Semantic Audio Analysis Utilities and Applications.

Fazekas, Gy¨orgy

Semantic Audio Analysis

Utilities and Applications

PhD Thesis

György Fazekas

Centre for Digital Music
School of Electronic Engineering and Computer Science,
Queen Mary University of London

April 2012

I certify that this thesis, and the research to which it refers, are the product of my own work, and that any ideas or quotations from the work of other people, published or otherwise, are fully acknowledged in accordance with the standard referencing practices of the discipline. I acknowledge the helpful guidance and support of my supervisor, Professor Mark Sandler.

Abstract

Extraction, representation, organisation and application of metadata about audio recordings are in the concern of semantic audio analysis. Our broad interpretation, aligned with recent developments in the field, includes methodological aspects of semantic audio, such as those related to information management, knowledge representation and applications of the extracted information. In particular, we look at how Semantic Web technologies may be used to enhance information management practices in two audio related areas: music informatics and music production.

In the first area, we are concerned with music information retrieval (MIR) and related research. We examine how structured data may be used to support reproducibility and provenance of extracted information, and aim to support multi-modality and context adaptation in the analysis. In creative music production, our goals can be summarised as follows: Off-the-shelf sound editors do not hold appropriately structured information about the edited material, thus human-computer interaction is inefficient. We believe that recent developments in sound analysis and music understanding are capable of bringing about significant improvements in the music production workflow. Providing visual cues related to music structure can serve as an example of intelligent, context-dependent functionality.

The central contributions of this work are a Semantic Web ontology for describing recording studios, including a model of technological artefacts used in music production, methodologies for collecting data about music production workflows and describing the work of audio engineers which facilitates capturing their contribution to music production, and finally a framework for creating Web-based applications for automated audio analysis. This has applications demonstrating how Semantic Web technologies and ontologies can facilitate interoperability between music research tools, and the creation of semantic audio software, for instance, for music recommendation, temperament estimation or multi-modal music tutoring.

Acknowledgements

I would like to take this opportunity to thank everyone in the Centre for Digital Music at Queen Mary University of London. My research would not have been possible without the help I received, the numerous discussions and wide-ranging collaborations I have had, and the stimulating and friendly atmosphere I have found in this group. I would particularly like to thank my supervisor, Mark Sandler, for his encouragement, guidance and support. His wide-ranging knowledge helped every aspects of my work. I would like to thank to my second supervisor Simon Dixon for his advice and meticulous corrections of my various PhD progress reports, and my internal accessor Nick Bryan-Kinns whose objective advice has helped to steered my research.

I owe a great deal to the work of people I frequently collaborated with. In no particular order, thanks to Dan Tidhar, who shared his knowledge of musical temperament with me, Thomas Wilmering for countless discussions that helped me in creating the Studio Ontology, Mathieu Barthet, Amélie Angalade, and Sefki Kolozali who I very much enjoyed working with at various music hack days, Kurt Jacobson and Matt Bernstein for their invaluable help in server configuration, Matthias Mauch and Katy Noland for enthusiastically sending me bug reports about software I wrote, and Matthew Davies for his encouragement in the early stages of my academic work. Special thanks to Yves Raimond, without his work and the Music Ontology, this thesis would not have been possible. I owe a lot to our early discussions which made me realise the importance of semantics, enabled me to understand Semantic Web technologies, and helped me to formulate my long term research goals.

Thanks to all past and present members of C4DM who created software I used in my research. The applications discussed in this thesis would not have been possible without the contributions of Chris Cannam, Mark Levy, Chris Sutton, Chris Landone, and the brilliant research of all members of the group that enable the work of Vamp plugins. I must also acknowledge the work of open source communities, including the enthusiastic people who created Audacity, Python, Numpy, Scipy, libRDF, rdflib, Matplotlib, BibDesk, LaTeX, TeXshop and numerous other open source libraries and software packages I used.

Finally, I thank my Mother and Father and all my family, for their encouragement and continued support in my endeavour in a distant country.

A note on language and style

Throughout this thesis the term ”we” is used (in line with recent trends in academic writing) to refer to the author’s work, which is influenced by numerous discussions with his thesis supervisor, faculty and colleagues. This fact is acknowledged through the use of the plural form, except in cases where its use invites the reader to think alongside, or may well disagree with the author. The thesis may reflect the sole opinion of the author in some cases however. Statements in this category are delineated using phrases such as ”in the author’s opinion” or the occasional use of first person singular.

License

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

Computers play an increasingly ubiquitous role in many areas of modern life. This is most apparent in creating, editing and managing multimedia content: text, graphics, video and audio. Bulky machines of the old days, for example, mechanical devices once used for typesetting or reel-to-reel tape recorders in the music studio, were gradually replaced by light elegant electronic hardware employing software with digital signal processing (DSP) technologies. The appearance of content-aware software applications is a relatively recent improvement in this field. A word processor which is able to find grammatical or spelling mistakes in comparison with a standard text editor is an every day example. Intelligent tools for media authoring with the ability of extracting and managing information — by means of content analysis, data aggregation, or logical inference — related to the meaning of the edited material have become imminent.

In the context of audio, this information is commonly termed as *audio semantics*. It may represent some perceptually or musically meaningful feature of sound, or some contextual information related to the creative processes of composing, performing or recording music. The primary motivation for this work is to design information and software systems to support intelligent content-aware audio editing, facilitate data collection about the music production workflow, represent audio engineering knowledge, and use this information to support diverse applications of semantic audio, such as multi-modal music retrieval, recommendation and other applications within the fields of music information retrieval (MIR) and musicological analyses.

1.1 Semantic analysis of musical audio

Music theory is the subject matter that springs into mind the most when hearing the term *music analysis*. Music however is a many-faceted phenomenon subject to a wide range of disciplines. It is important to emphasise this, since on one hand, we do not intend to, or even try to replicate analysis as a musicologist would do. On the other hand, we would not like to ignore theories related to the analysis and human perception of sound and music.

Most music we hear today starts out as human artistic expression in its fundamental origin. Through playing an instrument, it manifests in physical quantities constituting sound. Encoded in the variation of these quantities is the *message* or *meaning* as it emerges through human perception. Bertrand Russell [1948] divides sound perception as having physical, physiological and psychological aspects. Thinking along these lines, we can outline the relation of various concepts involved in music perception, and the sciences that best describe our understanding of them. This is summarised in Figure 1.1

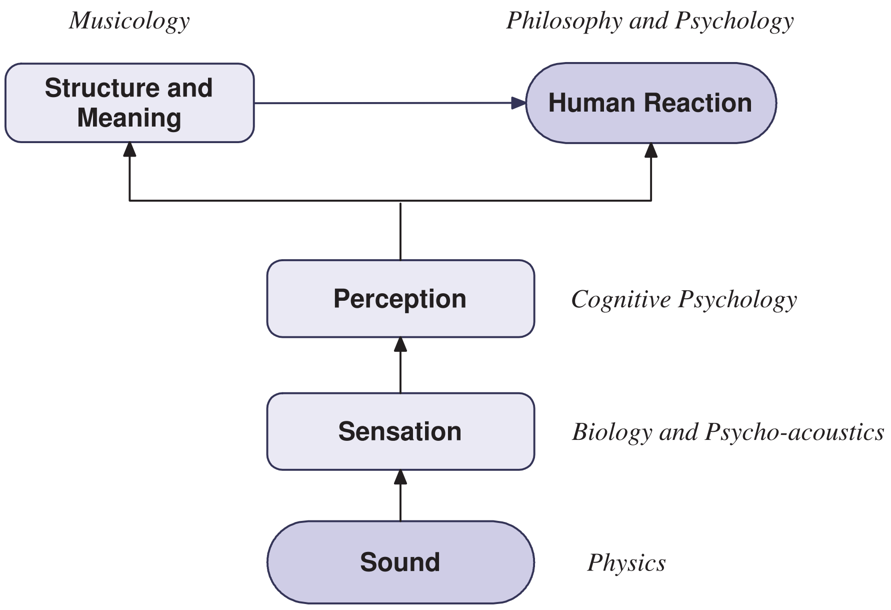


Figure 1.1: Elements of music perception

The production of sound waves and their propagation in air is studied by Physics. Together with Mathematics, the field provides us with tools (e.g. frequency transformations) for measuring its basic characteristics. Biology describes the human auditory system and the fundamentals of neurological response to sound; the sense of hearing. An inter-disciplinary field: Psycho-Acoustics studies the properties of the auditory system. The *absolute threshold of hearing*, setting the most quiet sound we can hear, or the *just noticeable difference*, the smallest change in sound stimulus that can be recognised, may be given as examples.

While sensation is the way we collect information, perception is a higher level cognitive process responsible for organisation and to some extent interpretation. It is studied by a branch of Psychology. Two models need to be noted related to *perceptual organisation* and *memory*. Grouping principles prescribed by the Gestalt school of Psychology may describe the way we organise sounds to form structure. In the context of music segmentation, short motives are formed by temporal *proximity* rules. We also group *similar* sounds together. The process of stream segregation [Bregman, 1990] separates similar sounds heard in a complex background. *Directionality* (or *continuity)* and *simplicity* are further important Gestalt rules [Zentz, 1992].

Memory also plays an important role in music perception. The recognition of the distinctive sound of an instrument or certain melodic patterns are related to learned templates, stored in long term memory. The models based on memory and Gestalt rules sometimes appear to be contradictory in the music analysis literature. See [Bod, 2001] for an example. Nevertheless, auditory sensory memory (helping the Gestalt conception) as well as short and long term memory, all play an important role in perception. Although no exact model of human music understanding exists, it appears that these models work in parallel inducing different levels of hierarchy. Musical meaning, structure and the reaction of the listener are depicted as emerging equally from the preceding steps in our model. It can easily be argued, that both individual percepts and higher level structural organisation result in some sort of human reaction. Russell [1948] asks if we can “compute” this reaction given an input and an exact model of one’s brain. Although it is unlikely that such a complete model of music perception will ever become reality, a certain amount of determinism exists in the process which can be modelled by a computer. However, the higher we try to climb this ladder, the more complex the model becomes. In Koestler’s view, cognition involves the whole arsenal of conscious and subconscious channels of sensation and internal constructs [Koestler, 1964].

Composer and musicologist Alfred Pike [1971] argues that the primary focus of music perception is at the motivic level. The motive is a short sequence of sounds, perceived as a coherent structural unit of a musical piece. While it can be accepted that this is the smallest unit in music, it is important to remember, that smaller units exists on the acoustic and sensory level. Music and an ‘intended’ meaning results from the conscious organisation of these elementary sounds. From a simplified perspective, a sound might be seen as a symbol from an alphabet. A single letter of an alphabet hardly means anything without context, but as soon as we look at an organised sequence of letters, we are able to associate, thus understand the meaning of a word. Meaning in music however is much more abstract than meaning in language. Even the existence of the concept is questioned, and it is subject to philosophical debate. Some aesthetes view music as a group of organised sounds without meaning. The other extreme is to define meaning outside music, only in its cultural context. In Cage’s view, as explained in [Pap, 2002], everything is music what we think of it as such. In prehistoric civilisations for instance, music was almost purely functional [Rezinkoff, 2004]. Some of this aspect of music is retained until today. A military march or a call signal of a radio station are good examples. In these cases the intention or meaning is clear, and for this very reason aesthetes often regarded them as non musical. In the more general case, music transmits its message through inducing emotions. Rezinkoff assigns elementary meanings to simple sound intervals of early modal music. In the correct context and temperament, intervals can express anything from simple concepts of brightness and darkness to emotions of joy or devotion. Today, this expressive capacity has disappeared, which is partly due to the introduction of equal temperament. Rezinkoff argues however, that expressiveness was recovered by using more complex harmonies and structure. More complex timbre, more complex motivic development with return and alterations of pitch and rhythm patterns of a motive play an important role in western classical music.

The same argument seems valid in case of contemporary popular music. Electronic instruments and computer based music production tools are there to create a more complex harmonic and structural repertoire. An extended structure is built from repetition and variation, a play with similarity and dissimilarity, to transmit a message and surprise. The alphabet of sounds and their combinations are infinitely extended. This makes music very different from language, and makes structure an important part of musical expression. Structure and meaning emerge from return and variation of *similar* sounds and motives. This is one reason why audio segmentation and similarity will be examined and discussed in this work. Also, beyond musical considerations, similarity (in some feature of music) is often the objective measure in segmentation algorithms. Semantic analysis of music involves finding the hierarchy of these individually coherent sections. We can define various levels of abstractions from an acoustic, cognitive or musical view point. In cognition, the resulting units are grouped together according to their place in an assumed hierarchy or perceived meaning. These units form the basis of higher level structural elements, such as the refrain of a song.

These considerations call attention to the interdisciplinary nature of semantic audio analysis and related research. In my view, the prevailing approach for extracting information from audio content, which views music solely as data — subject to general purpose statistical analyses, without the assessment of rich contextual information — has reached its natural limitation. In the following sections, the most important points of this argument will be highlighted, and a way in which complex information management and knowledge-based systems could help to overcome certain practical problems arising from these practices will be outlined. Although we build on relevant previous work, it appears that even a seemingly simple task of choosing an algorithm that yields good results in a specific analysis task requires deep insight into many of the subjects outlined in Figure 1.1, hence the complexity of systems that are able to perform such tasks. To summarise these views, this section shall be closed with a passage from Pike that best portrays the complex relationships in the perception and cognition of music.

> *"Musical perception, in its protensity, takes in relationships, differentiates, compares, groups, and forms structural and qualitative percepts. ... The process involves constant blending of the old and new, a process of continuous, progressive change which not only satisfies the listener's appetite for novelty but also widens his span of attention." – Alfred Pike [1971]*

1.2 Organisation of this work

In the rest of this chapter, after outlining the utilities of semantic audio, some basic methods for analysing music using computer algorithms will be discussed. The chapter is then closed with an outline of the music production environment and some associated problems, and the research methodology applied in this work. The rest of the thesis is organised as follows:

- In Chapter 2, the necessary background in information management and knowledge representation will be reviewed, and a model for intelligent semantic audio applications that drives the development of ontologies and software frameworks will be introduced.
- In Chapter 3, ontology engineering techniques, and the state of the art of multimedia information management will be presented using a unique, application specific view on this hard to navigate field.
- In Chapter 4, we briefly describe the Music Ontology framework, and present a large ontology extension called the Studio Ontology as one of the main contributions of this thesis.
- In Chapter 5, novel software frameworks that utilise ontologies will be introduced. This includes Sonic Annotator Web Application, a Web-based framework for automatic audio analysis, which is a major contribution in this thesis.
- In Chapter 6, we discuss the evaluation of the Studio Ontology together with case studies of using the ontology in Semantic Web and Semantic Audio applications.
- In Chapter 7, we outline our future work and conclusions.

The contributions of this thesis are listed in Section 7.2, while a detailed outline of associated publications with the author's specific contributions is provided in Appendix A. The appendices also provide a brief introduction to minor contributions that are important in the semantic audio applications discussed in the main body of this work.

1.3 Utilities and applications of semantic audio

The concept of *semantic audio* is seen in this work as the convergence of technologies for interacting with audio in human terms. The technologies involved should therefore enable the analysis of audio content in order for meaningful associations between the content and musical or other abstract concepts to be made, and enable the representation and management of these abstractions and associations in a digital computer. Using a musical example, a semantic audio tool might find notes in a recording and display how they are related to the temporal extent of an audio signal. It may enable a user to navigate the recording using this information, or enable the retrieval of recordings from a database using similar information. The capability of representing and structuring information about human or more specific musical concepts, and the analytical capability which provides for the association of these concepts with a representation of the recording are two crucial components of semantic audio applications.

Semantic audio has relevance in multiple fields of study. The broad and multidisciplinary field of music informatics, which brings together information and communications technology and music, and its specific sub-fields such as music information retrieval are good examples. We believe however that semantic audio is increasingly relevant for people who not just enjoy music, but who create music or participate in the music production process. In the following sections we briefly introduce these fields and discuss motivating examples for developing systems that facilitate diverse applications of semantic audio analysis.

1.3.1 Creative music production

Enabling applications of semantic audio in creative music production — for instance, building intelligent music production environments that rely on data collected from studio processes — is among the prime motivations for this work. The following sections provide a brief introduction to a specific area where semantic audio has increasing relevance. Audio editing and its role in music production is discussed in Sections 1.3.1.1 and 1.3.1.2, then, in Section 1.3.1.3, we outline how intelligent music production systems may support audio engineering workflows. Specific information needs of such systems and state of the art information extraction techniques will be discussed in Section 1.4, while motivating examples and problems arising in record production will be outlined in Section 1.5. The information management problems in these applications provide the motivation for developing the ontologies discussed in Chapter 4. First however, we briefly introduce the basic concepts of audio editing in a historical context.

1.3.1.1 Recording and editing music

The invention of recording before the turn of the twentieth century profoundly changed the way music is performed and appreciated. The invention brought the need for producing and distributing recordings at the best available technological level. Besides gradual advancements in the reproduction of sound, another important effect of technology on music has been implied by the ability to alter the characteristics of a performance itself by editing sound. This gave rise to a new profession: sound engineering.

There are undeniable impacts of the prevalence of edited recordings on both classical and popular music. Philip [2004] points out the most significant ones very clearly in case of classical recordings and concert hall performances:

"... by the beginning of the twenty-first century, musicians and audiences have become so used to hearing perfect performances created by editing that the general standards in the concert hall are also much higher than they used to be."

Later he elaborates on how recording changed performance practices:

"informality [in music performance] was acceptable, even welcomed, ... due partly to the fact people did not have the perfection of edited recordings as a yard-stick."

The earliest evidence of music editing appears in the process of producing piano rolls, perforated paper rolls used as medium for early automatic instruments. During the 1920s several companies were making reproducing pianos to capture real performances of famous pianists. The perforated paper rolls used as recording medium for these instruments encode the sounding notes and their durations on the piano organised into tracks of holes corresponding to each key of the instrument. The dynamics of notes were recorded onto separate side tracks. An notable addition to this fact is the use of binary code by the Ampico company for encoding the loudness of each note. This is probably the first use of such a code for a musical application[1]. These music rolls – somewhat similar to punch cards – were played back using pneumatic instruments capable of reading the roll and operate the hammers inside a piano. The player piano or *pianola* was such an instrument. Besides the fact that the reproducing piano — which was used for recording new rolls — has different action and acoustics from the instrument the rolls were played back on, piano rolls were criticised for an additional reason. During the final production of the music roll, timing inaccuracies were corrected, while much of the expressiveness — which is the most difficult to capture automatically — was added by technicians rather than the performing pianist [Philip, 2004]. Automatic instruments and music rolls had helped to overcome the serious bandwidth limitation thus poor sound quality of pre-electronic gramophone and phonograph recordings. Microphones and electronic recording devices however rapidly rendered them obsolete. Similarly to the original phonograph, the first electronic recorders were using wax cylinder medium. For this reason, editing, and even playback for monitoring purposes was impossible without damaging the recording. Although wax cylinders were replaced by direct-to-disc recordings, sound editing only became wide-spread after the introduction of the magnetic tape.

Tape recorders were first used in broadcasting during the 1930s. Soon, they became common in the music studio too, and they remain in use until today. Destructive editing, which involves physically cutting and joining pieces of tape, became generally accepted, although this technique was limited to patching mistakes of a master recording using successive takes recorded during a session. With multi-track tape recordings, it soon became impractical since these tapes contain several instruments recorded on different parallel tracks of a tape.

[1]See for instance instruments exhibited at the Musical Museum in London (`www.musicalmuseum.co.uk`)

The invention of using multiple takes to record instruments and voice separately dates back to the late 1940s. It is credited to the guitarist and composer Les Paul. This invention involved a modified tape machine such that an extra play-back head – placed before the erase head – enabled mixing previously recorded takes with new ones, and re-recording them on the same tape. This idea created the need for tape recorders with several parallel recording tracks. These machines were first developed at the Ampex company[2]. With the use of multi-track tapes, it gradually became a common practice to record instruments onto different tracks several times, then select the best parts for the final composition. Initially, analogue tape machines were used in the process, but more recently, they were superseded by hardware based digital recording equipment and finally computers.

In modern audio workstations the recording process as well as the user interaction is controlled solely by software. Despite the rapid change in studio technology, the basic concepts of recording and editing audio remains unchanged since the 1950s. Most software applications tend to recreate an analogue multi-track environment in which the engineer manually edits the best pieces together. Apart from audio effects, not much signal processing, not to mention machine intelligence, is applied to help the work. Using an analogy: a simple software for schematics design allows an engineer to draw a circuit diagram. Most of today's CAD (*computer aided design*) applications however include circuit simulation and automatic layout and routing of printed circuit boards, using the drawn schematics. In this sense, a true computer aided recording system, which uses music information, does not yet exist. We believe, that using music analysis tools in the production process is a promising way of enhancing computer based recording. For example, automatic extraction of audio fragments showing strong semantic relation can be used by an engineer as guidance. Labelling these fragments according to their similarity, musical qualities, or using them as editing constraints can assist in evaluating the numerous recordings, usually taken during a recording session.

1.3.1.2 Aesthetics of record production

Tape editing which involves manually cutting and splicing audio tape to select the best parts of a performance, multi-tracking, the recording of individual members of a music group using separate audio channels and storage, and the appearance of digital audio workstations (DAW) are among the most important historical milestones in the production of both classical and popular music records.

The introduction of computers with flexible music editing functionality, audio storage, and file transfer technology propelled an unprecedented change in the creative environment; separating recording, editing and mastering space, both in terms of geography and in terms of expertise. Although the use of computer software had already had a significant impact on music making, due to the complexities involved in computational analysis of music, machine

[2]Ampex historical tape recorders. For more details see http://www.ampex.com/l-history.html

intelligence is not widely used in the production process. It is difficult to make machine analysis comparable to human perception and understanding of musical audio. It may well be questioned, whether we should perform such analysis at all.

In the early days of tape editing, some producers maintained — especially at classical music labels — that only minimal or no editing should be applied to recordings [Philip, 2004]. Contrary to this position, the general tendency has leaned towards editing in the realm of subtleties. These developments have changed the way we experience music, and perhaps most notably, the way it is performed in the recording studio. Editing allows an unprecedented amount of perfectionism, impossible to achieve in live performance. It is outside the scope of our thesis to make an aesthetic judgement on this effect, however, a valid question can be asked: Why would we deny the instrument of revision, gradual improvement and polishing of work from the performing musician, observing that it has been practised in many different art forms for centuries? Similarly, why would we exclude the result of recent advancements in signal processing and music information systems from the creative process? Early versions of composition, text, poem, sculpture or painting are often regarded as inferior by the artist and the public alike. While they technically mature during refinement, it is unlikely that the fundamental message or semantics will be significantly changed. The novel technological features of advanced DSP and music information algorithms can improve the refinement process in many ways. Improvement of communication between an engineer and a less technically minded artist, for instance by visualising details intuitively, aiding the work of a sound engineer, or developing techniques which allow musicians to engineer for themselves more easily. These advancements may significantly change the way music is edited, mixed and produced in the studio. In our view, an intelligent music production system should consequently emphasise on *providing help*, by means of tools using automatically extracted and appropriately managed audio information, rather than automating the music editing process or taking over the work of an engineer. Since supporting engineering decisions is a complex task, an audio editor capable of this task can be seen as an intelligent system.

1.3.1.3 Intelligent music production systems

A computer-based system may be seen as intelligent if it accomplishes feats which require a substantial amount of intelligence when carried out by humans [Truemper, 2004]. In the context of audio engineering, we may think of tasks such as recognising a musical phrase with bad intonation, and finding one which is similar but superior in some sense. Easier jobs may include navigation by similarity or metric structure, alignment of phrases played by different instruments using corresponding beats, or the ability to join audio clips without noticeable artefacts. The literature on Artificial Intelligence (AI) and related fields discusses a wide variety of intelligent systems and their architectures. Most systems fall into the category of *expert systems* or decision support systems and *intelligent agents*. Expert systems on one hand are used interactively by definition [Truemper, 2004]. While the rules supporting their decisions are written by experts, these systems obtain information from non-experts in their use. Based on this information, they draw conclusions or give advice. Intelligent agents on the other hand are autonomous. They perceive their environment, and using the information they collect, they act on the environment.

Given some common audio editing use cases, and without the assumption that the aesthetic goals of an engineer can be achieved automatically, an intelligent music production system utilising semantic audio analysis would optimally fall somewhere in between these categories. It is an agent in one sense, because it is required to obtain information from the environment, which in our case is the edited audio material and the workflow context. It is an expert system, when the main objective is considered as being *workflow support* as opposed to automation. In Section 1.5.1 we will provide motivating examples, and discuss the information needs of intelligent semantic audio applications. First however, we briefly review another area if interest, namely music information retrieval.

1.3.2 Music Information Retrieval

The field of music information retrieval (MIR) evolved primarily in response to two chief user needs, *i)* managing large digital music collections on personal computers, and *ii)* accessing the increasing amount of music content on the Web. The field can be seen as a counterpart of information retrieval (IR), addressing the unique requirements and challenges presented by managing, searching or accessing musical audio as opposed to text. Music information retrieval may also be seen as a sub-field of multimedia information retrieval, which deals with heterogeneous content; audio, video, images and text. MIR is in our interest since many of the techniques developed for music retrieval may also be used in creative music production. Retrieval itself may bear relevance in applications like managing the back catalogues of recording studios. However, since MIR is a relatively young area in the centre of scientific research, many of its techniques are in the process of maturing and yet to be applied in end-user applications.

Despite the explicit reference to *retrieval* in its name, a review of the MIR literature[3] suggests that it is more commonly interpreted as *information extraction* from audio content motivated by retrieval use cases; but apart from a few exceptions, without addressing all issues related to encoding and accessing music related data. In this sense, a large part of MIR overlaps with the field of semantic audio analysis, particularly the need in both areas for extracting musically meaningful information from audio such as temp or chords. Other use cases, for instance, audio-based similarity that is commonly characterised by measuring distance in a low-level feature space, are usually seen more unique to MIR.

[3]See for instance papers presented at the International Society for Music Information Retrieval (ISMIR) conferences throughout the last decade available from: `http://www.ismir.net/`

In this work we take the broadest possible view of both fields however, and focus on all issues, including methodological ones, arising in the applications of semantic audio and MIR. This includes information extraction, information aggregation, information management, knowledge representation, as well as Web-based, and recording studio-based applications. The most relevant techniques used in semantic audio analysis and MIR will be reviewed in Section 1.4, while the necessary background on information management and Web technologies will be presented in Chapter 2. Observing that information management in MIR is underdeveloped, and that representing musical information and general Web content are similar in complexity and in their requirements, we place an emphasis on adopting *Semantic Web* technologies [Berners-Lee et al., 2001] for the purposes of semantic audio information management. The following two sections provide motivations for using these techniques and a brief introduction.

1.3.2.1 Music and the Semantic Web

Extracting semantic descriptors, such as note onset times, beats, chord progressions and musical keys from audio recordings can be useful in various applications such as audio collection management or music recommendation. Since no single solution can address the large variety of possible user needs, these data can become a valuable resource when interlinked with cultural or editorial metadata. For instance, events; concerts, tour dates, or artist relations are rapidly becoming available via a number of conventional Web services[4], as well as via the Semantic Web. These resources can be used to find connections in music in intuitive new ways as shown in the work of Jacobson [2011].

However, most existing information management solutions and previous research focus on different aspects of music related data in isolation. The main reason for this can be identified in the lack of comprehensive open-ended standards, or the lack of use thereof, for representing heterogeneous music related data, — for instance, the configuration parameters and results of audio analyses — in a common, machine-processable, and easily exchanged way. Several organisations have defined standards modelling different aspects of the audio domain, providing schema and their definition in different syntaxes, and various ways of encoding information. As a result, multitudes of incompatible standards and methods were produced without common grounds and principles. Since often a narrow area of a domain is addressed with a specific set of requirements, interoperability between applications is difficult, even if sharing otherwise overlapping information would be useful. The use of non-normative development and publishing methodologies, rather than flaws in the design of these schemata, is the most serious issue. In Section 3.2.1 we argue that common approaches, such as standardising syntax using the eXtensible Markup Language (XML), do not provide sufficient

[4]See for instance proprietary online resources like Last.fm (`http://www.last.fm/api`) or the EchoNest (`http://echonest.com/`).

ground for modularity and interoperability. One common problem is the ad-hoc definition of terms in various standards, while the lack of support in the language of choice for establishing meta-level relationships — such as equivalence and hierarchical relations — is another. These problems typically prevent interoperability, and the reuse of data expressed in most existing formats. For a detailed discussion on the problems of metadata standards see Section 3.2 in this work, or [Klein et al., 2001] and [Smith and Schirling, 2006] in the literature.

Web technologies for knowledge representation, information sharing and interlinking heterogeneous resources provide some remarkable solutions to the problems mentioned so far. The technologies developed to fulfil Tim Berners-Lee's vision of the Semantic Web include the Resource Description Framework (RDF) [Lassila and Swick, 1998]. This framework can be used in conjunction with the Uniform Resource Identifier (URI), and the access mechanism of the Hypertext Transfer Protocol (HTTP). These technologies enable the formation of the "Giant Global Graph" of machine-interpretable data. In the RDF data model, information is represented as statements consisting of *subject-predicate-object* terms. These terms are typically named by Web URIs, which may be dereferenced to access more information such as vocabulary definitions about their meaning. Although RDF provides a simple mechanism to express vocabularies: the RDF Schema (RDFS) vocabulary description language; this is generally insufficient for describing a *precise and explicit conceptualisation of a domain* similarly to the capabilities of database and software modelling languages like the Entity-Relationship (ER) model, or the Unified Modelling Language (UML). The task of providing precise and explicit conceptualisation is fulfilled by *Semantic Web ontologies*, which may be expressed using one of the several layers of the Ontology Web Language (OWL) [Patel-Schneider et al., 2004]. As opposed to most prior forms of knowledge representations, these languages have their grounding in mathematical logic, and therefore they support automated reasoning procedures (see e.g. [Horrocks, 2008]). A more detailed discussion of information management and Semantic Web technologies will be provided in Chapter 2, while the use of ontologies will be introduced in Chapter 3. In the next section, we outline how the use of these technologies may improve the field of MIR, and contribute to research and community efforts at large.

1.3.2.2 A Knowledge-based approach to MIR

Bridging the *semantic gap*, integrating *computational tools and frameworks*, and stronger *focus on the user* can be cited among the most important future challenges of music information research [Casey et al., 2008]. Prevailing machine learning tools that build statistical models typically from manually annotated data sets provide good solutions to specific problems in MIR; however they give little insight into our understanding of the musical phenomena they capture. In other words, they do not easily allow us to close the semantic gap between features and computational models on one hand, and musicological descriptors or human music perception on the other. While cognitive modelling, the use of contextual metadata in MIR algorithms, and the use of high-level reasoning are promising future directions; for a recent example see [Wang et al., 2010], some common agreement in how knowledge and information are represented in different systems is requisite for building on previous work by other researchers and for deploying complex systems.

Collins [2010] presents a musicological study where influences on composition are discovered through the use of the types of socio-cultural information mentioned above, combined with content-based audio similarity. While it is feasible to perform such studies using traditional techniques such as Web scraping, — extracting information from Web pages using natural language processing techniques — proprietary APIs of online data providers, as well as content based feature extractor tools such as Marsyas[5] [Tzanetakis and Cook, 2000], or jAudio[6] [McEnnis et al., 2005], building robust MIR systems based on these principles could be made easier through agreement on how diverse music related data is represented and communicated. As a result of such an agreement, the laborious process of aggregating information could be reduced to making queries to distributed resources of linked-data on the Semantic Web.

The use of a wide variety of programming languages, software libraries and Application Programming Interfaces (API) within the MIR community is an additional problem to be noted. This practice hinders the process of exchanging research results, and makes it cumbersome to build on prior work. The answer to this problem may be found in using common interfaces such as Vamp [Cannam, 2009], an API for audio analysis enhanced with easily extended Semantic Web ontologies. These ontologies, built on the foundations of the Music Ontology (MO) framework [Raimond et al., 2007] outlined in Section 4.1, can be used for describing both analysis results and algorithm configuration, as well as other aspects of the analysed piece of music. The utilities of the combined use of these technologies and ontologies is demonstrated by the Sonic Annotator Web Application (SAWA), a Web-based system for audio analysis (see Section 5.3). The motivation for developing this systems is twofold. On one hand, it provides an easily accessible exposition of MIR algorithms implemented as Vamp plugins, on the other hand, it aims to show how Semantic Web technologies can facilitate interoperability between service components within an MIR system, and provide access to external systems at the same time. This includes online databases containing editorial information such as MusicBrainz[7], or Semantic Web resources such as DBpedia [Auer et al., 2007]. The integration of computational tools through the use of interoperable software components and extensible metadata management was also in the focus of the OMRAS2 project[8]. The software frameworks and ontologies discussed in chapters 4 and 5 are novel contributions to this project. We aim to show how these components support reproducible research, the

[5]Marsyas: `http://marsyas.info/`

[6]jAudio: `http://jmir.sourceforge.net/jAudio.html`

[7]MusicBrainz is a community created open music encyclopaedia: `http://www.musicbrainz.org/`

[8]See `http://www.omras2.org` for details, and [Dixon et al., 2010] for an overview.

combination of different music analysis tools and resources, and collecting data about music production. First however, we outline some relevant signal processing and machine learning techniques for extracting information from audio content. These techniques are used as basis in most systems discussed in this thesis.

1.4 Signal processing techniques for semantic audio analysis

Extracting information from audio recordings is requisite for building semantic audio applications. An outline of the fundamentals of analysing music using computer algorithms, including low-level physical and perceptual, as well as high-level semantic feature extraction techniques is therefore crucial for describing the type of tools we discuss in this work. The following brief and critical review highlights some information management problems arising in the applications of audio analysis, and serves to inform about underlying needs for the design of ontologies and information management solutions discussed in the following chapters. First, we briefly review the basic categorical distinctions in the features of sound and music, and the relationships of these features in the physical, perceptual and musical domains.

Physical Quantity (or concept)	**Perceptual Quality**	**Musical Category**
frequency (fundamental)	perceived pitch	musical note
amplitude (intensity)	perceived loudness	dynamics
duration (time)	perceived duration	beat and tempo
waveform (or complex spectrum)	perceived timbre	tone quality (e.g. instrument)

Table 1.1: Principal dimensions of elementary musical sounds, based on [Olson, 1952].

Olson's taxonomy of musical dimensions [Olson, 1952] provides an insightful parallel view on how the qualities of sound and music are interpreted in various disciplines, and provides a basic terminology related to the concepts of physical and psychological qualities. Following this line of thought enables us to resolve ambiguities that often appear in relation to acoustical, perceptual and musical quantities.

Table 1.1 illustrates physical quantities used to describe elementary sounds and the most related perceptual and musical concepts. It is important to observe that the further we depart from basic physical quantities, the concepts become more complex, and it becomes more difficult to establish the correspondence between categories. Olson defines additional properties which can be understood physically, perceptually and musically. A sound may be characterised by *growth and decay,* or the attack and release times related to timbre, *harmonicity and inharmonicity*, regular or irregular spacing of frequency components, *vibrato* (frequency modulation), *tremolo* (amplitude modulation). In relation of two or more sounds we can talk about *consonance, dissonance or harmony*, the fundamental frequency ratio and coincidence of partials, *beating,*, that is, a perceived pitch and loudness fluctuation of sounds close in frequency, and *rhythm*, some uniform variation in duration and dynamics of a succession of sounds. Obtaining audio features corresponding to simple physical quantities, such as the fundamental frequency of a sound, is a question of measurement involving simple mathematical transformations. Extracting perceptual qualities require more complex computation, often involving auditory modelling, while musical interpretation of audio recordings (e.g. transcription, beat tracking) involves algorithms replicating a complex process of perception and to some extent cognition.

In semantic audio applications, the more help we wish to provide to users, the more important the reliance on higher level processing becomes. Due to the increasing ambiguity in conceptual relationships of quantities across different domains (e.g. how fundamental frequency or perceived pitch are related to a musical note), correct identification of musical categories, for example, recognising a musical note or an instrument will require more complex processing, such as pattern recognition and classification, or knowledge-based processing, which relies on logical inference using contextual information alongside directly measured physical quantities. In the following sections, we provide an overview of relevant audio feature extraction techniques.

1.4.1 Physical and perceptual features of sound

We may dichotomise audio features several different ways. Perhaps the most common, though slightly ambiguous distinction is between low and high-level features (please see a thorough discussion of audio feature categorisation in Section 3.2.5.1), where low-level features correspond to the first two categories discussed in the previous section, namely physical quantities and perceptual qualities. These features are generally non-musical, therefore they cannot be directly associated with high-level meaning. We can further subdivide low-level features such that perceptual features, (e.g. perceived pitch or complex timbre models of instruments) are though to be mid-level representations, while physical quantities (e.g. the attack time or fundamental frequency of a note) are though to be low-level features. In this section we discuss these two categories.

1.4.1.1 Fundamental frequency and perceived pitch

Most elementary stationary sounds, whether sinusoidal or complex, harmonic or inharmonic are assigned a pitch by the human auditory system, unless they are completely aperiodic. This is known as perceived pitch. Because of the complexities involved in perception (e.g. in case of a missing fundamental, there may be a perceived *virtual pitch*) and auditory modelling, a large variety of algorithms were proposed for deriving the perceived pitch both in the time domain and the frequency domain. These include techniques based on time domain autocorrelation and the Cepstrum. A comprehensive overview can be found in [Klapuri, 2004].

Despite this large variety, the most straightforward interpretation of the pitch of a complex tone is its *time domain repetition rate*, or fundamental frequency. Boersma [1993] showed, that the best candidate for finding the acoustic pitch of a sound is the highest peak of its autocorrelation function (ACF). In a recent thesis, Purwins [2005] demonstrated through perceptual experiments, that direct autocorrelation is capable of resolving the perceived pitch in most cases. Considering these results we compared an autocorrelation based method for pitch determination with some Cepstral methods.

The autocorrelation of an N-length signal x(n) is given by Equation 1.1. Through the efficient implementation using the Discrete Fourier Transform (DFT) (see Equation 1.2), we find the relation between the ACF and the Cepstrum. This can be written in a generalised equation given in (1.3). The unification of these methods appears in the literature as the *generalised autocorrelation* [Tolonen and Karjalainen, 2000], using the exponent 0.67, instead of power spectrum or log spectrum computation.

$$r_{xx}(i) = \frac{1}{N} \sum_{n=0}^{N-i-1} x(n) \cdot x(n+i) \tag{1.1}$$

$$X(\omega) = \sum_{n} x(n) e^{-j\omega n}, \tag{1.2}$$

where $\omega = 2\pi k/N$, and k is the index of frequency bin such that $0 \leq k \leq N$.

$$r_{xx}(i) = IDFT\left[f\left(|DFT(x_n)|\right)\right], \tag{1.3}$$

where

$$f() = \begin{cases} f()^2 \rightarrow \text{ACF} \\ \log() \rightarrow \text{Cepstrum} \\ f()^{0.67} \rightarrow \text{generalized-ACF} \end{cases}$$

and IDFT is the inverse transform of the DFT given in Equation 1.2.

In order to obtain low-level segmentation, which may be used for instance as guidance for selecting individual notes in an audio editor [Fazekas and Sandler, 2007b], we employed Boersma's algorithm. We first extract a continuous pitch track (see Figure 1.2/b) and use it in a pitch based onset detection algorithm proposed by Collins [2005]. Boersma's method provides an efficient and relatively accurate way of pitch detection. In an attempt to develop a general purpose algorithm for symbolic or quasi-symbolic representation of track, we first verified the results shown in [Monti and Sandler, 2000] for example, that ACF-based techniques perform well in finding the pitch of instruments with stable, harmonic sound. Although, the pitch based segmentation proposed by Collins was further enhanced and shown to work well for pitched non-percussive instruments, such as the violin, in [Schleusing et al., 2008], we found that reliable conversion from a continuous pitch track to symbolic notation turns out to be very difficult. This is especially the case when analysing unprocessed multitrack recordings (see [Viitaniemi, 2003] for detailed discussion). This is demonstrated in Figure 1.2 with a worst-case example using singing voice. The dashed black lines in the diagram represent the ground truth, obtained through a listening experiment, by manually tuning pure tones such that they sound unanimously with the sung notes.

