Audio Engineering Society

Convention Paper

Presented at the 120th Convention
2006 May 20–23 Paris, France

An ontology-based approach to information management for music analysis systems

Samer A. Abdallah [1] and Yves Raimond [1] and Mark Sandler [1]

[1] *Centre for Digital Music, Queen Mary, University of London*

Correspondence should be addressed to Yves Raimond (yves.raimond@elec.qmul.ac.uk)

ABSTRACT

We describe an information management system which addresses the needs of music analysis projects, providing a logic-based knowledge representation scheme for the many types of object in the domains of music and signal processing, including musical works and scores, performance events, human agents, signals, analysis functions, and analysis results. The system is implemented using logic-programming and semantic web technologies, and provides a shareable resource for use in a laboratory environment. The whole is driven from a Prolog command line, where the use of Matlab as a computational engine enables experiments to be designed and run with the results being automatically stored and indexed into the information structure. We present as a case-study an experiment in automatic music segmentation.

1. INTRODUCTION

Information management and retrieval systems are becoming an increasingly important part of many music related technologies, ranging from the management of personal music collections (e.g. with ID3 tags or in an iTunes database), through to the construction of large 'semantic' databases intended to support complex queries, involving concepts like mood and genre as well as lower-level or textual attributes like tempo and composer. In parallel with this, the development of systems to automate the analysis of music—perhaps in order to populate a semantic database—depends on the availability of annotated music databases, both as a ready source of test data in the initial development phase, and to support systematic evaluations of the analysis algorithms. The resulting computational systems will often produce a large amount of intermediate data; in any case, the combined multiplicities of source signals, alternate computational strategies, and free parameters will very quickly generate a large result-set with its own information management problems.

In this paper, we describe a system which represents, manages and retrieves all of these different types of information in a unified structure, using the language of first-order predicate calculus, in terms

of which we define a collection of predicates designed according to a formalised ontology covering both music production and computational analysis. By integrating these different facets within the same logical framework, we facilitate the design and execution of experiments, such as exploration of function parameter spaces, the forming of connections between given 'semantic' annotations and computed data, and even mundane tasks like producing annotated figures presenting the results of an experiment.

The paper is organised as follows: the motivation behind using logic for knowledge representation is discussed in section 2; the terminology of predicate calculus is reviewed in section 3; the components of the ontology and some implementation details are described in sections 4 and 5. In section 6, we present as a case-study an experiment that was managed using the system, and conclude in section 7.

2. KNOWLEDGE REPRESENTATION FOR SIGNAL ANALYSIS

Consider the following scenario: we have a collection of raw data in the form of recorded signals, e.g., audio or video data. We also have information about the physical circumstances surrounding the recording of each signal, such the time and place, the equipment used, the people involved, descriptions of the events depicted in the signals, and so on. Our first task is to represent this 'circumstantial' information in a flexible and general way.

2.1. The 'metadata' approach

One option is to 'tag' each piece of primary data with further data, commonly termed 'metadata', pertaining to its creation. For example, CDDB associates textual data with a CD, while ID3 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, creating new first-class entities for people, songs, albums, record labels etc., the more we approach a fully *relational* data structure.

2.2. The dictionary approach

Now consider a scenario where, as well as collection of signals, we also have a number algorithms we can apply to the signals in order to compute features of interest. The algorithms may be modular and share intermediate steps, such as the computation of a spectrogram or the fitting of a hidden Markov model, and they may also have a number of free parameters.

The data resulting from these computations is often managed as a dictionary of key-value pairs—this may take form of named variables in a Matlab workspace, files in a directory, or files in a directory tree. 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, clues in the form of the variable names one has chosen. The semantic content of these data, such as it is, is intimately tied to knowledge about which function computed which result using what parameters, and so one might attempt to ameliorate the problem by using increasingly elaborate naming schemes, encoding information about the functions and parameters into the keys, but once again, this is but a step towards a relational structure where such information can be represented explicitly and in a consistent way.

