a football match video (corner, penalty, ...), we will be able to express a conceptual match between *what* is done by the algorithm and a set of RDF statements conforming to this domain ontology.

However, in order to keep the overall framework in a consistent state, designing a new ontology must be done considering some points. These include modularity [5] and ontological 'hygiene' as addressed by the OntoClean methodology [6].

3.4 Handling Semantic Web knowledge

At this point, we still need to make Semantic Web data available to both Knowledge Machines and other entities wanting to make queries.

XsbOWL: creating SPARQL end-points In order to achieve this goal, we designed a program able to create SPARQL *end-points*: XsbOWL (see fig. 5). It allows SPARQL queries to be done through a simple HTTP GET request, on a set of RDF data. Moreover, new data can be added dynamically, using an other HTTP GET request.

Reasoning on Semantic Web data To handle reasoning on the underlying Semantic Web data, we bound XsbOWL to an XSB Prolog engine (which is more adapted to a large deductive database use case [7]). The latter, thanks to the inner XSB tabling mechanism, is able to provide reasoning on the positive entailment[11] subset of OWL Full [12]. XsbOWL is able to deal simultaneously with around 100000 RDF statements and still provides a really fast reasoning (less than 0.2 seconds per query). More scalability testing still has to be done.

Dynamically exporting knowledge to the Semantic Web We also integrated a planner in this XSB engine, in order to fully use the information held by the *semantic matching* ontology. This one is planning which predicate it needs to call in which Knowledge Machine (using the remote calling mechanism described in § 2.4) in order to reach a *state of the world* (the set of all RDF statements known by the *end-point*) which will at least give one answer to the query (see fig. 6). For example, if there is a Knowledge Machine somewhere that defines a

[11] see `http://www.w3.org/TR/owl-test/`
[12] see `http://www.w3.org/TR/owl-ref/`

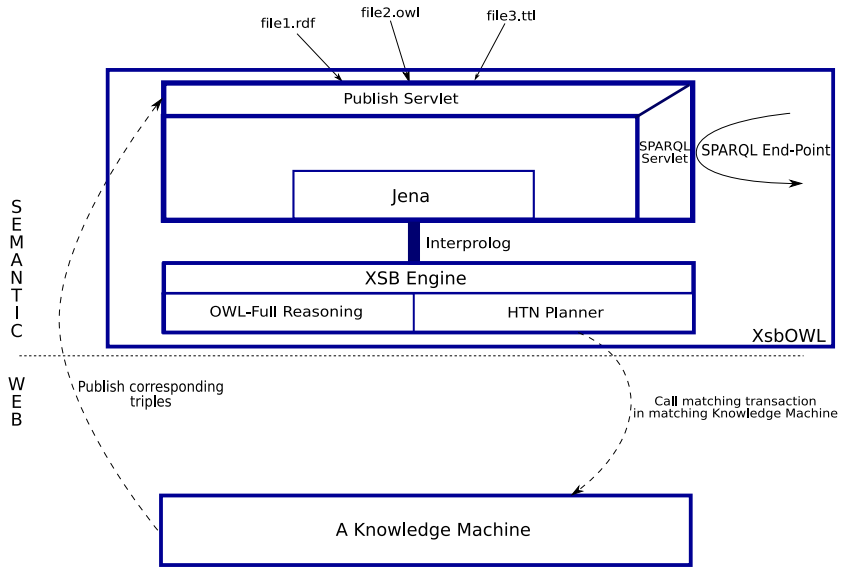

Fig. 5. XsbOWL: Able to create a SPARQL end-point for multimedia applications

predicate able to locate all the segments corresponding to a penalty in a football match, querying the *end-point* for a sequence showing a penalty during a particular match should automatically use this predicate.

4 Use cases

In this section, we describe two different use cases, to give an insight of the wide range of possibilities the proposed knowledge management framework brings.

4.1 Enhanced workspace for multimedia processing researchers

Working in a Knowledge Machine environment to develop multimedia processing algorithm helps to create what we could call a *semantic workspace*. Every object is part of the same logical structure, based on predicate calculus. Moreover, the Knowledge Machine framework provides a brand new programming environment, aware of an *open context*. Therefore, while developing a new predicate, we may access knowledge perhaps already available or newly created by an other Knowledge Machine, and this in a completely transparent way.