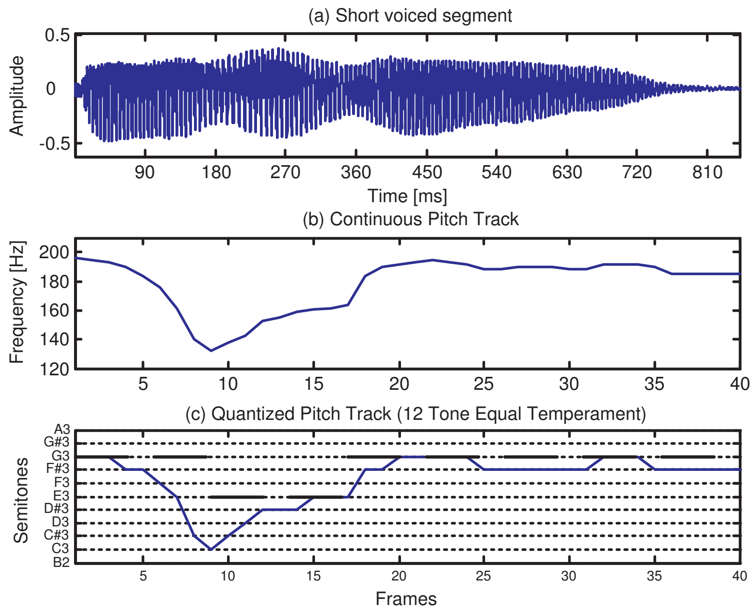


Figure 1.2: Pitch track (b) of a short voice segment (a) and its transcription (c)

Comparing the manual transcription with the quantised pitch track shows that human cognition ignores pitch instabilities and glides during legato singing. To solve this problem, Viitaniemi and Klapuri [2003] proposed a hidden Markov model (HMM) based probabilistic approach, which uses the pitch trajectory and a musicological model. This model was trained on a large database of folk songs to learn note transition probabilities, however, the transcription accuracy is still worse than human judgement, with error rates up to 20 percent. Common alternative implementations of pitch estimation include YIN, which can be seen as a variant of auto-correlation [Cheveigné and Kawahara, 2002]. It makes an attempt to resolve ambiguities in choosing the ACF peak which corresponds to the fundamental. A more recent algorithm SWIPE and a general comparison of several methods in the context of different instruments is provided in [Camacho, 2007].

Our subjective tests, as well as the relevant literature mentioned above suggest that pitch transcription alone is not sufficient to obtain a general purpose, reliable method for the symbolic representation of audio tracks. These results indicate however that knowledge-based music processing systems could benefit from the ability to select the most effective algorithm or parameters given some contextual information, such as the instrument or recording conditions, and inform us about knowledge and information management requirements in semantic audio production tools.

1.4.1.2 Low-level timbre features

Low-level timbre features are often used to characterise sound or serve as bases for higher-level semantic audio analysis tasks such as speech-music segregation. These features are commonly computed from the windowed short-time Fourier transform (STFT) of the signal given in Equation 1.4

$$X(m,k) = \sum_{n} s(n)w(mh-n)e^{-j2\pi nk/N} \tag{1.4}$$

where m is the frame index, h is the hop size, and $w(n)$ is an N length window function.

A typical example of low-level features is spectral centroid, see Equation 1.5, an important attribute of timbre. Its high correlation with the *brightness* of a sound was shown in psychological experiments using multi-dimensional scaling (MDS) [Kruskal and Wish, 1986]. In Equation 1.5 *freq(k)* is the centre frequency and $|X(m,k)|$ is the magnitude of bin k of the N point short-time discrete Fourier transform of frame m.

$$sc(m) = \frac{\sum_{k=0}^{\frac{N}{2}-1} \text{freq}(k) \cdot |X(m,k)|}{\sum_{k=0}^{\frac{N}{2}-1} |X(m,k)|} \tag{1.5}$$

Spectral variation or spectral flux (sf) is another commonly used low-level feature. It expresses the dissimilarity or shape change between the magnitude spectra of successive frames. It is usually implemented as the Euclidean distance (l_2 norm) between normalised vectors, however, depending on the application the implementations often vary. It may be computed by the normalised cross-correlation as in [Peeters, 2004], or using the l_1 norm, for instance, when applied as an onset detection function as in [Dixon, 2006]. This application is shown Equation 4.25, where the function $H(x) = (x + |x|)/2$ represents half-wave rectification.

$$sf_{df}(m) = \sum_{k=0}^{\frac{N}{2}-1} H(|X(m,k)| - |X(m-1,k)|) \tag{1.6}$$

Because of the ambiguities of use in different applications, spectral flux is a good example where a music processing system could benefit from using a knowledge-base. Given that the extractor is available as a low-level processing block, and our knowledge-base contains the associations between the descriptors and the applications, our system should be capable of choosing the implementation most appropriate for the task. In the broader view, for example in onset detection tasks, such a knowledge-base could encode the empirical results of larger-scale experiments like the one described in [Dixon, 2006].

1.4.1.3 Timbre models and spectral envelopes

Timbre has multiple definitions in the literature depending on how broad our interpretation is. In one view, timbre describes all aspects of sound but pitch and amplitude, and include features such as the attack time, or the temporal centroid of a musical note. Others mainly relate it to harmonic content. Dannenberg [1993] ironically noted: "*timbre is by definition that which we cannot explain*". More recently, especially in the contexts of music similarity and structural segmentation, timbre has been generally interpreted as the time-varying spectral envelope. Properties of the human auditory system, namely the finite resolution of auditory bands, allow a compact representation of the harmonic content. In this sense, the timbre space has to be just large enough to resolve the perceivable difference between the sounds of two instruments, textures or phonemes.

The two most common ways to obtain spectral envelopes are Linear Prediction and the Cepstrum, both are well established in speech communication research. Linear prediction for instance is commonly used in telephony and speech transmission. Linear predictive coding (LPC) approximates speech production using the source-filter model. In this model, the vocal tract is represented by an all-pole resonant filter (synthesis filter), which is the inverse of the analysis filter. The analysis filter coefficients are generally estimated from the short-time autocorrelation of the signal. Due to quantization sensitivity of LPC coefficients, they are commonly converted to reflection coefficients related to the vocal tract tube model, or line spectral frequencies (LSF), which are frequency domain representations of the filter poles and the related formant frequencies. A disadvantage of using direct-form LPC coefficients lies in the fact that some poles appear deep within the unit circle. This makes finding the roots of the filter polynomial computationally expensive. A solution to this problem is provided by using Line Spectral Frequencies [Kabal and Ramachandran, 1986].

$$A(z) = \frac{1}{2}\left(P(z) + Q(z)\right), \tag{1.7}$$

where

$$P(z) = A(z) + z^{-p+1}A(z^{-1}) \tag{1.8}$$

$$Q(z) = A(z) - z^{-p+1}A(z^{-1}) \tag{1.9}$$

The LPC analysis filter polynomial A(z) can be rewritten as a pair of symmetric and anti-symmetric polynomials P(z) and Q(z). This is shown in Equation 1.7. The most impor-

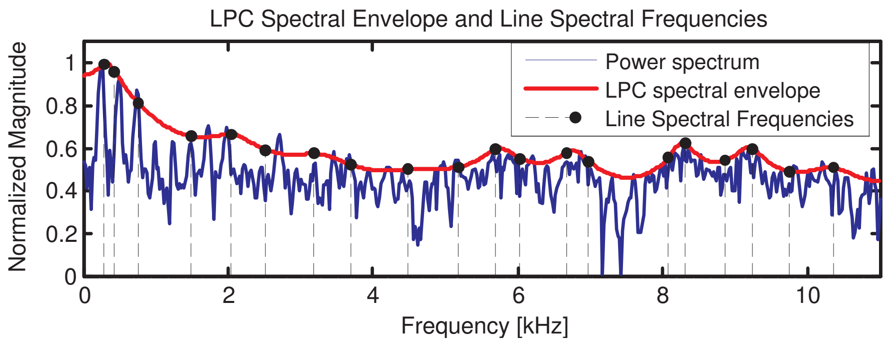


Figure 1.3: Spectral envelope derived by LPC and line spectral frequencies

tant property of these polynomials is that the roots always lie on the unit circle, and they alternately surround resonant peaks of the of the LPC filter. These properties make the computation of LSFs efficient using the iterative root finding algorithm of Kabal. This algorithm finds the polynomial roots by searching through the unit circle using bisection. Figure 1.3 exemplifies the spectral envelope and LSF coefficients.

Another representation of timbre we may utilise is *Mel-Frequency Cepstral Coefficients* (MFCC) derived from the computation of the *cepstrum* [Rabiner and Juang, 1993]. This representation is commonly used in speech recognition due to its flexibility in describing spectral envelopes at any desired resolution. The cepstrum separates the slowly varying components of a signal from the superimposed high frequencies or noise like components by taking the logarithm and an inverse transform of the power spectrum. This is shown in Equation 1.10. It can be viewed as a re-arranged spectrum, such that the higher the number of coefficients, the more spectral detail is retained.

$$c(n) = \frac{1}{2\pi} \int_{-\pi}^{\pi} \log |X(\omega)| \, e^{j\omega n} d\omega \tag{1.10}$$

Before computing the cepstrum, it has become common to use non-linear frequency warping in order to emphasise perceptually important low frequencies. Mel-scaling is a common method for modelling auditory bands. However, a simple log-frequency scale, the Bark scale, or Equal Rectangular Bandwidth (ERB) scale may also be used. In these processes, the spectral bins of a DFT are grouped according to an auditory model or scale. We already showed the efficient computation of the cepstrum through DFT and IDFT in Equation 1.3. The Discrete Cosine Transform (DCT) however is commonly used to compute the MFCCs. Typically the first twenty to thirty MFCCs are used to representation a spectral envelope. This is illustrated in Figure 1.4. In the example of Figure 1.4b, the first 22 coefficients are used to compute the spectral envelope shown in Figure 1.4a. Note that DC coefficient is set to zero. It can be seen that the first two harmonics are not resolved by this representation, however the accuracy can be increased by using more coefficients.

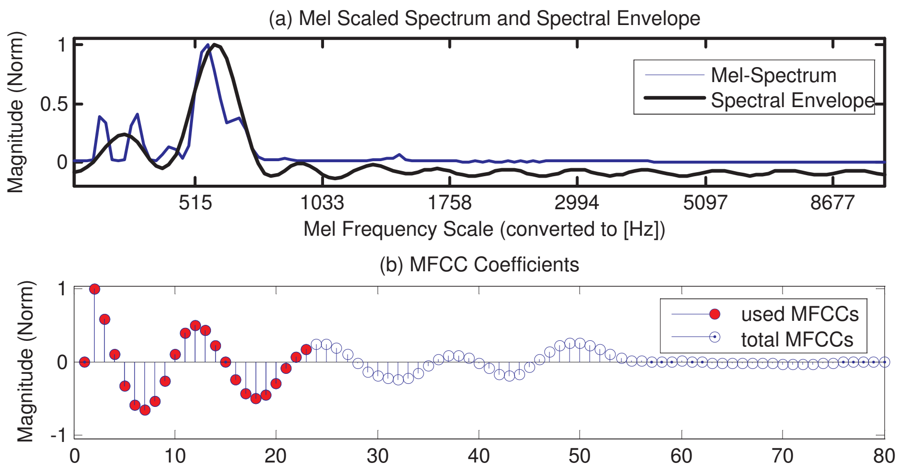


Figure 1.4: Spectral modelling of a single audio frame using MFCCs

1.4.2 Semantic features of audio and music

High-level or semantic features of audio closely correspond to musically meaningful concepts, for instance, notes and note onset times, rhythm, tempi, keys, chords or structural segments, such as musical phrases, movements, or choruses and verses in popular music. In this section, we review the most common techniques for extracting this information from audio recordings. We also discuss some of the limitations of the state of the art, and highlight some information management needs of semantic audio tools designed to deal with this information.

1.4.2.1 Onset detection, tempo estimation and rhythm analysis

Rhythmic structure and organisation is an essential feature of music. Since various sources provide different descriptions, it is difficult to find a universally accepted definition of rhythm. Still, we can identify its most important aspects: *Tempo* is the overall repetition rate of beats, while *metre* is the hierarchical structure of musical time. Lerdahl and Jackendoff [1983] identify *grouping* as an entity distinct from *metre*, expressing rhythmic structure. Thus *rhythm* can be seen as the manifestation of temporal-accentual patterns of musical events. Similar consensus seems to be accepted by music psychologists and musicologists. See [Clarke, 1999] or [Patel and Peretz, 1997] for detailed discussion and various possible definitions.

Rhythmic analysis of audio recordings usually employs note onset detection as the front-end feature extraction step. This may be performed in the time domain, but state of the art techniques use frequency domain processing. Typically, a detection function is computed first, and the onset locations are identified by selecting local maxima of the detection function [Hainsworth, 2004] which satisfy certain conditions, for instance, having a magnitude exceeding a predefined threshold.

A key difference between onset detectors is the choice of the detection function. This may be as simple as the energy or magnitude of linearly weighted short-time Fourier transform vectors, such as the high-frequency content detection function [Duxbury et al., 2002]. Distances between successive STFT frames may also be used to derive simple detection functions measuring the difference in successive spectral profiles. More complex functions, such as the complex-domain detection function [Bello et al., 2005] take phase information into account. Using this detection function, note onsets are emphasised either as a result of significant change of energy in the magnitude spectrum, or the deviation from expected phase in the phase spectrum caused by a pitch change.

Alternative techniques formulate onset detection as a classification problem, and attempt to decide whether two frames of a certain temporal distance could be coming from the same event [Kapanci and Pfeffer, 2006]. The combination of multiple onset detection functions were also explored. This may take the form of a linear combination of different detection functions, or involve the use of higher-level fusion rules to combine peak information obtained from several onset detectors. The use of information fusion methods facilitates gathering the efforts of the MIR community, which develops multiple signal processing algorithms for similar purposes [Degara-Quintela et al., 2009].

Algorithms for various rhythm related tasks such as tempo estimation and beat tracking may rely on onset detection directly. The method described in [Dixon, 2001], derives tempo hypotheses from clustering inter-onset-intervals. These are then used within a multi-agent system, forming multiple beat agents which compete based on how well each can predict beat locations. Other systems use multiple note onset detection functions derived from different frequency sub-bands of the signal, or incorporate harmonic change detection, and prior knowledge such as an assumed steady tempo. Comprehensive overviews of these techniques are available in [Gouyon and Dixon, 2005] and [Davies and Plumbley, 2007].

Despite the multitude of techniques proposed for onset detection, extracting onsets from audio recordings is not a fully solved problem, and can introduce undesirable errors such as missed detections and false positives. Therefore Davis uses a continuous detection function directly as the input to his beat tracker [Davies and Plumbley, 2007], without extracting onset locations first. In this method the beat period (the time between successive beats) is estimated using an adaptively filtered and half-wave rectified detection function. To emphasise the strongest peaks, an adaptive moving mean threshold is computed and subtracted from the onset detection function. Then, normalised autocorrelation of this function is computed, and the lag from the autocorrelation domain is mapped into a measure of tempo in beats per minute, using a weighted comb filtering technique. This tempo hypothesis is used as prior knowledge in a two-state model, which is able to adopt to tempo changes, as well as able to prevent the two most common beat tracking errors: *i)* switching between metrical levels (elements in the hierarchy of musical metre, e.g. beat and bar level, see [London, 2012]), for instance half or double the beat period, and *ii)* switching between *on* and *off*-beat.

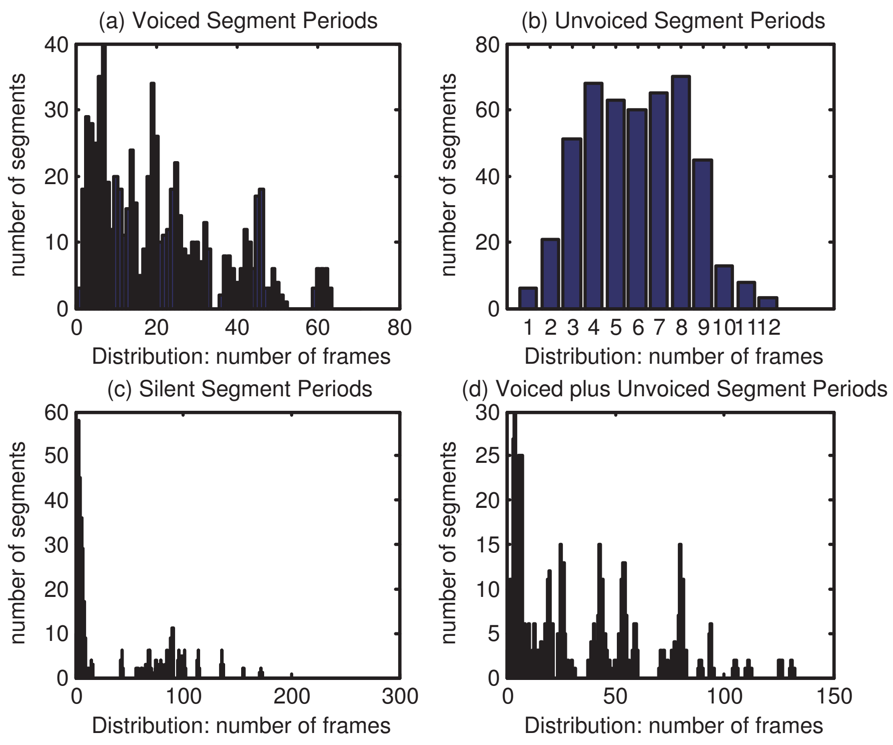


Figure 1.5: Using segment-duration histograms for periodicity estimation [frame size = 11.6ms]

This and other techniques mentioned in the literature above indicate that using contextual information as prior knowledge in audio analysis tasks can improve the results. For instance, finding beat locations can be made computationally simpler or more accurate given the priors such as tempo, time signature or metrical structure. This type of data about a recording is usually available during production. For instance, the tempo and time signature of a recording project is typically set by the engineer when generating a click track which guides the musicians during a recording session. We can therefore easily argue that semantic audio tools should be able to collect and manage this information, in order to facilitate semantic audio analysis. The results may be used in music information management and retrieval, or audio analysis applications that aid the recording process itself.

Tempo and beat analysis can be used to support various other audio analysis tasks. Beat synchronous analysis is more and more commonly used in tasks such as chord change detection (i.e. the most likely chord boundaries may coincide with beats), or structural segmentation, as described in [Levy and Sandler, 2006b] and [Mauch, 2010]. It is also possible to reverse the process and use for instance segmentation techniques to infer rhythm related information. We exemplify an application of processing separate tracks in a multitrack recording environment in [Fazekas and Sandler, 2007b]. Here, we obtain periodic segment duration histograms from coarse segmentation of voice recordings shown in Figure 1.5a. These segments are extracted using standard speech processing techniques such as voice activity detection, voiced/unvoiced segmentation and pitch change detection. Seeking for subdivision of these segments, we can use periodicity information obtained from the histograms to estimate temporal locations of expected note transitions. Periodicities can also be used to focus similarity search on segments that are clustered around a peak (thus similar in length).

In an even wider context, to exemplify an application of beat detection in multi-track processing, we may extract rhythmic information from percussive tracks, and superimpose this on non-percussive tracks, for example a violin recording or other string or wind instruments, where this information is harder to obtain, assuming that they follow the same tempo and metre. This can play an important role at later steps, for example in improving note segmentation or refining the detection of higher-level segment boundaries.

1.4.2.2 Harmony, tonality and chord estimation

Key, tonality and chord detection are of high importance in semantic audio analysis and prospective intelligent audio production systems. They can be equally useful in numerous different applications, such as navigation by harmonic structure, visualisation, content management or retrieval from music databases or recording studio archives.

Harmony in tonal music [Dahlhaus and Gjerdingen, 1990] relies on the ability of the human auditory system and cognitive processes to perceive simultaneously sounding pitches, and associate them with some musical function. Therefore, any algorithm designed to extract harmony-related information from audio (see for example chord recognition in [Fujishima, 1999; Harte and Sandler, 2005] or [Lee and Slaney, 2008]), has to rely on musically meaningful lower-level features such as the pitches of musical notes.

The coefficients of the traditional Fourier transform based spectrum are equidistant in frequency. However, the fundamental frequencies of musical notes (as well as the associated perceived pitches) are approximately equidistant in log-frequency, therefore, to obtain musically meaningful representations of audio, we have to move to log-frequency. This can be done by mapping the frequency coefficients of the STFT to log-frequency coefficients, however large transform windows are required to resolve the pitches of musical notes notes at lower frequencies, resulting in poor time resolution. Key and chord analysis algorithms therefore typically utilise time-frequency transforms, whose frequency spacing can be adjusted to match the geometric frequency spacing of musical notes. It may also be necessary to adjust the analysis to specific tuning systems and/or recording characteristics such as an unconventional concert pitch, which may diverge greatly from the standard 440 Hz of middle A.

The Constan-Q transform [Brown, 1991] shown in Equation 1.11 provides a solution to the problem above, by having a constant ratio of centre frequency to resolution

$$CQ(k) = \frac{1}{N(k)} \sum_{n=0}^{N(k)-1} w(k,n)x(n)e^{-j2\pi Qn/N(k)} \tag{1.11}$$

where $w(k, n)$ is a window function which depends on the frequency bin k, Q is the quality factor, the ratio between the central frequency f_k of the k-th bin and the corresponding resolution (filter width), and $N(k) = Q \times (f_s/f_k)$ where f_s is the sampling frequency.

Early chord extraction algorithms however were still dependent on the STFT. For example, the basis for chord detection in Fujishima's method [1999] is the pitch class profile (PCP) representation, computed by summing the power spectrum coefficients that are close to the frequency corresponding to either one of the individual pitches in a pitch class. This results in a 12 dimensional chroma vector, a chromatic representation of the harmonic content of an audio signal, where spectral content is essentially wrapped into a single octave [Bartsch and Wakefield, 2005]. Chord names are then estimated by finding the best matching chroma profiles given a chroma vector. Harte [2005] improves on these results by using a tuned Constant-Q transform instead of the STFT. The first chord detection technique that uses machine learning models was introduced in [Sheh and Ellis, 2003], which employes Gaussian models of chord templates learned from labelled audio data and a hidden Markov model. More recent techniques increasingly move towards the use of supervised machine learning for chord recognition, but the use of the Constant-Q transform, and PCP as the underlying feature remains predominant (see [Papadopoulos and Peeters, 2007] or [Mauch, 2010] for more comprehensive overviews). The use of musical contextual in chord recognition was introduced in [Mauch and Dixon, 2010b], where information such as metrical structure or the key of a piece are incorporated in the chord extraction process using Dynamic Bayesian Networks (DBN) [Murphy, 2002].

Different applications of chord detection in other semantic audio analysis tasks were explored by several studies. The use of chord extraction, for instance in structural segmentation (see Section 1.4.2.3), was demonstrated in [Bello et al., 2005], who found, not surprisingly, that the sequence of chord labels correspond well to the structure of pop songs. The reverse of this idea, improving chord recognition by structural segmentation, has been found to be useful in [Mauch et al., 2009], while Lee and Slaney [2008] introduced a key-dependent model for chord transcription. All these techniques show the high degree of interdependency between high-level musical features, which supports the notion that using a database, together with uniform representation of feature components shall lead to the generalisation of these techniques, as well as improved efficiency in practical applications, since the underlying low-level features, such as chromagrams, will not need to be recalculated.

Musical key estimation is based on similar techniques described above in the context of chord detection, i.e. matching compact frequency domain representations of audio (i.e. tone profiles, profiles of salient pitch classes) with templates of harmony or a model of harmony progressions, see for instance [Peeters, 2006] and [Noland and Sandler, 2009].

We can speculate that processing multitrack data and utilising information collected in the recording studio (e.g. metadata about instruments and recording conditions) provide a potential improvement on the state of the art. Using a knowledge base, in which we encode what instruments contribute positively to chord or key analysis, and which are the ones that degrade the performance, we can fine tune the processing to a multitrack recording. For instance, keyboard and guitar accompaniments almost certainly improve the analysis, while lead instruments or drums may degrade it. Then, an intelligent system can create an optimal sub-mix from the available tracks before the analysis.

1.4.2.3 Structural segmentation

Semantic music segmentation involves finding the hierarchy of individually coherent sections in a musical piece. Various levels of abstractions can be defined starting from notes, the most fundamental units in music, or other perceptually coherent small acoustical units, such as phonemes of speech or singing. Bars, motives and phrases are sequences of notes, grouped together according to their place in some hierarchy, such as the metrical structure, or a perceived meaning, often characterised by direct repetition or similarity within a sequence of events, such as a repeating melodic phrase or a chord progression. These units form the basis of higher-level structural structures, such as movements in classical pieces, or the refrain of a popular song.

In this section, we are concerned with methods for extracting high-level segments from music. These methods have several potential applications in music production tools that utilise semantic audio analysis. For instance, it is now standard in the studio to compose the final mix of a song from the numerous audio tracks taken during a recording session, and to select technically superior or artistically more expressive phrases played by an instrument and use them in a song release. Since takes recorded successively and repetitions at different song positions can both be used, the process involves exhaustive search and assessment of musically similar audio sections within or between tracks.

A sound engineer or producer working in a complex multi-track software environment would greatly benefit from using the song structure as guidance in the post-production process. Although manual annotation of audio tracks is possible in most applications, it is a time-consuming and often weary process. An intelligent audio editor would provide the song structure, visually laid out in the project window, together with simple semantic constructs, like *find the next similar segment*, in order to navigate around the recorded material. Based on segmentation techniques described in [Levy and Sandler, 2006b] and [Levy and Sandler, 2008], we developed a system for utilising automatic timbre analysis and structural segmentation in a multitrack audio editor [Fazekas and Sandler, 2007a]. This application provides one of the main motivations for using structure analysis in semantic audio tools. In the rest of this section, we review additional methods in the context music segmentation in the symbolic domain, as well as rule-based algorithm and statistical models for audio-based segmentation.

Symbolic and musicological approaches The problem of automatic hierarchical segmentation of music was first introduced in the symbolic domain by Tenney and Polansky [1980], with a view on applying the same principles in the audio domain. They introduce *multidimensional scaling*, originally used in cognitive psychological experiments, in search of applying principles of Gestalt perception to music segmentation. This may seem controversial, knowing that Gestalt psychology was developed for visual objects — Koestler [1964] and Bod [2001] provide a critical view — still, it has become widely accepted, since they seem to explain perception of musical hierarchy on both the auditory and a higher, cognitive level [Moore, 2003]. In Tenney's view music is a hierarchically ordered network of sounds, motives, phrases, passages, sections and so on. However, there are perceptual boundaries of sounds and sound configurations which are different from the segmentation used in music notation.

Tenney's Temporal Gestalt Units introduce the conception of distinct spans of time at several hierarchical levels, which are "internally cohesive and externally segregated" from adjacent segments. The hierarchy of perceptual units, starting from an element, the undivisible unit, can be extended through a piece and even further. An ordered song collection, a concert program, or the process of sequence editing during mastering, represent the highest level of the same hierarchy.

According to Tenney and Polansky [1980], *proximity* (in time) and *similarity* (in some acoustical or perceptual parameter) are "the primary factors in cohesion and segregation involved in music perception". They use these rules, plus the principle of *relativity* (emphasising on relative changes in magnitude), and find multilevel hierarchy using pitch intervals, elementary segment duration, and loudness. The use of timbre is introduced but not utilised. In their simple linear hierarchy, each high level perceptual unit is formed by two or more lower level units. The initiation of higher level units depend on the relative magnitude of change in the features. This is assessed in the context of both statistical averages and terminal elements of lower level units. They pose the important question, how to integrate the perceptual aspects of music into a single measure of change. The answer is in the use of a multidimensional feature space, in which the dimensions are acoustical or perceptual qualities of sound. In such space, the similarity or difference between two objects may be characterised by the *distance* between them. The most important questions arising in their application are: *i)* how to assess the relative significance of each dimension, and *ii)* what distance metric to use. Since this has become a common approach, we shortly review distance measures and related techniques in Section 1.4.2.4. Tenney proposes the l_1 norm, and uses empirical weights to express their subjective significance. For example, pitch intervals might be more important in a piece than loudness intervals. The method was successfully evaluated in obtaining segmentation of classical pieces in agreement with expert segmentations appeared earlier in the musicological literature. There are two important conclusions of the research. The empirical weights have to be adjusted in a piece by piece manner to obtain good agreement with human segmentation.

This indicates that the perceptual dimensions used are not as universal as they might be seen. Second, the segmentation becomes less and less reliable as we move higher in the hierarchy. This is due to the increasing significance of cultural elements (memory and expectation) in human interpretation. This is the boundary, where the points raised by critiques of Gestalt become more and more valid.

Rule-based approaches Temperley [2001] describes a segmentation system using rules and dynamic programming techniques. Based on [Lerdahl and Jackendoff, 1983], a large set of preference rules are implemented to extract the melodic phrase structure. This method stays completely in the symbolic domain. Considering the several critiques of the theory, see e.g. [Balaban and Elhadad, 1999; Cross, 1998; Hofmann-Engl, 2003], and the unknown effects of transcription inaccuracies on a strictly theoretical approach, makes it difficult to use in systems that work with audio. Hofmann-Engl [2003] provides an example of fusing musicological, cognitive and statistical approaches in the context of melodic music similarity. First, the cognitive aspects of basic features: pitch, loudness and duration are established through listening tests. A general transformation theory is developed starting from transposition, retrograde and inversion and applied to the full feature set. A vector space is used to allow algebraic transformations, however a set of empirical constants are included to allow weighting that reflects perceptual significance. For example, a melody interval vector can be weighted using an inverse Gaussian function, in order to reflect that similarity at the *beginning* and the *end* of a phrase is perceptually more important. Although the symbolic methods described so far are not usually considered for direct application in semantic audio tools, they provide a good source of principles and musical considerations that can be utilised in audio analysis.

Audio and model-based approaches Most audio-based segmentation algorithms start from extracting a sequence of low-level features and try to find boundaries within the sequence. The choice of feature depends on the underlying assumption for a segment model, for instance, assuming the consistency of timbre within a segment as in [Levy and Sandler, 2006b]. Although this might be as simple as picking the peaks of a difference function computed from these features, finding boundaries of higher-level segments requires the use of more complex post processing. This may involve clustering similar frames using k-means clustering [Peeters et al., 2002], fitting Gaussian models, or training hidden Markov models on labelled data [Aucouturier et al., 2005] and then compare observed frames against these models, or extract repeated segments from a self-similarity matrix — a square matrix that contains a measure of the similarity between pairs of frames [Foote, 2000] — using a rule-based approach [Goto, 2003]. A more recent, predominantly rule-based approach which uses chroma sequences and a greedy structure finding algorithm is presented in [Mauch et al., 2009]. Semantic segmentation and analysis of music in the context of individual instrument parts of master recordings is an open research question we introduced in [Fazekas and Sandler, 2007a]. We reviewed a large number of segmentation and semantic annotation methods, and applied the algorithm of [Levy and Sandler, 2006b] which uses timbre distribution clustering. The idea of processing multitrack audio for semantic segmentation was explored further in [Hargreaves, 2010], where standard features extracted from individual audio recordings are fused in a single vector before calculating a similarity matrix.

Audio segmentation techniques mentioned so far (as well as the wider literature of the topic) can be divided based on whether an algorithm assumes a *consistent state* corresponding to a segment and attempts to model this state using the features, or attempts to extract repeated segments from the *sequence of features* directly. A combination of these approaches was presented more recently in [Paulus and Klapuri, 2009]. The primary need for developing additional techniques is in the improvement of computational efficiency, more precise segment boundaries and optimisation for our specific content. A multi phase algorithm proposed by Ong and Herrera [2005] addresses some of these issues. Here, MFCCs are used to obtain a rough segmentation first, which is then refined using low-level spectral features, such as spectral centroid, roll-off rate, or spectral flatness. This algorithm avoids training a HMM for feature extraction, however it is using a fine grain similarity matrix with considerable memory and computational cost. This is an important factor considering the usual large number of tracks in a multitrack project. We describe a conceptually similar algorithm in [Fazekas and Sandler, 2007b] for voice segmentation, which compares models of longer segments,thus reduces the size of the similarity matrix.

1.4.2.4 Content-based audio similarity assessment

Audio domain similarity is important in several use cases, for instance, recommendation or similarity based navigation. The three main categories of similarity measurements are:

- **frame-level similarity**, e.g. similarity quantified by a distance between two feature vectors
- **model similarity**, e.g. quantified by the distance between distributions of feature vectors or the similarity of more complex statistical models fitted on a set of features
- **sequence similarity**, e.g. quantified by the distance between sequences of features or corresponding symbols of an alphabet such as the edit distance.

Here, we provide a short overview of the most common metrics we used so far, as well as some related techniques which can also be used to measure music similarity.

The most common distances, the Euclidean and the City-Block metrics can be unified under the Minkowski measure as shown in Equation 1.12.

$$d_{MI}(x, y) = \left(\sum_{i=1}^{D} |x_i - y_i|^p \right)^{\frac{1}{p}} \tag{1.12}$$

In Equation 1.12 the distance of two vectors, $\boldsymbol{x}$ and $\boldsymbol{y}$ are measured in a D dimensional space. The Minkowski metric reduces to the City-Block metric, also called as the Manhattan distance, given that *p=1*. If *p=2*, the equation describes the Euclidean distance. In case of music similarity, the significance and variance of different features may not be the same. For example, we can imagine a piece where dynamics vary in great extent while the pitch range is limited, still, pitch change may provide better discrimination between segments. In a multivariate feature space, the measure or scale of features can be very different, since often distinct physical quantities are combined in a single vector. Normalised measures can be used to account for this difference between the variance of features if they are equivalently significant. The normalised Euclidean distance is given by Equation 1.13, where σ is the standard deviation computed over the sample set of feature vectors.

$$d_{NE}(x, y) = \left(\sum_{i=1}^{D} \frac{(x_i - y_i)^2}{\sigma^2} \right)^{\frac{1}{2}} \tag{1.13}$$

In fact, any distance measure can be normalised by normalising the features over the standard deviation or variance. The Mahalanobis distance is scale invariant, and it also takes into account any correlations in a sample set. In Equation 1.14 Σ is the sample covariance matrix.

$$d_{MAH}(x, y) = \sqrt{(x_i - y_i) \cdot \Sigma^{-1} \cdot (x_i - y_i)^T} \tag{1.14}$$

In measuring audio similarity, it is often a requirement to assess distances independently of a single feature with great significance. As an example, loudness have too much influence on the length of timbre features in the Euclidean space. This effect is annihilated by using the Cosine distance, which measures the angle θ between two vectors.

$$d_{COS}(x, y) = 1 - \cos(\theta) = 1 - \frac{x \cdot y}{\|x\| \|y\|} = 1 - \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} (x_i)^2} \sqrt{\sum_{i=1}^{n} (y_i)^2}} \tag{1.15}$$

There are several ways of defining distance measures in order to optimise our approach to a given problem. The metrics described so far are defined in relation to two points or vectors. However in assessing similarities, it is important to measure differences between sets. In statistics and information theory, it is common to measure the distance between two probability distributions or sample spaces of two random vectors. In computer science, string metrics are used to measure the similarity between two sequences of symbols or strings.

The relation of these to music similarity can easily be shown considering that on a higher hierarchical level, we have to assess the similarity of a set of feature vectors, rather than single points representing audio frames or perceptually not divisible elements.

Probability distributions $\boldsymbol{p}$ and $\boldsymbol{q}$ can be compared using the Kullback-Leibler (KL) divergence as described by Equation 1.16. This divergence was defined in information theory in order to measure the difference in entropy between random variables.

$$KL_{pq} = \int p(x) \cdot \log \frac{p(x)}{q(x)} dx \tag{1.16}$$

With regards to normal distributions, Equation 1.17 describes the distance between Gaussians given by $p(x) = \mathrm{N}(x, \mu_p, \Sigma_p)$ and $q(x) = \mathrm{N}(x, \mu_q, \Sigma_q)$ [Mandel et al., 2005].

$$KL_{pq} = (\mu_p - \mu_q)^T \cdot \Sigma_q^{-1} \cdot (\mu_p - \mu_q) + Trace\left(\Sigma_q^{-1}\Sigma_p\right) + \log \frac{\Sigma_q}{\Sigma_p} - d \tag{1.17}$$

The divergence described by Equation 1.16 is asymmetric. Therefore, it is common to compute this metric with regards to each distribution and take the average to form a symmetric measure. In [Fazekas and Sandler, 2007b] we faced the problem of computing the similarity between sound objects with multi modal distributions. The short audio segments were modelled by a mixture of Gaussians. A GMM is a set of multivariate gaussians which can be used to model complex non-normal probability distributions. This is expressed in Equation 1.18.

$$p(x) = \sum_{i=1}^{M} \alpha_i \cdot \frac{1}{\sqrt{|2\pi\Sigma_i|}} \exp\left(-\frac{1}{2} \cdot (x - \mu_i)^T \cdot \Sigma_i^{-1} \cdot (x - \mu_i)\right) \tag{1.18}$$

There is no clear definition on how to measure the distance between these distributions. Some research suggests that the correct way of computation is using stochastic Monte Carlo sampling [Aucouturier and Pachet, 2002]. In this method, samples are drawn from each distributions and pairwise log-likelihoods are computed using samples of one distribution querying the model of another. The total distance is defined as the normalised sum of likelihoods. However, this method is computationally expensive. Therefore, we use an approximation by measuring the minimum distance between elements of the mixture model. This approach was suggested in [Vemuri and Jian, 2005] and [Goldberger et al., 2003] in the context of image similarity.

$$KL_{pq} = \sum_{i=1}^{n} \alpha_i \cdot \min_j \left[KL(p_i|q_j) + \log \frac{\alpha_i}{\beta_j}\right] \tag{1.19}$$

In Equation 1.19 the distributions $\boldsymbol{p}$ and $\boldsymbol{q}$ are multivariate models such as the one described by Equation 1.18. The weights α and β represent weights of the i-th or j-th element of each mixture model respectively. Each element of $\boldsymbol{p}$ is minimised over the elements of

$\boldsymbol{q}$. Thus the total distance is given by the sum of the smallest distances found. The Earth Mover's Distance (EMD) proposed as the minimum cost of transportation between multiple sources and destinations follows a similar intuition.

A disadvantage of using distributions in measuring the distance between sets of vectors is that the original structure is not preserved, hence cannot be taken into account. Using a crude example, if we play a piece of audio backwards it is unlikely to sound similar. yet, it produces exactly the same distribution of spectral or MFCC vectors. It is becoming more common in MIR to represent audio using symbols, and using alternative distance measures (audio hashing, or query-by-humming applications can be noted). For example, Weinstein and Pedro [2007] proposes fitting a GMM network on audio to represent 'music phones', and uses the edit distance as objective measure in learning the phone transcription. Once musical audio is transcribed into a set of symbols, whatever the symbols represent, their temporal organisation is important. Therefore, we provide a brief overview on string metrics and dynamic programming approaches.

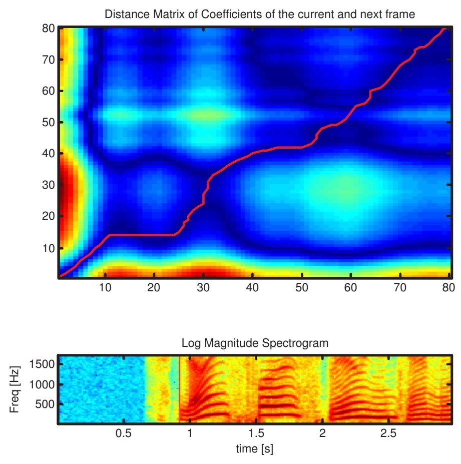


Figure 1.6: Cost function of transforming cepstra of successive transient frames (a bar in the spectrogram indicates the frames compared)

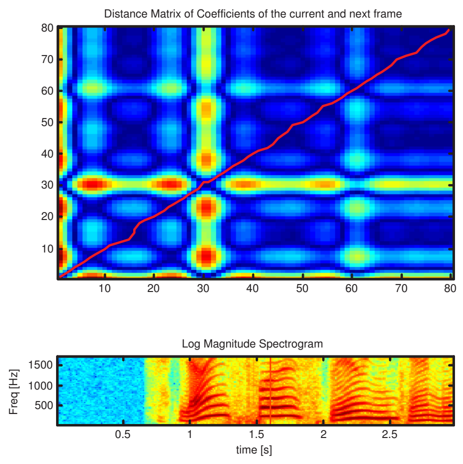


Figure 1.7: Cost function of transforming cepstra of successive steady-state frames (a bar in the spectrogram indicates the frames compared)

The distance between two sequences can be characterised by the number of operations required to transform one sequence into another. This is known as the *edit distance* and most often used in the context of strings. Various interpretations and implementations exists. In the simplest case of equal length binary strings $\boldsymbol{x}$ and $\boldsymbol{y}$, the edit distance is the sum of ones in $\boldsymbol{x}$ xor $\boldsymbol{y}$ (exclusive OR). This is known as the *Hamming distance*. By placing N-length strings into an N dimensional space, it becomes equivalent to the City-Block metric described earlier. A generalisation of this measure is the *Levenshtein distance*, which takes into account the minimum number of insertions, deletions and substitutions required to transform one string into another. The implementation of the edit distance is similar to related dynamic programming techniques.