2.3. Relational and logical data models

Both of the scenarios mentioned above point to a relational data model [1], where different relations are used to model the connections between signals, 'upstream' circumstantial data, and 'downstream' derived data. Tuples in these relations represent propositions such as '*this* signal is a recording of *this* song at *this* sampling rate', or '*this* spectrogram was computed from *this* signal using *these* parameters'. From here, it is a small step to go beyond a relational database to a deductive database, where logical predicates are the basic representational tool, and information can be represented either as facts or inference rules. For example, if a query requests spectrograms of wind music, a spectrogram of a recording of an oboe performance could be retrieved by making a chain of deductions based on some general rules encoded as logical formulæ, such as 'if x is

an oboe, then x is a wind instrument'. In the next section, we review some of the terminology of propostional and predicate logic so that we can see how statements such as these can be encoded.

3. A REVIEW OF PREDICATE CALCULUS FOR KNOWLEDGE REPRESENTATION

The propositional calculus provides a formal mechanism for reasoning about statements built using atomic propositions and logical connectives. An atomic proposition is a symbol, e.g. p or q, standing for something which may be true or false, e.g. 'snarks have 7 tentacles' or 'Norb is a snark'. The logical connectives $\vee$ (or), $\wedge$ (and), $\neg$ (not), $\supset$ $(implies)$ and $\equiv$ $(equivalent\ to)$ can be used to build composite formulæ such as $\neg p \vee q$ and $p \supset q$. Given a collection of axioms, new statements consistent with the axioms can be deduced. Thus, a knowledge-base could be represented as a set of axioms, and questions of the form 'is it true that ...?' could be answered by attempting to prove or disprove the query.

The propositional calculus is rather limited in the sort of knowledge it can represent, because the internal structure of the atomic propositions, evident in their natural language form, is hidden from the logic. It is clear that the propositions given above concern certain objects which may have a number of tentacles, but there is no way to express these concepts within the logic.

The predicate calculus extends the propositional calculus by introducing both a domain of objects and a way to express statements about these objects using predicates, which are essentially parameterised propositions. For example, given the binary predicate *tentacles* and a domain of objects which includes the individuals Norb and Zork as well as the natural numbers, the formulæ $\mathit{tentacles}(\mathsf{Norb}, 7)$ and $\mathit{tentacles}(\mathsf{Zork}, 5)$ express propositions about the numbers of tentacles belonging to those individuals.

The introduction of variables and quantification increases the power of the language yet more. For example, the two examples of atomic propositions given at the beginning of the section can be expressed as

$$\forall x.\mathit{snark}(x) \supset \mathit{tentacles}(x, 7), \quad (1)$$

$$\mathit{snark}(\mathsf{Norb}), \quad (2)$$

where x is a variable which ranges over all objects in the domain. In this form they are much more amenable to automatic reasoning; for example, we can infer $\mathit{tentacles}(\mathsf{Norb}, 7)$ as a logical consequence of the above two axioms. We can also pose queries using this language, for example, we can ask, 'which (if any) objects have 7 tentacles?' as $\exists x.\mathit{tentacles}(x, 7)$. An inference engine would attempt to prove this by searching for objects in the domain for which $\mathit{tentacles}(x, 7)$ is true. In this way, a query can retrieve data satisfying given constraints, something which is obviously necessary for a practical information management system.

The logic-based language is more powerful than the SQL commonly used to access a relational database management system, but nonetheless, each predicate can be likened to a table in a database, with each tuple of values for which the predicate is true corresponding to a row in the table. The calculus allows predicates to be defined using rules rather than as an explicit set of tuples, but these rules can be more complex than those allowed in SQL views.

A large part of building a logic-based information system is deciding what types of objects are going to be in the domain of discourse and what predicates are going to be relevant. Designing an *ontology* [2] of the 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. In the next section, we give an overview of the ontology of music production and computation which we have used in our information management system.

4. AN ONTOLOGY OF MUSIC AND COMPUTATION

A review of the literature on ontology development highlighted a number of points to consider when designing an ontology. These include modularity [3] and ontological 'hygiene' as addressed by OntoClean methodology [4]. In addition, we have adopted or made reference to some of the ontological structures to be found in previous ontology projects, including MusicBrainz [5], DOLCE [6], SUMO [7], and the ABC/Harmony project [8], though none of these was deemed suitable as a direct base for our system, being either too general or too specific.