While working on a multimedia feature extraction predicate, it is possible to access the knowledge environment *inside* the predicate. For example, while working on a melody extraction algorithm, we are able to state that a particular sub-algorithm is to be used if an audio signal was created by a particular instrument. This could lead to the transparent use of an instrument classification predicate exported by an other Knowledge Machine.

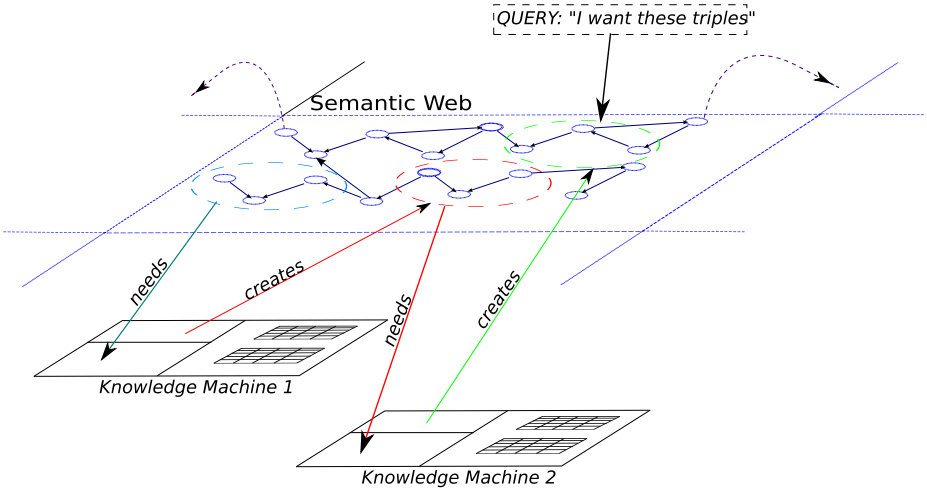

Fig. 6. Planning using the semantic matching ontology

4.2 End-user Information Access

Once the shared information layer holds a substantial amount of knowledge, it can be useful for other entities (not part of the Knowledge Machines framework) to use a SPARQL end-point. For example, an interactive graphical viewer application (such as Sonic Visualiser[13]) should be able to submit simple queries to compute some features of interests (or to retrieve previously computed ones) .

Moreover, as expressed in fig. 7, multimedia information retrieval applications can be built *on top* of this shared environment, through a layer interpreting the available knowledge. For example, if a Knowledge Machine is able to model the *textural* information of a musical audio file, and if there is an interpretation layer that is only able to compute an appropriate distance between two of these models, an application of similarity search can easily be built on top of all of this. We can also imagine more complex information access systems, where a large number of features computed by different Knowledge Machines can be combined with social networking data, all part of the shared information layer too.

5 Knowledge Management for Music Analysis

In this section, we will describe how this framework has been used for a music information management (this is explained in greater details in [2]). We will detail two Knowledge Machines, respectively dealing with format conversion and segmentation.

[13] see http://www.sonicvisualiser.org

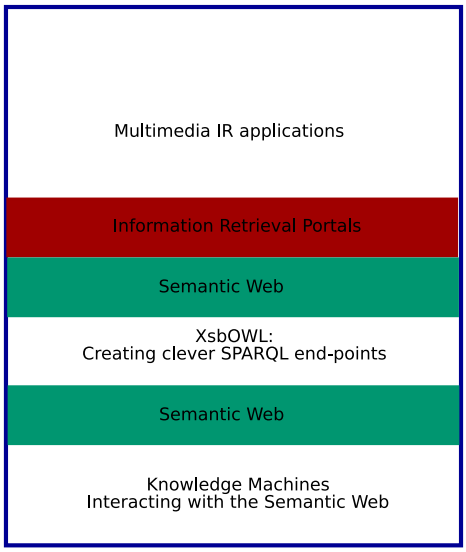

Fig. 7. The Multimedia Knowledge Management and Access Stack

5.1 An ontology of music

Our ontology must cover a wide range of concepts, including non-physical entities such as a musical *opus*, human agents like composers and performers, physical events such as particular performances, informational objects like digital signals, and time. We will focus on the two main aspects of our ontology: physical events and time representation.

An ontology of events Music production usually involves physical events that occur at a certain place and time and that can involve the participation of a number of physical objects both animate and inanimate.