Dynamic Time Warping (DTW) is a common method used for sequence alignment. It also has an increasing use in music and audio matching [Dixon and Widmer, 2005]. In speech recognition this technique was frequently used to compare word utterances with different speed. It is also closely related to the Viterbi search algorithm used in the context of hidden Markov models [Gold and Morgan, 2000]. DTW can be used to measure the similarity between two sequences by measuring the length of the cost function. Using this property, for instance a similarity measure of cepstral vectors can be implemented, based on the observation that during steady-state, the peaks of the cepstrum have a slow drift in one direction. If significant change happens in the audio content, these peaks collapse or reappear at different locations. Measuring the cost of transforming one vector into another can yield an improvement over standard Euclidean-distance based similarity measures. This is illustrated in Figure 1.6 and Figure 1.7. The curve shows the DTW path for aligning adjacent cepstra during a transient and a steady-state frame, as indicated by the marker in the spectrograms.

Having outlined some of the main techniques for audio information extraction, from the detection of acoustical and perceptual qualities to complex musically meaningful features such as musical structure and similarity, we will continue by outlining some of the applications of these techniques, and mention some problems that guide us in developing ontologies that satisfy information needs of complex semantic audio applications.

1.5 Machine intelligence in music production

In popular music production, the dual role of an engineer can be characterised by the aim of fulfilling some artistic goal on one hand, but also by the use of very specific domain knowledge on the other. This knowledge is used for the adaptation of tools at hand to the specificity of a set of recordings comprising a music release. While artistic goals are defined purely by human factors, such as aesthetic decisions made by the producer or the recording artist, the domain knowledge mentioned above mainly concerns the appropriate use of tools. Capturing this domain knowledge for the benefit of the engineer, and thus building context-adaptive audio processing systems are important use cases of semantic audio. Achieving this goal requires work on two fronts; one is the development of formalised data models to structure and represent necessary information, the other is the adaptation of existing music processing tools for these purposes.

The most prevalent paradigm for the design of today's digital audio workstations (DAW, see [Huber and Runstein, 2005]) is a model of real-world analogue counterparts of recording and signal processing components, such as multi-track tape recorders, mixers and effect units. These systems however have an inherent limitation in that they blindly process the signals they receive, that is, adaptation to contextual information, such as the processed instrument or the tempo of the recording, can only be made manually or by using some inferred characteristics of the signals themselves. This presents further limitations which are both theoretical and practical in nature. Firstly, there is an upper limit in the robustness of high-level features extracted from audio (which may be improved by considering contextual information), secondly, the implementation of adaptive tools within this paradigm leads to highly intermingled components, which often fail to generalise in the context of real-world audio recordings. To avoid these pitfalls, system components need to be interchangeable, their properties need to be fully described, and the information flow and data encoding between them need to be fully formalised.

These design principles distinguish the types of semantic audio applications considered here from related work, for instance, Verfaille's adaptive audio effects [Verfaille et al., 2006b], where signal adaptation serves mainly an artistic goal, or adaptive systems where metadata is used as basis for more general audio transformations [Amatriain et al., 2003]. These systems use metadata without considering the wider context of processing, and a general knowledge representation framework enabling the use of domain knowledge, signal-derived information as well as incidental information available in a host environment. In the following sections we outline some semantic audio applications and relevant problems. We focus on supporting music production workflows with a particular emphasis on audio editing.

1.5.1 Audio engineering workflows

We already noted the serious impact recording and editing had on recent musical developments in Section 1.3.1.1. Recording became an crucial activity in music production, while editing music became a form of art.

> *"The job of an engineer can be best described as an interpreter in a techno-artistic field. The engineer must be able to express the artist's music and the producer's concepts through the medium of recording technology"*
> *— [Huber and Runstein, 2005]*

The roles of the *producer*, the *artist* and the *engineer* are becoming more and more intermingled. In popular music production involving commercial record companies, the producer and the engineer have taken over significant artistic roles. At the same time, with the advent of self-publicity through the internet, the independent artist has become an engineer. Technology is a main driving force behind the change in the way music is created. It is well signified by common views that internet publication may overthrow a well established industry. Naturally, the change in music production has its feedback on technology too; creating new needs and in addition, while the budgets of recording projects shrink, efficiency too is becoming more important in the studio.

The independent artist also poses new challenges when he or she tries to replicate the "professional sound" of recordings created by a staff of engineers working for a record label and an associated professional studio. In this scenario, whether artistic freedom is propelled or limited by certain music production tools depends on the technical skills of the user. Musicians however can not generally be expected to be engineers or technically minded individuals. Therefore tools that capture engineering workflows, allow sharing know-how, and provide workflow support based on common practices may well be desirable elements of future music production systems. The use of semantic audio tools in music production (see Section 1.5.2) can be exemplified in different parts of the workflow of a typical production sequence. This workflow consists of the basic steps outlined below [Huber and Runstein, 2005]:

1. Preparation
2. Recording session
3. Correction techniques (e.g. punching-in, bouncing etc...)
4. Overdubbing
5. Mixdown
6. Mastering and Sequence Editing

A wide range of activities from planning a recording session to physically arranging the equipment such as microphones or amplifiers constitute the process of *preparation*.

During a *recording session*, or over several successive sessions, different versions of the same song are recorded. Depending on engineering practice, all of these takes are retained and assessed later. Alternatively, the best parts are selected right on the spot. In both cases, a tool that is able to identify structural segments of the music and use this for navigation (e.g. find the next chorus in the guitar track) can speed up the selection process. This allows jumping between parts and, for instance, listening to different versions of a verse repeatedly appearing at various times or in separate recording takes. Such a tool may be seen as a particular example of a semantic audio tool.

A destructive process in which an audio section – judged to be incorrect – is re-recorded is known as *punching-in*. Semantically constrained selection, for example, using the hierarchical or metric structure of the music, can help to quickly activate the desired region of a track for recording. *Bouncing* is less frequently used nowadays due to the high number of virtual tracks available in digital audio workstations. It is a method used to create preliminary mixes from a subset of audio tracks before a final mix is produced. In tape editing with limited number of channels, it was crucial to free up tracks that do not require further editing, thus can be mixed together. Bouncing selected segments from a single track can be used to create a single final track from several takes of an instrument. Using a manual process, each section needs to be selected and bounced individually. However, if a hierarchical object based decomposition of the audio is presented to the user, it is enough to drag ”musical objects” – representing an audio section – to a new track.

During *overdubbing*, new instruments, usually short fills or voice parts may be added to a largely completed project. The final assessment before *mixdown* is possibly the process where the use of semantic audio tools can yield the most benefit for the engineer. Although in some cases mixdown is interpreted strictly as setting the right balance and panning, many engineers assess all retained takes before creating the final mix and apply additional effects where needed. Semantic segmentation and navigation tools may be extensively used during this phase (see for instance [Fazekas and Sandler, 2007a]).

Mastering provides the final touches: overall dynamics and equalisation of mixed tracks and relative settings between tracks. The project is often handed over to a specialised mastering studio to do this work. Improved music representation and semantic audio tools may help a mastering engineer in becoming familiar with the material more quickly. It can help the work by subjectively or objectively comparing important parts of each song. For instance, in a final music release it is often required to adjust levels the ensure equal (or nearly equal) perceived loudness across tracks, which is often verified by randomly listening to or monitoring different structural segments. This leads us to the process of *sequence editing*, assembling the final order of songs on an album. Musical key, tempo, or best matching ‘mood’ or ‘feel’ of the song sequence are among the most important factors in this judgement. We can assume, that techniques for playlist generation may provide an initial ‘good guess’ on the desired final sequence.

1.5.2 Motivations

Throughout the workflow described in the previous section, we can enhance the interaction between the audio engineer and the machine using intelligent semantic audio tools. In this work, we pay special attention to developing and examining use cases that aid *correction techniques*, *low-level editing* and *mastering*. The most common representation of music in audio editors is a waveform, a pressure-level graph which is not an easy reading material for the untrained eyes it, let alone finding a section in music using this sole cue. Little research has attempted to improve on this by finding a better alternative. However, we should not argue for the replacement of the waveform in semantic audio applications. Instead, we argue for additional visual cues to enhance user interaction. The following are just a few examples: i) labelling audio tracks and displaying performance related information; ii) using colours to reflect the semantic structure, or iii) following (and reversing) the ecological approach of Gaver [1993] who first proposed ‘well thought-out’ *auditory icons* for *human-computer interfaces* (HCI), we can develop a set of visual elements (objects) that associate or resemble an underlaying sound in some feature. These objects can be used for easy ‘drag and drop’ editing or clicked to expand into a waveform or reveal additional data contained in the object.

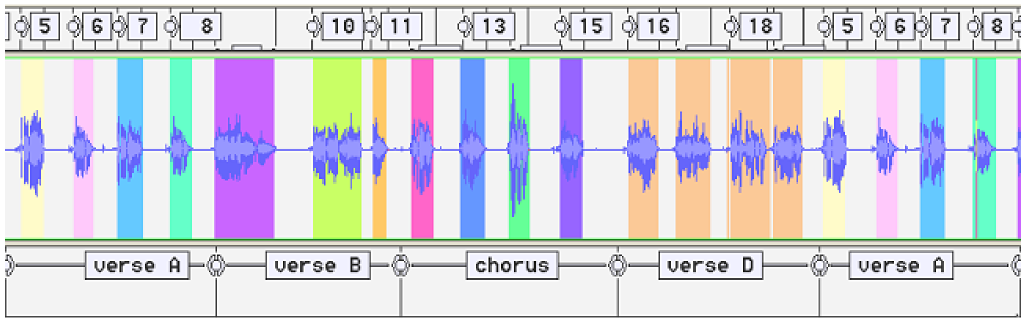


Figure 1.8: Visualising structural segmentation using colours.
The matching sequences at the beginning and the end indicate two occurrences of the same verse. The colours in this diagram were obtained by mapping audio similarity, derived using simple features such as segment duration and amplitude envelope, to a finite set of colours in a discrete colour space.

The latter idea suggests an *object-orientated* (OO) design, which we may see as a first step towards more complex knowledge-based representations. In an object based representation, sounds are decomposed and organised into a hierarchy of inter-related objects, an approach proposed by Pope [1996] in the context of music description languages. Although traces of these ideas appear in modern audio editors, for example in Logic Audio, a little pictogram of an instrument can be assigned to each track representing the object which holds data for that particular track, see [Duignan, 2008], a formalised hierarchical system does not exist. Even object-based environments like Max/MSP are often criticised for the lack of visual formalism, well-defined hierarchy and for an ad-hoc design [Balaban, 1999]. The thorough examination of music production software presented in [Duignan, 2008] seems to indicate that this has not changed significantly to date. High level music description and representation is far from a fully solved problem. More general music representation needs and issues are discussed for instance in [Dannenberg, 1993]. The majority of problems detailed in the literature however remain unsolved to date.

Returning to audio editing, in the author's opinion, audio editors available today are conceptually not more than waveform displays with an increasing collection of sound processing and manipulation tools. This statement seems to be valid, even if we consider the most advanced multi-track environments, where the expression *advanced* may only stand for the number and (with some good-will) quality of available tools. We provide a simple example. In state of the art audio editors, it is often undefined what sort of data may be processed by a particular tool or plugin considering the *particularities of the signal* or the *musical context.* If the user is unaware of certain restrictions, an error message appears, or in the worst case, the application crashes. A knowledge-based system could try to choose the right tool, or indicate correct usage by employing some appropriate visual formalism.

In the light of these problems, we propose a *context-dependent* approach for developing semantic audio tools, which requires the tools to have definite *knowledge* about the content and its context. This knowledge can be used to constrain how the tools work on certain kinds of audio based on structure, similarity or another specified feature. The knowledge can also be shared with plugins and external applications. In a broader context, metadata can also be saved and used as basis for a semantic search interface, for instance in a studio database. These ideas call for more examples in studio use. A *de-esser* for instance is a common tool used for correcting distorted syllables in singing voice. The problem usually affects fricative consonants, plosives and affricates (s,z,p,t,ch[ts]). A de-esser is generally implemented as a band-stop filter. Although it is possible to set the centre frequency and bandwidth of the filter as well as an energy threshold to limit the area of intervention, it is hard to avoid the filtering to be applied where not needed thus distorting the content. Manual selection of fragments of the waveform is often the only solution to this problem. However, if we are able to group applicable sounds based on similarity, it would be enough to select one example and apply the filter to all similar sounds, or all sounds occurring in the same context. A second example of the possible use of content or context-based metadata is a search scenario. Studios often keep a large database of multi-track recordings. During a recording session, an audio example may be the best way to communicate a conceptual or artistic idea. Engineers often try to find these examples from earlier recordings, which can be made easier if metadata collected about the recording process and audio content is stored and later reused. For instance, we could find a bass phrase with a set of given parameters; duration, frequency range or musical note range.

The development of intelligent semantic audio systems offering context-dependent functionality requires an extensible information management framework with a supporting ontology. The theoretical and methodological basis for knowledge representation in music software development is described in [Balaban and Elhadad, 1999] and [Balaban, 1999]. Our recommendation for developing semantic audio tools is largely based on this approach. For the purposes of sensing, a semantic audio tool may rely on content analysis using MIR techniques. We can obtain contextual information [Turney, 1996] by taking auxiliary data — for instance, the name of the instrument associated with a track, or the tempo of the click-track associated with a recording project — entered using the human interface of an audio editor into account. In text based information retrieval, the importance of this sort of contextual information is studied more deeply [Fazekas, 2009], and an elaborate view is presented for example in [Robertson, 2008].

Knowing the primary sources of information a system may rely on, supporting the audio engineering workflow in semantic audio tools may be achieved by using by a *set of processes* performing content analysis and data collection in the background, and a set of *tools and visual enhancements* which the user may invoke while carrying out editing tasks. A common requirement across these systems is the need for managing structured, or more precisely, semi-structured information. The term "semi-structured" will be used here to refer to information that is open and can not be strictly constrained by a schema, and which is often but not necessarily self-describing [Buneman, 1997]. The Web is a good example of this kind of information source.

Due to the need for approaching the complexity of human-level interpretation of music signals, almost no tool that is meaningful to an end-user may be built relying on a single source of information and simple techniques, without the use of formalised and complex information management methodology. It is easy to see that human information processing and decision making is based on a complex set of *a priori knowledge*, *empirical evidence* and *contextual information.* If we were to build systems that support human decisions, we have to make an attempt to collect, manage and build on multiple of sources of information. The use of a semi-structured data model seems to be of key importance to represent this kind of information in an open-ended way.

The need for incorporating multiple information sources has already been recognised by MIR researchers who have been using editorial, cultural and social information, in conjunction with content-based features of audio. These data, however, remain focussed on relatively simple editorial information about artists or songs, data obtained from collaborative filtering — the analysis of user behaviour — or features resulting from the analysis of commercially released audio mixtures. These features have certain limitations, due to the limited robustness of feature extraction from rich polyphonic mixtures, or due to being collected after the recording process.

We argue that an invaluable source of information exists pertaining to the composition context, history, production and pre-release master recordings of music. Production data may include microphone arrangements and characteristics, configuration, connection, decomposition and operation of audio signal processing devices, such as mixers and effect units, projects and edit decisions in post production workstations, and features extracted from individual tracks of pre-release master recordings. One reason for these data have been neglected so far is not necessarily unawareness within the community, or the limited use of this type of information, but the fact that production data is not easily retained and therefore it is not available in sufficient quantities to be effectively used in MIR applications. We argue that the lack of comprehensive open standards and methodologies for collecting production data is a central issue, and the reason why these data are simply lost.

Gathering information about recording can also contribute to the creative process itself, besides enabling the use of a previously unavailable data source, and thus building better models for music information retrieval. Music making is an increasingly social activity. We believe that the Semantic Web could become a platform for sharing not just music, but ideas between artists and engineers. To facilitate this process, the ontologies detailed in Chapter 4, and our particular contribution, the Studio Ontology discussed in Section 4.2, can be utilised to denote information about music production, and propagate it through the recording workflow. Data expressed using these ontologies enable answering queries such as: *How was this song produced? What effects and parameters were used to achieve that particular sound of the guitar? How was the microphone array configured when recording this orchestra?*

1.5.3 Studio specific problems

The application of semantic audio analysis and metadata in the studio environment is among the main motivations for our work. Primary reasons for collecting metadata in music production include the effective organisation of musical assets, such as sounds in a sample library, or previously recorded takes of multitrack master recordings. Although various applications in creative music production such as visualisation, or semantic navigation, and search within recording projects are frequently mentioned as motivating examples for the development of semantic audio and music information retrieval technologies; very few research addresses specific issues arising in music production environments. Using state of the art MIR therefore does not immediately enable the applications outlined in the previous section. While the use of more complex machine learning and statistical models in addition to signal processing often improves performance, the models are usually trained on cleaned and post-processed audio, or well mastered polyphonic mixtures, thus their application in the studio environment presents difficulties. We can outline the most common artefacts that may be encountered in real-world recordings as follows:

- amplitude beating or vibrato as a result of specific playing techniques which affects, for instance, most string and wind instruments,
- frequency beating and dissonance due to accidentally played notes or polyphony of the recording,
- effects, reflections, reverberation due to room characteristics or effects applied for aesthetic considerations, e.g. recording in a highly reverberant room deliberately,
- intentional distortion and other artistic effects,
- phase problems due to specific recording techniques, such as the use of multiple microphones and the mixing of signals thereof,
- noise, including ambient noise, as well as noise from amplifiers, mains, or cables.
- cross-talk problems resulting form the use of long instrument cables or mixing desks with poor channel isolation,
- excessive dynamic range of unprocessed recordings, (noting that final *master* recordings are typically compressed in several stages before and after mixing).

Instrument track	Length	Labelled onsets
bongo drums	2m40s	888
acoustic guitar	4m04s	541
drum kit mix	4m04s	914
fuzz guitar	4m09s	935
synth bass	4m04s	766
tambourine	4m00s	289

Table 1.2: Different instrument recordings taken from multitrack master recordings, and the number of onset labels used as ground-truth.

Specific recording techniques — for instance, the common practice to record an electric guitar using two or more microphones via different amplifiers to take advantage of their specific response characteristics, or the use of multiple microphones placed at different locations — can cause artefacts in studio recordings such as phasing effects due to different delay times or accidental inversions, and comb filtering when the signals are mixed.

In order to investigate these effects in the context of different instrument takes of multitrack master recordings, we examined one particular problem, the effect of reverberation on onset detection tasks [Wilmering et al., 2010]. In this experiment, several individual instrument takes shown in Table 1.2 were labelled manually, and compared with four onset detection techniques, namely, high-frequency content, spectral difference, complex-domain, and broadband energy rise [Bello et al., 2005]. To synthesise the effect of a reverberant space, artificial reverberation was applied to each recording with increasing gain. Two standard configurations were used to simulate a medium and a large room. This resulted in 6000 reverberated audio files per room setting. An implementation of the Schroeder-Moorer reverb model, Freeverb [Smits, 2009] was used in all experiments based on four Schroeder serial all-pass filters and eight parallel Scroeder-Moorer comb-filters [Schroeder, 1961, 1962; Moorer, 1979]. The best onset detector parameters were determined using the original *dry* signals by maximising the F_1 score which combines precision and recall with equal weight:

$$F_1 = \frac{2PR}{P+R}, \tag{1.20}$$

where P is the *precision* given by:

$$P = \frac{O_{TP}}{O_{TP}+O_{FP}}, \tag{1.21}$$

and R is the *recall* given by:

$$R = \frac{O_{TP}}{O_{TP}+O_{FN}}, \tag{1.22}$$

where the number of true-positive onsets (O_{TP}) were counted within a tolerance time window of $50ms$. Undetected onsets were considered false-negatives (O_{FN}), while onsets outside the window of ground-truth onsets, as well as multiple onsets within the same window were considered false-positives (O_{FP}).

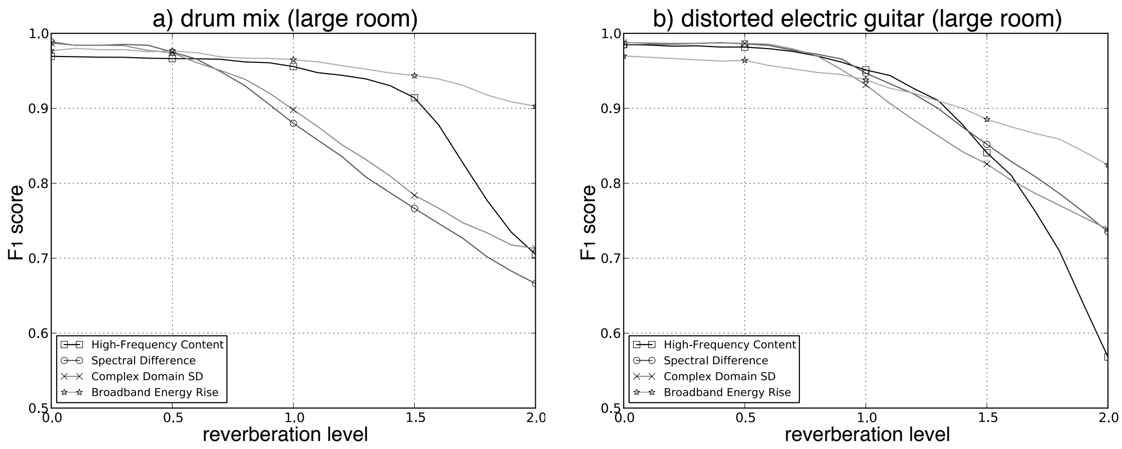


Figure 1.9: Onset detection in reverberant recordings showing degradation of performance in a) drum mix and b) electric guitar recording with increasing reverberation levels.

Keeping the onset detector parameters fixed, we applied the algorithms to the data set of reverberated material. The results are exemplified by Figure 1.9. Our finding supports the intuition that onset detection performance generally decreases with increased reverberation level, and the loss in performance is more pronounced with larger room settings. These results, together with our earlier observations, indicate a strong need for

- using feature extractors before audio effects are applied in the studio, and
- collecting as much information as possible about recording conditions

in order to improve on the current state of the art of semantic audio analysis and MIR.

Our investigation also suggests that published results are often better than what we may achieve when the same algorithms are applied to 'raw' master recordings. This difference can be attributed to the fact that evaluation databases are typically free from the artefacts of 'unprocessed' master recordings listed above. The problems however, if ever mentioned, are considered pathological or simply put aside as engineering as opposed to scientific problems. Since many of these problems are related to recording conditions, we see the solution in the adaptation of analysis techniques and parameters to these conditions. This requires a framework for describing the recording process itself using logical models, and a high-level layer in the analyses which is able to 'supervise' underlying signal processing and modelling techniques. We think of the information necessary to achieve this as the environmental context of recording.

1.6 Research scope and methodology

In previous sections, we outlined several existing and potential applications of semantic audio analysis. An initial investigation of prevailing techniques revealed some important problems and difficulties in applying state of the art audio analysis methodologies directly in semantic audio applications. Therefore, in order to achieve the full potential of semantic audio, we argue for advancements in information management, knowledge representation, and formalised software systems, and consider these advancements vital in applications like music information retrieval and creative music production.

An underlying hypothesis for this work is that collecting data in the recording studio can improve semantic audio analysis, and enhance the applicability of audio analysis in general. This may be achieved by enabling tool adaptation to the context of recording, or using high-level semantics in complex systems, given some contextual information collected about the recording and audio engineering processes. This hypothesis, however, may seem weak in the eyes of the falsificationist, who require all scientific theories to be directly refutable.

The idea of using falsifiability to delineate scientific hypotheses from non-scientific premises was put forward by Karl Popper in his seminal work published in 1934 [Popper, 1959]. While this theory is often regarded as fundamental to the scientific method, it has been criticised [Duhem, 1954; Quine, 1951; Kuhn, 1962; Lakatos, 1970; Mayo, 1996], and all but the most sophisticated forms have been refuted by their inability to explain the growth of science on historical grounds, see [Lakatos, 1970], or [Chalmers, 1999] for a summary. For one, falsification is paradoxical, because it requires the observation statements that may lead to the falsification of a theory to be also fallible. Then, in case of a negative result, we have no way to know whether the theory is falsified, or the observation criteria are wrong. The assumption that hypotheses may be tested in isolation is also problematic (see [Duhem, 1954; Quine, 1951]), due to the complexity of realistic test situations that always involve auxiliary premises and initial conditions. In case of a negative observation, we can only conclude that at least one of the premises are false, but we do not know which.

A characteristic problem within the scope of this work is related to the evaluation datasets that are typically drawn from carefully engineered audio recordings, which are at least free from the artefacts one may find in unmastered studio stems. Algorithms tested on these datasets will therefore perform better than what we may expect in most real-world situations (see Section 1.5.3). We can then hypothesise that information about the recording process and conditions are useful. The complexity involved in utilising this information, and the knowledge that enables one to act upon it requires sophisticated management and formal representation of information and knowledge, especially if one wishes to go beyond ad-hoc or case-specific solutions. The paradigm of competing research programmes offered by Lakatos [1970] as a solution the problems of falsifiability and hypothesis testing is applicable in our case. While it has to be noted that finding conclusive evidence is beyond the scope of this (or any single) work, a research programme that includes audio engineering knowledge in its set of hypotheses has the potential to replace research programmes that do not.

When the ontology comprising this knowledge is seen as a hypothesis in itself, we face the problem of failing under Popper's criterion of demarcation again. While the ontology may be validated or verified and proved wrong under a set of criteria, it is not directly falsifiable or testable in isolation. We can test if an ontology is syntactically correct, semantically consistent and logically sound (see Section 6.2). Failing under any of these criteria may incur necessary amendments, but will not refute the underlying hypotheses and conceptualisations, of which many are plausible and may all be correct. A possible way to evaluate a scientific or engineering ontology then is by appeal to the criteria proposed in [Lakatos, 1970] as alternatives to falsification, that is, *novelty and competition*, where novelty is what discriminates between two lines of tradition [Hattaingadi, 1987]. We can examine *i)* how well does an ontology fit new facts, observations or requirements that were unknown or that were deemed outside of its scope when it was first proposed, and *ii)* evaluate an ontology by competition, i.e., does one theory complex underlying the ontology fits better to new facts or requirements than the conceptualisation in some other ontology? The ontology evaluation presented in Chapter 6 follows these philosophical guidelines.

Since data collection in the studio is a difficult task in itself, we focus on information management, knowledge representation and software requirements in this task. We argue that the complexity of this task is similar to providing machine-processable representations of general Web content, which is the reason why we adapt Semantic Web technologies, and develop a Web ontology for describing studio production. We aim to show that this ontology is able to capture information about music production. In addition, we observe that knowledge representation requirements and thus ontologies are dynamically evolving artefacts. Therefore, as a further aim of this work, we design software systems which are able to accommodate changing ontologies, and examine how these systems can be applied in various tasks. The ontologies and software frameworks form building blocks of an intelligent, content-aware music processing system and semantic audio tools which are able to use several interlinked information sources.

The utilities of these frameworks are evaluated in *i)* a Web-based environment, which can be used for analysing audio recordings and express the results using formal ontologies (see Section 5.3), and a desktop environment and software library for facilitating manual data collection in the studio (see Section 5.1). Building these frameworks doesn't fit well with the observational methods commonly used in laboratory experiments. The work can not easily be divided into self-contained hypotheses, rather, it is presented as a collection of interdependent building blocks which may be designed and developed *iteratively*. This implies that our current research method is closer to *design based techniques* of information science [Hoadley, 2003], an iterative process with the gradual improvement of components.

Chapter 2

Information Management and Knowledge Representation for Audio Applications

In the previous chapter, several motivating examples of intelligent semantic audio applications were discussed alongside prevailing methods in semantic audio analysis, and some challenges in their use in complex environments such as music production. In order to meet these challenges, and for future improvements of semantic audio tools, we argued for improved information management, and highlighted the importance of collecting data about the recording process.

In this chapter, we examine more specific problems, review the information management technologies we use, and outline how Semantic Web technologies can satisfy information management and knowledge representation needs in semantic audio. First however, we attempt to clarify the meaning of fundamental concepts often taken for granted in this field.

2.1 Data, Metadata, Information and Knowledge

In order to be able to talk about information management and knowledge representation problems, it is useful to define the meaning of Data, Metadata, Information and Knowledge. The taxonomy of knowledge [Ackoff, 1989] can be used to explain the relationships between these concepts. This taxonomy also includes *Signal* as a carrier of data but excludes *Metadata*. Similar definitions are given in [Aamodta and Nygardb, 1995] and [Raimond, 2008].

- **Data** is a set of symbols with no definite meaning. Although, it is easy to see that without a frame of reference, a single datum removed from its context means nothing, one may argue that structure can be learned from a set of symbols which gives rise to meaning. An important question to consider here is that of "sameness of reference" studied by Quine [1995] in the context of denotation and truth and semantic agreement. Without a shared conceptual framework and common references, neither data nor emergent pattens and relations carry useful information. An array of values resulting for instance from the short-time Fourier transform of an audio file has no definite meaning in and of itself without knowing how its columns relate to temporal segments of the audio data, and what its rows represent. Furthermore, the precise interpretation of raw data is difficult without knowing certain signal processing parameters underlying the process, for instance, the type of window function used for short time segmentation.

- **Metadata** is commonly explained as *data about data*. Its meaning however carries ambiguities, inconsistencies, and variation. For one, it may refer to a datum related to another datum such as the musical key of a piece. However, from this, we do not necessarily know the exact relation between the two items. Metadata may also refer to data structures or containers which describe data records, that is, the relations between data items. Metadata can also be associated with other metadata, leading to relational or hierarchical structures. This is illustrated by Figure 2.1.

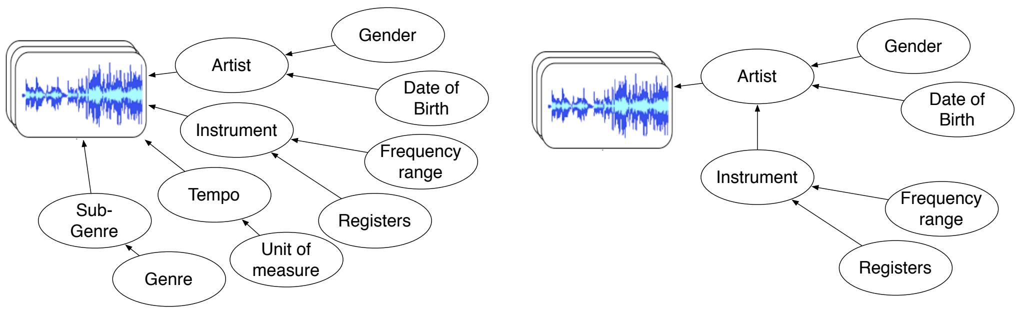


Figure 2.1: Illustrations of different possible arrangements of audio related metadata

- **Information** is data structured within a certain context such that the relations between datum are formalised. For instance, if we know how the columns of the array of our example above is related to an audio signal, and what each row represents, then our data constitutes useful information.

- **Knowledge**, in a pragmatic definition, is information structured in such ways that logical reasoning is permitted. In epistemology, noting that this view may be challenged [Steup, 2010], knowledge is commonly defined as *justified true belief*. This requires that *i)* a proposition **p** is true, *ii)* the subject **s** believes that **p** is true, and *iii)* **s** is justified in believing that p is true. This implies that knowledge involves learning or some other form of justification.

- **Information Management** is the process of collecting, organising, maintaining, storing and distributing of information. For example, the process of collecting metadata about user interaction or audio feature extraction, and maintaining its consistency in a database is information management.

- **Knowledge Representation** is concerned with methods of structuring information such that automatic interpretation or formal reasoning becomes possible. The Dublin Core metadata standard and the Music Ontology are examples of knowledge representations.

The rest of this chapter discusses information management in semantic audio applications. First, we discuss a number of different options for managing metadata in audio applications through a practical example. A brief summary of the logical foundations of information management and knowledge representation is discussed next. These foundations become crucial when we discuss one of the main contributions of this work, the Studio Ontology, in Chapter 4. Finally, the application and utilities of Semantic Web technologies for the above purposes is introduced.

2.2 Information Management in Semantic Audio applications

Semantic audio and its applications are in the confluence of a multitude of technologies. The signal processing techniques outlined in the previous chapter enable the extraction of characteristic features from audio. These features can be used in intelligent algorithms to associate raw feature data with meaningful representations of audio content such as a sequence of chord labels, or elements of musical structure. However, in order to interact with semantic audio, information management tools which allow for the effective organisation of these data, and knowledge representation tools which provide a frame of reference and enable the representation of abstract concepts in a machine-processable way are vital. These technologies also facilitate automatic data aggregation and high-level inferences. In this work, we define semantic audio applications as tools where these methods are working together in a harmonised manner.

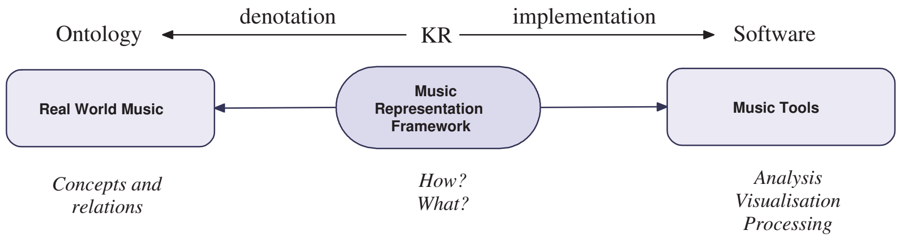


Figure 2.2: Balaban's knowledge representation (KR) framework for music software development

These tools may be Web-based or mobile, and aim at facilitating communication between providers and consumers of musical content, but they may also be desktop or embedded in hardware, and may focus on assisting the artist or the engineer in music production. An intelligent search engine that is able to answer complex semantic queries by dynamically aggregating and analysing large amounts of musical data on the Web, or an intelligent audio editor that uses content analysis as well as the information available in the desktop environment to help the work of an engineer are prominent examples.

The newly recognised role of information management and knowledge representation in the development of semantic audio applications can be highlighted using a system that exemplifies intelligent audio editing. This problem is particularly interesting, since it enables looking at the creation of semantic audio tools from many different angles. Here however, we are mainly concerned with the caveats of information management, knowledge representation and software engineering.

Balaban and Elhadad [1999] outline a knowledge representation framework and requirements for designing music processing tools from a general software engineering point of view. An updated version of this framework is depicted in Figure 2.2. According to Balaban and Elhadad, the role of knowledge representation is to describe real world requirements, user demands and the available information. An *ontology* describes problem entities, operations, relations and structures. In the context of semantic audio tools, the entities may be sounds or sound objects, while relations and structures are described by their hierarchy. Operations describe the available tools and their context. Using the examples from Section 1.5.2, a *navigation tool* for instance uses the hierarchical music structure, a *de-esser plugin* operates on unvoiced high-frequency sounds of certain characteristics, and finally musical phrases of a track that sound similar can easily be interchanged by an intelligent *editing tool* if represented as high-level objects corresponding to musical or perceptual boundaries within the signal. A discussion on information management and knowledge representation requirements of these tools can be facilitated by a model for building semantic audio tools. In the next section, we present such a model focussing on applications in music production.

2.2.1 A Semantic Audio Tool

A system for integrating components that allow the implementation of the ideas mentioned so far in this work may be modelled as shown in Figure 2.3. This model has three *analysis layers* corresponding to audio feature extractors, three *information layers* corresponding to ontologies for describing tools and the results of audio analysis, and three *application layers* corresponding to tools that can be built using this information and their descriptions. In the following, we outline the role of the three layers and the components that may be utilised in the model.

Analysis Layers: Audio feature extraction. Some of the signal processing components required for extracting information from audio content are well researched and may be adapted. Basic feature extraction techniques standardised in speech recognition are successfully applied to music, for example, pitch determination and timbre modelling techniques. We have to note however that they are not well adapted to work with 'unmastered' raw recordings in a multitrack environment, which may require the use of contextual information, structured and uniform knowledge representation and interpretation. High and mid-level feature extraction is in the focus of MIR research. Semantic segmentation of polyphonic audio is described for instance in [Mauch et al., 2009; Levy and Sandler, 2006b], or [Ong and Herrera, 2005]. High-level segmentation of audio recordings played by a single instrument and the analysis of master recordings however were not considered by previous research. In the symbolic domain, the problem was first examined in [Tenney and Polansky, 1980]. Although utilising successful symbolic approaches by obtaining symbolic representation first is well worth considering, obtaining clear symbolic music representation from audio is generally not a solved problem. It may require a high-level cognitive model [Klapuri, 2004] (e.g. interpreting a glide or vibrato as a single note, or separating tones with smooth changes or legato). Expressive performance models and their visualisation were discussed for instance in [Widmer and Goebl, 2004] and [Dixon et al., 2002]. The identification of vibrato, glide, legato or glissando are also required to obtain a detailed symbolic representation. Signal processing techniques applicable to these problems are described for instance in [Wen and Sandler, 2008].

Information Layers: Ontologies as frames of reference for describing the domain. The Music Ontology and its extensions can be used to provide a frame of reference and the basis for the information management layer considered here. This ontology can be used to represent for instance a musical piece and its relation to corresponding signals. Its basic components allow for associating domain entities with time-based events. This is crucial in representing audio features, and serve as the basis for the Audio Features Ontology[1], which covers basic features in a non-exhaustive way. It does not provide however for the description of interdependency between features, which we consider important for efficient feature extraction (e.g. to avoid processing overlaps and enable the reuse of low-level features stored in a database). The Studio Ontology discussed in Chapter 4 provides extensions for the Music Ontology for describing studio concepts. This includes the Multitrack[2] ontology to associate information with tracks in a multitrack audio editor. While it is not immediately apparent from the figure, ontologies for describing audio analysis algorithms, ideally, including even their low-level DSP components, and ontologies that allow for describing audio processing tools are equally important in building intelligent music processing environments. Currently there are no ontologies describing specific signal models or performance related data. These shall be developed in the future as the need arises.

[1]http://purl.org/ontology/af/
[2]http://purl.org/ontology/studio/multitrack/

Application Layers: Visualisation and interaction. Examples of visualisation, navigation and context-dependent tools were described in Section 1.5.2 in more detail. The key to their development is a well-defined and structured representation of the music and its various representations a semantic audio tool may operate on. The ontological needs of describing applications include the ability to describe dependencies (i.e. to create a knowledge base), that is, the information needed as input for the successful use of a specific tool. This information can be used to retrieve data, invoke feature extractors if needed, or ask for user interaction. A relevant example is the LV2 plugin ontology[3], which enables the description of input/output requirements of audio processing plugins.