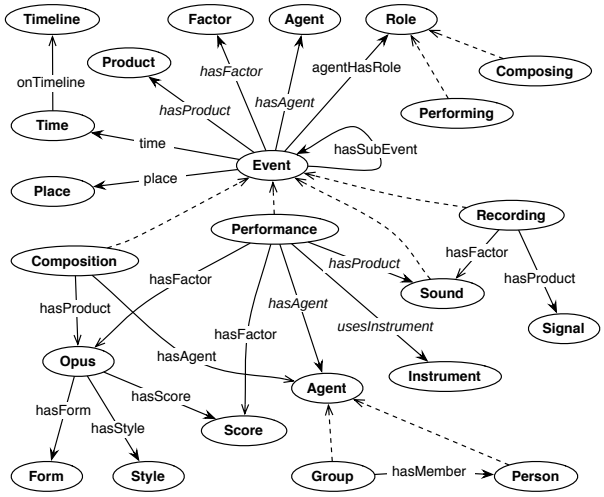

Fig. 1: Some of the top level classes in the music ontology The dotted lines indicate subclass relationships, while the labelled lines represent binary predicates relating objects of the two classes at either end of the line.

Given that we wish to represent information about music and music analysis, our ontology must cover a wide range of concepts, including non-physical entities such as *Mahlers's Second Symphony*, human agents like composers and performers, physical events such as particular performances, occurrent sounds and recordings, and informational objects like digital signals, the functions analyse them and the derived data produced by the analyses.

The three main areas covered by the ontology are (a) the physical events surrounding an audio recording, (b) the time-based signals in a collection and (c) the algorithms available to analyse those signals. Some of the top-level classes in our system are illustrated in fig. 1 and described in greater detail below.

4.1. Events

Music production usually involves physical events, which occur at a certain place and time and which can involve the participation of a number of physical objects both animate and inanimate. Because of the richness of the physical world, there can be a large amount of information associated with any given event, and finding a way to represent this flexibly within a formal logic has been the subject of much research [9, 10, 11, 12, 13, 14].

More recently, the so-called *token reification* [15, 16] approach has emerged as a consensus, where a first-class object or 'token' is used to represent each individual event occurence, and a collection of predicates is used to relate each token with information pertaining to that event. For example, partial information about Martin Luther King's 'I have a dream' speech could be stated as follows:

$$\begin{aligned} &\mathit{speech}(\mathsf{e1}) \\ \wedge\, &(\exists t.\mathit{eventTime}(\mathsf{e1}, t) \wedge \mathit{during}(t, 1963)) \\ \wedge\, &\mathit{participant}(\mathsf{e1}, \mathsf{M.L.King}, \mathsf{speaker}). \end{aligned}$$

Note that the subsequent acquisition of more detailed information, such as the precise date or location, does not require a redesign of the predicates used thus far and does not invalidate any previous statements.

Regarding the ontological status of event tokens, we largely adopt the view expressed by Allen and Ferguson [17]:

> [...] that events are primarily linguistic or cognitive in nature. That is, the world does not really contain events. Rather, events are the way by which agents classify certain useful and relevant patterns of change.

We might also expand the last sentence to say that events are the way by which cognitive agents classify arbitrary regions of space-time. Hence, the event token represents what is essentially *an act of classification*. This definition is broad enough to include physical objects, dynamic processes (e.g. rain), sounds (an acoustic field defined over some space-time region), and even transduction and recording to produce a digital signal. It is also broad enough to include 'acts of classification' by artificial cognitive agents, such as the computational model of song segmentation discussed in § 6. A typical description of some events involved in a recording process are illustrated in fig. 2.