The event representation we have adopted is based on the *token-reification* [8] approach. We consider an event occurence as a first class object or 'token', acting like a hook for additional information pertaining to the event. Regarding the ontological status of event tokens, we consider them as being the way by which cognitive agents classify arbitrary regions of space-time. Our definition of an event is broad enough to include sounds (an acoustic field defined over some space-time region), performances, compositions, and even transduction and recording to produce a digital signal. We also consider the existence of *sub-events* to represent information about complex events in a structured and non-ambiguous way. A complex event, perhaps involving many agents and instruments, can be broken into simpler sub-events, each of which can carry part of the information pertaining to the complex whole. For example, a group performance can be described in more detail by considering a number of parallel

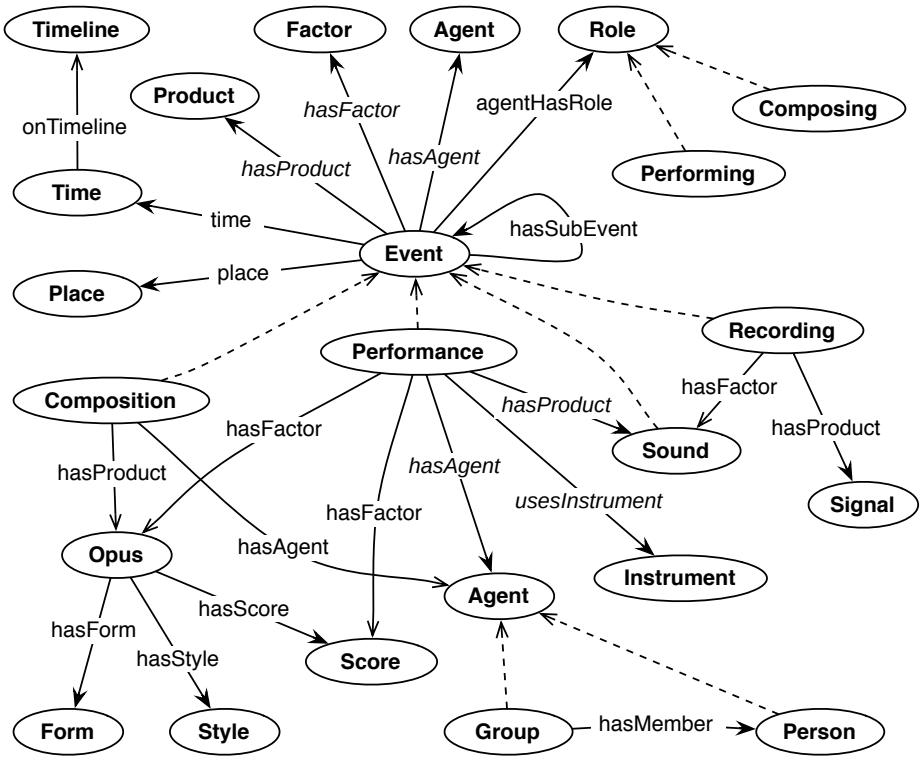

Fig. 8. Some of the top level classes in the music ontology

sub-events, each of which representing the participation of one performer using one musical instrument (see fig. 8 for some of the relevant classes and properties).

An ontology of time Each event can be associated with a time-point or a time interval, which can either be given explicitly, e.g. 'the year 1963', or by specifying its temporal relationship with other intervals, e.g. 'during 1963'. Both must be related to a *timeline*, continuous or discrete, representing linear pieces of time which may be concrete—such as the one underlying a signal or an event, or more abstract—such as the one underlying a score. Two *timelines* may be related, using *timeline maps*. For example, an instance of this concept may represent the link between the continuous physical time of an audio signal and the discrete time of its digital representation.

5.2 Examples of Knowledge Machines

So far, two main Knowledge Machines are *exporting* knowledge to the shared knowledge environment.

A format conversion knowledge machine The simplest one is about converting the format of raw audio data. Several predicates are exported, dealing with sample rate or bit rate conversion, and encoding. This is particularly useful, as it might be used to create, during the development of another Knowledge Machine, test sets in one particular format, or even to test the robustness of a particular algorithm to information loss.