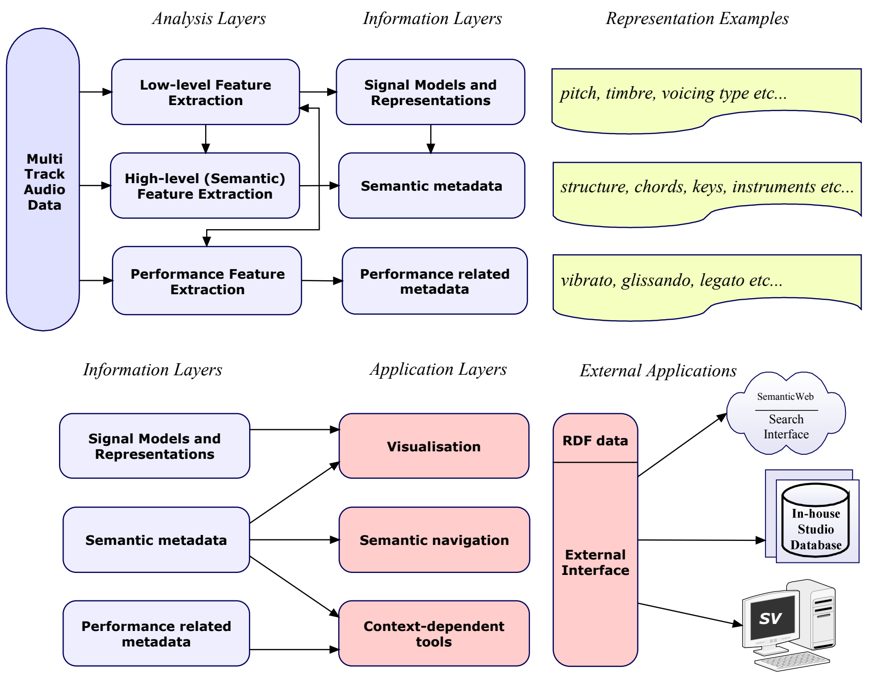

Analysis Layers
Information Layers
Representation Examples
Multi Track Audio Data
Low-level Feature Extraction
High-level (Semantic) Feature Extraction
Performance Feature Extraction
Signal Models and Representations
Semantic metadata
Performance related metadata
pitch, timbre, voicing type etc...
structure, chords, keys, instruments etc...
vibrato, glissando, legato etc...
Information Layers
Application Layers
External Applications
Signal Models and Representations
Semantic metadata
Performance related metadata
Visualisation
Semantic navigation
Context-dependent tools
RDF data
External Interface
SemanticWeb
Search Interface
In-house Studio Database
SV

Figure 2.3: Knowledge representation model for intelligent audio editing systems

2.2.2 Information Management and Knowledge Representation

Loss of information in the media production workflow chain is a major issue when it comes to collecting, managing and repurposing metadata about various aspects of the music production process itself, or the media under consideration. This, as well as the problems and requirements mentioned previously indicate the need for advanced information management solutions and formal knowledge representation in semantic audio applications. In the following, we examine some options available for these purposes.

The first and obvious choice for structuring information in audio applications is to treat

[3]http://lv2plug.in/spec/

all data describing tools, stakeholders and audio files as metadata. Then, we can use a combination of existing metadata standards to fulfil our information management requirements. However, there are several problems with this approach. While there are numerous musical applications of metadata discussed in the literature, see for example [Gomez et al., 2003], [Verfaille et al., 2006b], [Pampalk et al., 2004a], [Wright et al., 1999], and the author's own work detailed in [Fazekas and Sandler, 2007a], these examples focus on specific case-based implementations. However, no generic formats for metadata, and more generally, information management practices emerge from previous research and industrial solutions. Disjoint purposes for covering overlapping musical domains produce a abundance of disharmonious standards and methods (see [Smith and Schirling, 2006] or Section 3.2.1). This seriously impairs the exploitation of metadata in developing ubiquitous creative applications. A part of this problem can be identified in the use of non-normative development and publishing techniques, rather than flaws in design. We recognise that common approaches to overcome this, including standardising syntax through the use of XML or creating a reference library, such as those accompanying Sound Description Interchange Format (SDIF) [Wright et al., 1999] or the Advanced Authoring Format[4] (AAF) do not provide sufficient ground for modularity and interoperability. The heart of the problem lies in the assumption that musical information can be expressed using strictly structured schemata and static data sets. In our research, we challenge this general assumption and hypothesise that musical information is better modelled as dynamic semi-structured data.

Perhaps the most important issue arising from common metadata management practices is related to the association of *first class* or *primary objects* and *metadata tags*. Suppose we want to associate an audio file with a recording artist. This can be done by extending the data structure representing audio files with an additional string attribute where the artist's name can be stored. Then, we want to describe the instrument played by the artist, and use this information to infer some properties of the signal. This can be used for example to support the adaptation of audio processing algorithms to a particular instrument. Since the instrument in this case is more closely related to the artist rather than the audio file, it would be illogical to link the instrument to the file directly. Moreover, there may be other attributes associated with the artist, for example gender, (which might be useful to know about in voice processing applications). A solution to this problem is to represent the artist as a first class object with its own set of attributes and relationships. Similar problems arise when we attempt to describe content-derived features of audio, or audio effects and their parameters as applied to a recording. Although the solution remains the same, the further we go in the direction, and start introducing more interlinked first class objects to represent audio signals and their features, audio effects, instruments or artists, the closer we get to a data structure best described as a *semantic graph* or a *Semantic Network*, a graph structure for representing

[4] `http://aaf.sourceforge.net/`

information and knowledge in patterns of interconnected nodes and arcs [Sowa and Borgida, 1991].

There are several ways we can conceptualise, encode and store the types of information described previously. Parts of this information has a corresponding relational data model [Codd, 1970]. A relational schema is defined in terms of relations and dependencies among them, which can be formalised using *First Order Logic* (FOL) formulæ. Relational schema is often described using the Entity-Relationship (ER) model and corresponding diagram. While this is common in relational database design, just like FOL, the ER model is more expressive than what can be described by a set of relational tables. Fundamental to both the relational, and the more recent Semantic Web data models described in Section 2.4, first order logic allows us to precisely define the semantics of data structures, database schemata or ontologies. A brief overview of this logic and its foundations is provided in Section 2.3.

Although the relational data model has solid mathematical foundations in first order logic, its *implementations* in the form of relational tables have several disadvantages. We may mention the loss of explicit identity of first class objects such as an artist or an instrument from our example in Section 2.2. Objects can only be represented indirectly, characterised by a set of attributes in a table. Similarly, the identities of relationships have no explicit representation in the database. They can only be discovered using queries based on some external knowledge about the domain—that is, the model has *hidden semantics* [Chen et al., 2005]. Explicit taxonomical relationships (known as *is-a* relationships) have no corresponding representation in the relational model, although they can be expressed in FOL. Moreover, certain types of heterogeneous, semi-structured data, spatial data or temporal data cannot be easily accommodated in the model. Since these types of information are often present in a multimedia environment, this requirement points us to seeking other methods such as object-based models or models that are closer to logic based knowledge representation. The relationship between these and the relational model is discussed for instance in [Motik et al., 2007].

Object-based data models have become popular due to their direct correspondence with modern object-orientated programming languages, and the fact that additional mapping between the object graph of an application and a database is not required. Many XML-based information management solutions are also based on this idea. However, object models in general have no sound theoretical foundations, and there is lack of support for efficient query evaluation or logical reasoning. An alternative to object-based or purely relational data models is provided by the Resource Description Framework and more expressive Description Logic (DL) systems, and corresponding Semantic Web ontology languages [Horrocks et al., 2003]. These systems are not only concerned with simple relations between terms or data structures, but with the representation of knowledge about a domain using a logical formalism.

Recognising the similarities and problems shared between semantic audio tools and the

Web in need for representing diverse, virtually unbounded information; we turn to Semantic Web technologies to fulfil these requirements. In particular, we use RDF, a fundamental data model, and Semantic Web ontologies such as the Music Ontology as the basis for our domain model. In the following sections, we review the logical foundations, as well as the logics and languages that are used to describe ontologies in the rest of this thesis.

2.3 Logical Foundations

Providing a precise description of data models and ontologies is a common requirement in information management and knowledge representation. Logic languages allow for these descriptions to be denoted and manipulated. In the following sections we outline the logical foundations of these fields. We start by reviewing some important concepts in propositional logic and first order logic. The latter is particularly interesting, since it is often used for formalising ontologies. This language will be used for describing ontologies throughout this thesis.

Essentially, a logic is a language for the formalisation or axiomatisation of a domain of information. It allows for a representation of a world using a formal syntax and associated semantics (meaning), such that some new information (conclusions) can be drawn from this representation. There are many information management solutions relying on data models which have no sound theoretical foundations. For this reason, query evaluation in these systems may be less efficient, while automated reasoning is not possible. An important advantage of using Semantic Web technologies is that the RDF data model allows for expressing and using vocabularies in knowledge representation languages which have solid grounding in mathematical logic, in particular, First Order Logic (FOL) and the Description Logics outlined in the following sections.

2.3.1 Propositional Logic

In propositional logic simple statements or *propositions* such as "This audio track has a tempo of 120 bpm" are used, together with the logical connectives $\land$ *(and)*, $\lor$ *(or)*, $\rightarrow$ *(implication)*, $\neg$ *(negation)* or $\leftrightarrow$ *(equivalence)* in order to build a composite formula. The truth value of propositional formulæ however are dependent on the truth value of atomic propositions, while the internal structure, that is, the meaning or *semantics* of propositions is inaccessible in this logic.

2.3.2 First Order Logic

First Order Logic extends propositional logic with functions, variables (that range over a fixed domain), universal and existential quantification, and a way to describe relations between domain objects using predicates. Therefore it is also called Predicate Logic. Table 2.1

summarises the syntax of FOL and provides some examples.

Elements of FOL	
Constant symbols	piano, apple, bach, me
Variables	$a,b,c...x,y$
Logical connectives	$\wedge$ (conjunction), $\vee$ (disjunction), $\rightarrow$ (implication), $\neg$ (negation), $\leftrightarrow$ (equivalence)
Quantification	$\exists$ (existential) $\forall$ (universal)
Predicates	$\mathsf{Signal}(a)$ (unary), $\mathsf{Title}(artist, album)$ (binary), $\mathsf{P}(term_1, term_2, ...term_n)$ (n-ary)
Functions	+(1,2)

Table 2.1: Elements of First Order Logic (FOL)

First Order Logic formulæ are interpreted as statements describing relationships between entities. Given for instance the predicates (relations) *Signal* and *hasSignal*, in the domain of an audio editor, we may define an *AudioClip* using the formula:

$$\forall x(\mathsf{AudioClip}(x) \leftrightarrow \exists y(\mathsf{hasSignal}(x,y) \wedge \mathsf{Signal}(y))) \tag{2.1}$$

Predicates are atomic statements whose arguments may be constant symbols, functions, or variables bounded by quantifiers. In this example, the variable x is universally quantified, which corresponds to the English terms *all, each, any.* In data modelling terms, we may think of unary predicates such as $\mathsf{Signal}(y)$ as identification or instantiation, and binary predicates like $\mathsf{hasSignal}(x,y)$ as relations. Our example audio clip can be described further as follows:

$$\exists x(\mathsf{AudioClip}(x) \wedge \mathsf{tempo}(x, 120) \wedge \mathsf{name}(x, myrecording)) \tag{2.2}$$

where the variable x is existentially quantified, corresponding to the notion of *some* or *there exists*, which imply a minimum cardinality of one.

2.3.2.1 Interpretations and Models

A complex FOL formula may be true or false with respect to a given interpretation $\mathcal{I}$. An interpretation is a mapping of symbols to objects, relations or functional relations in an arbitrary non-empty set Δ, which is called the domain (of elements) or domain of discourse.

A *model* for the formula is any interpretation where the formula is true. In a more general sense, models are formally structured worlds that are used to calculate the truth value of logical sentences. Conventionally, this is written as $\langle \Delta, \bullet^{\mathcal{I}} \rangle$, where $\mathcal{I}$ is a function that maps

- any constant symbol a to elements of Δ: $a^{\mathcal{I}} \in \Delta$
- any n-ary predicate symbol P to relations over Δ: $P^{\mathcal{I}} \subseteq \Delta^n$
- any n-ary function symbol f to functions over Δ: $f^{\mathcal{I}} \in [\Delta^n \rightarrow \Delta]$

where Δ^n is the set of all ordered n-tuples of elements of Δ, and $\bullet^{\mathcal{I}}$ is used as a shorthand for $\mathcal{I}(\bullet)$.

The truth value and satisfiability of ground terms (terms that do not contain variables), predicates (ground atoms), and atomic formulas are evaluated with respect to such a model. For instance, the value of the predicate $P(t_1, t_2, ...t_n)$, is true, if and only if the terms referred to by the arguments $t_1, t_2, ...t_n$ are in the relation referred to by the predicate P itself.

2.3.2.2 Variable Assignment

For the set of all variables V we define the function $\alpha : V \rightarrow \Delta$, that maps variables to elements of the domain. The interpretation of terms under $(\mathcal{I}, \alpha)$ is then:

- for constant symbol a: $a^{\mathcal{I},\alpha} = a^{\mathcal{I}}$,
- for variable x: $x^{\mathcal{I},\alpha} = \alpha(x)$,
- for function f: $f(t_1, ...t_n)^{\mathcal{I},\alpha} = f^{\mathcal{I},\alpha}(t_1^{\mathcal{I},\alpha}, ...t_n^{\mathcal{I},\alpha})$.

2.3.2.3 Satisfiability and Tautology

Satisfiability in FOL is given in terms of an interpretation $\mathcal{I}$ under a variable assignment α. In general, an interpretation is a model of the formula ϕ under α if $(\mathcal{I}, \alpha) \models \phi$, where $\models$ stands for entailment or logical consequence, defined in the next section. Satisfiablility in atomic formulas is given, for instance, by $(\mathcal{I}, \alpha) \models P(t_1, t_2, ...t_n)$ iff $\langle t_1^{\mathcal{I},\alpha}, t_2^{\mathcal{I},\alpha}, ...t_n^{\mathcal{I},\alpha} \rangle \in P^{\mathcal{I}}$.

The formula ϕ is then said to be

- satisfiable, if there is some $(\mathcal{I}, \alpha)$ that satisfies ϕ,
- a tautology (valid), if every $(\mathcal{I}, \alpha)$ satisfies ϕ,
- falsifiable, if there is some $(\mathcal{I}, \alpha)$ that does not satisfy ϕ,
- unsatisable, if there is no $(\mathcal{I}, \alpha)$ that satisfies ϕ.

2.3.2.4 Entailment and Reasoning Procedures

The formula ϕ is a logical consequence of ψ, iff ϕ is true in all models of ψ, denoted $\psi \models \phi$. The notion of entailment can be extended to a (large) set of statements as follows: If $M(\phi)$ is the set of all models of ϕ, and KB is a set of statements called a Knowledge Base, $KB \models \phi$ iff $M(KB) \subseteq M(\phi)$. That is, the knowledge base entails ϕ, if ϕ is true in every model in which all the sentences in KB are true.

A method to derive ϕ from KB is called an inference or reasoning procedure, denoted $KB \vdash_i \phi$, where i is a reasoning procedure. The two most important properties of reasoning procedures are *soundness* and *completeness*, defined as follows: A procedure i is

- sound, if whenever $KB \vdash_i \phi$ is true, $KB \models \phi$ is true,
  that is, no false consequences are drawn,
- complete, if whenever $KB \models \phi$ is true, $KB \vdash_i \phi$ is true,
  that is, all correct conclusions are drawn.

Inference in knowledge bases is typically based on rules that allow the symbolic computation of sentences that are entailed by the knowledge base. The two main approaches used in practice are based on the *modus ponens* rule of inference $(a \wedge (a \rightarrow b) \vdash b)$, used in forward-chaining algorithms which are executed every time some new facts are added to the knowledge base, or the *modus tollens* rule of inference: $(\neg b \wedge (a \rightarrow b) \vdash \neg a)$ which is typically used in backward-chaining executed at query time.

Other reasoning problems include satisfiability, subsumption and model checking. The Tableaux calculus for instance is a decision procedure which solves the problem of satisfiability for a FOL formula. That is, it finds a model of the formula, by decomposing it and exhaustively evaluating all possibilities.

2.3.2.5 Decidability

Given a logical system, a reasoning problem is decidable if there is an algorithm which solves the problem in a finite number of steps. Decidability is therefore a crucial property of a logical system, as opposed to the decision procedure itself. In FOL, the problem of logical implication is only semi-decidable [Church, 1936]. There is no decision procedure that determines whether an arbitrary sentence is valid or not, since, for any arbitrary sentence, if the sentence is false, the procedure may not terminate. A logic can typically be made decidable by restricting its expressiveness.

The properties of FOL discussed so far only cover the most important aspects of this logic. A more detailed discussion about model theoretic semantics [Hodges, 1993] outlined in Section 2.3.2.1 to 2.3.2.3, as well as details about reasoning procedures and decidability can be found for instance in [Franconi, 2002] and [Boolos et al., 2007].

2.3.3 Higher Order Logics and reification

First Order Logic only allows for the quantification of variables that range over individuals that are elements of a domain of discourse. Second order logic relaxes this limitation, by allowing variables that range over sets of individuals, as well as the quantification of predicates. Higher order logics use quantification of even higher types, for instance, relations between relations. Although these logics are more expressive, they lose many desirable properties of FOL, thus it becomes more difficult to create effective, sound, and complete reasoning procedures.

The above properties of second order logic allows for writing statements about statements, which is a very useful tool from a knowledge representation point of view. However, this feature can also be expressed in FOL through the process of *reification*. This process consists of creating a *constant* associated with a predicate, which then can be used in other sentences. Note that the term 'reification' will be used in a somewhat broader sense throughout this thesis, and will include, for instance, the notion of representing relations as concepts in an ontology in order to allow for expressing further details about a relation.

2.3.4 Temporal and Modal Logics

Several types of logics exist which extend standard formal logic. These include *modal logic* which deals with possibility, probability and necessity. *Temporal logic*, or *tense logic* can be interpreted over temporal structures. For instance, the Interval Temporal Propositional Modal Logic allows for incorporating Allen's temporal relationships [Allen, 1983] in its formulæ. Most of these logics however are undecidable for most interesting problems. In other cases the problems may be expressed in FOL, using for instance, the above discussed reification process.

2.3.5 Description Logics

A particular problem of FOL discussed in Section 2.3.2.5 is related to the fact that it is only semi-decidable. In order to create effective and terminating reasoning procedures the expressiveness of FOL has to be limited. Description Logics (DL) [Baader et al., 2003] are different restricted fragments (sub-languages) of FOL, with properties providing a trade-off between expressiveness and decidability making DL languages more suitable for certain applications. Description Logics have been increasingly used in software engineering and information management for formalising various aspects of applications; domain models, application logic and data communication [Borgida, 1995]. They have also been used as basis for modern ontology languages of the Semantic Web such as the Ontology Web Language (OWL)[5] [Horrocks et al., 2003]. The stack of ontology languages recommended by the mediator of Web stan-

[5] http://www.w3.org/TR/owl-ref/

dards, the World Wide Web Consortium (W3C), grows progressively closer to Description Logic languages with OWL-DL corresponding to $\mathcal{SHOIN}(\mathcal{D})$ while OWL-2 building upon $\mathcal{SROIQ}(\mathcal{D})$.

DL languages are related to conceptual models in that they allow precise specifications to be made such as cardinality constraints, for instance, using the features of OWL such as owl:minCardinality and owl:maxCardinality, or restricted existential and universal quantification expressed by owl:someValuesFrom and owl:allValuesFrom respectively. Examples of using DL based Web ontologies include modelling contextual information in software [Turhan et al., 2006], or augmenting data with semantic annotations, which is one desirable property we exploited in our research as described in Chapter 5 Section 5.1.

The symbols constituting the name of a logic describe the features and constructs allowed in a DL language [Horrocks, 2008]. For instance, $\mathcal{N}$ in $\mathcal{SHOIN}(\mathcal{D})$ tells us that the language is capable of expressing cardinality restrictions, while $\mathcal{Q}$ in $\mathcal{SROIQ}(\mathcal{D})$ stands for qualified cardinality restrictions constraining the number of values of a particular type for a property. The symbols $\mathcal{S}$ common in both languages is an abbreviation for $\mathcal{ALC}$, an extension of the basic attributive language providing minimal constructs such as atomic negation and existential quantification limited to the top concept. This forms the basis for the DL family $\mathcal{SH}$ (where $\mathcal{H}$ stands for role or property hierarchies, i.e. the use of rdfs:subPropertyOf in OWL and the RDF Schema language, see Section 2.4.4), and also the basis for the design of Semantic Web ontology languages, in particular, all versions and layers of OWL. More details about DL languages, their constructs and the naming scheme can be found in [Baader et al., 2003].

2.3.6 Knowledge Representation and Ontologies

Knowledge Representation is mainly concerned with providing a formal representation of a domain of a logic-based system. This includes a *vocabulary*, which is essentially a set of objects or entities that are included in the domain, a set of possible *relations* between the entities, and also includes a set of *constraints* over the use of the objects and relationships. These choices usually reflect the knowledge of a domain expert, and constitute the design of an *ontology*. A formal definition of ontology is provided in Chapter 3 and in [Gruber, 1993a].

Several kinds of languages are available for this purpose, including the Entity-Relationship conceptual data model, or UML mentioned in Section 2.2 which may be expressed as diagrams. Description Logic languages and corresponding Semantic Web ontology languages however are more and more commonly used to provide machine processable definitions.

2.3.7 Taxonomies and Partonomies

The vocabulary of entities as well as their relationships within an ontology are often organised hierarchically. The two most common organising principles are *i)* taxonomies which express a hierarchy of *is-a* or *type-of* relationships between objects and/or their properties independently, and *ii)* partonomies which express meronomical or *part-of* relations between objects. Ontologies however do not need to be deeply hierarchical. They define the semantics of concepts in terms of their properties and possible relations, which are often neither taxonomical nor mereological. Other types of knowledge representation such as the use of restrictions or constraints on the use of domain entities are often more suitable for a particular application.

2.4 Semantic Web Technologies

The basic idea behind the Semantic Web [Berners-Lee et al., 2001] is to ease information seeking tasks for humans or Semantic Web user agents, by facilitating the aggregation of Web content, as well as automated reasoning over this content. Representing the heterogeneous information found on the Web, however, is a difficult task from a knowledge representation point of view, therefore it became an active area of research. The set of techniques, often termed Semantic Web technologies, amalgamate different methods for representing and linking information.

These technologies encompass a set of Web standards for communication and information sharing. The Uniform Resource Identifier (URI) provides us with a conventional unique naming scheme for ontological concepts and relationships. In the context of the Web these are called *resources*. The Hypertext Transfer Protocol (HTTP), which provides basic methods for obtaining resources identified by HTTP URIs, is the fundamental linking mechanism used on the Web, and on top of these we find the Resource Description Framework (RDF) [Lassila and Swick, 1998] and various Semantic Web ontologies built on the foundations of this framework.

A key enabling concept in the success of the Web is the URI. It solves the problem of identifying and linking resources (web pages, data, or services) in a simple and efficient manner. Together with the access mechanism of HTTP, it enables the formation of a large interlinked network of documents: the Web as we know and use it today. However, this infrastructure is not yet used as widely and effectively as it could be. In particular, the flow of data and access to networked services are cluttered by incompatible formats and interfaces. In a more technical sense, the Semantic Web aims at resolving this issue, in the wider context of bringing intelligence to the Web, by creating a "Giant Global Graph" of machine-interpretable data. In the rest of section we provide an overview of Semantic Web technologies, first however, we briefly outline the basic idea behind the Semantic Web data model and why it is useful for musical applications.

2.4.1 Information Management and the Semantic Web

Since information on the Web can stand for just about anything, developers of the Semantic Web are faced with a major challenge: How to represent and communicate diverse information such that it can be *understood* by machines? The answer lies in standardising how information is published rather than trying to arrange all human knowledge into rigid data structures. Based on this recognition, several remarkable yet simple technologies emerged promoting the development of the Semantic Web.

We believe that this musical information is just as diverse as information expressed on the general Web. Moreover, we cannot presume to plan for all possible uses of our system components. Therefore, we need data structures that are interoperable with other systems, and extensible even by end-users. This poses problems similar to building the Semantic Web itself, hence the development of Semantic Web ontologies and the use of technologies produced in this community became a key element in this work.

Similarly to the general Web: an interlinked network of documents, the Semantic Web is a heterogeneous network of interconnected data and services. This network may only work if various disjoint data sets and services speak the same language. Thus, they have to follow some common data model or structured schema. The problem however is that the Web exposes unbounded, diverse information, making it hard, if not impossible to design this schema. Yet, the Semantic Web provides a surprisingly simple solution to this problem: the Resource Description Framework.

2.4.2 Resource Description Framework

RDF is a conceptual data model providing the flexibility and modularity required for publishing diverse semi-structured data—that is, just about anything on the Semantic Web. It is also the basis of more complex description languages, such as the OWL Web Ontology Language, which provides a way of publishing extensible data schema. The model is based upon the idea of expressing statements in the form of *subject – predicate – object* (s,p,o), also known as triples. A collection of triples can be seen as a graph, with nodes representing subjects and objects, and edges representing predicates as shown in figure 2.4. Therefore, a large set of statements form a complex network of semantic relationships. All elements of a triple may be named by a URI, which enables RDF graphs to be linked and distributed across the Web.

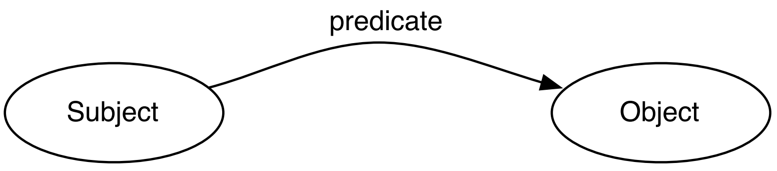


Figure 2.4: The graph structure of the basic RDF triple

2.4.2.1 RDF syntax and semantics

Elements of RDF statements may be resources, literals or blank nodes. Concepts and relationships are represented by resources named by a vocabulary of URIs. This provides the model with its core semantics, an unambiguous way of referring to things since dereferenceable URIs are globally unique, and permits using HTTP as a linking mechanism. Literals are used to express data such as the name of an artist, or a numerical parameter of an algorithm. A blank node is an unnamed resource whose use in an RDF statement corresponds to existentially quantified variables of logical sentences. This is useful when describing complex hierarchies where a certain resource may not be, or may not need to be explicitly named.

Formally, an RDF graph is a set of triples of the form (s, p, o). The property, or predicate p is drawn from a set of URIs R. The subject s is drawn from the union of R and a set of blank nodes B. The object o is drawn from the union of R, B and a set of literals L. Blank nodes act as existentially qualified variables over the domain of a particular graph. RDF graphs can be drawn as a graph using a union of R,B,L as nodes and drawing an edge for every triple labelled with the property URI. Non-blank nodes are labelled with a URI or a literal string in quotes [Howe et al., 2004].

2.4.2.2 RDF/XML and RDFa

RDF/XML specifies a syntax for serialising (representing) RDF graphs. Albeit this is the oldest and most adopted format, more concise and human readable formats are rapidly gaining popularity. A somewhat related syntax, RDFa uses a set of XHTML attributes to augment visual data with "machine-readable" data, that is, information which is liked with explicitly defined meaning (such as an ontology) such that machines can automatically process or interpret it. RDFa enables the inclusion of RDF data in traditional Web content.

2.4.2.3 NTriples and Turtle

As we mentioned, the RDF conceptual model does not specify a syntax for encoding information in itself. While RDF/XML is a common serialisation format for RDF data, more compact and efficient representations exist such as Turtle, which is both human and machine readable. Turtle is based on the simple NTriples syntax, exemplified in Listing 2.1. Its graph rendering is given in Figure 2.5 showing how this statement related to an RDF graph structure. Here, the URIs corresponding to resources are written out in full. This syntax therefore is rather verbose. In order to provide a more compact and readable representation, the Turtle syntax uses a shorthand notation exemplified in Listing 2.2.

In our example, the explicitly written URI identifies a web resource representing an artist. We make two statements about this artist. The first triple, where the predicate `rdf:type` and object `mo:MusicArtist` are written using the namespace prefix notation expresses the fact that this resource is a music artist. Such URI references are expanded using a namespace declaration after a `@prefix` directive like the ones in our example. A prefix can also remain empty, in which case it is bound to the local namespace of an RDF file or data store.

```
<http://dbpedia.org/resource/Dave_Brubeck>
        <http://xmlns.com/foaf/0.1/name>
                "Dave Brubeck" .
```

Listing 2.1: RDF statement in N-Triples

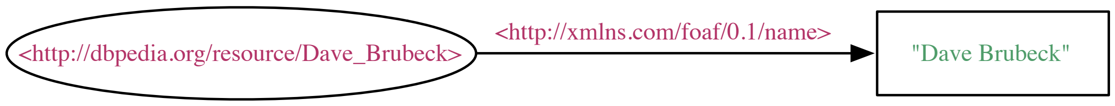


Figure 2.5: Graph of an N-Triples statement with URI references: In a common graphical formalism resources are depicted using ovals, while literals may be represented by ovals or rectangular boxes.

2.4.2.4 RDF linking mechanism

The second triple in our previous example after the semicolon refers to the same resource; our artist. Here, the semicolon is used to group RDF statements about the same resource. The use of the URI exemplifies how RDF may be utilised to link resources. We can now follow the `owl:sameAs` link to a resource within DBpedia[6], which holds structured data extracted from Wikipedia. Expanding the prefix notation by concatenating `type` to the URI corresponding to the prefix, we can also follow the `rdf:type` link to get more information about what `mo:MusicArtist` means.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix mo: <http://purl.org/ontology/mo/> .

<http://www.bbc.co.uk/music/artists/1545000730-525f-4ed5-aaa8-92888
    f060f5f#artist>
    rdf:type mo:MusicArtist ;
    owl:sameAs <http://dbpedia.org/resource/Dave_Brubeck> .
```

Listing 2.2: Linking two resources representing a music artist.

2.4.3 Semantic Web ontologies

Although RDF provides a fundamental data model, it does not have the facilities for expressing complex relationships required for modelling a domain. In order to precisely communicate

[6]http://dbpedia.org/

information using RDF statements, we have to be able to define and later refer to concepts such as a specific algorithm we use for audio processing, its concrete implementation and its parameters. We also need a vocabulary of well defined relationships existing in an application.

To use a musical example, an RDF statement may stand for a simple piece of information like: *"Take Five" - "composed by" - "Paul Desmond".* Should we try to express this information without referencing an already established data set, we immediately face the need of having to say more than what is conveyed by our simple statement. For example, we might need to add that *Take Five* is a jazz piece. Similarly, one might think of Paul Desmond as the saxophonist in the Dave Brubeck Quartet. Still, with regards to our particular statement, he is a composer acting as an agent in the composition event that produced the piece *Take Five*. This more logical view is compliant with the Music Ontology, as opposed to the simple example statement above. In order to avoid ambiguities and be precise in our statements, we need to be able to define (and later reference) concepts, for instance a *Song*, a *Composer*, a *Plugin* for audio processing or the *FFT size* parameter in an algorithm performing the Fast Fourier Transform. We also have to specify relationships or roles (such as the one associating a song with its composer) pertinent to our application. Ontologies are the tools for establishing these necessary elements in a domain model. Building ontologies, a process which will be discussed in Section 3.1 in more detail, is often referred to as knowledge engineering. In the next section, we provide a very brief introduction to the Ontology Web Language (OWL), or more precisely, a family of languages most commonly used for building Semantic Web ontologies.

2.4.4 RDFS and the OWL

Semantic Web Ontologies are created using the same conceptual model that is used for communicating the data. However, additional vocabularies are needed for expressing formal ontologies as well as for greater machine interpretability. A hierarchy of languages are recommended by the W3C for this purpose. This includes the RDF Schema Language (RDFS) [Brickley and Guha, 2004] for defining classes and properties of RDF resources and OWL [Patel-Schneider et al., 2004] for making RDF semantics more explicit. OWL has three layers, corresponding to three levels of expressiveness, OWL-Lite, OWL-DL and OWL-Full. OWL-DL is the most commonly used layer and it closely corresponds to previously mentioned Description Logics, although other OWL flavours also have DL equivalents. Using this language we can impose for instance restrictions on the range and domain types of properties, or constraints on cardinality, the number of individuals linked by a property. An updated language OWL-2[7] has been released to overcome some of the limitations and increases the expressiveness of OWL. This language remains backward compatible with the previous version. For a discussion on how OWL is related to Description Logic languages please see Section 2.3.5.

[7]See the comparison of OWL-2 to OWL-1 in: `http://www.w3.org/TR/owl2-overview/`

A standardised way of accessing information represented in RDF is an important requirement in building applications using Semantic Web technologies. In the next section, we briefly outline a language most commonly used for this purpose.

2.4.5 SPARQL: a query language for the Semantic Web

The *SPARQL*[8] *Protocol and RDF Query Language* has recently emerged as a W3C recommendation from a number of similar languages previously designed for accessing RDF data stores. SPARQL is a query language somewhat similar to the Structured Query Language (SQL). It allows complex joins of disparate RDF resources in a single query. A Web interface executing a SPARQL queries is commonly referred to as a SPARQL end-point. The language can be used in multiple ways. In the simplest case, a query consisting of triple patterns is matched against a database. Results are then composed of variable bindings of matching statements based on a *select* clause specified by the user. For example, the query shown in listing 2.3 retrieves all triples about the DBpedia resource Bill Evans. In this example, the HTTP URI identifies the artist in DBpedia's database. Terms starting with a question mark represent free variables that are matched when the query is evaluated.

Using SPARQL is the easiest way of accessing the semantic web from an application while creating an end-point is a standard way of publishing data. Most modern programming languages have SPARQL libraries, and several open-source RDF stores (including Openlink's Virtuoso, Garlik's 4Store, Joseki or the D2R Server) are available for creating an end-point. The standardisation and increasing support of SPARQL promotes the adoption of RDF itself as a prevailing metadata model and language.