The event representation we have adopted is based on the token-reification approach, with the addition of *sub-events* to represent information about complex events in a structured and non-ambiguous way. A complex event, perhaps involving many agents

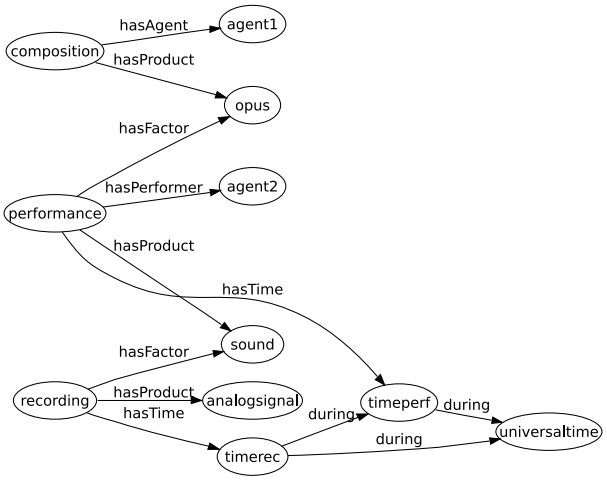

Fig. 2: Some events involved in a recording process. In this graph, the nodes represent specific objects rather than classes.

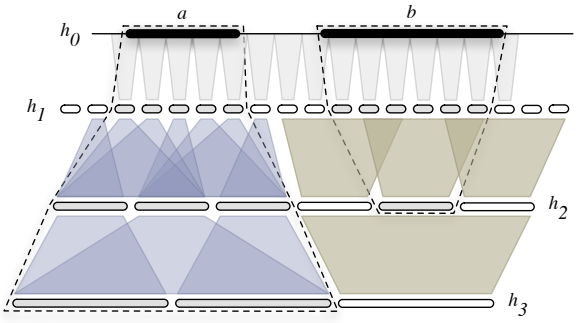

Fig. 3: An example of the relationships that can be defined between timelines using timeline maps. The continuous timeline h_0 is related to the three discrete timelines below through cascade of maps. The dotted outlines show the images of the continuous time intervals a and b in the different timelines. On the left, the potential influence of values associated with interval a spreads *out*, while on the right, the discrete time intervals which depend *solely* on b get progressively narrower, until, on timeline h_3, there is no time point which is dependent on events within b alone.

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 sub-events, each of which represents the participation of one performer using one musical instrument (see fig. 1 for some of the relevant classes and properties).

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'. Relationships between intervalse can be specified using the thirteen Allen [10] relations: *before, during, overlaps, meets, starts, finishes*, their inverses, and *equals*. These relations can be applied to any objects which are temporally structured, whether this be in physical time or in some abstract temporal space, such as segments of a musical score, where times may not be defined in seconds as such, but in 'score time' specified in measures and beats.

4.2. Time-based signals

A fundamental component of the data model is the ability to represent unambiguously the temporal relationships between the collection of signal fragments referenced in the database. This includes not only the audio signals, but also all the derived signals obtained by analysing the audio, such as spectrograms, estimates of short-term energy or fundamental frequency, and so on. It also includes the temporal aspects of the event ontology discussed above: we may want to state the relationship between the time interval occupied by a given event and the interval covered by a recorded signal or any signal derived from it. The representation of a signal simply as an array of values is not sufficient to make these relationships explicit, and would not support the sort automated reasoning we wish to do.

The solution we have adopted is in a large part a synthesis of previous work on temporal logics [10, 18, 19], which attempt to construct an axiomatic theory of time within the framework of a formal logic. This involves introducing several new types of object into our domain of discourse. Multiple *timelines*, which may be continuous or discrete, represent linear pieces of time underlying the different unrelated events and signals within the system. Each timeline provides a 'backbone' which supports the definition of multiple related signals. Time coordinate systems provide a way to address time-points