A segmentation knowledge machine This Knowledge Machine is able to deal with segmentation from audio, as described in greater details in [9]. It exports just one predicate, able to split the time interval corresponding to a particular raw signal in several ones, corresponding to a machine-generated segmentation. This Knowledge Machine was used to keep track of hundreds of segmentations, enabling a thorough exploration of the parameter space, and resulting in a database of over 30,000 tabled function evaluations. This Knowledge Machine provides a segmentation ability on raw audio files. Along with the format conversion Knowledge Machine, it brings the ability to segment *all* available audio files by using our *planning* component.

6 Conclusions and further work

In this paper we described a framework able to deal with information management for multimedia analysis systems. This is built around two main components: *Knowledge Machines*, enabling to wrap and work on analysis algorithms using *predicate calculus* and *function tabling*, and a shared *Semantic Web* knowledge environment. Knowledge Machines can interact in two different ways with this environment. They can build an *interpretation* of the theory that it is holding at a given time as a set of logical predicates, thus to use them when building compound predicates. They can also specify a *match* between a set of logical predicates and a set of RDF triples, to export results to the Semantic Web layer, or to dynamically compute new ones in order to satisfy a query.

We can think of the Knowledge Machine framework as an artificial way of accessing *concepts*, which are defined by the domain ontologies held by the shared knowledge environment. Thus, a network of Knowledge Machines can bring an artificial and approximate *cognition* for multimedia related materials, against a *culture* which is defined by the different ontologies. It leads to a distributed intelligence, as mentioned in [10], and thus is similar in some ways to agent technologies. However, it brings an artificial *cognition* of available multimedia materials instead of *services*—getting things done.

There are several possible extensions to this framework, such as handling *trust* in the Semantic Web environment. For example, we may want to express that a computer-generated segmentation of an audio file is less accurate than a human-generated one, and we may also want to *quantify* this accuracy. We could also do a statistical analysis to judge whether or not a particular algorithm has successfully captured a given concept, and if so, to declare a match between a wrapping predicate and this concept so that the algorithm gains a semantic

value; subsequent queries involving this concept would then be able to invoke that algorithm (using the planner component) even if no key annotations are present in the shared environment. This would be an example of 'closing the semantic gap'.

7 Acknowledgments

The authors acknowledge the support of both the Centre For Digital Music and the Department of Computer Science at Queen Mary University of London for the studentship for Yves Raimond.

References

1. D. S. Weld, "Recent advances in ai planning," *AI Magazine*, 1999.
2. S. Abdallah, Y. Raimond, and M. Sandler, "An ontology-based approach to information management for music analysis systems," in *Proceedings of 120th AES convention*, 2006.
3. E. F. Codd, "A relational model of data for large shared data banks," *Communications of the ACM*, vol. 13, no. 6, pp. 377–387, 1970.
4. F. Baader, I. Horrocks, and U. Sattler, "Description logics as ontology languages for the semantic web," in *Essays in Honor of Jörg Siekmann*, ser. Lecture Notes in Artificial Intelligence, D. Hutter and W. Stephan, Eds. Springer, 2003.
5. A. L. Rector, "Modularisation of domain ontologies implemented in description logics and related formalisms including owl," in *Proceedings of the international conference on Knowledge capture*. ACM Press, 2003, pp. 121–128.
6. N. Guarino and C. Welty, "Evaluating ontological decisions with ONTOCLEAN," *Communications of the ACM*, vol. 45, no. 2, pp. 61–65, 2002.
7. K. Sagonas, T. Swift, and D. S. Warren, "XSB as an efficient deductive database engine," in *SIGMOD '94: Proceedings of the 1994 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM Press, 1994, pp. 442–453.
8. M. P. Shanahan, "The event calculus explained," in *Artificial Intelligence Today, Lecture Notes in AI no. 1600*, M. J. Woolridge and M. Veloso, Eds. Springer, 1999, pp. 409–430.
9. S. Abdallah, K. Noland, M. Sandler, M. Casey, and C. Rhodes, "Theory and evaluation of a bayesian music structure extractor," in *Proceedings of the Sixth International Conference on Music Information Retrieval*, J. D. Reiss and G. A. Wiggins, Eds., 2005, pp. 420–425.
10. J. Bryson, D. Martin, S. McIlraith, and L. Stein, "Toward behavioral intelligence in the semantic web," *IEEE Computer, Special Issue on Web Intelligence*, vol. 35, no. 11, pp. 48–55, November 2002.