```
SELECT ?predicate ?object
WHERE {
      <http://dbpedia.org/resource/Dave_Brubeck> ?predicate ?object .
}
```

Listing 2.3: A simple SPARQL query.

2.4.6 Notation 3 and RIF

The primary aim of Notation 3 (N3) is to create a language that allows for integrating the expression of logical rules within the RDF framework, using a compatible data model and a concise syntax [Berners-Lee et al., 2008]. Notation 3 extends the RDF data model with features often found in logics, such as variables, universal quantification, and formulæ, which allow N3 graphs to be quoted within N3 graphs. Quantification is discussed in Section 2.3.2.

[8]http://www.w3.org/TR/rdf-sparql-query/

As we previously mentioned, existential quantification can be expressed in pure RDF using blank nodes.

Syntactically, N3 is a superset of Turtle, and adds elements, such as a set of new keywords, and curly brackets, which can be used to denote a quoted formula. For instance, the statement `{ [ a mo:MusicArtist ] mo:member [ a mo:MusicGroup ] } a log:Truth .` expresses the truth value of the set of statements in braces, which represent the sentence *'There exists a music artists who is a member of some music group.'*. Universal quantification can be expressed using the notation `?x`, while existential quantification can be expressed using blank nodes, as in our previous example, or using the notation `_:x`. There is a shorthand for expressing rules, i.e. the notation `=>` corresponds to `log:implies`, while the use of the symbol `=` corresponds to the term `owl:equivalentTo`. These extensions to the RDF model and semantics yield a simple rule language whose statements may be interpreted using, for instance, the Closed World Machine (CWM) [Berners-Lee et al., 2006], a forward-chaining reasoner for deriving new statements from RDF data using N3 rules.

It should be noted however that N3 is not a standardised or the only rule language used on the Semantic Web. The Rule Interchange Format (RIF) is a current W3C recommendation[9] which facilitates the exchange of rules between many existing rule systems. RIF provides a model for expressing rules that can be mapped onto RDF triples.

2.4.7 Linked data

The ultimate goal of the Semantic Web is to enable machines to interpret Web resources and thus execute very complex search tasks currently requiring human-level intelligence. The more pragmatic goal of Linked Data is to facilitate this process by publishing information in an open format that shares a common conceptual framework, that is, RDF. The data sources published in recent years cover a wide range of topics: from music (MusicBrainz[10], Magnatune[11] and Jamendo[12]) to encyclopaedic information (Wikipedia) or bibliographic information (Wikibooks[13], DBLP bibliography[14]).

The 'Linking Open Data on the Semantic Web' community project [Bizer et al., 2007] of the W3C Semantic Web Education and Outreach group[15], aims at making data sources available on the Semantic Web and creating links between them using the technologies described in the previous sections. This includes creating SPARQL end-points, exposing partial or full RDF representation of Web resources by serving both HTML for traditional browsers, and RDF for Link Data browsers (the correct version is typically accessed using a process called

[9]http://www.w3.org/TR/rif-overview/
[10]http://musicbrainz.org/
[11]http://magnatune.com/
[12]http://www.jamendo.com
[13]http://wikibooks.org/
[14]http://dblp.uni-trier.de/
[15]http://www.w3.org/2001/sw/sweo/

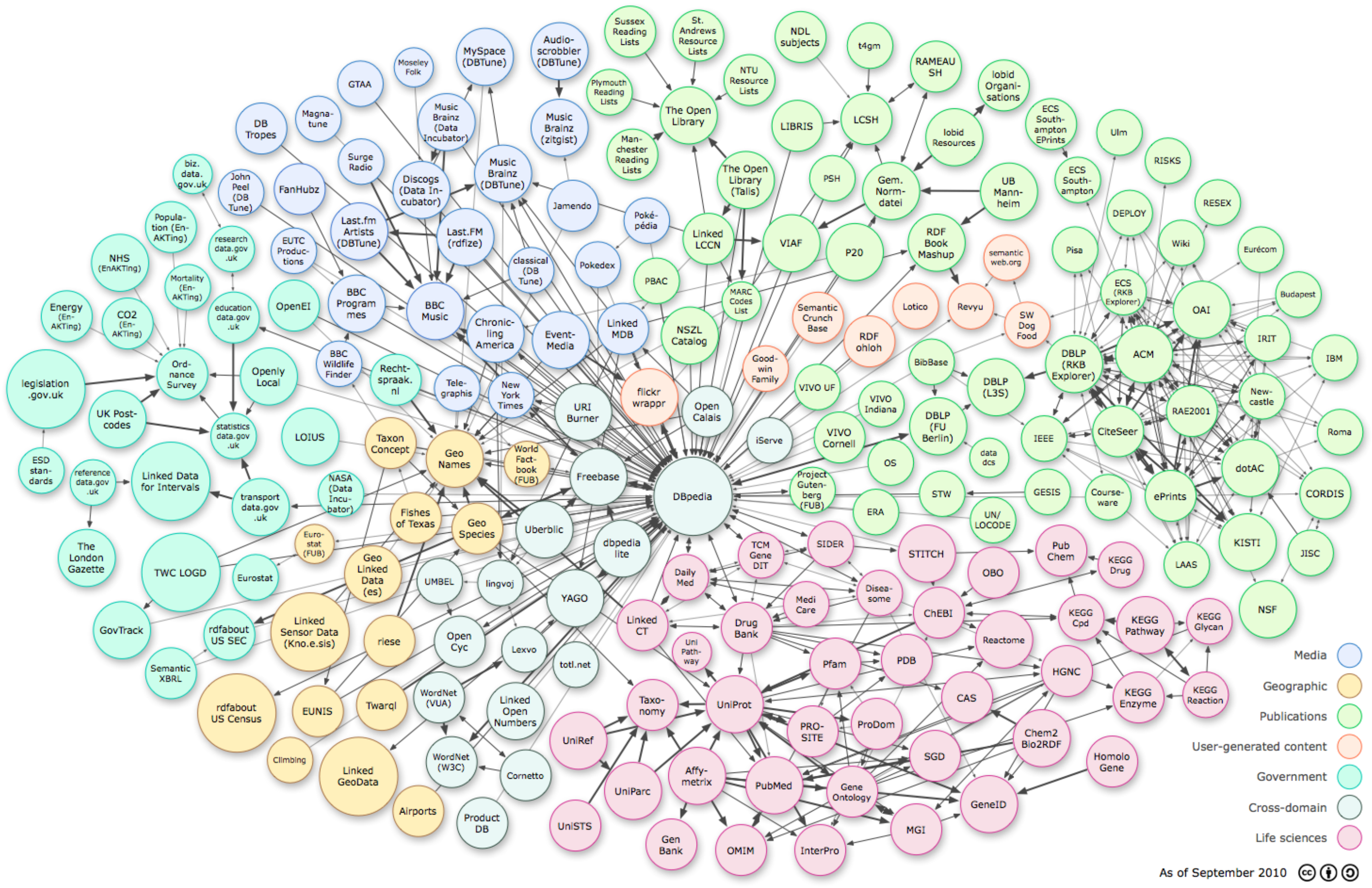


Figure 2.6: Datasets published by the Linking Open Data community, September 2010. Each node corresponds to a particular dataset. Diagram by Richard Cyganiak, and available on-line at http://richard.cyganiak.de/2007/10/lod/

content negotiation), publishing downloadable RDF dumps, and enriching Web pages using the RDFa format mentioned in Section 2.4.2.2.

The growing community behind Linked Data aims at making information freely available on the Semantic Web at large. The various data sets published by the end of 2010 is depicted in Figure 2.6. This includes the DBpedia [Auer et al., 2007] project which extracts structured information from *fact boxes* the Wikipedia community-edited encyclopedia, the Geonames dataset[16] which exposes structured geographic information. Finally, the BBC datasets [Kobilarov et al., 2009] cover a wide range of information from programmes[17] to artists and reviews[18]. Creating bridges between previously independent data sources paves the way towards a large machine-processable data web, gathering interlinked Creative Commons licensed content[19], such as encyclopaedic information, domain-specific databases, taxonomies, and cultural archives.

Linked Data has relevance in this work for two reasons. Firstly, the ontologies presented in the following chapters facilitate publishing data about music production on the Semantic Web, thus we contribute a new and presumably valuable source of information to the music research and Linked Data communities. Secondly, with the growing trend of using multiple modalities in MIR applications — see for instance the growth of this field in the proceedings of the ISMIR conference[20] — we can hypothesise that semantic audio tools relying on Semantic Web technologies can utilise Linked Data resources more easily and more successfully.

2.5 Summary

In this chapter, we reviewed the basics of information management and knowledge representation with a particular interest in using these technologies in audio applications. In Section 2.2, we outlined an information management framework for semantic audio tools. We examined the requirements of utilising semantic audio tools in music production, and argued that such systems would benefit from using Semantic Web technologies. The fundamentals of Semantic Web technologies, as well as the logical foundations of knowledge representation and Semantic Web ontologies expressed using RDFS and OWL were then outlined in Sections 2.3 and 2.4.

In the next chapter we discuss how these technologies may be utilised in more specific knowledge representation problems, outline the design principles governing the creation of shared ontologies, and examine the application of these principles via looking at a narrower domain with relevance in our work, namely, ontologies for multimedia applications.

[16]http://geonames.org/
[17]http://www.bbc.co.uk/programmes
[18]http://www.bbc.co.uk/music
[19]http://creativecommons.org/
[20]http://www.ismir.net/

Chapter 3

Ontology Engineering and Multimedia Ontologies

In previous chapters we saw how information can be obtained from audio material by means of analysis, what information requirements need to be considered in intelligent semantic audio applications and music research tools, and reviewed the kinds of technologies currently available to manage this information. We have seen how Semantic Web technologies and Semantic Web ontologies can be used in information management, and made a stand for utilising these technologies due to their solid logical foundations and open ended information model.

In this chapter we go deeper into ontological questions pertaining to semantic audio applications. We review the most prominent ontologies existing for music and multimedia related information, and describe some principles which we use in the design of ontologies related to the information framework outlined in Chapter 2.

3.1 Ontologies and basic ontological decisions

The definition, interpretation, etymology and meaning of *ontology* vary throughout history and differ in specific areas of philosophy and science. In metaphysics, ontology refers to the formal study of existence; designations of basic categories and relationships of entities. In the sciences, this notion can be subdivided further. In artificial intelligence, ontology is seen as the conceptualisation of a world (a universe or domain of discourse) used for knowledge sharing between technological artefacts. In computer science and software engineering in particular, it may be seen as the specification for the domain model of an application or a database. In this work, noting that knowledge models are distinct from data models, we are primarily interested in the latter interpretations. When seen in this sense, creating an ontology is, in essence, collecting and organising objects that form a coherent application domain, and defining their semantics by means of declaring relationships between them, as well as constraints over their interpretation and use. These activities amount to a set of *ontological decisions*, closely related but not equivalent to the design of software application models, where ontology is seen as the class model of an application, expressed for instance using the Unified Modelling Language (UML), creating database schemata, or devising communication protocols. The importance of these aspects of ontologies will become clearer in later chapters. Here, we focus on basic decisions governing the design of ontologies we describe in the rest of this chapter.

3.1.1 Ontology and Philosophy

Making ontological decisions is far from straightforward and often leads to philosophical problems. Enumerating and correctly identifying objects in a domain for instance, while only a part of the design process, can be a difficult task in itself. Returning to somewhat philosophical enquiries, we may ask with regards to object identity whether a *glass* for instance, equates to the material it is made out of. This example highlights different sources of ambiguities. The first arises from language, due to the polysemous nature of the word glass. If we know that it refers to a container to drink from, we arrive to the philosophical question of identity[1], that is, can we distinguish between co-localised objects occupying the same spatial and temporal co-ordinates? The common sense answer, based on certain properties of the objects in question, seems to be *yes*, however there are different subtle distinctions that can be made. For example, is a metal can the same entity after squeezing it slightly or squeezing it fully? Is an audio signal the same entity after some transformation? These questions are not easily answered. An attempt to give an account on available options of interpretations would require a careful review of metaphysics, epistemology and related branches of philosophy. Our insight suggests that the answers are deeply dependent on context, and lay in finding the granularity in which information have to be represented and understood. To avoid getting bogged down in questions of this sort, we rely instead on mature and well-established secondary sources to evaluate ontological decisions. These sources are secondary in the sense that they are one or more steps removed from the original study of philosophical problems related to ontological enquiries, and contain application specific analyses. For instance [Genesereth and Nilsson, 1987], and [Gruber, 1993a,b], provide useful definitions and discuss design criteria in artificial intelligence, [Smith, 1995; Brickley et al., 1999; Sowa, 2000; Guarino and Welty, 2002; Gangemi et al., 2002; Masolo et al., 2003a], provide important modelling principles, [Kania, 2008] discusses methodological aspects of designing musical ontologies and implications of taking a descriptivist or a revisionary approach (see Section 3.1.4.7), [Horrocks, 2008] discusses Semantic Web ontologies in the context of Description Logics, while [Berners-Lee, 2006] hints at ontology design issues related to best practices of Linked Data publishing. In what follows, we discuss basic ontology engineering and design principles drawn from multiple resources.

[1]The problem mentioned here is modelled after the metaphysical problem of material constitution, and the classical example of how the vase and the clay it is formed of are related.

3.1.2 Ontology engineering

Although the creation of ontologies for computational purposes have long standing tradition in artificial intelligence [Sowa, 2000], in many other disciplines, including software engineering, life sciences, library science or multimedia and music research, their importance have only become recognised more recently. The revival of the field is partly due to the advancements in research and technologies related to the Semantic Web, however it is also due to new needs arising in various research and user communities.

The development of ontologies, often called *ontology engineering* is a complex design cycle, which aims at fulfilling diverse needs and requirements. These requirements may come from a single field or domain, in most cases however they reach across several overlapping domains. This is for instance the case in multimedia and the music domains mentioned above. Analysing the domain in order to determine the requirements and scope of an ontology is often cited as the first tasks in ontology engineering. The complete design cycle can be subdivided into the following main tasks:

- Requirements analysis
- Define scope and consider reusability
- Ontology design (domain modelling)
- Evaluation and refinement
- Application and evolution

The above partitioning roughly follows the methodology described in [Sure et al., 2009], however many authors (see [Haase et al., 2005a] or [Fensel, 2001]) argue that the the design process is fully circular as the domain knowledge behind most ontologies evolves continuously. Indeed, we can think of the above process as a single cycle or multiple cycles, since almost any stage can feed back information to an earlier stage. It is important to note that domain modelling can be subdivided further as shown below, and this process is often circular in itself. For example, defining relationships and constraints often reveals errors in the initial class hierarchy or necessitates the introduction of new terms.

- Enumerate concepts, properties and individuals
- Define classes and create a taxonomy
- Define relationships between classes
- Define constraints (e.g. cardinality and value restrictions)
- Add individuals (if necessary)

We do not intend to give a more detailed account on ontology engineering here. The process has a rich literature, see e.g. [Sure et al., 2009] for an overview. The rest of this section provides basic use cases constituting to our requirements analysis and describe some design principles resulting from this, or taken from the previously mentioned literature. These principles provide the basis for the design of the ontology library detailed in Section 4.2.

3.1.3 Some use cases for ontology design

It is important to consider generic needs and requirements for developing ontologies. These consideration lead to common design principles guiding the development, and may also alleviate the need for ex post facto harmonisation between elements of an ontology library. In this section, we discuss three use cases of ontologies relevant in our work. We also outline some basic ontology engineering principles through examples in the following areas ranging from more general to specific:

- Knowledge Representation and Knowledge Sharing
- Information Management and Multimedia Workflows
- Data Exchange in Music Production

3.1.3.1 Knowledge Representation and knowledge sharing

Knowledge sharing has become important especially in research for streamlined flows of work pertaining to reproducibility of experiments, and information exchange between groups and communities. For example, in Music Information Retrieval, it is desirable to reach a common agreement on the meaning and representation of audio features such as Mel-Frequency Cepstral Coefficients described in Section 1.4 that can be computed and interpreted in several different ways.

Knowledge sharing requires a shared *conceptualisation* of a domain. Formally, conceptualisation is a set of relations **R** over a universe of discourse D [Genesereth and Nilsson, 1987]. If this conceptualisation is represented and published in a syntactically and semantically interoperable formal language, we are talking about a shared ontology. While the concept of *shared* is important in our use cases, we keep the scope of this discussion constrained, and refrain from further explanation. Please see [Gruber, 1993b] and [Borst, 1997]. Although using a formal language facilitates syntactic interoperability in itself, it does not guarantee semantic interoperability, that is, a common interpretation and *ontological commitment* between agents interpreting our data. We say that an agent commits to an ontology if its observable actions are consistent with the definitions in the ontology [Gruber, 1993a]. Making an ontological commitment pertaining to the meaning of terms and expressions, higher level constructs — such as definitions that constrain the interpretation and use of terms — are required, leading to a logical system. The presence or lack of this logical system signifies the difference between database schemata or simple data models expressed using the XML schema language, and knowledge representations expressed for example in first order logic or Semantic Web ontologies. Genesereth and Nilsson [1987] and Gruber [1993a] provide formal definitions of the terms conceptualisation, ontology, and ontological commitment in knowledge sharing applications.

3.1.3.2 Information Management and multimedia workflows

Another important use case for the development of shared ontologies is provided by considering information management requirements in broad term multimedia workflows, such as the production, annotation and search of multimedia assets. Some recent motivating examples include [Troncy et al., 2004; Aubert et al., 2006; Arndt et al., 2007; Raimond, 2008]. The works of Arndt et al. and Raimond consider the use of distributed computational tools and services. Arndt et al. for example envision a multimedia presentation workflow, where multiple web services for face recognition provide semantic annotation of images, and argue that existing industry standards (see [Ossenbruggen et al., 2004] or [Smith and Schirling, 2006] for an overview) are insufficient for the requirements of the outlined workflow. This is due to syntactic and semantic incompatibilities of various standards, and the lack of *formal semantics*—that is, some logical formalism for the interpretation of terms and relationships in their definition. Raimond develops a distributed music information system with Semantic Web ontologies in its core. We demonstrate (see Section 5.4) how a tool for visualising content-based audio features may interoperate with a web service for automated audio analysis using the ontological framework developed in this system. The most important conclusions we learn from the examples and implementations mentioned above are the requirements for a conceptualisation and corresponding ontology to be *modular*, *extensible*, *open* and *reusable*). Defining shared *formal semantics* if we expect multimedia information management systems to co-operate is vital. In Section 3.1.4 we discuss these concepts and their consequences in more detail.

3.1.3.3 Data exchange in music production

In our final example, we describe why using an explicit conceptualisation (an ontology) can facilitate data exchange between media production and authoring tools. In music production, various authoring tools such as sample editors and multi-track recorders are used in different stages of the creative process. Interoperability between these tools however is often limited to raw data exchange, leading to the loss of valuable information such as recording conditions, signal processing parameters, or editorial metadata attached to recordings.

Most software applications use a set of abstractions to represent concepts such as audio tracks, plugins, controls and so on. They share a common metaphor, often relating these concepts to physical hardware equivalents if possible. There are also significant differences as demonstrated in [Duignan, 2008] in the context of commercial music software. The main problem however is the lack of explicit and shared conceptualisation of this domain. The implicit conceptualisation present for instance in the class hierarchy (or UML model) of a tool represents objects in different contexts, within different hierarchies, and having different relations and attributes. This makes standardisation efforts difficult. We argue that by modelling fundamental abstractions (such as an audio mixer device) independently from the scope and implementation of an actual tool, we can make rich data exchange possible thus retaining valuable information throughout the production workflow. This idea is related to an important design principle, namely, *separation of concerns* [Dijkstra, 1982]. In ontology design, this essentially requires that knowledge about individuals and entities in a domain should be represented separately from administrative issues related to their specific representation in a tool or a document. A similar example is provided in [Arndt et al., 2007] arguing why MPEG-7 [Martínez, 2004] is insufficient for exchanging media annotation data. Having outlined some important basic use cases, we shall proceed to consider the fundamental design principles that help to achieve these goals.

3.1.4 Ontology design principles

Gruber [1993a] identifies five design principles we find equally important in our use cases. These principles were originally developed to fulfil special requirements for interoperability between knowledge based systems, which communicate using statements in a formal language, grounded on specific knowledge representations. First, we summarise the criteria described by Gruber and provide examples how they influence ontological decisions. We also extend this framework with additional principles drawn from experience in developing Semantic Web ontologies, and additional resources such as the ontology design patterns outlined in [Gangemi and Presutti, 2009]

3.1.4.1 Clarity

According to the clarity criterion, definitions in an ontology should be independent of social and computational context. They should favour the description of *necessary and sufficient*[2] conditions — that is, once a modeller is committed to a specific use of a term, complete concept definitions are preferred over primitive concept definitions (describing necessary conditions only). It is also preferred that each definition is presented in a logical form and documented in natural language. In the context of Semantic Web ontologies, these requirements can be fulfilled using RDF and OWL (see chapter 3 for definitions) language constructs describing for instance subsumption relations, (`rdfs:subClassOf`) or cardinality restrictions

[2]In logic, A is said to be a *sufficient condition* for B, whenever A being true is all that is required for B being true. A is said to be a *necessary condition* for B if B cannot be true without A.

on certain properties. The documentation requirement is usually fulfilled using `rdfs:label` and `rdfs:comment` properties.

3.1.4.2 Coherence

The coherence principle dictates that an ontology should only permit those inferences that are consistent with its logical axioms and its natural language definitions. This criterion requires that the ontology should express the designer's intent as precisely as possible, and avoid inconsistencies. For example, it may be possible to define a class incidentally, that can have no instances, by subsuming disjoint classes. From a logical point of view, this means that the ontology describes an unsatisfiable theory. There are methodologies such as OntoClean [Guarino and Welty, 2002] to discover inconsistencies, however, it only provides limited support to verify that an ontology matches the modeller's intent in all aspects.

3.1.4.3 Extensibility

The extensibility criterion prescribes that definitions should allow the extension of an ontology monotonically—that is, without a revision of existing definitions. This requirement seemingly contradicts with the need for clarity favouring complete definitions. The contradiction may be resolved by anticipating specific uses of a term outside the modelled domain, and allow more specialised terms to be defined subsuming a concept or property in our ontology, committing to a more general definition. However, as we will see later, specialisation through subsumption is not the only way we can make connections between vocabularies. When classes are loosely related in two different domains, we may relate their individuals for instance using the `owl:sameAs` property, or relate terms using `rdfs:seeAlso` indicating that a resource might provide additional information about a subject in a different domain.

3.1.4.4 Minimal encoding bias

This criterion requires that a conceptualisation remains on the knowledge level without depending on implementations—that is, representation choices should not be made for implementation and notational convenience. This is indeed a tighter and more specific application of the separation of concerns principle mentioned in the previous section. This consideration makes us restrain from very specific data type definitions in most of our ontologies, since type mapping and similar sort of arbitration should be solved at a lower (implementation) level.

3.1.4.5 Minimal ontological commitment

According to this criterion an ontology should aim at specifying only those definitions that are essential for communication or knowledge sharing. From a logical point of view, this means that the ontology should admit many possible models of the world, (i.e. by specifying a weak theory making as few claims as possible). According to Gruber, the key of resolving the apparent conflict between this and the clarity principle (requiring tight concept definitions) is in the recognition that ontological commitment concerns conceptualisation in a more general sense, as opposed individual concept definitions. That is, once a term is included in a conceptualisation, a complete definition should be provided. Another possible way of resolving this and similar conflicts between design criteria is modularisation, a largely presentational solution described next.

3.1.4.6 Modular design

Modular design requires that ontologies adhere to clear conceptual boundaries. They should aim at describing small and coherent domains or sub-domains precisely. Although this was not included in the original set of criteria considered by Gruber, we see modularisation as a useful tool for resolving apparent conflicts between other design criteria. A modular ontology may be presented as a single document with clearly defined annotations (e.g. describing different levels of a conceptualisation) or as a library or framework of harmonised ontologies each contributing to a wider shared domain model. In OWL we may use the `owl:imports` construct to reference another ontology containing definitions whose meaning is considered to be part of the importing ontology.

3.1.4.7 Reusability and scope

An important design decision in the ontology engineering process concerns designating the scope of an ontology—that is, whether it should focus on a well defined small domain, serve as a core ontology representing a wider area of knowledge with less depth, or remain at a rather general level. These decisions are also influenced by whether an ontology is intended to be reused across multiple domains. The above designations suggest the distinction of three different types of ontologies:

- Domain Ontologies
- General or Core Ontologies
- Foundational or Upper-level Ontologies

Domain ontologies are centred around either an abstract or concrete concept, for example an ontology of time, an ontology of people, or an ontology describing a well defined process, workflow or task. In the music domain we may think of an ontology of instruments, or an ontology for describing how audio processing devices may be connected. Domain ontologies tend to be small and aim at providing complete coverage of an area of knowledge. Concrete examples include the OWL Time Ontology[3] describing temporal concepts, [Hobbs and Pan, 2006] or the Device Ontology presented in Section 4.2.3.2 describing artefacts of technology.

Core ontologies describe a wider domain defining fundamental concepts and relationships. They can be seen as more general domain ontologies that do not aim at full coverage. Instead, they are designed to be extensible (and often modular) allowing more specific ontologies to be plugged under fundamental concepts. Core ontologies may also import specific domain ontologies and may form an ontological framework. The Music Ontology, the Core Ontology for Multimedia (COMM) [Arndt et al., 2007] or the CIDOC Conceptual Reference Model (CIDOC CRM) [Crofts et al., 2010] are examples of core ontologies. We take a closer look at these ontologies later in this chapter.

Foundational ontologies do not concern a specific domain, rather they describe fundamental concepts reflecting categorical distinctions discussed in philosophy or simply collect terms that are the same across several knowledge domains. Examples include the Upper Mapping and Binding Exchange Layer (UMBEL)[4], the Standard Upper Ontology (SUO)[5] or the Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE)[6]. Although the definitions above suggest common objectives, the role and scope of foundational ontologies vary considerably.

UMBEL for instance provides a reference structure and base vocabulary for a large number of concepts extracted from OpenCyc[7] with the aim of helping content interoperate on the Web. It is popular in the Linked Data community for its mappings to other resources such as DBPedia[8] or GeoNames[9]. It also contains links to Music Ontology terms. Other foundational ontologies describe few generic concepts grounded on philosophical considerations and provide rigorous axiomatisations. DOLCE for instance draws from Strawson's examination on the conception of basic particulars as general spatio-temporal concepts [Strawson, 1959].

The philosophical status of foundational ontologies is controversial due to myriad of arguments for and against the feasibility of general purpose ontologies. See a discussion by Smith in [Floridi, 2004] for example. Due to this reason, and to avoid the complexity resulting from the reuse of ontologies like DOLCE, we argue the need for extending a foundational ontology directly, as it is advocated by the developers of COMM and others, described in [Arndt et al., 2007] and [Gangemi et al., 2002]. We see the role of foundational ontologies in services mediating between resources committed to different domain ontologies and in ontology alignment tools. We also use them as reference and observe their design considerations. For example, it is useful to think about whether an ontology should be *descriptive* or *revisionary*,

[3]OWL-Time Ontology: `http://www.w3.org/TR/owl-time/`
[4]UMBEL: `http://umbel.org/`
[5]SUO: `http://suo.ieee.org/`
[6]DOLCE: `http://www.loa-cnr.it/DOLCE.html`
[7]OpenCyc is a large manually assembled knowledge base: `http://www.opencyc.org/`
[8]DBPedia provides linked data extracted from Wikipedia: `www.dbpedia.org/`
[9]A geographical database exposed as linked data: `www.geonames.org/`

or take a *multiplicative* or *reductionist* approach with regards to the number and complexity of definitions. Descriptive ontologies are grounded on common sense, language and human cognition, while the revisionary approach requires a model to be always defensible on scientific grounds. The reductionist approach aims at defining the smallest number of primitives, while the multiplicative approach trades simplicity for a more expressive system. For similar reasons described in [Masolo et al., 2003a], we opt for developing descriptive ontologies, which tend to be multiplicative. We also focus on particulars, and do not include the kind of meta-properties in our ontologies described by Guarino and Welty [2002] in the context of OntoClean. The methodology is applied instead during the design process.

3.1.5 Summary

In this section, we discussed the role of ontology in artificial intelligence and computer science, outlined the difficulties of making ontological decisions and reviewed some basic concepts such as the use of a shared conceptualisation as frame of reference, and the meaning of ontological commitment. We also outlined the typical life cycle of ontologies. Through a number of relevant use cases, we reviewed the fundamental guiding principles that are used in the development of ontologies described in the following chapters, as well as the semantic audio tools that rely on these ontologies.

The apparent conflict between some of the the design principles — for instance, clarity and minimal ontological commitment — can be resolved by carefully scoping the application of the principles, as well as by defining small ontologies with clear scope and domain boundaries and harmonising them in a larger framework. Another important principle is reusability, which encourages the modeller to think outside the goals of any large framework, and allow the fundamental parts of an ontology to be used in other applications. This principle, as well as best practices for Linked Data publishing also suggest that one should attempt to reuse existing ontologies, and incorporate conceptualisations and ideas proved successful in other domains. To this end, it is useful to seek reference points, and review relevant areas such as ontologies created for describing creative works and multimedia documents. Among the few sources we are conceptualisations for various multimedia information management tasks, including the standards and ontologies outlined in the next section.

3.2 Conceptualisations of Music and Multimedia Information

In this section, we seek some reference points for the implementation of the framework for semantic audio information management outlined in the previous chapter. The discussion is not limited to explicitly conceptualised ontologies, but includes previously published standards, protocols or exchange formats in the areas of music and multimedia. We are interested in systems that can be used to associate media items and their content with different kinds of information, as well as systems for describing media production processes, workflows and provenance. We are also interested in cases where widely adopted standards fall short of certain design principles described in the previous section. This broadened view allows us to examine considerably different data models based on different needs, examine how they are related to the information management requirements in our framework, how much they overlap with each other, as well as our needs, and to what extent they could be reused.

3.2.1 Metadata standards

In order for applications or research tools to interoperate, some agreement has to be in place between developers, researcher and a user community. This agreement can take the form of an industry standard such as those published by the Moving Picture Experts Group (MPEG), or a highly adapted open framework, consisting of a conceptual data model and a formal description language.

Emerging metadata standards already proved invaluable in multimedia life-cycle management. For instance, content based search facilities can be provided using features defined by the MPEG-7 standard [Martínez, 2004]. The de-facto standard ID3 tag used in MP3 audio files empowers the online music community, despite its serious flaws in expressing basic editorial information (see Section 3.2.3). The usefulness of associating media items such as audio files with additional information often termed simply as *metadata* is obvious, however the questions *how this association should be made*, and *how metadata is represented and organised* present problems that are not easily solved.

Several academic and industrial organisations defined constructs ranging from simple data models and file formats through detailed standards to complex knowledge representation frameworks. They provide definitions and schemata in a variety of languages and syntaxes and define different ways of encoding information. The result is a large number of largely incompatible standards [Ossenbruggen et al., 2004; Smith and Schirling, 2006]. Since each addresses domain or sub-domain with a specific set of requirements, interoperability between applications is difficult, even if sharing otherwise overlapping information would be useful. We observe that the areas of music and multimedia information management exhibit a number of problems related to semantic and syntactic interoperability. In particular, many systems for describing *content* and *production* have little in common.

The problems are rooted in the lack of harmonisation in the development and publishing methodologies, and the lack of modularity and extensibility of the standards themselves. Most standardisation efforts stop at providing a data model, and then formalising how information is encoded according to that model. This may be for instance an XML based format, or the definition of a binary file encoding. However, standards rarely formalise the meaning of terms within a model, beyond a textual description in the specification document. See the comparison of XML Schema and Web ontologies in [Klein et al., 2001] for more details.

The lack of semantics, or in some cases the lack of even an implicit conceptualisation has far reaching consequences, which make us move away from reusing metadata standards.

Re-engineering existing standards in a Semantic Web ontology is tempting to solve these problems, since it enables harmonisation using a core or a foundational ontology. This is exemplified by the works of [Hunter, 2003] and [Arndt et al., 2007]. However, these efforts do not always solve all problems present in the original metadata model [Nack et al., 2005], leading to unnecessary complexity or the need for ground up design. The Music Ontology [Raimond et al., 2007] takes this approach, although it builds on previously published ontologies that fit its requirement. It provides a particular conceptualisation of the music domain described in [Raimond, 2008], which is largely void of the problems mentioned above. However, it does not in itself justify our decision for abandoning metadata standards in favour of the Music Ontology. In the following sections we discuss several frameworks and the kind of information they can represent. We also discuss their advantages and limitations, which provide interesting reference points for our work. First however, we outline the basic categories of information these frameworks can represent.

3.2.2 Basic categories of information about intellectual works

Metadata related to music, multimedia and intellectual works in general can be grouped into the following categories:

- **Bibliographic Information**: Editorial type information associating works with *creators*, and *publishers*, simple *subject headers*, and items with *location*, *condition* and other cataloguing information.
- **Cultural Information**: Social information pertaining to *consumption* and *appreciation* of works by different social groups, *relations between works or creators*, *intellectual property rights* and more generally, information that needs to be interpreted in a certain *cultural context*.
- **Content-based Information**: Particular *features* of content that facilitate organisation, search, navigation or similarity calculation. Features may result from *simple transformations* or complex *semantic analysis* resulting in representations that can be used to answer *meaningful queries*.
- **Provenance and Workflow Information**: A detailed description pertaining to the *origin* and *production process* of intellectual works, such as elements of the publishing and production workflow chains.

The boundaries between these categories however are not always clear leading to many different possible conceptualisations. For example, we can envisage a hierarchical system for content-based annotation of audio files blurring the boundaries between automatically extracted low-level features, and higher level musicological terms which often exhibit a strong cultural bias. In the following, we attempt to clarify the meaning of these categories by providing examples in various information management solutions, primarily focusing on music related information. This is followed by a brief overview of harmonisation efforts and frameworks covering many or all of these categories.

3.2.3 Bibliographic information

Bibliographic information can be defined as data pertaining to the history, physical description, comparison, and classification of books and other works [de Lavieter L. , Ed.]. In most metadata formats as well as complex frameworks the interpretation is usually narrower, and concerns only editorial type information such as the title and author associated with a work.

3.2.3.1 ID3

Perhaps the most commonly used metadata format for encoding bibliographic information in the music domain is ID3[10]. It defines a set of metadata containers (tags) originally designed to be included in MP3 audio files, but later adopted by other formats such as AIFF, WMA and MP4. Its first version ID3v1 provides a limited set of tags, for instance, *artist*, *album*, *title*, *year* and *genre*, each able to hold a fixed number of characters. The second, incompatible version ID3v2 relaxes this limitation by allowing variable length frames, and extends the type of information that can be included.

Unfortunately it is still very common that the ID3v1 tag set is the only metadata available in a music collection. ID3v1 has a strong bias for representing information about popular music, for example, it has no tags to describe a composer and a performing artist separately, making it highly unsuitable for describing classical recordings. Although ID3v2 introduces new elements for this purpose, the specification remains ambiguous. For example, the TPE2 frame may stand for *band*, *orchestra* or *accompaniment*. For these reasons ID3 is an example of a format which lacks coherence and extensibility. It is also limited to one level of description strictly related to audio items as identified by Raimond [2008] — that is, we cannot use it to provide information about a band or an album associated with a song other than their name or title.

3.2.3.2 DCMI

The Dublin Core Metadata Initiative (DCMI) maintains a basic set of ISO standardised metadata elements[11]. These include information such as *title, description* or *creator*, *language* and *rights* related to intellectual works in general or particular items. It is not specific to any

[10]ID3 audio tagging format: http://www.id3.org/
[11]Dublin Core Metadata Terms: http://dublincore.org/documents/dcmi-terms/

media, rather, it provides a cross-domain way of describing a wide range of resources. Dublin Core was among the first standard to recognise that one particular syntax specification would not fit all applications, hence no implementation is defined. However, most applications that use Dublin Core metadata are based on RDF, which brings the widest range of possible uses forth through its resource linking mechanism.

3.2.3.3 FRBR

The Functional Requirements for Bibliographic Records (FRBR) aims at providing a framework that identifies and clearly defines the entities of interest to users of bibliographic records, the attributes of each entity, and the types of relationships that operate between entities [Plassard, 1998]. It encompasses more than 40 years of development in the digital libraries community by building on previously published standards.

products of intellectual work	**creators** of intellectual work	**subjects** of intellectual work
work	person	concept
expression	corporate body	object
manifestation		event
item		place

Table 3.1: Three groups of entities defined in the FRBR model.

FRBR defines a set of entities divided into three groups as shown in Table 3.1. The first group depicted in Figure 3.1 describes products of artistic or intellectual works. This model includes entities ranging from abstract to concrete.
Work may stand for a poem, the lyrics of a song, or a classical composition. *Expression* represents a particular realisation that remains intangible and reflects artistic qualities, such as a recital of a musical piece, or illustrations in a book. *Manifestation* represents all the physical embodiments of an expression, that bear the same characteristics, with respect to both intellectual content and physical form, for example a book about the Semantic Web published in 2011. *Item* is the only concrete entity in the model, a single exemplar of a manifestation, for instance, a copy of the aforementioned book on my shelf, a compact disc in the collection of the British Library, or an audio file on my computer.

FRBR is particularly interesting as the above model provides useful concepts and relationships to describe the production workflow of intellectual works, however it is not sufficient in itself for this purpose. For example, we cannot express temporal relations between certain stages of the work. FRBR is available in RDF[12]. Its relationship to the Music Ontology is defined in [Raimond, 2008] and briefly outlined here in Section 4.1.

[12]Namespace for FRBR core concepts: `http://vocab.org/frbr/core.html`

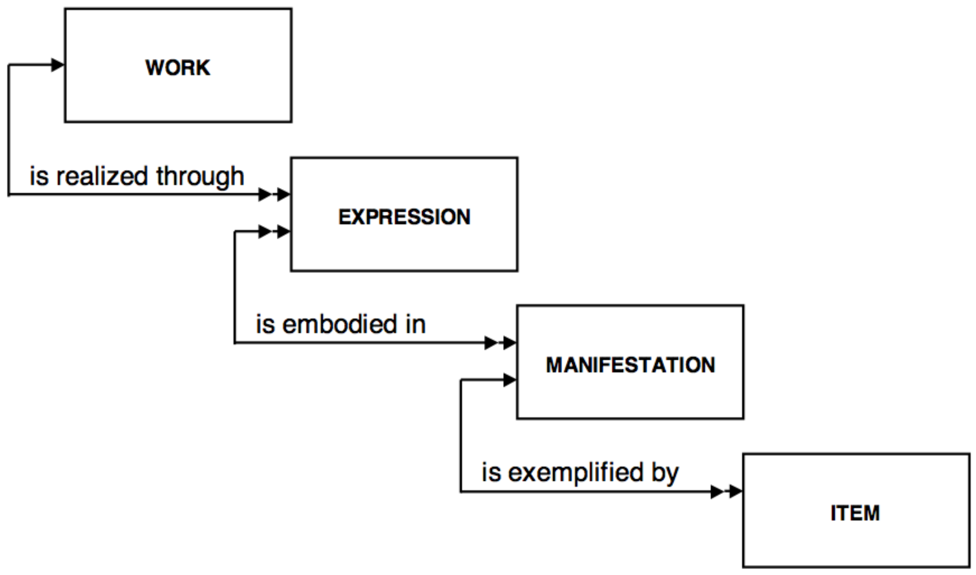


Figure 3.1: Entities related to products of intellectual works in the FRBR model [Plassard, 1998]. (Double arrows represent 1:N or N:N relationships.)

3.2.4 Cultural information

The meaning and boundaries of cultural information are perhaps the most difficult to define. We argue that it can be characterised by at least two distinct factors, *social information* related to the cultural environment such as the popularity of a work, and interpretations or high level *content semantics*, which exhibit a strong cultural bias.

3.2.4.1 Social information and content semantics

Pachet [2005] defines cultural information in the music domain as knowledge produced by the environment or culture. He mentions collaborative filtering, the analysis of online listening habits, as the particular source of such information. Although this information has undoubtedly cultural elements, we believe it is more precisely termed as *social.* According to the original definition, cultural information may be seen as all data that represents the status and content of intellectual works with regards to a particular environment and cultural context. We may easily argue for example, that musicological representations, such as a set of notes or chords, may only make sense in the context of a particular musical tradition or theory, such as the western tonal system [Cope, 1997]. In this sense, information bearing high level semantics related to content is deemed predominantly cultural.

3.2.4.2 CIDOC CRM

Perhaps the richest source for ontologies centred around the notion of cultural information is the museums and cultural heritage conservation community. Examples include the CIDOC Conceptual Reference Model (CIDOC CRM) a formal ontology intended to facilitate the integration, mediation and interchange of heterogeneous cultural heritage information [Crofts et al., 2010] or the MIDAS Heritage data standard[13] for the preservation of the knowledge about historic environments. These ontologies and standards are interesting for their conceptualisation of the temporal evolution of objects related to events or actions.

3.2.4.3 OntoMedia

OntoMedia is a Semantic Web ontology for the annotation of multimedia documents to capture and describe knowledge that is implicit in the context of the given media unit [Jewell et al., 2005]. Rather than focusing on requirements to represent information resulting from machine analysis of media, it aims at providing high-level semantics largely inaccessible for state of the art feature extraction tools. For example, using OntoMedia, we may describe character development in fiction, a type of information whose interpretation is dependent on cultural context. OntoMedia is strongly concerned with people and events, for these reasons it shows similarities with CIDOC CRM mentioned above.

A particularly interesting feature of the ontology is its event hierarchy and model. Subsuming a general *Event* concept, the concepts of *Gain*, *Loss*, *Transformation* and *Action* can be used to describe changes in the status of a character. For instance, a change in location (travel) is modelled as locational transformation. OntoMedia allows for the modelling of event chains — situations arising as a result of another — using ordering properties that are independent from the actual temporal context of events. Temporal associations are made through an extra layer provided by occurrence representations, which allow for the same event to occur multiple times, possibly on several timelines.

3.2.5 Content-based information

Describing the content of audio or media items for interoperability or facilitated search is the core objective of many information management solutions. As opposed to encoding the content for reproduction, the aim is to represent some aspects of it, such that the results can be transmitted, stored, identified and compared more economically, as well as queried on the level of their semantics. In this section, we outline some possible conceptualisations and organising principles of content-based information (often termed content-based features or descriptors) centred around different characteristics outlined in Table 3.2. We argue that these characteristics are paramount in the representation of knowledge about features, and

[13]MIDAS Heritage standard: `http://www.english-heritage.org.uk/`

are particularly useful in communication. First we discuss what they mean, and how they might be applied. Next, we contrast these organising principles with strictly hierarchical systems or taxonomies. Finally, we review some existing systems for the representation and encoding of this information.

3.2.5.1 Conceptualisations of feature representation

Content-based information, may represent anything extracted form the original material itself: an abridged textual description of prose, the temporal extents and locations of choruses in a pop song, or an object identified in an image. These features may be classified in a multitude of ways. Some ontologically relevant distinctions are shown in Table 3.2 (compare with Table 1.1 in chapter 2).

conceptualisation	**examples**	
hierarchical level	symbolic	sub-symbolic
	perceptual	physical
interpretation	cultural (contextual)	definitive (primary)
temporal characteristics	instantaneous	durational
data density	dense (compact)	sparse (scattered)
provenance	automatically extracted	manually extracted
computation domain	signal domain	transform domain

Table 3.2: Some ontologically relevant categorical distinctions of content based features

Although we are most interested in music related features, these categories are kept intentionally domain independent to support a more general view point. Here, we provide examples in the music domain, which is followed by a short discussion of alternative, taxonomical organisation.

Hierarchical level: The difference between symbolic and sub-symbolic representations can be seen in the music domain as distinction between musicological and acoustical features, for instance, a musical note or the fundamental frequency of a sound. In this context we call a representation “subsymbolic”, if it is made by constituent entities that are not representations in their turn, e.g., pixels or sound images as perceived by the ear, or signal samples [Schomaker et al., 1995]. This distinction is similar to classifying features into high and low level ones, however, these terms seem more ambiguous, as there is no common agreement on what is meant by high-level or low-level features. Symbolic features may also be called semantic, when they represent a feature that is meaningful according to some theory (e.g. western music theory), or represent a concept associated with higher level cognitive functions, rather than physical or perceptual phenomena like fundamental frequency or perceived pitch.

Another distinction can be made between features that express a directly measurable physical quantity and some perceptual quality (e.g. fundamental frequency and perceived pitch). From a musicological point of view, these features remain on the sub-symbolic level. We can similarly divide symbolic features further (e.g. notes vs. structural segments), ultimately leading to a hierarchical organisation, which may be encoded in a taxonomy. However we argue the usefulness of such taxonomical representation later in this section.

Interpretation: From the interpretation point of view, we may consider whether a feature is meaningful outside a cultural context or not. For example, a spectrogram or a chromagram resulting from a straightforward mathematical transform is definitive in the sense that its meaning is unconditional. A note name on the other hand assumes a specific tonal system. A note name cannot be safely assigned to a fundamental frequency extracted from the audio signal, without knowing for example the tuning system (temperament) used during the analysed recording. This distinction is closely related to Turney's investigation into the context dependency of features in machine learning and classification [Turney, 1996]. These differences are not commonly reflected in multimedia information management frameworks, which has far reaching consequences. We highlight the problem in Section 4.6, and outline a possible solution using a Semantic Web ontology.

Temporal characteristics: Regarding their temporal characteristics, we can differentiate features that describe a segment with a fixed duration from features that describe instantaneous events. One may question however, whether any event can be seen as instantaneous. This leads to philosophical problems discussed by Russell [Newton-Smith, 1984], and a number of theories regarding the representation of time [Hayes, 1996]. Different interpretations of a note onset is a good example to consider. In physics, only Planck time — currently held as the smallest measurable time interval[14] — may be considered instantaneous. It can be argued however that this has no relevance in music or multimedia, therefore instants are outside of the domain of discourse. From a practical point of view, it is useful to consider events that are *perceived* as instantaneous, reflecting a cognitive or perceptual bias in ontology design discussed in [Masolo et al., 2003b].

Data density: From a representational point of view, it is useful to consider the data density of features. Especially on the symbolic level, descriptors may represent objects that are spatially or temporally scattered with respect to the original content. Examples include note onsets, or structural segments. Other features, such as those resulting from straightforward mathematical transforms of a digital media items, are best described as dense data structures.

[14] http://astronomy.swin.edu.au/cosmos/P/Planck+Time

Provenance: In most applications it is useful to know the origin or *provenance* of a set of descriptors. Although many different sources are possible, perhaps the most useful distinction is between features resulting from machine analysis and features produced by human experts. Considering a knowledge base in scientific research for example, the latter may be used as ground truth to evaluate the former. It strongly depends on the application what the best way to conceptualise and represent provenance information is. In most cases, it is more useful to categorise the sources, rather than the features themselves. The notion of provenance is closely related to the notion of *trust*, since descriptors from one particular source may be more valuable than others. Provenance encoding is also related to workflows. An instantiated workflow template, or simply a description of how something was derived can be seen as a detailed account on provenance. Such a description may be provided using provenance models and corresponding ontologies such as the Open Provenance Model [Moreau et al., 2010] and its encoding in OWL. Complex provenance models are outside the scope of our present discussion, however we will revisit the problem in Section 3.2.6.3 in the context of workflows. A detailed overview of provenance encoding in ontologies is provided in [Ding et al., 2010].

Computation domain: We can differentiate between features based on their domain of computation. Most importantly, whether they are computed from the original raw data (signal) or some type of simplified or transformed representation of it. In text processing for instance, a tokenised or stemmed bag of words may be seen as the transform domain of a document, often used for computing features to derive document similarity. In audio signal processing, the time domain data is seen as the original signal domain, while the transform domain representation can be derived from a large number of different transformations, including conventional time-frequency transforms, and more recently sparse representations. Since certain types of features are often computed from a particular domain, it is tempting to use it as the main organising principle in feature classification, however, as we will see in the next section, this may lead to a rigid knowledge representation, as it is very often possible to derive features with the same or very similar meaning form two or more different domains.

3.2.5.2 Taxonomies

There are many alternative ways content based features may be organised. A deeply hierarchical ontology may be based on how features are computed and attempt to encode their hierarchical dependencies. The use of signal processing and machine learning techniques in the extraction process can also be an organising principle. A taxonomy of audio features based on these ideas is presented in [Mitrović et al., 2010], aiming to improve on work by Tzanetakis [2002] and Peeters [2004].

The highest level of this taxonomy is defined by the transform (or signal) domain (*time,* *frequency, ceprstral, eigen, etc...*). This is followed by further dichotomies such as *physical* or *perceptual*, however the choice of transform domains where this distinction is actually reflected remains ad-hoc. It is unclear whether the overall organising principle for the taxonomy is the transform domain, the semantic meaning of features or the extraction process. This ambiguity is apparent in the placement of particular features under certain branches of the taxonomy tree. For example, it blurs the concepts of fundamental frequency and perceived pitch, yet it does not reflect the various methods that use different domains as starting point to compute these features.

We argue that strict taxonomies do not help in clarifying the meaning of features. They lead instead to rigid, application specific knowledge representations, which cannot reflect the heterogeneity of computations and results. They are inappropriate in many applications, as they cannot accommodate for the conflicting views and needs of researchers and developers. See for instances [Hepp and de Bruijn, 2007] for a detailed discussion on how ontologies are related to inconsistent taxonomies and how these problems may be resolved. In our opinion, issues regarding computation are perhaps best encoded as scientific workflows (see Section 3.2.6.3). We argue that the domain independent conceptualisations listed in Table 3.2 are the most useful for communication, as they adhere to most design principles such as extensibility, minimal ontological commitment and minimal encoding bias.

Different ontologies and standards exist, however, that commit to a diverse set of organising principles. The MPEG-7 standard (described next) presents a hierarchical model, separating low-level descriptors from high-level annotations. The Audio Features Ontology published as part of the Music Ontology framework (see [Raimond, 2008] and also Section 4.3 in this work) differentiates between sparse and dense features on the highest level. In the following, we outline some commonly used frameworks to represent content based features of audio.

3.2.5.3 MPEG-7

The Multimedia Content Description Interface (MPEG-7) [Martínez, 2004] is arguably the most well known standard for the annotation of media content that includes still images, audio, video, audiovisual items, as well as rich multimedia presentations. Providing an XML-based metadata framework for high and low level content description appears to be its core objective. It also contains elements to encode—at least to some extent, editorial and production information (content management), as well as data related to storage, transmission, navigation, access and user interaction.

MPEG-7 provides a rich set of *descriptors* (**D**) to define how individual features are represented in a document, and *descriptor schemes* (**DS**) that specify the structure of descriptors and other descriptor schemes. In the MPEG-7 Audio Framework in particular, we find the low-level feature categories shown in Table 3.3.

Concrete descriptors include *AudioSpectrumEnvelope* which represent a logarithmic-frequency spectrum, or the *AudioSpectrumBasis* and *AudioSpectrumProjection* descriptors which describe basis functions derived from the singular value decomposition [Golub and Kahan, 1965] of a normalised power spectrum, and low-dimensional features of a spectrum after projection upon these bases. The MPEG-7 Audio Framework — exhibiting a predominantly hierarchical structure — also provides high level tools. These allow the representation of fingerprints, musical instrument timbre, or monophonic melodic information, using, for instance, the *MelodyContour*, or *MelodySequence* descriptor schemes.

category	**some examples**
Basic	AudioWaveform, AudioPower
Basic Spectral	AudioSpectrumEnvelope, AudioSpectrumCentroid
Signal Parameters	AudioHarmonicity, AudioFundamentalFrequency
Timbral Temporal	LogAttackTime, TemporalCentroid
Timbral Spectral	HarmonicSpectralCentroid, HarmonicSpectralSpread
Spectral Basis	AudioSpectrumBasis, AudioSpectrumProjection

Table 3.3: MPEG-7 Low level audio tools (D and Ds)

The usefulness of MPEG-7 descriptors in musical applications has already been noted during early developments of the standard. See for example [Casey, 2002] and [Kim et al., 2005] where several applications and extensions are discussed. However, later adaptation remains low especially in terms of real-life applications. Nack et al. reported the lack of real-life applications in 2005, four years after the initial publication of MPEG-7 [Nack et al., 2005]. To the author's knowledge, there are still very few applications, and reviewing the relevant scientific literature to date suggests that the standard is not widely used in research anymore. However, a few important conclusions can be drawn from early applications.

MPEG-7 provides a closed set of descriptors. Although it was designed with extensibility in mind — it provides a language, the MPEG-7 *Description Definition Language* (**DDL**) to define additional descriptors and descriptor schemes — the need for using the standardised descriptors in order to remain compliant [Casey, 2002], is a serious constrain on extensibility. The standard also defines certain computational steps associated with descriptors, but allows for variability without providing sufficient tools for describing what exactly was computed. This hinders interoperability, and obscures the aim of the standard, making it difficult to decide whether its primary aim is knowledge representation or data exchange. We mentioned its broad goals, and the diversity of data the standard aims to represent. However, this is provided without an explicit conceptual model which would tie its descriptors into a common framework. A number of additional concerns were reported for instance in [Hunter, 2001; Ossenbruggen et al., 2004; Troncy et al., 2004; García and Celma, 2005; Nack et al., 2005; Troncy et al., 2007] and [Raimond, 2008]. We summarise these concerns in the following:

Lack of formal semantics: The lack of formal semantics of MPEG-7 descriptors and descriptor schemes is perhaps the most serious problem, a direct consequence of the use of XML as basis for the standard. XML can only represent a document's hierarchical and syntactical structure, it is insufficient in itself to communicate semantics in a machine-processable way. Moreover, as pointed out in [Troncy et al., 2007], MPEG-7 allows diverse syntactic variations to be used with the same intended semantics while remaining valid under the standard.

Non modular schema language: The MPEG-7 DDL extends the XML Schema Language. It only provides however the extensions needed to define higher dimensional data structures (vectors and matrices), but no support for modularity or the definition of semantic relations. Although the schema language is provided, the structural and syntactical definitions that should be part of the language are defined instead as part of concrete schemata. These two factors make the reuse and integration of descriptors, descriptor schemes and the metadata itself outside the context of the standard very difficult, especially in applications like ours, which cut across several domains, and where content description is only a small part of the problem space.

Additional concerns: Without further details, some other problems we find important to mention include the lack of linking mechanism between certain descriptors types and the content, the monolithic structure of MPEG-7 documents, and the inadequate reasoning services provided by the standard. In the context of the Web, Raimond [Raimond, 2008] mentions the difficulty of mixing vocabularies from heterogeneous domains, the inability to use URIs as identifiers, and finally the lack of a conceptual model for complex music-related information. Some of these concerns led to harmonisation efforts described for instance in [Hunter, 2003; García and Celma, 2005], and [Arndt et al., 2007] which provide mappings of MPEG-7 elements to Semantic Web ontologies mixed with other ontologies. However, they neither offer an efficient solution, nor do they solve the fundamental problem of lack of semantics in the standard specification itself. We will discuss these works in Section 3.3.

Although some of the above concerns may be resolved for specific applications using MPEG-7 profiling [Troncy et al., 2010], we can conclude that from a structural point of view, the use of the standard provides little benefit over simpler frameworks such as the Multimedia Contents Description Language (MCDL) [Furini, 2007] or the Synchronized Multimedia Integration Language (SMIL) [Bulterman et al., 2008].

3.2.5.4 SDIF

The original aim of creating SDIF (Sound Description Interface Format) [Wright et al., 1999] was the efficient binary representation of frame-based spectral data in an analysis/synthesis framework. However, this was soon extended to include more complex types and higher level features. Although this format is available in existing audio applications, unfortunately its rigid specification is highly particular to its original purpose. This compromises semantics and extensibility. Moreover, its binary representation is not well suited for searchable persistent database storage. Therefore this format cannot ideally be used in a general information management framework.

3.2.5.5 ACE XML

The goal of ACE XML [McKay et al., 2009b] is to meet the representational needs of MIR research in general, and automatic music classification in particular, with maximum extensibility and simplicity. For these reasons, the utilities of this format has to be considered in our framework. Close examination of its design choices and their consequences reveals however that the format works best for the particular use cases of music classification described in [McKay et al., 2009a] and [McKay et al., 2005], rather than as a general purpose exchange format, or an application outside of ACE XML's originally defined context.

Rather than defining a monolithic single document format, ACE XML is a framework, a collection of multi-purpose file formats developed for the jMIR package. It enables communication between jMIR components such as the jAudio [McEnnis et al., 2005] feature extraction library and the ACE classification engine. Within this framework, we find five types of files, for instance, *Feature Value Files* describing feature values extracted from audio items, or *Feature Description Files* describing abstract information about the features themselves. A notable benefit of this design is the ability to reuse feature files in different classification tasks. However, it makes the use of the framework cumbersome in data exchange due to the issue arising from the need for parsing and interpretation of different files formats, as well as keeping related files together. The use of *Project Files* seems more like a workaround rather than a satisfying solution for this purpose.

ACE XML is defined using the XML Document Type Definition (DTD) Language. This choice is justified by two design goals: simplicity and human readability. The language however is very limited, allowing only the syntactic structure of XML documents to be defined. Compared with the XML Schema language, it leaves even data typing issues aside. Consequently, when reading ACE XML documents, fetching data types from separate feature description files has to be dealt with explicitly by the user, as opposed to be part of the standard parsing process. The choice of DTD as opposed to the XML Schema language requires the understanding of two, albeit simple, languages. Compared to more recent serialisation formats of RDF such as Turtle (see Section 2.4.2.3), these languages are less human readable, and provide no way to express an explicit conceptual model, while Turtle can be used to represent both an ontology and data in a single, more compact and more easily readable format.

ACE XML provides flexibility and extensibility on the expense of not specifying an ontology or a conceptual model. This leaves the issues of semantic interoperability entirely to the user. It may be argued that such interoperability is not required in all circumstances. However, we may ask in this case what is the benefit of using ACE XML outside the context of jMIR, apart from the use of standard XML parsers. To highlight this issue, we may imagine the case of combining two large projects containing arbitrarily defined features sets. Since the format does not specify an explicit model of feature representation, mapping the feature files into a common format is far from trivial if at all possible, and may require linguistic analysis of elements of feature definition files, which clearly defeats the purpose of a machine-processable file format. Although version 2 of ACE XML allows the association of URIs with instances within feature files and feature descriptions, this is not a requirement, hence retroactive conversion of projects is made very cumbersome. The overall conceptualisation implied from the different file format definitions shows that it largely remains on the level of audio items. It does not deal with more complicated production workflows, and provides only non standard ways to encode cultural or provenance information.

Albeit ACE XML is very well designed, complete, and provides the required flexibility for diverse classification tasks with potentially unforeseen requirements. Given the problems above, and general concerns regarding XML-based formats mentioned earlier, we conclude that it isn't suitable for our framework.

3.2.6 Provenance and workflow information

Describing subtle details about the origin and production of intellectual works, rather than simple bibliographic information, pertains to describing their provenance and production workflow. We are interested in, for example, how a piece of music was produced in the recording studio, or what facilities were used in the production chain. Numerous models and systems have been developed to describe workflow and provenance information, however the interpretation of these concepts may be very diverse in different communities and application areas. In this section, we first provide a brief outline and attempt to clarify the interrelation of workflow and provenance, before proceeding to review the state of the art for capturing workflow and provenance information related to audio signal processing and audio engineering.

3.2.6.1 Provenance

Provenance may simply mean *'source of origin'* (of an artefact) as it is often used by archeologist. This simple meaning is not uncommon in the context of the Semantic Web, when we are only interested in the source of a statement, such as a particular database, or a person or method responsible for the assertion. The RDF specification described in Section 2.4.2 allows for making statements about statements — a process called *reification* in knowledge representation — in order to facilitate the attribution of a statement to a particular resource. The DCMI vocabulary mentioned in Section 3.2.3 defines the property `source`, to describe that a resource may be derived from a related resource.

A more complex interpretation, often used in computing as well as by historians and museums, is *'chain of custody'* or origin to present. DCMI supports this level of provenance expression by defining the concept `ProvenanceStatement` and the property `provenance` to allow describing changes in ownership and custody of a resource since its creation.

The e-Science community [Taylor et al., 2006] in turn is typically interested in *'process provenance'* — that is, the full execution history of processes which were utilised to compute some data. In this sense, provenance encoding is a specific type of workflow encoding, and a crucial component of workflow systems. The Open Provenance Model [Moreau et al., 2010], described in the next section, is a prime example focussing on process provenance. The majority of complex provenance encoding systems however are also concerned with some kind of denotation of process and data dependencies. The Proof Markup Language (PML) [da Silva et al., 2006] for instance builds on proof theoretic foundations to describe processes such as logical reasoning and data mining. The justification of a conclusion can be expressed using the concept `InferenceStep` linked to rules using the property `hasRule`. Rules are denoted by the term `InferenceRule`, and can be linked to variables through variable bindings. A chain of dependencies may be expressed using the properties `isConsequentOf` and `hasAntecedent` linking inference steps with antecedents and a consequent. Other provenance ontologies for describing information manipulation include the Provenance Vocabulary [Hartig and Zhao, 2009], the Provenir Ontology [Sahoo and Sheth, 2009], and the Workflow Driven Ontology (WDO) [da Silva et al., 2007]. WDO extends the PML provenance ontology to represent workflows in an abstract way, describing a plan for a workflow without an execution bound to concrete services. It uses the concept `Method` aligned with PML's `InferenceRule` to represent the class of actions to be executed, and the concept `Data` to represent the information to be operated on by an action in the workflow.

The World Wide Web Consortium (W3C) is also interested in the problem of provenance encoding, and created and incubator and later a working group[15] to investigate technical requirements for provenance encoding on the web. This group issued a report, but no candidate recommendation yet.

We saw from the above discussion, how complex interpretations of provenance are directly related to the denotation of workflows. In the next section, we describe a concrete provenance model, then we examine workflow models and workflow representation systems.

3.2.6.2 Open Provenance Model

The Open Provenance Model (OPM) [Moreau et al., 2010] is a general purpose knowledge model for the description of workflow or process provenance and information manipulation. OPM's language independent specification is synchronised with an OWL ontology. Similarly to other provenance ontologies, it defines three core concepts: artefacts, processes and agents. An *artefact* is defined as an *immutable piece of state* which may refer to an actual physical object, a digital representation or some digital data. A *process* represents an action that

[15]W3C Provenance Working Group: http://www.w3.org/2011/prov/wiki/Main_Page

creates artefacts, either by acting on an existing artefact or by creating a new one. *Agent* describes an entity involved in a process by enabling or controlling its execution. The basic relationships between the concepts defined in OPM is shown in Figure 3.2, reflecting a process orientated view. The studio ontology described in Section 4.2 borrows from this model for describing provenance in audio engineering workflows, however it has a data centric rather than a process centric conceptualisation.

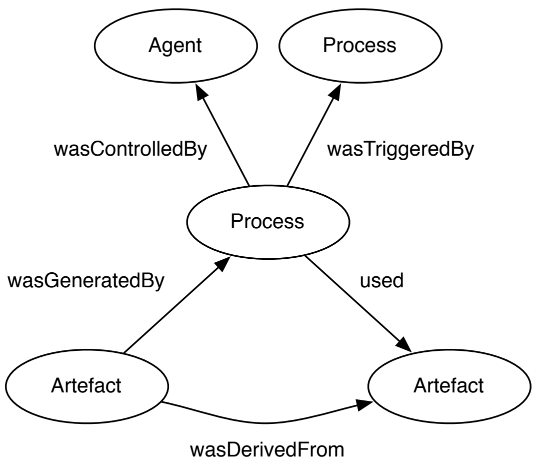


Figure 3.2: Basic relationships in the Open Provenance Model

3.2.6.3 Workflow

The concepts of workflow and workflow management have attracted interest in the business and more recently, in the e-Science communities. In fact, workflow management is considered as a predecessor, and an important part of Business Process Management (BPM), concerned with the automation of business processes using distributed (Web) services and procedural rules [Taylor et al., 2006; Ko et al., 2009]. Although BPM yielded a wide variety of tools and languages, having additional goals and requirements that are unique to science, the e-Science community adopted BPM standards only partially, focussing on workflow systems that facilitate the reproducibility of scientific experiments.

Workflows are relevant in the context of both MIR research and audio engineering. In MIR research, workflow can be a useful tool for reproducibility and sharing, but also in an attempt to generalise music analysis algorithms that are predominantly developed in a highly specific context. In audio engineering, capturing the workflow pertaining to the production of individual audio items is the first step in sharing audio engineering know-how, and may lead to systems that enable the reproduction of audio processing steps. Additionally, workflow or process provenance provide invaluable data for music information management tools.

Workflow in itself has many different meanings however. A broad review of the literature (e.g. [Taylor et al., 2006; Ko et al., 2009; De Roure et al., 2009; Gil et al., 2007; Ding et al., 2010; Moreau et al., 2010; da Silva et al., 2007; Goble et al., 2003]) reveals the following interpretations, with representations existing on all three levels:

- **Workflow plan**: *A plan or template describing abstract data access and manipulation steps in a domain specific but service independent way.*
- **Workflow execution**: *An instantiated workflow template, or a description bound to specific data items and service end-points.*
- **Workflow description**: *A denotation of a workflow instance with participating Agents, Processes and Artefacts (data items). This can be seen as a record of provenance.*

General workflow plans as well as workflow instances that describe a specific computational process can be seen as *prescriptive*, in the sense that they are to be interpreted and executed by a computational agent or workflow engine. See for example the Scufl language used in the Taverna [Oinn et al., 2004] graphical workflow modelling system, the Meandre workflow engine and its ZigZag language [Llorà et al., 2008] or the N3-Tr framework proposed in [Raimond, 2008]. This framework extends the N3 model [Berners-Lee et al., 2008] discussed in Section 2.4.6 with concurrent transaction logic ($\mathcal{CTR}$) [Bonner and Kifer, 1996], and represents workflows essentially as logic programs, using *built-in* predicates to represent computational services.

A denotation of a specific computational process can be seen as a *descriptive* workflow, also called workflow provenance or process provenance. It was shown that prescriptive workflows can be easily translated to provenance graphs or the other way around. For example, Missier and Goble [2011] describe Taverna workflows using the Open Provenance Model (OPM) [Moreau et al., 2010], and also translates OPM graphs to Taverna workflows. Since we are interested in systems for describing audio production workflows, we review some relevant workflow systems, as well as metadata models that contain elements for workflow description in the sections that follow.

3.2.6.4 Workflow systems

Workflow is commonly equated to *experiment* in the context of e-Science. In the $^{\text{my}}$Grid project [Goble et al., 2003] for instance, it is used as a synonym for experiments where database access and computational analysis is used, instead of hypothesis testing in the laboratory.

$^{\text{my}}$Grid builds an architecture for such *in silico* experiments in biology by binding command line tools and scripts exposed using Simple Object Access Protocol (SOAP) Web services. The project is pioneering the use of Semantic Web technology in service and workflow discovery, metadata management and service registers. The use of RDF however mainly concerns metadata management for provenance tracking and workflow linking, but workflows themselves are treated as black-boxes form the semantics point of view. They are expressed in a dedicated XML-based language: the simple conceptual unified flow language (Scufl), developed for the Taverna [Oinn et al., 2004] graphical workflow modelling application. Sculf represents each step within a workflow as an atomic task. It can be used to describe processors, data links, and coordination constraints, linking two processors and controls their execution. However, atomic processes in workflows are not uniquely identified or interlinked in a distributed way.

Similarly to Taverna, Trident [Barga et al., 2008] features a graphical workflow editor. This system is aimed at scheduling and executing scientific workflows in high performance clusters with possible Web-based deployment. It features a sophisticated workflow control system with primitive execution patterns such as sequential, split, merge, synchronisation, and choice. Trident offers two scheduling strategies: *cache aware scheduling*, to minimise data transfer or *schedule aware caching*, to optimise the order of execution and processing expense.

While the above mentioned systems focus on facilitating the creation of workflow templates and the execution of workflows over heterogeneous services, $^{\text{my}}$Experiment [De Roure et al., 2009] focusses on reuse and social aspects of both workflow templates and workflow instances, deployed over a service oriented architecture (SOA). It essentially provides a virtual research environment for social sharing of workflows originating form different, disjoint workflow systems. Key design decisions in $^{\text{my}}$Experiment include the support for *federation*, by providing an ability to link workflow instances, an Application Programming Interface (API), that enables *embedding functionality* in existing interfaces, *research objects*, that collect data, results, provenance, tags and documentation, and *experiment objects*, for exporting data using RDF named graphs [Carrolla et al., 2005]. The metadata available about these workflows can be accessed via a SPARQL end-point.

The Networked Environment for Music Analysis (NEMA) [West et al., 2010] project is an effort parallel to $^{\text{my}}$Experiment in many respects. It aims to implement a similar workflow sharing approach, possibly with joining the two services. NEMA addresses two fundamental problems in music research: the unavailability of large music data sets due to copyright issues — a problem first addressed in [McEnnis et al., 2006], and the heterogeneity of programming languages, algorithms, and data exchange formats used within the community. Hence, the project focuses on building a distributed workflow environment to facilitate computation over remote audio and resource collections, interoperability between data formats and types, and sharing evaluation procedures. NEMA builds on the foundations of previous projects such as Music-to-Knowledge (M2K) [Downie et al., 2005], a prototyping and evaluation platform which allows for the specification of data flows operating on multimedia data, but without the ability of sharing or distributed execution, and the On-demand Metadata Extraction Network (OMEN) [McEnnis et al., 2006], a distributed feature extraction system which is built by coordinating instances of the jAudio [McEnnis et al., 2005] feature extractor engine running on different machines.

NEMA itself extends these ideas by adding support for workflows related to MIR experiments, which include feature extraction as well as training, applying and evaluating machine learning models against different data sets. NEMA primarily uses ACE XML [McKay et al., 2009b] with a view of extending its data model to incorporate Semantic Web ontologies developed in the OMRAS2 project [Fazekas et al., 2010]. Workflows are executed using the Meandre workflow engine [Llorà et al., 2008] and denoted using ZigZag, its dedicated flow language. This language may use Web URIs as identifiers of workflow components facilitating the creation of a distributed system. However, due to the concerns discussed in Section 3.2.5 regarding the present use of ACE XML for data communication, the system does not enable interoperability between heterogeneous systems, or the creation of inter-framework workflows. Although it allows semantic annotations of Meandre workflows, workflow components themselves are not described, i.e. inputs, outputs and configuration are not communicated in a more interoperable format like RDF. Even provenance information is stored using a proprietary package. Workflows are still treated as black-boxes, without a shared conceptualisation of workflows and workflow components.

While the NEMA approach is well suited to evaluate the most common MIR algorithms, not concerned with using multiple resources (i.e. cultural information obtained from the Semantic Web), a more transparent system utilising fine grained workflow description and execution, (perhaps facilitated by a common data exchange and workflow language), and the ability to tie in several, potentially unknown systems in the execution, supported by a common ontology and uniform resource identification, would be desirable in testing future MIR algorithms.

Two systems developed in OMRAS2 move towards these goals. One is the N3-Tr framework mentioned previously, and its implementation: Henry, described in [Raimond, 2008]. This system focuses on automated execution of music analysis workflows on the Semantic Web, using a SPARQL end-point. The other is our SAWA framework [Fazekas et al., 2009] described in Section 5.3. This system features a Web based user interface, and the ability to express algorithm configuration as well as the returned features within the same ontological framework described later in this chapter. This framework also includes an ontology for the transparent denotation of signal processing workflows, see Section 4.2. Although the aims and complexity of these systems are not comparable to NEMA on the whole, they provide a model for building truly semantic systems.

There are many other applications in the music domain that are related to the execution of music processing or music signal analysis workflows. Marsyas [Tzanetakis and Cook, 2000] provides a framework for distributed music processing [Bray and Tzanetakis, 2005]. CLAM [Amatriain, 2007] features a graphical music analysis workflow editor and execution engine. Pure Data (PD) [Puckette, 1997] provides similar facilities for real-time audio synthesis and signal processing, and commercial tools like MAX/MSP[16] and Reason[17] (for an analysis of commercial music software see [Duignan, 2008]) exists, geared toward the needs of creative professionals. However, these systems are closed in many sense. Lacking a common conceptualisation of audio processing components, configuration data and exchange formats, they cannot communicate with each other. They do not provide information on process provenance in an interoperable way, as their internal workflows are coupled with implementation details, without abstract workflow models. This loss of information aggravates the problems of experiment reproducibility and code reuse in MIR research, and the lack of production information both in MIR and audio engineering applications.

In this present work, we are concerned about descriptive denotation of workflow provenance rather than workflow execution. Next, we outline some frameworks and standards that contain metadata elements for this purpose.

3.2.6.5 AES31

The AES31 standard of the Audio Engineering Society (AES) is concerned with network and file transfer of audio as well as metadata storage and transmission. It includes the latest extension to the Broadcast Wave Format (BWF-E) defined in AES31-2. In our particular interest, the recently revised third part of the standard AES31-3[18] encodes information related to audio workflows.

AES31-3 defines a simple textual markup language called Edit decision markup language (EDML), which — although strictly speaking not related to XML as none of the standard definition languages such as DTD or XML Schema are used — results in a simplified XML-like document when implemented. An EDML compliant document is divided into sections identified by start and end tags such as `<EVENT_LIST> ... </EVENT_LIST>`. Each section may contain a number of entries starting with a specific keyword. The core of an EDML document is the Audio Decision List (ADL) section, which contains some project and system specific information, and enables the restoration of the track and clip composition of a project, as well as fades, transitions and to some extent automation. Within an ADL header section, we can refer to the source audio material used in a project, and define the possible destination tracks. The clip composition is described using a number of edit events paced in the *event list* section of the ADL. This however is limited to two types of events, *Cut* or *Silence*, corresponding to placing some audio source material on a destination track, or defining a silent region without a specific source. A Cut entry may look like as follows:

```
(Entry) 0010 (Cut) I 1234 1 2 03:00:00:00/0000 01:00:00:00/0000 01:00:10:00/0000 _
```

[16]A visual signal processing environment: http://cycling74.com/products/maxmspjitter/
[17]A virtual studio environment: http://www.propellerheads.se/products/reason/
[18]AES31-3-2008 Standard: http://www.aes.org/publications/standards/search.cfm?docID=32

The above entry denotes the source index of an audio file, source and target channel designations, and specifies three time codes in time-code character format (TCF) to provide sample-accurate position and synchronisation information. These are used to index into the source audio file, and to identify the start and end of the corresponding audio clip on the destination track. Similar data structures may be used to describe fades, cross-fades, and automation of volume fader and pan parameters for spatial positioning.

The simple data model of AES31-3 outlined above only provides for encoding the state of a recording project, so that it can be saved and restored. There is no explicit conceptualisation of audio clips, events and timelines assumed by the standard specification, therefore we cannot attach additional information to events, or link them with different timelines to denote a sequence of modifications in a production workflow. The main motivation behind AES31-3 is to facilitate simple project transfer. It was published in text, without the use of a definition language, therefore its semantics is highly intermingled with its syntax specification. This makes it difficult to implement, reuse or extend. Although AES31-3 influenced our ontology designs detailed in Section 4.2, due to its overly simple conceptualisation highly optimised for project transfer, we could not directly map or reuse concepts defined by this standard in our ontologies.

3.2.6.6 AAF and OMF

The Advanced Authoring Format[19] (AAF) is in our concern because of its ability to describe media production workflows. It is the successor of Open Media Framework (OMF) and considered to be a superset of the AES31-3 standard. AAF specifies an object-orientated class model for interchanging audio-visual content as well as associated metadata. This maps well on the typical class hierarchy of application storage containers. For extensibility, the format specifies Meta-Classes which can be included in Meta-Dictionaries and sent along with an AAF file. Yet, the specification contains statically coded information such as media types. Therefore it fails to support a dynamically extensible system. Most extensions would require generating and recompiling source code. This does not permit easy adaptation to user's needs in situations such as describing new signal processing elements of a production system. The rigid data model consisting of closely joint objects and properties, and the typically used binary encoding, do not support the development of query interfaces to assist workflow management, nor does it support automated inferencing on previously computed and stored metadata. Building an interlinked database over various facilities would also be problematic. Finally, the supplied SDK confines monolithic, single language implementation. These drawbacks led to the need for mapping AAF terms onto XML based vocabularies and data encoding [Beenham et al., 2000] types. However, the format is still lacking semantic associations, therefore the afore-mentioned problems are not fully solved.

[19]AAF specification and supporting documentation available at: www.aafassociation.org

3.2.6.7 IXD

The Integra Extensible Data (IXD) [Bullock and Frisk, 2007] format was developed within the Integra project for the libIntegra library. This software library provides interlinking and persistent storage facilities for audio processing and live composition environments (such as PD and Max/MSP) in a software independent manner. It is based around the concept of multimedia module encapsulating signal processing functionality. The IXD format appears to be well designed and well suited for representing module information for the purposes of storing module states and parameters in the local file system or in a remote database. It provides a good example of an XML-based schema which doesn't entirely overlook semantics and ontological considerations. For instance, it is possible to define hierarchical inheritance relations between modules. However, it is not possible to do so in defining module attributes, closely tied with module definitions in an object-orientated manner. Although the format moves in this direction, it does not permit the definition of semantic associations rich enough to be used in a more generic way.

3.3 Metadata harmonisation using core and foundational ontologies

There exists a conflict, inherent between certain ontology design principles and best metadata management practices, and some goals fundamental to the Semantic Web and heterogeneous information management systems in need of a universal knowledge representation. In particular, the modularity principle, discussed in Section 3.1.4, serves us well in avoiding unnecessary complexity when describing a small specific domain. It also reduces the temptation to represent entities in an ontology that are more fundamental then what is required by the granularity of the resulting data in a target application, thus creating multitudes of universal ontologies. Furthermore, it is desirable that metadata practitioners focus on a specific domain of expertise, since no single person or community is able to create *the* universal ontology for the Semantic Web.

Foundational ontologies do exist however (see Section 3.1.4), which tend towards a universal status, regardless of the philosophical controversies around them discussed for instance in [Floridi, 2004]. In brief, it is disputable whether a universal ontology accepted by all communities can ever be created. Information management solutions which have to deal with parts or all of the different kinds of information discussed so far in this chapter, ontology alignment tools, and metadata model harmonisation provide the best cases for the need of foundational and core ontologies. In this section, we outline some solutions which aim at facilitating interoperability between multiple metadata standards by developing core ontologies.

3.3.1 ABC

The ABC model [Lagoze and Hunter, 2002] is the result of harmonisation between the digital libraries and museum communities. It takes several metadata packages such as the earlier mentioned FRBR [Tillett, 2004; Plassard, 1998] and CIDOC/CRM [Crofts et al., 2010] models into account. However, rather than attempting to build a universal ontology, it simply aims to find the most frequently occurring set of entities and relationships (e.g. events, places or people), that can be considered domain independent. ABC is intended to facilitate machine interoperability as a conceptual model. It is not concerned with the implementation language or data interchange syntax, but it builds fundamentally on RDF, and provides trivial mappings to RDFS and OWL ontologies[20]. The main goals of ABC can be summarised as follows (see [Lagoze and Hunter, 2002] for a more detailed discussion):

- **Guide ontology analysis or design**: *Provide a base vocabulary that can be used in the analysis of existing, or the development of new community-specific vocabularies.*
- **Facilitate metadata mapping**: *Provide a common logical model, that is more interoperable than one-to-one mapping between specific domain ontologies.*

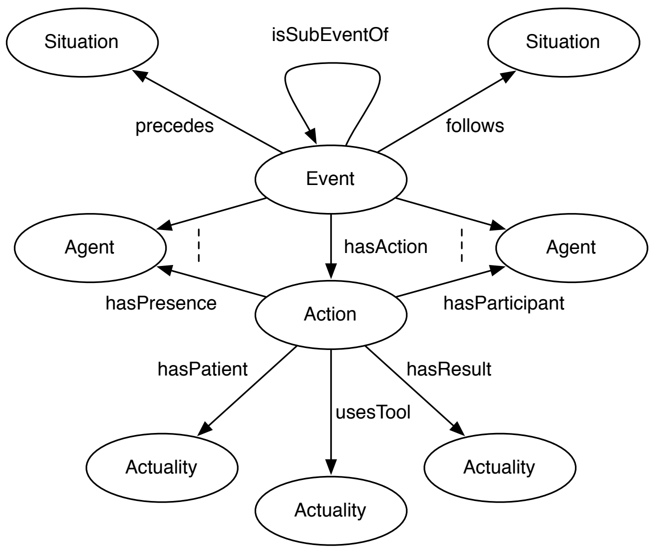


Figure 3.3: ABC model showing terms related to situations events and actions (partial).

The core of ABC's model are *i)* a layered conceptualisation of the life cycle of intellectual works (i.e. books, works of art, music, etc...), and *ii)* an event model centred around three concepts: *situation, event, action*, as shown in Figure 3.3.

[20]http://metadata.net/harmony/ABC/ABC.rdfs and http://metadata.net/harmony/ABC/ABC.owl

ABC's conceptualisation of the life cycle of intellectual works is similar to the FRBR model described in Section 3.2.3. It includes entities ranging from abstract to concrete such as *work*, *manifestation*, and *item*, however, it leaves the *expression* layer out, which turns out to be of paramount interest in describing music production workflows as described in [Raimond, 2008]. ABC considers events as transitions between situations, linked with actions performed on *actualities* (agents or physical artefacts). This can be used to describe for example the transformations of an item. Using the property `hasPresence`, events or actions may be linked with agents that are simply present during an event. Agents that actively participate are linked using `hasParticipant`, a subproperty of `hasPresence`.

With regards to the intended use of ABC mentioned above, the first kind of usage is demonstrated by the Music Ontology for instance, in that it borrows from ABC's event-based conceptualisation for its model of music production workflows by refining and simplifying this model (see Section 4.1 and [Raimond, 2008]). When describing audio transformations, we also use a model similar to how actions perform transformations in ABC, but rely instead on refined concepts from the Music Ontology. The second usage is shown in [Hunter, 2003], fusing a museum ontology (CIDOC/CRM), multimedia content description (MPEG-7), rights management (MPEG-21), and a biomedical ontology (ON9.3) under ABC's common framework, based on principles of an event-aware integration model described in [Hunter and James, 2000].

3.3.2 DOLCE and COMM

The Core Ontology for Multimedia (COMM) [Arndt et al., 2007] is one of the many approaches which aim to reuse MPEG-7 to create a harmonised multimedia ontology for Semantic Web applications, and alleviate the problems of the standard discussed in Section 3.2.5.3. As opposed to alternatives proposed in [Hunter, 2001, 2003], [Tsinaraki et al., 2004], [García and Celma, 2005], based on one-to-one mapping from MPEG-7 descriptors to OWL entities, COMM uses a completely re-engineered version of MPEG-7. This was required to fully capture the intended semantics of the written standard, and to avoid interoperability problems resulting from the many different ways it provides to encode the same information.

Similarly to the way Hunter extends ABC to harmonise her MPEG-7 ontology with other domain ontologies [Hunter, 2003], COMM enables harmonisation using a foundational ontology called DOLCE [Masolo et al., 2003a; Gangemi et al., 2002] already mentioned in Section 3.3.2. A thorough review of these ontologies is beyond the scope of our work. Instead, we focus on the extension approach and its consequences, and discuss why we do not take this route.

Integration in COMM is achieved by using two refined design patterns from DOLCE: the Description and Situation (D&S) and Ontology of Information Objects (OIO) patterns. This is demonstrated in Figure 3.4, where the concepts `parameter` and `abstract-region` defined

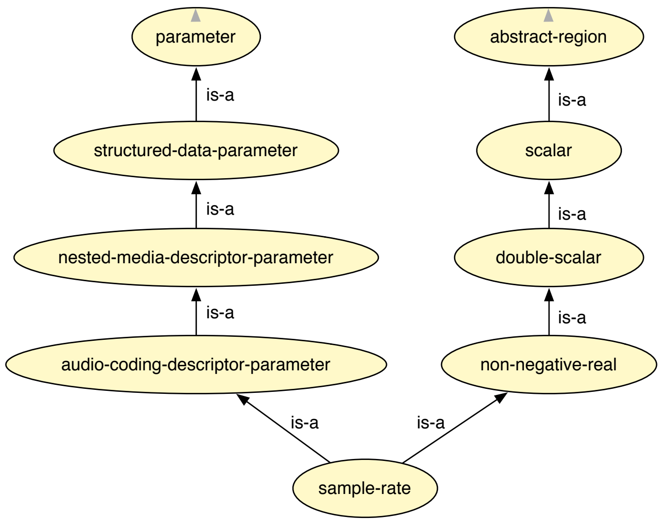


Figure 3.4: Concept hierarchy of the Audio Sample Rate concept in COMM. Only the five most relevant hierarchical levels are shown. (Reproduced from the COMM OWL ontology using the OWL-Viz Protégé plugin.)

in DOLCE are subsumed to describe multimedia specific concepts, such as the `sample-rate` of an audio file. More precisely, the `parameter` concept is defined in the Description and Situation Ontology extending DOLCE. D&S implements a reification schema for descriptions, and a basic framework for situations and for their elements, where *description* and *situation* are disjoint concepts, and can be thought of as synonyms for *conceptualisation* (or *representation*), and *configuration* (or *state*) respectively. A situation can satisfy a description in many ways through a hierarchy of satisfaction (referenced by) relations.

The ramifications of using this schema are adverse in domain specific applications in need of simple knowledge representation and information access. D&S forces reification of even simple data type properties such as a sample rate, and requires the use of foundational properties which encode role semantics between concepts (for example, `requires`, `defines`, or `valued-by`), rather than direct semantic relations between a domain object and its attributes. This has the following consequences: *i)* the RDF data encoded using this ontology becomes exceedingly verbose, a direct result of reification, *ii)* it is very difficult to write queries for data expressed using COMM without a deep understanding of the philosophical foundations of DOLCE, *iii)* query complexity and execution cost increases with data complexity, *iv)* the use of generic properties obstructs the use of many RDFS/OWL features in simple knowledge representation.

We can see from Figure 3.4, that COMM uses subclass axioms to represent data type constraints. This is a less flexible approach than range constraints and OWL restrictions expressed over properties. This is demonstrated in Listing 3.1. Here, we define `sample_rate` as a functional property — a global cardinality constrain meaning that it may only have one value — and as a data type property, which means that its object has to be a data value. A simple global data type constrain is also expressed by specifying the *range* as an XML Schema data type. Furthermore, we can also define class specific restrictions. In the `ex:CDtrack` definition example, we state that the value of `sampe_rate` is constrained to be 44100 when describing tracks of an audio CD. In more precise logic terms, we define the class **CDtrack** as the set of things, that form a subset of things having a sample rate attribute with the value 44100, noting that the subsumption relation expresses necessary but not sufficient conditions, and the value restriction in this case expresses existential quantification. These knowledge representation options become very cumbersome (and in some cases impossible) to use in the context of COMM and DOLCE without introducing even more complexity in an already excessive structure.

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix ex: <http://example.org/ex_ontology#> .

ex:sample_rate rdf:type owl:FunctionalProperty , owl:DatatypeProperty ;
       rdfs:range xsd:double .

ex:CDtrack a owl:Class ;
       rdfs:subClassOf
           [ rdf:type owl:Restriction ;
             owl:onProperty ex:sample_rate ;
             owl:hasValue "44100"^^xsd:nonNegativeInteger
           ] .
```

Listing 3.1: Property restriction examples

When to use reification is among the most complex decisions is ontology design. Representing all domain entities as concepts on principle eases this choice, but it affects both data and query complexity. It is one of the most powerful modelling constructs on the other hand, therefore we opt for using reification, even when simpler models are possible, when its use provides flexibility that would be hard to achieve otherwise.