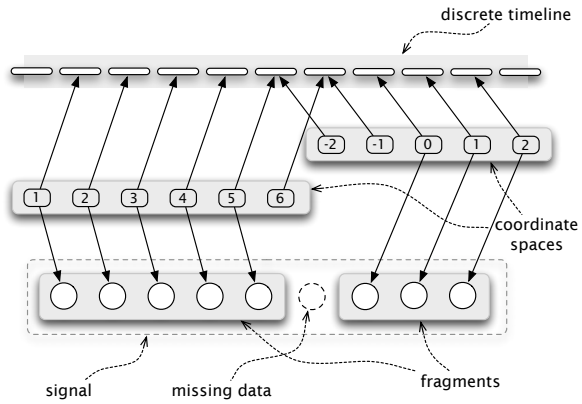

Fig. 4: The objects and relationships involved in defining a discrete time signal. The signal is declared as a function of points on a discrete timeline, but it is defined relative to one or more coordinate systems using a series of fragments, which are functions on the coordinate spaces.

numerically. The relationship between pairs of timelines, such as the one between the continuous physical time of an audio signal and the discrete time of its digital representation, is captured using timeline maps—see Fig. 3 for an example.

A particular signal is then defined in relation to a particular timeline using one or more coordinate systems to attach the signal data to particular timepoints—Fig. 4 shows an example of a (rather short) signal defined in two fragments (which could be functions or Matlab arrays); these are attached to a discrete timeline via two integer coordinate systems.

4.3. Computation and derived data

To keep track of computed data, we recognise that most of the computations we are interested in are functional in the mathematical sense, that is, a set of input-output pairs where each input has at most one output. Our data model, on the other hand, is relational, that is, composed of sets of tuples. Since every function is also a relation, a computational system built from a network of functions *automatically* defines a relational schema which can be used to store the results of each computation—it amounts to tabling or memoising each function evaluation. The data can then be retrieved using a query which closely parallels the expression used to compute that data in the first place. Essentially, we treat each function like a 'virtual table', any row of which can be computed on demand given a value in the domain of the function (which may may be a tuple corresponding to several columns). However, we can also arrange that each time a row is computed in this way, it is stored as a row in an actual table. These tabled rows can subsequently be enumerated and provide a record of previous computations. Our approach is similar in spirit to the tabling implemented in the XSB Prolog system [20], but we only allow tabling of predicates which correspond to functions.

4.4. Closing the semantic gap

Having expressed both *circumstantially* related information—which may have some 'high level' or 'semantic' value—and *derived* information in the same language, that of predicate logic, we are in a good position to make inferences from one to the other; that is, we are well placed to 'close the semantic gap'. For example, the score of a piece of music might be stored in the database along with a performance of that piece; if we then design an algorithm to transcribe the melody from the audio signal associated with the performance, the results of that computation are on the same semantic footing as the known score. A generalised concept of 'score' can then be defined that includes both explicitly associated scores (the circumstantially related information) and automatically computed scores. Querying the system for these generalised scores of the piece would then retrieve both types.

4.5. Extensibility of the ontology

We do not claim to have achieved complete expressiveness for music production knowledge, in the sense that we have not included every concept that might be useful in some situation. There are specific classes, however, which are intended to be specialisable (by subclassing) in order to be able to describe specific circumstances. For example, any instrument taxonomy can be attached below the root intrument class, or any taxonomy of musical genre could be placed under the root genre concept. Similarly, new event classes could be defined to describe, for example, novel production processes.

5. IMPLEMENTATION

In our current implementation, the ontology is coded in the description logic language OWL-DL. The different components of the system are integrated using Jena, an open source library for semantic web applications. The database is made available as a web service, taking queries in SPARQL (a SQL-like query language for RDF triples). A Prolog client has been implemented to allow standard Prolog-style queries to be made using predicates with unbound variables and returning matches one-by-one on backtracking. This style is expressive enough to handle very general queries and logical inferences. It also allows tight integration with the computational facet of the system, built around a Prolog/Matlab interface.

The computation-management facet of the system is built around a Prolog-to-Matlab interface which uses Matlab as an external engine to evaluate Prolog terms representing Matlab expressions. The service is provided through the binary predicate `===` same way as standard Prolog allows certain terms to be evaluated using the `is` binary predicate. Matlab objects can be made persistent using a mechanism whereby the object is written to a `.mat` file with a machine-generated name and subsequently referred to using a locator term. These locator terms can then be stored in the database, rather than storing the array itself as a binary object.

6. A CASE STUDY: SEGMENTATION FROM AUDIO

To illustrate how a unified information structure can be of use in practice, we describe an experiment in segmentation from audio, which examined the performance of several segmentation algorithms on a collection of popular songs. The experiment is described in greater detail elsewhere [21]; here we focus on the information management framework that was used to support it. Note that Prolog predicates are conventionally referred to by their name and arity in the form `Name/Arity`, e.g. `is/2`; we will adopt this convention in the remainder.

6.1. Representation of songs and manual segmentations

The audio database contained 14 popular songs represented as 14 performance events, each of which is associated with an agent (the performer), the name of the piece, and one or more audio files (e.g. with different sampling rates or encodings). The Prolog predicate `piece_signal/2` provides a simple way to query this information from the command line by wrapping RDF statements, for example:

```
?- piece_signal('The Beatles':T,S@R), int(L)===length(S).

>> reading wave header...done.

T = 'A Hard Days Night'
S = wavedata(''/audio/The_Beatles_##_A_Hard_Days_Night.wav')
R = 11025
L = 1762848 ;

>> reading wave header...done.

T = 'Love Me Do'
S = wavedata(''/audio/The_Beatles_##_Love_Me_Do.wav')
R = 11025
L = 1661472
```

The variables `T`, `S`, `R`, `L` are bound to the song name, the signal data, the sampling rate, and the length of the signal respectively. Note that the signal data is represented as term, which when evaluated as a Matlab expression through the Prolog-Matlab interface, returns an object containing the signal data. The `===/2` operator triggers a Matlab evaluation, in this case to find the length of the signal; 'reading wave header' is actually a message from the `wavedata` function.

The human segmentation of each song is also available through a similar mechanism, for example, the following query returns a the songs with more than 10 segment types:

```
?- piece_structure(P,E), structure_numclasses(E,N), N>10.

P = 'Cranberries':'Zombie'
E = seg_read(''/struct/Cranberries_##_Zombie.structure.txt')
N = 11 ;

P = 'Deus':'Suds and Soda'
E = seg_read(''/struct/Deus_##_Suds_and_Soda.structure.txt')
N = 13
```

Note that `E` is bound to a Matlab expression which would read a text file and return a Matlab structure describing the segmentation, and `structure_numclasses/2` is a Prolog predicate which calls Matlab to discover the number of segment-types in a segmentation. The existence of a segmentation implies the existence of one classified event for each segment.

6.2. Managed computations

A number of predicates are defined to manage the different computations that make up the analysis. For example, consider the following query:

```
?- piece_signal(_,Y),
   powspec(Y,ms(100)/ms(200),X,_,_),
   constq(X,50--5000,1/12,Z,T,F).

Y = wavedata(''/audio/A_Ha_##_Take_On_Me.wav')@11025
X = h_powspec(h_fparms(11025, [100, 1000], [200, 1000]),
       wavedata(''/audio/A_Ha_##_Take_On_Me.wav'))
Z = mat:d0509/m90318|x
T = h_timescale(h_fparms(11025, [100, 1000], [200, 1000]),
       2501568)
F = 11025*as_cqedges(11025, [50, 5000], 1/12)
```

The first time this is executed, it will trigger the computation of a power spectrogram (with 100 ms hop size and 200 ms frame length) and a constant-Q spectrogram (50–5000 Hz in 1/12-octave bands). The constant-Q spectrogram (bound to `Z`) is represented as a *locator* term which points to a newly created `.mat` file containing the spectrogram as an array. The evaluation is also recorded as a new row in the database, so that if the query is repeated, the old result is returned rather than being recomputed.