Although foundational ontologies like DOLCE provide very sound logical and philosophical foundations to an extending ontology, and they are useful to eliminate semantic interoperability problems of ambiguous metadata schema like MPEG-7, due to the complexity issues discussed above, we see their role only in manual or automatic harmonisation of schemata, and in reasoning agents requiring to understand data from heterogeneous, cross-domain resources. Mapping from domain ontologies to foundational ontologies can be provided separately, freeing the user of the domain ontology (both human and machine) form the burden of dealing with extra complexity. Such a separate mapping ontology is demonstrated by Tsinaraki et al [2004], although in a different context, and also advocated by Lagoze and Hunter [2002] in the context of the ABC ontology.

The Music Ontology framework takes the approach of defining some core, domain independent elements, that are no more fundamental than what is required within a knowledge representation framework for describing music. Then, these terms are extended with music specific concepts in a modular ontology library. We follow and advocate this approach.

3.4 Reflections on design principles

The Studio Ontology described in the next chapter focuses on practical issues regarding the description of music production processes in the recording studio environment. Due to its many desirable properties detailed in Section 4.1, we align this ontology with the Music Ontology. We also describe additional Music Ontology extensions unrelated to the Studio Ontology itself, but contribute to the general framework for an intelligent music production environment outlined in Chapter 2. The Temperament Ontology (see Section 4.6) is an example of such an extension. We opt for a modular and extensible ontology library design. Some fundamental elements of this library remain general as they are intended to be reusable across different domains. Examples include the Device Ontology described in Section 4.2.3.2. We also aim at reusing previously published ontologies. Conforming to Linked Data principles [Berners-Lee, 2006], we try to avoid the redefinition of terms available elsewhere, unless it is necessary to fulfil alternative design goals.

We develop a novel framework. Although this includes elements covered by other ontologies, many parts of the knowledge domain we describe have not been explicitly conceptualised before. Consequently, we have few reference points and grounds for comparison. Among the few reference points we use are conceptualisations for multimedia information management outlined in Section 3.2. The structural evaluation of ontologies, — an important part of the development cycle rather than the evaluation of an ontology as a theory — is performed through examining how they reflect certain design principles outlined in this chapter. We acknowledge however the existence of many possible conceptualisations of the domains covered in this work, and do not see the ontologies presented here as rigid, definitive or the best for all purposes. This view is signified by the citation opening the next chapter.

Chapter 4

Ontologies for Semantic Audio Information Management

"When I use a word, it means just what I choose it to mean — neither more, no less." — Humpty Dumpty, Through the Looking-Glass by Lewis Carroll, 1872.

Utilising the design principles as well as the lessons learned from examining state of the art methods for representing music and multimedia related information, we develop a set of ontologies for describing the process of music production. The Studio Ontology detailed in this chapter is closely related to the information management framework for semantic audio tools outlined in Chapter 2. It is designed to satisfy some of its requirements, for instance, the need for collection information about production, and uses the technologies deemed to be most appropriate for managing heterogeneous information in an open ended way. The ontology library presented here builds on the Music Ontology, however it also diverges from it in the conceptualisation of a few fundamental entities relevant in the domain of recording. The Studio Ontology allows for describing music production in more detail than what was possible using previously published ontologies.

4.1 Overview of the Music Ontology Framework

The aim of the Music Ontology [Raimond et al., 2007] is to provide comprehensive, yet easy to use and easily extended domain specific knowledge representation for describing music related information. Integration of music related resources (Web services and data repositories) on the Semantic Web [Raimond and Sutton, 2007], and facilitation of service integration and data communication in distributed music processing environments [Abdallah et al., 2006], [Raimond, 2008] are paramount among its existing applications. Compared to the ontologies and metadata frameworks discussed so far, the Music Ontology has certain properties which make it particularly suitable as basis for a general semantic audio information management framework as well as data collection in the studio. In this section we provide a brief overview.

4.1.1 Utilities of the Music Ontology

The Music Ontology covers and integrates the different kinds of information we discussed in the context of intellectual works in Section 3.2.2. However, it does not do so in equivalent detail. The Music Ontology is particularly well suited to describe the following types of information:

- **Editorial metadata**: Concepts and relationships involving *artists, bands, labels, albums, tracks, audio files* or *downloads* and their identifiers in various databases.
- **Music production workflow**: The life cycle of musical works from *composition* through *performance*, to the produced *sounds and recorded signals* and their *publication.*
- **Event decomposition**: Further details about particular events in the production workflow such as *individual performances* by different musicians in a recording.
- **Content annotation**: Audio *signals* and *temporal annotation* of their content.

Rather than building a monolithic structure, the Music Ontology defines a framework involving some domain independent components which provide the basis for describing the above information. These are extended with music specific terms, which may be further specialised in ontologies extending the framework. The key design considerations of this framework can be summarised as follows:

- **Reuse of existing models**: The Music Ontology is built upon *existing Semantic Web ontologies* and *metadata modelling frameworks* by design.
- **Modularity**: The Music Ontology is published as a framework forming a *modular ontology library.*
- **Extensibility**: The Music Ontology is designed to be *extensible* instead of being cluttered by highly domain specific components.

4.1.2 Domain independent components

An ontology dealing with strictly editorial or bibliographic information may safely assume that the entities in its domain are stable and unchanging. Therefore, it can set aside the description of temporal relationships, as well as the precise conceptualisation of man made artefacts and their life cycle; staring from *ideas* to different possible *manifestations.* This is not the case if we build an ontology about music. Musical works may have a myriad of conceptual forms existing at different times, for example, two performances of the same piece happening perhaps centuries apart, or an audio CD containing the recording of one of these performances.

The ABC ontology described in Section 3.3.1 offers an event-aware integration methodology for tying up entities existing at different times and in different forms. The Music Ontology refines this model, both by simplifying it, and by extending it with elements important for describing music. Two ontologies are defined for this purpose: the Timeline Ontology [Raimond and Abdallah, 2007] and the Event Ontology [Raimond and Abdallah, 2006]. They include a clear conceptualisation of time, incorporating Allen's calculus of interval relationships [Allen, 1983], and the ability to relate events to different time-based multimedia items, similarly to the time model of the HyTime ISO standard [Goldfarb, 1991]. In the following, we provide a brief overview, and some examples that are important in our use cases. The formal definitions of these ontologies can be found in [Raimond, 2008].

4.1.2.1 Timeline Ontology

The Timeline Ontology [Raimond and Abdallah, 2007; Abdallah et al., 2006; Raimond, 2008] extends OWL-Time [Hobbs and Pan, 2006], which includes two important temporal concepts: *instants* and *intervals*, as well as Allen's interval relationships. The Timeline Ontology adds the ability to define specific timelines which may be *continuous* or *discrete*, supporting different temporal coordinate systems, or *abstract*, without an absolute coordinate system. Timelines may be linked through *timeline maps*, as shown in Figure 4.1. Indeed, this is a reification mechanism, which permits useful information to be attached about the relation of two timelines, in this case, a continuous timeline of an analogue signal, and a sampled discrete timeline of a digital signal. This allows the coordinate system most appropriate for the task to be conveniently used.

The conceptualisation on timelines and temporal entities provided by the Timeline Ontology is particularly rewarding in two use cases. The first is the association of dense content-based features or signal transformations obtained through shifted (overlapping) windows. This results in a data rate, different from the original audio sampling rate. A similar problem is encountered when associating low data rate signals controlling some signal processing parameter with an audio signal. In audio engineering terms, this is called *automation*, for example, a set of events or a discrete time signal controlling fader movements on a mixing console. We will thoroughly discuss this problem in the context of recording.

4.1.2.2 Event Ontology

The Event Ontology [Raimond and Abdallah, 2006; Abdallah et al., 2006; Raimond, 2008] aims to provide a way to classify spacial and temporal regions. For example, we may want to describe and localise a *performance* happening at a particular place and time, the temporal location and extent of a *chorus* relative to a pop song, or the location of a note *onset*. The Timeline Ontology provides the fundamental concepts such as instants and intervals for temporal localisation, but it is not sufficient to classify a performance or a chorus per se.

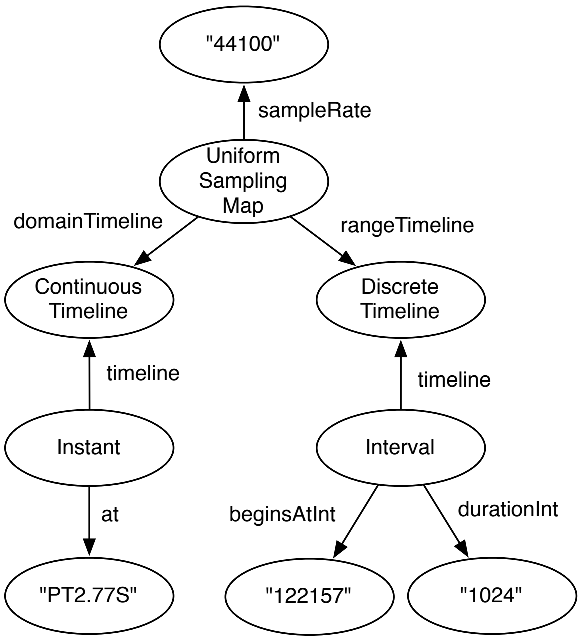


Figure 4.1: Timeline Ontology example: An Instant and an Interval starting at the same time, expressed using a continuous and a discrete timeline, linked through a timeline map.

The Event Ontology allows for such classifications to be made by considering *event tokens* as first-class entities [Vila and Reichgelt, 1996]. Although this approach is similar to ABC's event model shown in Figure 3.3, here, events are defined as the way by which cognitive agents classify arbitrary regions of space-time, an act of classification [Raimond, 2008], as opposed to transitions between situations. Compared to ABC, the Event Ontology also eliminates the need for describing *actions*, since event decomposition can easily substitute this facility. Then, the Event Ontology simply defines an `Event` concept, which can be linked with objects at a particular *place* and *time*, and participating *agents*. Events may have *products* and *factors* such as tools, or products of other events, and may be decomposed into one or more *sub_events*.

Using the Event and Timeline ontologies alone, we can already say quite a lot about events. For example, we may describe a recording session[1] as shown in listings 4.1 followed by its FOL translation. Throughout this chapter, taking advantage of the precision of FOL sentences compared to natural language, the FOL syntax described in Section 2.3.2 will be used. Unary and binary predicates typeset in sans serif correspond to concepts and properties in specific ontologies. Individuals are set in small case italic font, variables are set using capital letters. Colon separated namespace prefixes are given in Table D.1, Appendix D.

[1]Further examples are available at: `http://wiki.musicontology.com/index.php/Examples`

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix event: <http://purl.org/NET/c4dm/event.owl#> .
@prefix tl: <http://purl.org/NET/c4dm/event.owl#> .
@prefix : <http://example.org/>     .

:rs a event:Event ; # recording_session
  event:place <http://example.org/my_studio> ;
  event:agent <http://foaf.me/fazekasgy#me> ;
  event:time [
         a tl:Interval ;
         tl:timeline tl:universaltimeline ;
         tl:beginsAtDateTime "2011-03-20T09:00:00Z"^^xsd:dateTime ;
         tl:durationXSD "P1DT"^^xsd:duration
         ] ;
  event:sub_event :rec ;
  event:product <http://example.org/album> .

:rec a event:Event ; # recording
  event:factor [ rdfs:label "Shure SM58" ] ;
  event:product <http://example.org/signal> .
```

Listing 4.1: Using the event and timeline ontologies

$$\begin{aligned}
&\mathsf{event\!:\!Event}(rs) \wedge \mathsf{event\!:\!Event}(rec) \wedge \mathsf{event\!:\!place}(rs, my\_studio) \wedge \mathsf{event\!:\!agent}(rs, me)\\
&\wedge \exists\, I(\mathsf{tl\!:\!Interval}(I) \wedge \mathsf{tl\!:\!timeline}(I, universaltimeline)\\
&\wedge \mathsf{tl\!:\!beginsAtDateTime}(I, \text{"}\mathit{2011\text{-}03\text{-}20T09\text{:}00\text{:}00Z}\text{"})\\
&\wedge \mathsf{tl\!:\!durationXSD}(I, \text{"}P1DT\text{"}))\\
&\wedge \mathsf{event\!:\!time}(rs, I) \wedge \mathsf{event\!:\!sub\_event}(rs, rec) \wedge \mathsf{event\!:\!product}(rs, song)\\
&\wedge \exists\, F(\mathsf{rdfs\!:\!label}(F, \text{"}ShureSM58\text{"})) \wedge \mathsf{event\!:\!factor}(rec, F)\\
&\wedge \mathsf{event\!:\!product}(rec, signal)
\end{aligned} \tag{4.1}$$

Although we are not able to define and identify all factors precisely, these ontologies provide the basic logical framework for describing events. More domain specific ontologies are required however to distinguish between the individual events *recording session* (`:rs`), and *recording* (:rec) and designate them appropriately, or to specialise the meaning of properties like *place*, *agent* or *factor* in order to refer to a *studio*, a *sound engineer*, or a *microphone*, for example.

The Music Ontology provides domain specific concepts and properties to describe musical works and the music production workflow. It provides a level of detail sufficient for archival purposes, and the use cases of most online music applications or linked data services, however further extensions, as well as a few more basic components — such as an ontology of signal processing and recording devices — are required to describe audio engineering workflows precisely. In the following, we review the core components of the Music Ontology and its extensions, then we describe the Studio Ontology.

4.1.3 Core music specific components

The Music Ontology itself is the core component of a framework of related ontologies. It provides music related refinements of the previously discussed ontologies [Raimond, 2008]. The Music Ontology can be roughly divided into three parts or levels according to the specificity and expressiveness of concepts and relationships defined on these levels.

4.1.3.1 Editorial level

The first level builds on the Friend of a Friend (FOAF) Ontology [Dumbill, 2002] in order to describe *agents*; people and groups (e.g. composers, bands or engineers), organisations, (e.g. record labels), and also makes use of basic relations defined by Dublin Core (DC) such as `dc:title`. Furthermore, it extends FRBR discussed in Section 3.2.3. FRBR defines four layers of abstractions in the life cycle of intellectual works. These layer are ranging from abstract to concrete. Based on this model, the Music Ontology defines the concepts: `Musical Work`, `Musical Expression`, `Musical Manifestation`, `Musical Item` and specialises them as shown in Figure 4.2. Please note that only some relevant concepts are shown for brevity.

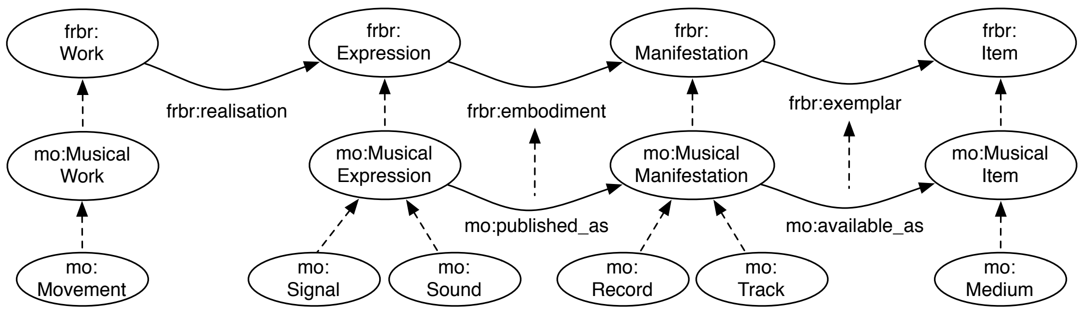


Figure 4.2: Relation of FRBR and some selected Music Ontology terms. (Dashed lines represent subclass or subproperty relationships.)

This is sufficient to describe artists and their relations, their works, published records, individual items in a collection such as audio files available locally or on the web.

$$\begin{aligned} &\mathsf{mo:MusicGroup}(dave\_brubeck\_quartet) \wedge \mathsf{mo:MusicArtist}(paul\_desmond) \\ &\wedge \mathsf{mo:member}(dave\_brubeck\_quartet, paul\_desmond) \\ &\wedge \mathsf{mo:Track}(take\_five) \wedge \mathsf{foaf:maker}(take\_five, dave\_brubeck\_quartet) \\ &\wedge \mathsf{mo:Record}(time\_out) \wedge \mathsf{mo:track}(time\_out, take\_five) \\ &\wedge \mathsf{mo:available\_as}(take\_five, file:///Library/iTunes/music/take\_five.mp3) \end{aligned} \tag{4.2}$$

For example, the above sentence (4.2) describes how an audio file is related to a track on a published record as well as the performing artists. However, we cannot yet explain that the song *Take Five* was composed by *Paul Desmond*. More precisely, we cannot describe on the editorial level, how a *composition* is related to a performance and its recording, a resulting track, or an individual audio item. The Music Ontology provides a workflow model for this purpose.

4.1.3.2 Workflows, events and signals

Higher levels of the Music Ontology define *events* that establish links between musical concepts corresponding to the different conceptual layers of the FRBR model. This allows for instance, to describe what it takes for a particular musical expression, such as a *sound*, to become represented in a manifestation, e.g. a track on a record. Essentially, this level provides a workflow model illustrated in Figure 4.3. This allows tying together different concepts using an event-aware methodology similarly to the ABC ontology discussed in Section 3.3.1.

The model extends the Event Ontology by subsuming the *Event* concept and specialising its properties. For example, the concept `mo:Composition` is a subclass of `event:Event`, while the property `mo:produced_work` is a subproperty of `event:product`. We note that Figure 4.3 shows a largely simplified model. For example, there are several inverse relations as well as short-cuts that facilitate simpler descriptions in situations where the full workflow model is not required by the application.

There are also more complex event and expression types which permit the description of real-life recording and publishing scenarios. The latest revision of the Music Ontology defines some concepts which are particularly interesting in studio production. These are summarised in Table 4.1. The inclusion of recording sessions and the ability to group their products (signal group) addresses the need for collecting the recordings of songs that are recorded and intended to be released together on an album, or by some other means. This works well in the broad production workflow of the Music Ontology, however more granularity and precision is required in certain studio specific use cases. We will discuss these in Section 4.2.

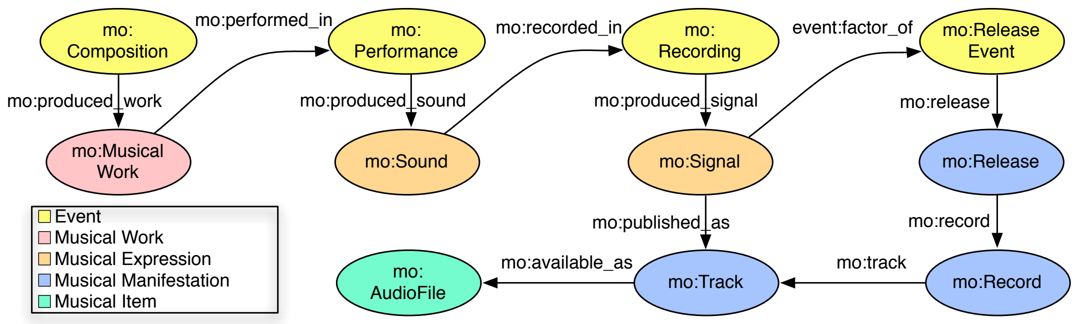


Figure 4.3: Music Production Workflow Model: key concepts and selected properties showing how events (top row) make connections between the layers of the FRBR model.

concept	basic type	description	selected properties
Recording	Event	Takes a sound as a factor and produces a signal.	mo:recording_of, mo:produced_signal, mo:engineer
RecordingSession	Event	A set of performances, recordings and mastering events.	mo:produced_signal_group, mo:engineer
Signal	Expression	A signal produced by recording a performance.	mo:records, mo:channels
SignalGroup	Expression	A group of signals, e.g. a set of masters resulting from a recording session.	mo:signal
SoundEngineer	Agent	A sound engineer associated with a performance.	mo:engineered

Table 4.1: Some Music Ontology concepts most relevant in studio production

Returning to our previous example of Listing 4.1, we can now refine the description of the two events introduced there. This is shown in Listing 4.2. For instance, we can clearly distinguish between recording sessions and recordings. We can say that the resource `<http://foaf.me/fazekasgy#me>` represents an engineer, instead of some arbitrary agent. We can also more clearly describe the products of the events. However, ambiguities still remain, as the Music Ontology does not allow for describing all details arising in a recording scenario. More specifically, we do not know precisely what `event:factor` and `event:place` refer to. The Music Ontology does not deal with tools such as recording devices used in the production workflow, therefore we cannot identify passive factors in recording events. The current use of `event:place` in Music Ontology applications is also rather unclear. Sometimes it refers to an exact geographical coordinate (latitude, longitude, altitude), but it may refer to a country, a city, a studio or a room. When we describe a recording however, we are more interested in the distance of the microphone from the sound source, rather than in its exact geographical location. Furthermore, the concept `mo:Recording` is too broad in the studio context. Even though it is defined as single recording event, in its common use, it may refer to the recording of a complete song with many constituent recordings corresponding to separate instruments. We will discuss these problems, as well as the above examples further in Section 4.2, in the context of the Studio Ontology.

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix event: <http://purl.org/NET/c4dm/event.owl#> .
@prefix tl: <http://purl.org/NET/c4dm/event.owl#> .
@prefix mo: <http://purl.org/ontology/mo>
@prefix : <http://example.org/>     .

:rs a mo:RecordingSession ;
   event:place <http://example.org/my_studio> ;
   mo:engineer <http://foaf.me/fazekasgy#me> ;
   event:time [
          a tl:Interval ;
          tl:timeline tl:universaltimeline ;
          tl:beginsAtDateTime "2011-03-20T09:00:00Z"^^xsd:dateTime ;
          tl:durationXSD "P1DT"^^xsd:duration
          ] ;
   event:sub_event :rec ;
   mo:produced_signal_group :album_signals .

:album_signals a mo:SignalGroup ;
       mo:signal :signal1, :signal2 .

:signal1 a mo:Signal ;
       dc:title "Take Five"^^xsd:string .

:signal2 a mo:Signal ;
       dc:title "Blue Rondo a la Turk"^^xsd:string .

:rec a mo:Recording ;
   event:factor [ rdfs:label "Shure SM58" ] ;
   mo:produced_signal :signal1 .
```

Listing 4.2: Using the Music Ontology to describe a recording session

4.1.3.3 Content annotation and event decomposition

The third conceptual level of the Music Ontology provides models for the temporal annotation of audio content, and events that can be described as parts of more complex events, using the event decomposition model defined by the Event Ontology. In fact, most facilities of this level are defined in other ontologies, such as the extensions outlined in the next section. Accordingly, we discuss these concepts elsewhere, see Sections 4.1.2.2, 4.3 and 4.1.4.

4.1.4 Extensions

The Music Ontology Framework contains a number of extensions [Raimond, 2008] that deal with more specialised domains whose inclusion in the core ontologies would result in too specialised and less flexible ontologies. Most notable extensions include the Symbolic Ontology dealing with scores and symbolic music notation, and the Chord Ontology describing chords corresponding to Harte's notation [Harte et al., 2005]. This generalises common chord notations used in jazz and popular music. The Audio Features Ontology (see Section 4.3) defines common high and low level features such as `af:Spectrogram` or `af:Note`, and deals with content annotation, — that is, how to describes features and associate them with the audio content relying on the Event and Timeline Ontologies. The primary scope of this ontology is feature representation and association. Therefore it does not attempt to classify them, describe their interrelationships or their computation. We make use of this model in our SAWA framework (see Section 5.3) for Web-based audio analysis developed as a demonstrator of OMRAS2 components. The workflow model provided by the Music Ontology is also extensible. It defines only basic events required to describe music production workflows in broad terms, however further extensions are needed to describe more specific production procedures.

4.1.5 Summary

In this section we summarise the features which make the Music Ontology more suitable for our work than its alternatives, for instance, COMM [Arndt et al., 2007], the Kanzaki Music Vocabulary [Kanzaki, 2007], the SIEMAC Ontology [García and Celma, 2005], or the metadata harmonisation framework advocated by Hunter [2003]. Detailed comparison with some of these Web ontologies are available in [Raimond, 2008].

- **Workflow-based conceptualisation** of the music domain: The Music Ontology can be used to describe different entities in the life-cycle of intellectual works — defined in the Functional Requirements for Bibliographic Records (FRBR) — ranging from abstract to concrete entities: Musical-*Work*, *Expression*, *Manifestation*, *Item*. These layers can be tied together in complex descriptions using events.

- **Event decomposition model:** Events are modelled as first-class objects (event tokens) with participating agents and passive factors, and may be decomposed into sub-events.

- **Timelines and temporal entities** can be used to localise events on different timelines: *abstract*, *discrete*, or *continuous*; *relative*, or *physical*.

- **Modular and extensible design**: The Music Ontology is published as a modular ontology library whose components may be reused or extended outside of its framework.

- **Ease of use:** The Music Ontology provides only the terms required for descriptive knowledge representation without more foundational elements. It extends or simplifies models described elsewhere.

- **Adaptation**: It has become a de-facto standard for publishing music-related data on the Semantic Web with several existing applications[2] in industry and academia.

The above models provide the basis for content annotation as well as the decomposition of events in complex workflows, so that we can precisely say *who* did *what* and *when.* While elements of these models can also be found in other ontologies, they are not present all at once in a single unified framework. The Music Ontology provides a model to describe the production workflow from composition to delivery, including music recording, but it lacks some very basic concepts to do so in detail. The Studio Ontology fills this gap.

[2]http://www.musicontology.com

4.2 The Studio Ontology Framework

In this section we discuss knowledge representation issues arising in studio production, and introduce the Studio Ontology framework for describing and sharing detailed information about music production in the recording studio. The primary motivation for creating this ontology is to facilitate capturing the nuances of record production by providing an explicit, application and situation independent conceptualisation of the studio environment. We may use the ontology to describe real-world recording scenarios involving physical hardware, or (post) production procedures on a personal computer.

The Studio Ontology framework builds on Semantic Web technologies for knowledge representation and knowledge sharing. It extends the Music Ontology framework in order to take advantage of the foundations it provides for dealing with music-related information. In the rest of this section, we first provide some motivating examples and discuss some design criteria. Then we outline the foundational and core elements of the Studio Ontology, followed by a review of its extensions dealing with highly specific domains such as microphone techniques, mixing consoles and audio effects.

4.2.1 Motivation

Our most important motivating use cases for creating an ontology of studio production are centred around the following applications:

- **Music Information Retrieval**: *Exploit production data* in MIR systems (e.g. music recommendation).
- **Audio engineering notation**: *Capture the creative contribution* of the engineer or producer to a musical work.
- **Intelligent audio editing**: *Provide a knowledge representation model* that supports intelligent audio editing systems.
- **Recording studio database**: *Facilitate information management* in the studio through the collection of production information for archival, search and educational applications.

Music Information Retrieval: Recognising that simple metadata based approaches are insufficient in complex music information management and retrieval scenarios, researchers have been focusing on cultural information — obtained predominantly from crowd sourcing, for instance, the analysis of listening habits published by online radio stations — as well as the use of content-based features extracted from commercially released audio mixtures. Certain types of cultural and contextual information are rapidly becoming available on the Semantic Web and via a number of Web services. For example, events (concerts, tour dates) or artist relations can be obtained and used in intuitive ways to find connections in music. See for example Passant [2010], Jacoboson [2009] or an overview in [Fazekas et al., 2010]. However, these data remain largely editorial and focussed on artists as opposed to music and production. We argue that another invaluable source of information exist, largely neglected to date, pertaining to the cultural context, history, production and pre-release master recordings of music. The reason for this neglect is not ignorance or the ineffectiveness of these data, but the fact that they are unavailable in large quantities, hence their use hasn't been explored yet. I firmly believe that the reason is the lack of comprehensive open standards and methodologies for their collection, therefore production information is simply lost.

Audio engineering notation: It can be argued that contributions from the producer or the sound engineer are just as important in popular music production as composition, however, we have no way to record his/her actions and choices, with the same transparency music is denoted using scores. The engineer has a dual role which can be characterised by the aim of fulfilling artistic goal on one hand, and by the use of very specific domain knowledge on the other, (see [Huber and Runstein, 2005] for an overview of audio engineering practice). This knowledge is used for the adaptation of tools at hand to the specificity of a set of recordings comprising a music release. While artistic goals are defined purely by human factors, such as aesthetic decisions made by the producer or the recording artist, the domain knowledge mentioned above mainly concerns the appropriate use of tools. Capturing this domain knowledge for the benefit of the engineer can also lead to building context-adaptive audio processing systems which are in the primary focus of our research. Achieving this goal requires work on two fronts; one is the development of formalised knowledge models to structure and represent necessary information, the other is the adaptation of existing music processing tools for our purposes.

Intelligent audio editing: The most prevalent paradigm for the design of today's digital audio workstations (DAW) is a model of real-world analogue counterparts of recording and signal processing components such as multi-track tape recorders, mixers and effect units. For an overview of state of the art music production software see [Duignan, 2008]. These systems however have an inherent limitation in that they "blindly" process the signals they receive, while adaptation to contextual information such as the processed instrument or the tempo of the recording can only be made manually, or by using some inferred characteristics of the signals themselves. This presents further limitations which are both theoretical and practical in nature. Firstly, there is an upper limit in the robustness of high-level features extracted from audio (which may be improved by considering contextual information), secondly, the implementation of adaptive tools within this paradigm leads to highly intermingled components. Typically, this fails to generalise in the context of real-world audio recordings. For example, we can think of the limitations of supervised machine learning algorithms for audio analysis, whose performance almost certainly drops when the characteristics of the analysed music or signal are different from what was included in the initial training set. To avoid these pitfalls, we argue for a system where components are interchangeable, their properties are fully described and the information flow and data encoding between them is fully formalised. Then, higher-level logical supervision becomes possible.

Recording studio database: Information management in the recording studio is a largely underdeveloped area. Few metadata standards exist, but they are disharmonious and concerned only with project transfer between audio workstations. They do not provide an application independent conceptualisation of the studio domain. Fortunately, Semantic Web technologies provide a solution to the complex information management problems arising in music production, and facilitate sharing the captured data. This may include microphone arrangements and characteristics, configuration, connection, decomposition and operation of audio signal processing devices such as mixers and effect units, projects and edit decisions in post production workstations and features extracted from pre-release master recordings. The ontologies we describe are designed to propagate this information through the production workflow to enable building better models for music information retrieval, hence facilitate applications like the retrieval of songs based on production procedures. This information can be utilised in the daily audio engineering routine, and it also has a considerable educational value.

4.2.2 Design decisions

We discussed ontology design principles in a more general context in Section 3.1.4. While we aim to adhere to these principles, some design considerations are specific to the development of the Studio Ontology Framework, and are paramount in finding the best balance in the application of general principles. These considerations can be summarised as follows:

- **Monotonic extension of ontologies**: We attempt to reuse existing ontologies, and where possible, do so monotonically, without the need for updating the meaning of terms in the base ontology.
- **Modular design**: We separate our ontology into a set of harmonised modules forming a framework.
- **Balanced use of reification**: While flexibility is an important objective, we try to achieve a flexible design with a balanced use of reification, without significantly increasing complexity.

- **Reusing the Music Ontology**: For reasons detailed in Section 4.1.5, we use elements of the Music Ontology Framework as the primary basis for the Studio Ontology.

Although these design decisions seem to be easily implemented, we found that this is not always the case. For example, monotonic extension of terms defined elsewhere can lead to models that are more complex than necessary, or less precise when capturing domain specific knowledge. However, the benefits of reusing an existing ontology often outweighs these drawbacks. Avoiding the use of reification — which often leads to a more complex model — while maintaining design flexibility is also a difficult task. Our solution to this problem is to offer different models with increasing complexity, — that is, allow the user to choose a desired level of granularity for the application. However, this solution has to be used sparingly in order to maintain consistency. We will discuss alternative design choices where applicable.

Despite the above concerns, our framework is designed to reuse existing terms and models published elsewhere that fit its requirements. It is presented as a modular and extensible ontology library, containing some general, domain independent elements, a set of core concepts and relationships to describe the studio domain, and some extensions covering more specific areas like microphone techniques and multitrack production tools. In the following sections we outline the most important parts of this framework.

4.2.3 Foundational elements

The Studio Ontology is grounded on similar concepts as the Music Ontology itself, therefore we can use and extend some of its components directly. However, it does not provide every basic element required to create a studio ontology. For instance, it does not contain elements for describing audio processing devices, or a descriptive workflow model for signal processing. While the N3-Tr framework described in [Raimond, 2008] (see also Section 3.2.6.3) takes steps in these directions, it is concerned with concurrent execution of signal processing workflows in audio analysis, as opposed to the description of audio processing. It does not include an ontology of audio or signal processing devices.

Here, we first outline how the core components of the music ontology are reused in our framework, then, we introduce the basic components for describing audio engineering workflows. Although we have domain and task specific implementation goals, these ontologies approach a foundational, domain independent status in the sense described in [Masolo et al., 2003b] or [Lagoze and Hunter, 2002], and can be seen as generalisations of more domain specific ontologies.

4.2.3.1 Workflows, Events and Timelines

We distinguish between two types of workflows: *prescriptive* and *descriptive* (see Section 3.2.6.3). Prescriptive workflows are best understood as templates describing common data access and manipulation steps. Descriptive workflows can be seen as denotation of specific instances of the above. Broadly speaking, a workflow step is a description of *who* (or *what*) produced *what*, *when*, and *how*, *using what*. Such a description requires a conceptualisation of entities existing at various stages of the workflow. The Music Ontology provides such a conceptualisation: A composition (*musical work*) may be performed producing a sound which may be recorded producing a signal (*musical expressions*). This is exemplified in the following sentence (4.3), where the predicate `mo:MusicalWork` identifies a work (*take_five*), while `mo:Sound` and `mo:Signal` define musical expressions (*sound,signal*) resulting from its performance and recording:

$$
\begin{aligned}
&\mathsf{mo:MusicalWork}(take\_five) \wedge \mathsf{dc:title}(take\_five, \text{"}Take\ Five\text{"})\\
&\wedge\, \mathsf{mo:Performance}(perf) \wedge \mathsf{mo:Sound}(sound)\\
&\wedge\, \mathsf{mo:performance\_of}(perf, take\_five)\\
&\wedge\, \mathsf{mo:produced\_sound}(perf, sound)\\
&\wedge\, \mathsf{mo:Recording}(rec) \wedge \mathsf{mo:Signal}(signal)\\
&\wedge\, \mathsf{mo:recording\_of}(rec, sound)\\
&\wedge\, \mathsf{mo:produced\_signal}(rec, signal)
\end{aligned} \tag{4.3}
$$

In the Studio Ontology, we follow this workflow model and hook into it at the level of expressions. Audio engineering workflows concern the manipulation of sounds and signals. For example, the arsenal of recording techniques includes placing a powered speaker cabinet into a specially treated room with a microphone, in order to record an electric guitar such that its sound is influenced by the room characteristics or the qualities of the amplifier-speaker combo[3]. This can be seen as the manipulation of sound by the engineer. The manipulation of signals by means of signal processing devices is a more common case.

In FRBR terms, when the sound engineer manipulates a sound or a signal, new expressions are created to which additional information can be attached on how it was produced. In order to describe this process, we need to be able to talk about *events* (performance, recording, mixing, transformation) which may be spatially and temporally localised and linked with *agents* (engineer) and *factors* (tools). We use the Event and Timeline Ontologies (see Section 4.1.2) for this purpose. The Music Ontology sets aside the problems of *how* and *using what* in its workflow model, — that is, it doesn't have a model for describing tools and how they are used. The ontologies we outline in the rest of this section provide a solution.

[3]Vintage equipment, such as certain brands of guitar amplifiers are often used for their unique characteristics, such as the non-linear distortions introduced by vacuum-tube amplifiers and coupling transformers.

4.2.3.2 Technological Artefacts

One of the most important things in modelling audio engineering workflows is to be able to describe technological artefacts, — that is, tools, such as microphones, amplifiers, mixing consoles, recording devices, plugins, audio editing software, and complex digital audio workstations. We also have to consider how these devices are used, for example, how to describe their variable parameters during an event. The first fundamental element we add to the set of core ontologies is the Device Ontology. It deals with the kind of tools mentioned above in a very general sense. Terms in the Device Ontology are kept entirely domain independent, therefore we may also use it for instance, as a base ontology to describe laboratory instruments.

Signal processing devices are a special class of devices commonly used in audio engineering to manipulate audio signals. We develop a Signal Processing Device Ontology which builds on the Device Ontology, and serve as basis for describing more complex tools such as mixing consoles or audio effect units. A small separate ontology is used to describe how signal processing devices are interconnected, and describe their input and output terminals, with associated physical connectors and communication protocols.

4.2.3.3 The Device Ontology

The Device Ontology can be used to describe artefacts of technology. Central to its model is the *Device* concept which may be subsumed by a term in a more specific ontology for representing any man-made object; concrete or abstract. The Device Ontology generalises terms from the ontologies described in [FIPA, 2002], and [Bandara et al., 2004] which are specific for their application domains, namely smart phones and computer networks. Its basic model is shown in Figure 4.4. We first provide the definition of terms in this ontology, then we discuss the rationale behind these definitions, and outline some applications. We define the Device concept as follows:

$$\forall\, D(\mathsf{device\!:\!Device}(D) \rightarrow \mathsf{owl\!:\!Thing}(D)) \tag{4.4}$$

A device may participate in an event as a passive factor, providing a particular *service* in a particular *state*. (We note that the use of these concepts is optional).

$$\forall\, S(\mathsf{device\!:\!Service}(S) \rightarrow \mathsf{owl\!:\!Thing}(S)) \tag{4.5}$$

$$\forall\, S(\mathsf{device\!:\!State}(S) \rightarrow \mathsf{event\!:\!Event}(S)) \tag{4.6}$$

$$\forall\, D, S(\mathsf{device\!:\!state}(D, S) \rightarrow \mathsf{device\!:\!Device}(D) \wedge \mathsf{device\!:\!State}(S)) \tag{4.7}$$

$$\forall\, D, S(\mathsf{device\!:\!service}(D, S) \rightarrow \mathsf{device\!:\!Device}(D) \wedge \mathsf{owl\!:\!Thing}(S)) \tag{4.8}$$

A *state* is considered as an abstraction of changeable attributes of a device, whose concrete parameters hold for a specific time and at a specific location. For example, it can be used to represent a configuration, such as variable polar pattern or sensitivity settings of a microphone during a recording. This provides a reification mechanism, when we need to consider multiple device states in a single event. The concept of *service* is useful to describe multifunctional devices, for instance a multi-effect unit, whose specific application is not clear from its identity. This can be seen as the generalisation of hardware and software description terms considered in [Bandara et al., 2004].

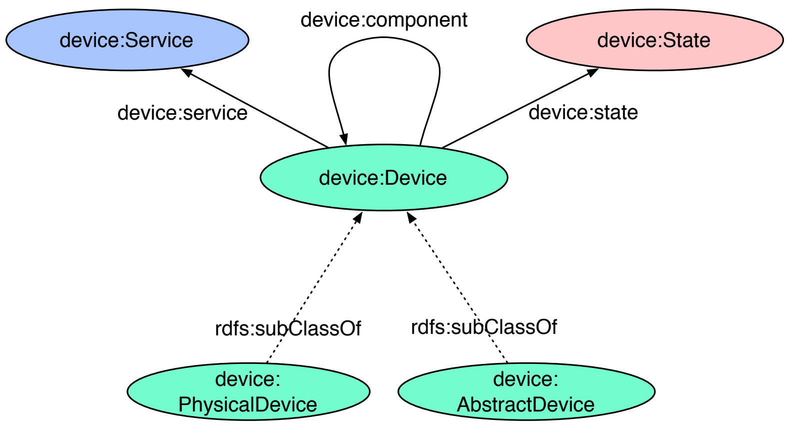


Figure 4.4: Overview of the Device Ontology

A device may be decomposed into abstract or concrete components, such as an extension module or firmware. For instance, we may want to describe a digital monitoring system in a studio, linked with its constituent loudspeakers and a firmware component implemented across the whole system. (In our example, the firmware typically implements the audio networking and device control protocols, and it is distributed across many physical constituents of the system.) We define a component relation for this purpose as follows (4.9).

$$\forall\, D1, D2(\mathsf{device\!:\!component}(D1, D2) \rightarrow \mathsf{device\!:\!Device}(D1) \wedge \mathsf{device\!:\!Device}(D2)) \tag{4.9}$$

Form a mereological point of view, a component is held as an identifiable *part of* an object [Tversky, 1989]. Device decomposition therefore expresses a partial order relation over the set of components of a device, which is, in the most general sense, a reflexive, transitive and antisymmetric property. Here, we discuss only two of the mereological problems: *i)* whether we should consider a component a *proper part* (an irreflexive relation) and *ii)* whether we should consider the component relation transitive. In the light of the arguments presented in [Varzi, 2003], it seems more favourable in both cases to support the weakest theory or the most general definition. For example, one may argue that transitivity does not hold for direct functional relations — that is, if x is a ϕ-part of y and y is a ϕ-part of z, x need not be a ϕ-part of z: the predicate modifier ϕ may not distribute over parthood [Varzi, 2006].