A very similar query can be used to retrieve and plot all the previously generated constant-Q spectrograms along with the parameters used to generate them (the `??` operator passes the following expression to Matlab for evaluation, in this case, producing an image of the spectrogram):

```
?- constq(_,R,Q,Z,_,_), ??imagesc(log(Z)).

R = 50--5000
Q = 1/12
Z = mat:d0509/m90318|x ;

R = 62.5--16000
Q = 1/8
Z = mat:d0509/m08805|x
```

Further down the processing chain, the core of the segmentation is accessed through the `segmentation/5` predicate, which has the form

```
segmentation(Method,NumTypes,TimelineMap,InSeq,OutSeq).
```

Without going into a detailed description, note that the first argument is a term which specifies which of several variant algorithms is used, as well as any variant-specific parameters it may require, so that a goal of the form `segmentation(_,K,TM,X,Y)` abstracts away the details of each segmentation algorithm and retrieves the union of their result-sets.

6.3. Segmentation results

Since all the information about each test piece is accessible within the same framework, the system can manage the evaluation of the results and even the generation of properly annotated figures, rendered using Matlab and stored directly to disk. Fig. 5 shows the result of such a process on the song *Smells like teen spirit* by Nirvana. At this stage of the computation, where several segmentation algorithms are available, we can begin to see the utility of applying ontological concepts to systems of functions for assigning semantic value to functions and computed data.

As noted above, the tuple-set for each segmentation function is a subset of a more general 'segmentation' relation. By finding some degree of overlap between these putative segmentation relations and the relation representing the 'true' notion of 'segmentation' (perhaps by looking at segmentations generated by humans), we gain some justification in stating that the result of a function is indeed 'a segmentation', that is, assigning the function a particular semantic value. Returning to our event ontology, the existence of a machine-generated segmentation implies the existence of a number of events just as in the case of the human segmentation; the only difference is that these events are acts of machine cognition, rather than human cognition.

7. CONCLUSIONS AND FURTHER WORK

In this paper we described how we can represent information about both music production and signal analysis in a single logic-based structure. We also described an ontology of music production and computation, which we have implemented using standard semantic web technologies.

Describing computational and derived data in a unified system model based on an ontology helps to create what we might call a *semantic workspace*. Every piece of data added to the structure automatically gains semantic value by virtue of its relationships with other existing data. In the experiment described in § 6, this information structure helped greatly in keeping 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.

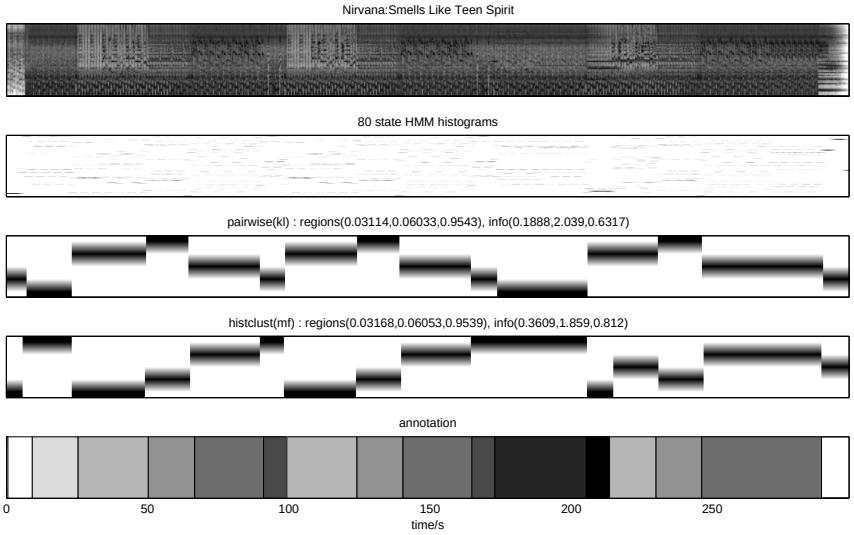

Fig. 5: Segmentation of the song *Smells like teen spirit* by Nirvana using two different algorithms. The panels contain, from top to bottom, a constant-Q spectrogram of the audio signal; a sequence of short-term HMM state occupancy histograms resulting from an HMM fitted to the spectrogram; two segmentations produced by two different algorithms, and finally, the 'ground-truth' segmentation provided by a (human) expert listener.

The analysis of the results was also managed from within the system, using Prolog queries to define which results to aggregate for each figure and calling Matlab to produce the figures themselves.

A possible extension to this would be to do a statistical analysis to judge whether or not a particular algorithm has successfully captured a given concept, and if so, to add this to the ontology of the system so that the algorithm gains a semantic value; subsequent queries involving this concept would then be able to invoke that algorithm even if no key annotations are present in the knowledge base. This would be an example 'closing the semantic gap'.

The ontology we have developed, along with all the reasoning layers (both OWL-based reasoning and computation-related reasoning), could potentially form the basis of a *musical semantic web*. We are currently working on adapting the system so that information can be distributed in a peer-to-peer network environment, where each peer can provide database or computational services either in software or by consulting a human. In this way, we hope to bring the vision of the semantic web to fruition for music processing applications.

8. ACKNOWLEDGMENTS

The authors acknowledge the support of the UK Engineering and Physical Science Research Council (EPSRC) for support of the SeMMA project (GR/S84743), and both the Centre For Digital Music and the Department of Computer Science at Queen Mary University of London for the studentship for Yves Raimond.

9. REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] C. Welty and N. Guarino, "Supporting ontological analysis of taxonomic relationships," *Data and Knowledge Engineering*, vol. 39, pp. 51–74, 2001.

[5] A. Swartz, "Musicbrainz: A semantic web service." *IEEE Intelligent Systems*, vol. 17, no. 1, pp. 76–77, 2002.

[6] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, and A. Oltramari, "Wonderweb deliverable d18: Ontology library (final)," Laboratory for Applied Ontology - ISTC-CNR, Trento, Italy, Tech. Rep., 2003.

[7] A. Pease, I. Niles, and J. Li, "The Suggested Upper Merged Ontology: A large ontology for the semantic web and its applications," in *Working Notes of the AAAI-2002 Workshop on Ontologies and the Semantic Web*, Edmonton, Canada, 2002.

[8] C. Lagoze and J. Hunter, "The ABC ontology and model," *Journal of Digital Information*, vol. 2, no. 2, 2001. [Online]. Available: http://jodi.ecs.soton.ac.uk/Articles/v02/i02/Lagoze/

[9] J. McCarthy and P. J. Hayes, "Some philosophical problems from the standpoint of artificial intelligence," in *Machine Intelligence*, B. Meltzer and D. Michie, Eds. Edinburgh University Press, 1969, vol. 4, pp. 463–502. [Online]. Available: http://www-formal.stanford.edu/jmc/mcchay69/mcchay69.html

[10] J. Allen, "Towards a general theory of action and time," *Artificial Intelligence*, vol. 23, pp. 123–154, 1984.

[11] R. Kowalski and M. Sergot, "A logic-based calculus of events," *New Generation Computing*, vol. 4, pp. 67–95, 1986.

[12] A. Galton, "The logic of occurrence," in *Temporal Logics and their Applications*, A. Galton, Ed. London: Academic Press, 1987, ch. 5, pp. 169–196.

[13] Y. Shoham, *Reasoning about Change: Time and Causation from the Standpoint of Artificial Intelligence*. Cambridge, MA: MIT Press, 1988.

[14] 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.

[15] A. Galton, "Reified temporal theories and how to unreify them," in *Proceedings of IJCAI'91*, 1991.

[16] L. Vila and H. Reichgelt, "The token reification approach to temporal reasoning," *Artificial Intelligence*, vol. 83, no. 1, pp. 59–74, 1996. [Online]. Available: citeseer.ist.psu.edu/vila93token.html

[17] J. F. Allen and G. Ferguson, "Actions and events in interval temporal logic," University of Rochester Computer Science Department, Tech. Rep., 1994.

[18] P. Hayes, "A catalog of temporal theories," Beckmann Institute, University of Illinois, Tech. Rep. UIUC-BI-AI-96-01, 1995.

[19] L. Vila, "A survey on temporal reasoning in artificial intelligence," *AI Communications*, vol. 7, no. 1, pp. 4–28, 1994.

[20] 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.

[21] 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.