However, this just shows the non-transitivity of the specialised relation which can be expressed using explicit predicate modifiers. We also note that only transitivity may be expressed in OWL-DL, which requires that no local or global cardinality constraints should be declared on a transitive property or its super-properties. Expressing the other logical characteristics requires the ontology to be updated to OWL-2, therefore they are not currently expressed.

Our ontology commits to a categorical distinction of devices, the difference between physical and abstract objects (4.10, 4.11). This distinction is worth making for the following considerations: Physical and abstract objects have different primary characteristics. For instance, physical devices have size and weight, and may be decomposed into physical or abstract components. Abstract devices on the other hand may be intangible models of physical devices.

$$\forall\, D(\mathsf{device\!:\!PhysicalDevice}(D) \rightarrow \mathsf{device\!:\!Device}(D)) \tag{4.10}$$

$$\forall\, D(\mathsf{device\!:\!AbstractDevice}(D) \rightarrow \mathsf{device\!:\!Device}(D)) \tag{4.11}$$

We use this mechanism primarily to distinguish between hardware and software components, for instance, elements of digital audio workstations. Software and other intangible components are subclasses of `device:AbstractDevice`, but *abstractions*, having both physical and abstract components directly subsume `device:Device`. See for example our model of device terminals discussed in Section 4.2.3.9. This is useful since digital interfaces typically consist of a physical connector, some circuitry and a communication protocol, that is, they have both tangible and intangible components.

$$\begin{aligned}&\forall\, D1, D2(\mathsf{device\!:\!component}(\mathsf{D1}, \mathsf{D2}) \wedge \mathsf{device\!:\!PhysicalDevice}(D1)\\&\rightarrow \mathsf{device\!:\!PhysicalDevice}(D2) \vee \mathsf{device\!:\!AbstractDevice}(D2))\end{aligned} \tag{4.12}$$

$$\begin{aligned}&\forall\, D1, D2(\mathsf{device\!:\!component}(D1, D2) \wedge \mathsf{device\!:\!AbstractDevice}(D1)\\&\rightarrow \mathsf{device\!:\!AbstractDevice}(D2))\end{aligned} \tag{4.13}$$

4.2.3.4 Modelling variable device parameters

In many cases it is insufficient to provide a static description of a device. For instance, the application of a tool in a recording event may involve several changeable characteristics and the modification of many different variable parameters. Although we can think of recording a concert for example, such that a mixing console is configured before the performance, and then left untouched until the end, an ontology which can only be used to describe this situation would not reflect reality in most circumstances. This consideration raises the question of how to model change in RDF. Several solutions were proposed to solve this problem.

Suppose we have a device with a sole variable parameter *sensitivity* modelled as a datatype property. A static description can be given as shown in Listing 4.3.

```
:e a event:Event ;
       event:factor :d .

:d a device:Device ;
       device:sensitivity "20"^^xsd:int .
```

Listing 4.3: A Device description with one parameter

This model implies that the parameter is fixed during the event. However, we want to describe several changes made by an engineer. One solution would be the *decomposition of the event* into several sub-events, each linked with a device with different parameter settings. The problem with this solution is that our model would not reflect reality. We are talking about a single event involving a single device whose parameter is changing, instead of several different individuals whose parameters are different. Moreover, event decomposition in this way could lead to an excessive amount of data, considering a large number of changes.

```
# A) Reification of RDF statements
s: rdf:Statement ;
       rdf:subject :d ;
       rdf:predicate device:sensitivity ;
       rdf:object "20"^^xsd:int ;
       dc:date "2011-03-20"^^xsd:date .

# B) Named graphs
:G1 { :d a device:Device ;
       device:sensitivity "20"^^xsd:int . }

:G2 { :d a device:Device ;
       device:sensitivity "30"^^xsd:int . }

:G1 dc:date "2011-03-20"^^xsd:date ;
```

Listing 4.4: Reifications of a device description using (A) a reified statement and (B) named graphs.

Typically we would want to express that a parameter value holds for a certain amount of time in a given situation. Various forms of reification provide a solution, since they allow linking additional information, such as a time object to a statement. In Listing 4.4 we exemplify two solutions; the first (A) is using the RDF reification syntax [Hayes, 2004], the second (B) is based on named graphs [Carrolla et al., 2005], which — though strictly speaking not equivalent to reification — is often used in place of the former syntax.

Reification of RDF statements is considered obsolete for several theoretical and practical reasons. A theoretical problem is related to the fact that the reified statement is equivalent to the original from an RDF model theoretic [Hayes, 2002] point of view. This solution is therefore paradoxical, since two equivalent statements cannot be part of the same set of statements. Furthermore, if a simple RDF triple is restored from a reified statement, the attached information is lost [Miller, 2000]. It is also problematic to write queries for data stored in a reified form.

The above problems can be resolved using named graphs, where statements are isolated in separate RDF graphs. Different graphs may include conflicting statements, while their use permits additional data to be attached to a graph specifications as shown in Listing 4.4. Although this approach is widely adopted, it requires an extension of the RDF data model related to RDF data-sets, originating from the development of SPARQL. This model lacks an effective means of distinguishing between graph sources once the information is published. In terms of storage, it requires the use of *quads* instead of triples, such that a unique graph identifier is attached to each statement. The application of named graphs for tracking modifications of audio files parameters in an audio editor was explored in [Fazekas and Sandler, 2009], but this model was dropped in favour of the one described in Section 4.2.4.3.

Another commonly used form of reification equates to promoting properties to first class objects, — that is, modelling relationships as concepts. Two alternatives of this approach are shown in Listing 4.5. The first form of property reification (C) simply treats the parameter *sensitivity* as a concept. The second form (D) is more flexible in that it considers a general purpose reified parameter `device:Parameter`, and enables the description of any device attribute using this concept.

The property reification approach (C) is similar to how the relationship between timelines is modelled in the Timeline Ontology (see Figure 4.1) in order to provide additional information on how two timelines are related. This works well if the number of relationships we need to reify in a model is relatively low. However, considering the potentially large number of device attributes we may want to describe, this can lead to the introduction of countless additional concepts in our ontologies. The approach essentially forces the modeller to follow similar design patterns discussed in the context of COMM, such that all relationships are modelled as concepts. This results in a large, relatively obscure, and difficult to manage structure, exhibiting the same knowledge representation issues discussed in Section 3.3.2.

The second approach to property reification (D) shown in Listing 4.5 is basically a variation of the first one. This is especially useful in cases where devices have parameters unknown to the ontologist. Although this approach does not require the introduction of new concepts for each property, from a knowledge representation point of view it is very limited. For instance, we cannot express constraints over the described parameters. Furthermore, we need to describe each parameter using a unique instance of `device:Parameter` (or using a blank node), and in both cases, binding parameters that change together or represent a single state is difficult. From a data access point of view, all reification models mentioned so far suffer from the issue of being quite verbose, and exhibit a problem when we need to find parameters which hold in a certain situation, for instance, when we have to find all values bound to the same time object.

```
# C) Reified parameter (using property reification)
:d a device:Device ;
       device:parameter :p .

:p a device:Sensitivity ;
       device:value "20"^^xsd:int ;
       device:time [ a time:TemporalEntity ] .

# D) Reified parameter (using a generic parameter concept)
:d a device:Device ;
       device:paramter :p .

:p a device:Parameter ;
       device:parameter_name "sensitivity" ;
       device:parameter_value "20"^^xsd:int ;
       device:time [ a time:TemporalEntity ] .
```

Listing 4.5: Reifications of a device description using (C) a reified property (D) a general purpose parameter concept.

4.2.3.5 Consolidated reification

We propose an alternative reification model which we call *consolidated reification* in the context of the Device Ontology. In this approach, we make use of the concept `device:State`. This concept is thought of as an abstraction of those aspects or conditions of a device which represent its changeable attributes and relationships. A device may have several states linked with specific parameter values using traditional data type or object properties. Listing 4.6 shows an example device description using this model.

Recall that we defined the concept `device:State` as a subclass of `event:Event`. This allows us to treat a device state as specific form of classification of space-time in the life cycle of a device, and link it with temporal objects, expressing that the parameters collected by the state hold during a specific space-time extent. Note that we only allow intervals to be linked to states.

```
# E) Consolidated reification using a device state model
:d device:Device ;
       device:state s1, s2 .

:s1 a device:State ;
       device:sensitivity "20"^^xsd:int ;
       device:threshold "0.5"^^xsd:float .

:s2 a device:State ;
       device:sensitivity "30"^^xsd:int ;
       device:threshold "0.8"^^xsd:float .
```

Listing 4.6: Device description using consolidated reification.

The benefits of this approach can be summarised as follows: Our device description stays within RDF/OWL specifications with regards to theory, model and syntax, as opposed to the approaches using the RDF reification syntax or named graphs. It is significantly less verbose than reifying each property whose object may be changing during the use of a device. Our model requires a minimum of N+2 triples when reifying N predicates, while reifying each predicate individually will result in a minimum of 3N triples using to the most concise alternative (C). This is because the consolidated model requires only two additional triples to declare a state and link it to the device, while reifying properties individually require 3 triples for each; one instantiating a reified concept, one linking it to the device, and one describing the value. Additionally, parameters that change together can be easily grouped using our model, and it is also easier to query for all parameters pertaining to a specific state.

Finally we have to consider *i)* how this model is linked to audio engineering workflows, *ii)* how to ensure device states are non-overlapping in time, and *iii)* how to describe individual decisions or conditions pertaining to changes in device parameters.

The first problem is easily solved using the Event Ontology. We can define events related to audio engineering tasks such as mixing or transformation (these are defined in the main part of the Studio Ontology), and link these to devices having one or more of their state parameters defined as sub-events. If the application does not require this level of detail, we may also forgo a detailed device state description. Therefore we have two degrees of complexity so far in our model. However, the second and third problems from above require a third level, in which we model the device life-cycle more accurately.

4.2.3.6 An accurate device state model

On the third and most complex level, we can still use the concise reification model described above, however we provide a mechanism to describe conditions of state changes.

Let a device $\mathbf{D}$ be a factor of a main event $\mathbf{E_M}$ having a temporal extent. This can be a recording for example. We can decompose this event into a number of instantaneous events $\mathbf{E_n}$ pertaining to changes in the states $\mathbf{S_n}$ of the device. A graphical illustration is provided in Figure 4.5. The predicates binding device states to events are defined as follows:

$$\forall\, S, E(\mathsf{device\!:\!entry}(S, E) \rightarrow \mathsf{device\!:\!State}(S) \wedge \mathsf{event\!:\!Event}(E) \\ \wedge\, \mathsf{event\!:\!sub\_event}(S, E)) \tag{4.14}$$

$$\forall\, S, E(\mathsf{device\!:\!exit}(S, E) \rightarrow \mathsf{device\!:\!State}(S) \wedge \mathsf{event\!:\!Event}(E) \\ \wedge\, \mathsf{event\!:\!sub\_event}(S, E)) \tag{4.15}$$

$$\forall\, S(\mathsf{device\!:\!State}(S) \rightarrow \exists^{\leq 1} E1, E2(\mathsf{event\!:\!Event}(E1) \wedge \mathsf{event\!:\!Event}(E2) \\ \wedge\, \mathsf{device\!:\!entry}(S, E1) \wedge \mathsf{device\!:\!exit}(S, E2))), E1 \neq E2 \tag{4.16}$$

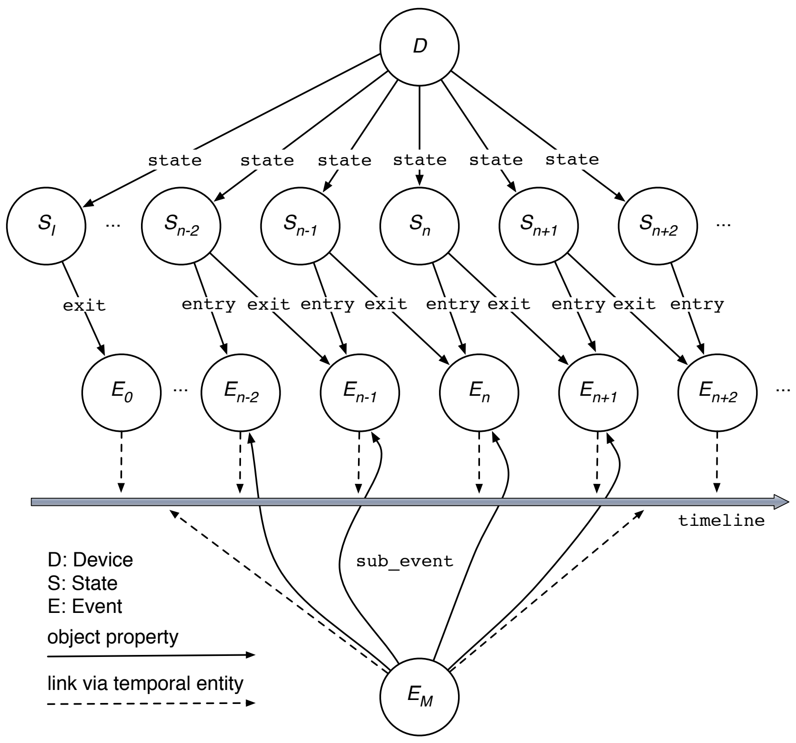


Figure 4.5: Accurate Device State Model

The counting existential quantification in the above definition (4.16) represents a global cardinality constraint on the properties: we may only define one entry and one exit event for each state. We may also forgo this description depending on the granularity of knowledge representation in a particular application. These constraints therefore are best encoded in OWL using the built-in classes `owl:FunctionalProperty` and `owl:InverseFunctionalProperty` which ensure pair-wise unique identification and exactly the required cardinality restriction.

In the above model, we borrow knowledge representation elements from the OWL description of UML state machines provided in [Dolog, 2005]. Our model however is considerably simpler, and resembles the paradigm of event driven finite state machines, in that it describes events related to an application of the device as the reason for state changes. These events are tied together as sub events of a main event representing the application. The model has the benefit of encoding chains of state changes during an event (with or without a temporal context), and the ability to assign additional information to entry and exit conditions which are modelled as events themselves. This information may be a link to an engineer, so we can encode fine details such as an *option* turned on by one engineer, and then turned off by another, or classifications of events causing state changes (e.g. automatic control, fault condition, engineering decision etc...).

Having defined a basic logical framework for describing devices and their parameters, we consider how simple or complex signal processing devices in audio engineering workflows may be described using this model, and how their connections can be represented.

4.2.3.7 Describing Complex Devices

An important class of devices in music production are artefacts for generating or manipulating audio signals. Here, we provide a conceptualisation of signal processing devices that represents their different aspects and supports versatile device descriptions. First, we outline a general model, that is useful, for instance, to describe audio effect units, and how they are related to corresponding physical phenomena. Then, we focus on representing concrete signal processing devices.

Signals are representations of some aspect or feature or the physical world. They may also be generated using a fine or coarse model of the world or an imaginary process, For instance, the physical model of a non-existing instrument. Similarly, signal processing devices are typically representations of real world phenomena, such that they reproduce the effect of a modelled phenomenon in the domain of signal representations.

We propose a layered conceptualisation of signal processing devices which accommodates this view. Not surprisingly, our model comes very close to the layered conceptualisation of intellectual works proposed in the FRBR model (see Section 3.2.3). However, due to having very different base entities in this domain, our ontology does not directly derive from FRBR. Figure 4.6 shows the basic entities and relationships we consider. It also illustrates the resemblance of our model to the relationships between the first group of FRBR terms.

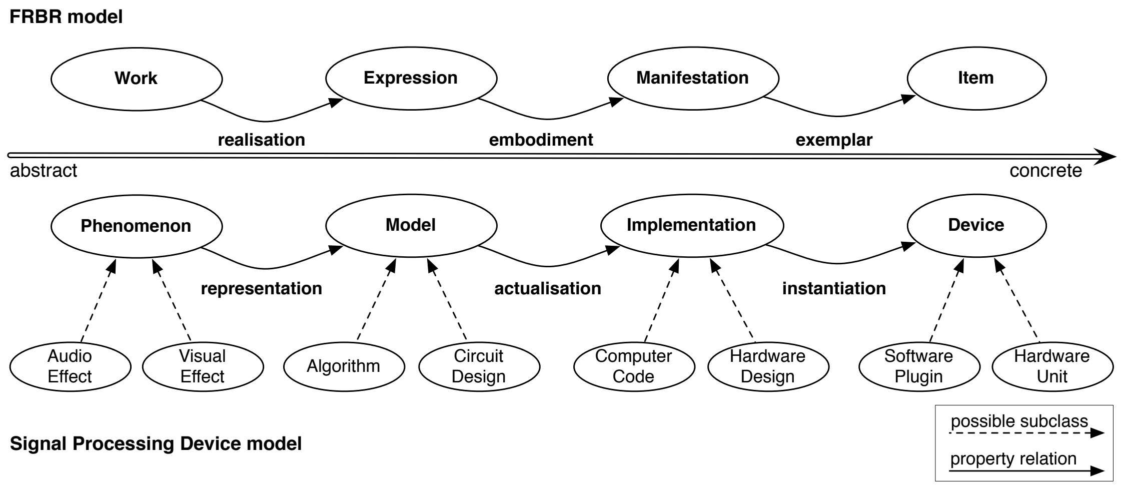


Figure 4.6: Fundamental entities and relationships in the model of signal processing devices

A *phenomenon* in our domain is best conceptualised as some perceived effect of a physical process. For example, the reflection of sound waves (echo) produces an audio effect, condensation of moisture in the air (fog) obscures visibility thus produces a visual effect. Here, the model is introduced through an image processing use case to emphasise its domain independence. Audio related examples will be discussed in the context of the core Studio Ontology. A physical phenomenon in this sense is the most abstract element of our model. It loosely corresponds to the abstract conception of intellectual works in FRBR. An algorithm, that reproduces the blurry effect of fog in a digital image, is essentially a *model* that approximates the perceived effect. Similarly to how a work may be realised thorough several expressions, a physical process may be represented in a different domain using several models. A model may have many different *implementations*, (loosely corresponding to manifestations), for instance, an algorithm implemented (actualised) in different programming languages, or a circuit design implemented using discrete components or using a Field-Programmable Gate Array (FPGA).

An instance (exemplar) of an implementation is a concrete *device*, for example, a plugin for an image editor producing the foggy effect, running on a computer which is in someone's possession. Thus our model is very similar to FRBR's conceptual layering. We can describe concrete instances of signal processing devices, linked to an actual implementation of a model or algorithm that represents a physical phenomenon.

We define the terms of this general model in the device ontology in order to keep it general, however, it is not a valid model of every possible technological artefact. For instance, we could not conceptualise a microphone this way. Our framework uses it to facilitate the description of complex signal processing devices. Since we consider this model more fundamentally applicable, it is also a good candidate for further modularisation. The specialisations of the entities shown in Figure 4.6 are examples only. Concrete classes deriving from this model are scattered throughout our framework. Next, we focus only on concrete devices, — that is, we provide more specific subclasses of the `device:Device` concept.

4.2.3.8 Signal Processing Device Ontology

We define the concept `SignalProcessingDevice` as a subclass of the more general device concept described in Section 4.2.3.3, having inputs or outputs for signal connectivity. From an ontological point of view this is sufficient to identify a signal processing device. The concept is interpreted broadly, and may stand for anything from a basic filter to a complex device such as a mixing console or an audio effect unit.

$$\forall\, D(\mathsf{spd\!:\!SignalProcessingDevice}(D) \rightarrow \mathsf{device\!:\!Device}(D) \tag{4.17}$$

$$\begin{aligned}&\forall\, D,T(\mathsf{spd\!:\!input}(D,T) \rightarrow \mathsf{spd\!:\!SignalProcessingDevice}(D) \wedge \mathsf{con\!:\!Terminal}(T)\\ &\wedge \mathsf{device\!:\!component}(D,T))\end{aligned} \tag{4.18}$$

$$\begin{aligned}&\forall\, D,T(\mathsf{spd\!:\!output}(D,T) \rightarrow \mathsf{spd\!:\!SignalProcessingDevice}(D) \wedge \mathsf{con\!:\!Terminal}(T)\\ &\wedge \mathsf{device\!:\!component}(D,T))\end{aligned} \tag{4.19}$$

We define `SignalProcessingDevice` subsuming `Device`, see (4.17), in a dedicated ontology called the Signal Processing Device Ontology (SPDO, prefix `spd:`), together with some fundamental signal processing components. More complex devices however are defined in the core Studio Ontology, where we also provide an appropriate workflow model (see 4.2.4.3).

A particular problem we need to consider at this point is how signal processing devices may be interconnected. While it appears convenient to link devices directly to audio signal entities defined in the Music Ontology, we have to be able to describe the relation of device characteristics and signal characteristics (e.g. analogue or digital or number of channels). In the above definitions (4.18 and 4.19) we make use of an additional fundamental element of our framework, the Connectivity Ontology (prefixed `con:`) which serves this purpose. This ontology provides essentially a reification mechanism to describe device connections precisely, and follows a paradigm used in audio processing workflow environments like MAX/MSP mentioned in Section 3.2.6.3.

4.2.3.9 Device Connectivity

The Connectivity Ontology allows for describing how signal processing devices, or other tools, such as microphones, in a recording and processing workflow are interconnected. Its central concept `con:Terminal` represents inputs and outputs in an abstract way, encompassing electrical or software interfaces and may be linked with a particular physical connector and communication protocol. In Figure 4.7 we illustrate its basic structure. The Connectivity Ontology incorporates the following three main aspects of device connectivity:

- **Concrete physical level**: dealing with types, form factors, pin layout and pin function of physical connectors. (This can be relevant in certain industrial applications of the ontology.)
- **Terminal characteristics**: defines a basic classification of terminal types: *electrical or optical, analogue or digital, input or output, balanced or unbalanced, mono, stereo or multichannel*, and permits describing, for instance, electrical characteristics of terminals.
- **Data communication level**: enables linking terminals to communication protocol standards, and describe their modes.

In Figure 4.7, the exemplified instances of `con:Connector` and `con:Protocol` can be thought to represent the output of a digital microphone having a 3 pin male XLR connector, and using the AES42 digital microphone interface protocol. The ontology defines some individuals of connectors and protocols common in audio production, however we may also define individuals of connectors or protocols in concrete device descriptions, where we need to go further than simple identification, for instance, if we need to describe the functionality of each pin of a complex connector type. Our ontology provides primitives for this purpose.

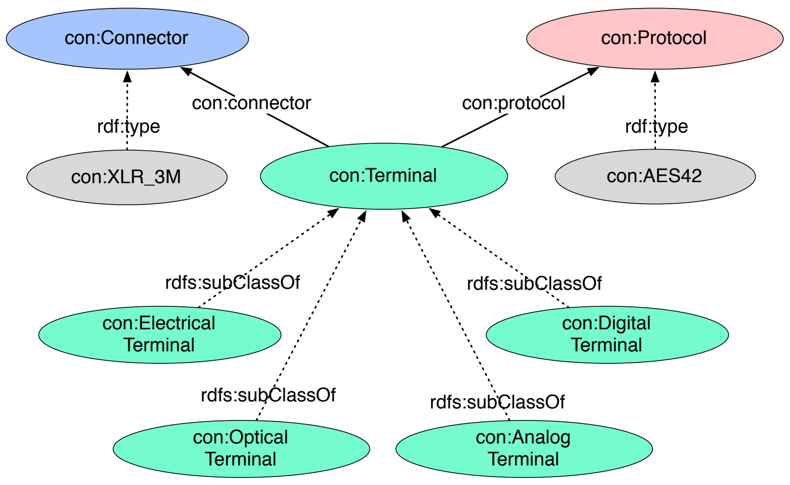


Figure 4.7: Overview of the Connectivity Ontology (with some subclasses and simplified examples)

```
:device1 a spd:SignalProcessingDevice ;
       spd:input :in1 ;
       spd:output :out1 .

:sig1 a mo:AnalogSignal .

:in1 a con:AnalogInput, con:BalancedTerminal ;
       con:connector con:XLR_3F ;
       con:terminal_channels "1"^^xsd:int ;
       con:signal :sig1 .

:out1 a con:DigitalOutput, con:BalancedTerminal, con:StereoTerminal ;
       con:connector con:XLR_3M ;
       con:protocol con:AESEBU ;
       con:terminal_channels "2"^^xsd:int ;
       con:signal :sig2 .

:sig2 a mo:DigitalSignal ;
       mo:channels "2"^^xsd:int .
```

Listing 4.7: Description of an analogue input and a digital output with corresponding signal connections.

An important feature of the ontology, illustrated in Listing 4.7, is the ability to match signal characteristics to interface characteristics. For instance, we can describe how many channels of a particular signal are accepted by a device terminal. In our example, we describe a device which has a balanced analogue input, and a balanced digital output using the AES3 digital audio standard[4] published as IEC 60958 Type I, (it prescribes balanced 3-conductor cabling with a 3 pin XLR connector), commonly called the AES/EBU standard in professional installations. Please note that the predicate `con:terminal_channels` is redundant in the output description, since we defined our terminal as `con:StereoTerminal` which has two channels by definition, provided in the ontology. This statement is included to illustrate the type of data that may be inferred, or used as constraint in a real-world application.

[4]see the European Broadcasting Union specification of the AES/EBU interface: http://tech.ebu.ch/docs/tech/tech3250.pdf

4.2.4 Core components

Having outlined the fundamental components required to describe the work of audio engineers, we now turn to more specific refinements of these ontologies that facilitate the description of the studio domain.

The Studio Ontology is the core component of our framework. It parallels the three levels of expressiveness of the Music Ontology and provides basic, recording-studio specific terms. On the first level, it provides for describing recording studios and facilities which loosely corresponds to the editorial level of the Music Ontology. The second level includes complex events, such as different types of recording and post production sessions, and provides for describing the production workflow on the level of audio transformations and signal processing (see Section 4.2.4.3). The third level provides some extension points to describe specific tools, such as multitrack audio production software (see Section 4.2.5.4); the audio editing workflow and project structure.

4.2.4.1 Describing recording studios

On the first level, the Studio Ontology provides for describing recording studios and facilities. This level builds on FOAF, Dublin Core, MO, and the Device, and Signal Processing Device ontologies. For example, we can differentiate between mastering, project and home studios:

$$\forall\, S(\mathsf{studio\!:\!RecordingStudio}(S) \rightarrow \mathsf{foaf\!:\!Organization}(S)) \tag{4.20}$$

$$\forall\, S(\mathsf{studio\!:\!HomeStudio}(S) \rightarrow \mathsf{studio\!:\!RecordingStudio}(S)) \tag{4.21}$$

$$\forall\, S(\mathsf{studio\!:\!ProjectStudio}(S) \rightarrow \mathsf{studio\!:\!CommercialStudio}(S)) \tag{4.22}$$

$$\forall\, S(\mathsf{studio\!:\!MasteringStudio}(S) \rightarrow \mathsf{studio\!:\!CommercialStudio}(S)) \tag{4.23}$$

$$\begin{aligned}&\forall\, S(\mathsf{studio\!:\!CommertialStudio}(S) \rightarrow \mathsf{studio\!:\!RecordingStudio}(S)\\ &\wedge \mathsf{mo\!:\!CorporateBody}(S))\end{aligned} \tag{4.24}$$

We can describe basic studio facilities using the terms:

$$\forall\, F(\mathsf{studio\!:\!StudioFacility}(F) \rightarrow \mathsf{owl\!:\!Thing}(F)) \tag{4.25}$$

$$\begin{aligned}&\forall\, F(\mathsf{studio\!:\!RecordingRoom}(F)\\ &\vee \mathsf{studio\!:\!MasteringRoom}(F)\\ &\vee \mathsf{studio\!:\!ControlRoom}(F)\\ &\vee \mathsf{studio\!:\!ListeningRoom}(F) \rightarrow \mathsf{studio\!:\!StudioFacility}(F))\end{aligned} \tag{4.26}$$

$$\begin{aligned}&\forall\, S, F(\mathsf{studio\!:\!facility}(S, F) \rightarrow \mathsf{studio\!:\!RecordingStudio}(S)\\ &\wedge \mathsf{studio\!:\!StudioFacility}(F))\end{aligned} \tag{4.27}$$

We also define different audio engineering and music producer roles, and enable linking engineers to different studios or studio facilities. This is useful when we need to describe commercial studios, where the name of the engineer or producer associated with the studio is often the most important factor.

$$\forall\, E(\mathsf{studio\!:\!SoundEngineer}(E) \rightarrow \mathsf{foaf\!:\!Person}(E)) \tag{4.28}$$

$$\forall\, P(\mathsf{studio\!:\!Producer}(P) \rightarrow \mathsf{foaf\!:\!Person}(P)) \tag{4.29}$$

$$\begin{aligned}&\forall\, S, E(\mathsf{studio\!:\!enigneer}(S, E)\\ &\ \rightarrow \mathsf{studio\!:\!RecordingStudio}(S) \vee \mathsf{studio\!:\!StudioFacility}(S)\\ &\ \wedge \mathsf{studio\!:\!SoundEngineer}(E) \vee \mathsf{studio\!:\!Producer}(E))\end{aligned} \tag{4.30}$$

Audio engineering roles such as `studio:mastering_engineer`, `studio:mixing_engineer`, `studio:lead_engineer` and so on, are defined as subproperties of `studio:engineer`. The above definition is a case in point where the Music Ontology could not be extended monotonically. Although it defines `mo:SoundEngineer`, it tightly links the term to its production workflow by defining that a necessary and sufficient condition for being an engineer is having to engineer a performance. This is not appropriate for our use case since we need to describe studios independently, therefore we link the term to the Music Ontology using `rdfs:seeAlso` only. We also define the term `studio:Producer`. This is a versatile role in music production. Producers often own or manage recording studios, participate in arrangements, and, especially in smaller studios, they fulfil engineering roles.

Finally, we define a vocabulary of tools. Instead of a detailed description, we list only some concepts in this vocabulary in Table 4.2. The terms typeset in bold are top level concepts in the Studio Ontology. Since these concepts are all conceptualised ultimately as some form of man made object, they are all derived from some subclass of `device:Device` (see Section 4.2.3.3). For the static description of studios, we define the predicate `studio:equipment` to link devices to recording facilities.

$$\begin{aligned}&\forall\, S, D(\mathsf{studio\!:\!equipment}(S, D) \rightarrow \mathsf{device\!:\!Device}(D))\\ &\mathsf{studio\!:\!RecordingStudio}(S) \vee \mathsf{studio\!:\!StudioFacility}(S)\end{aligned} \tag{4.31}$$

Now, we can provide descriptions such as:

> ”George Martin is a producer associated with AIR Studios including Studio1, which features a custom made 72 channel analogue Neve console George Martin and Rupert Neve made together.” (see 4.32)

or

> ”My bedroom studio features a digital audio workstation which consists of a Mac running Cubase and a Fireface800 audio interface.” (see 4.33)

$$\begin{aligned}&\mathsf{studio:Producer}(george\_martin) \wedge \mathsf{studio:CommercialStudio}(air)\\ &\wedge \mathsf{studio:producer}(air, george\_martin)\\ &\wedge \mathsf{studio:RecordingRoom}(studio1) \wedge \mathsf{studio:facility}(air, studio1)\\ &\wedge \mathsf{mx:AnalogConsole}(neve) \wedge \mathsf{studio:equipment}(studio1, neve)\\ &\wedge \mathsf{foaf:maker}(neve, rupert\_neve) \wedge \mathsf{foaf:maker}(neve, george\_martin)\\ &\wedge \mathsf{mx:channel\_count}(neve, 72)\end{aligned} \tag{4.32}$$

$$\begin{aligned}&\mathsf{studio:Engineer}(http://foaf.me/fazekasgy\#me) \wedge \mathsf{studio:HomeStudio}(my\_studio)\\ &\wedge \mathsf{studio:engineer}(my\_studio, http://foaf.me/fazekasgy\#me)\\ &\wedge \mathsf{studio:DigitalAudioWorkstation}(daw) \wedge \mathsf{studio:equipment}(my\_studio, daw)\\ &\wedge \mathsf{studio:AudioInterface}(fireface800) \wedge \mathsf{device:HardwareDevice}(mac)\\ &\wedge \mathsf{device:DAWSoftware}(cubase) \wedge \mathsf{device:component}(daw, cubase)\\ &\wedge \mathsf{device:component}(daw, mac) \wedge \mathsf{device:component}(daw, fireface800)\end{aligned} \tag{4.33}$$

Please note that `mx:AnalogConsole` (a subclass of `studio:MixingConsole`) and the property `mx:channel_count` is defined in a dedicated Audio Mixer Ontology (see Section 4.2.5.2). In fact, most categories of tools listed in Table 4.2 would ideally require an ontology extension, with the Studio Ontology tying together the top level concepts. Currently, we only cover a fraction of the domain of tools, most important in describing audio engineering workflows.

The vocabulary of tools provides only a shallow taxonomy of devices. We make a high level distinction between physical and other devices (such as software), as early as possible, based on the principle that some tools cannot exist without a physical manifestation. For example, microphones may have software models, (e.g. certain types of audio effects that are virtual models of highly regarded microphones) but these cannot replace the functionality of a physical microphone, similarly to how audio effect units and software effect plugins are functionally equivalent. These types of decisions therefore need to be deferred, sometimes until defining a particular individual, relying on the type of polymorphism provided by the ability to declare an individual as a member of several classes. We found in fact, that taxonomical organisation in describing complex audio production tools such as Digital Audio Workstations, is less important than partonomies [Tversky, 1989], an organising principle based on the *part-of* relation as opposed to the *kind-of* relation. Although partonomies are not directly supported by OWL, our device decomposition model discussed in Section 4.2.3.3 can be used to describe complex devices as shown for instance in Sentence 4.33.

Having defined the basic entities in the studio environment, we now turn to describing how they may be applied in audio engineering workflows.

concept	**base type**	**some subclasses**	**comments**
Microphone	PhysicalDevice	CondenserMicrophone, DynamicMicrophone, RibbonMicrophone, SurroundMicrophone,...	subclasses are defined in an extension (see Section 4.2.5.1)
AudioInterface	PhysicalDevice	-	typical part of a DAW
Amplifier	PhysicalDevice	PowerAmplifier, PreAmplifier	a more basic term is available in SPDO (see Section 4.2.3.8)
MixingConsole	Device	AnalogConsole, DigitalConsole, SoftwareConsole,...	subclasses are defined in an extension (see Section 4.2.5.2)
RecordingDevice	PhysicalDevice	AnalogRecorder, DigitalRecorder, TapeRecorder, DiskRecorder,...	excludes pre-electrical units
DigitalRecorder	RecordingDevice	HardDiskRecorder, CDRecorder, DATMachine, ADATMachine,...	e.g. a DATMachine is a DigitalRecorder and a TapeRecorder
DigitalAudioWorkstation	DigitalRecorder	-	the most complex recording device
DAWSoftware	SoftwareDevice	-	e.g. Audacity, ProTools, Cubase,...
MonitoringDevice	PhysicalDevice	NearFieldMonitor, FarFieldMonitor, Headphone,...	-
MeteringDevice	Device	SPLMeter, RT60Meter, CorrelationMeter, MeterBridge,...	-
EffectUnit	PhysicalDevice	-	see Section 4.2.5.3
EffectPlugin	SoftwareDevice	-	see Section 4.2.5.3

Table 4.2: Some terms in the vocabulary of music production tools

4.2.4.2 Describing audio engineering workflows

The second level of the Studio Ontology defines terms which enable tying together tools, agents and other entities in audio engineering workflows. It is based on the broad conceptualisation illustrated in Figure 4.8. This conceptualisation is supported by the author's experience as a sound engineer, and aligned with text book descriptions [Huber and Runstein, 2005] of audio engineering procedures, discussed in Section 1.5.

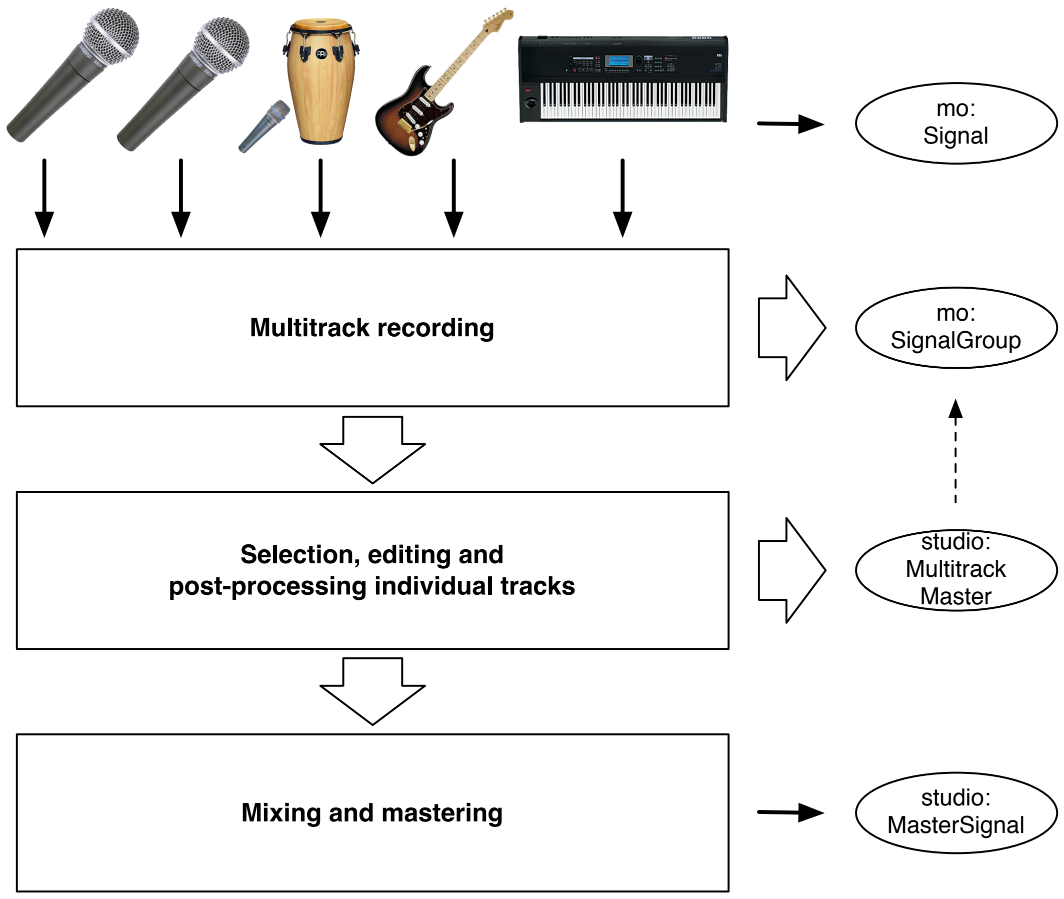


Figure 4.8: Recording studio workflow and corresponding signal entities (`MultitrackMaster` and `MasterSignal` are subconcepts of corresponding MO terms `SignalGroup` and `Signal`)

Here, we are most interested in describing audio *post-production*, providing workflow provenance in the context of an intelligent music production environment. Our model extends the workflow model of the Music Ontology. More precisely, it is embedded into its model, by defining additional events pertaining to audio signal transformations by an engineer, using the tools we discussed previously, and refining the concepts related to the grouping and the outcome of these transformations. We start by introducing production session types, loosely corresponding to the group of activities shown in the two bottom boxes of Figure 4.8.

The Music Ontology provides two concepts `mo:Recording` and `mo:RecordingSession` (see Table 4.1) related to audio engineering workflows. `mo:RecordingSession` corresponds to all activities which lead to a record that may be published. We can decompose a session into individual `mo:Recording` events, such as a recording of a song or individual recordings of instruments. This broad conceptualisation works well, if we want to express information such as: "The songs on this album were produced during a recording session lasting 3 days and include all the recordings of the songs." Audio engineering however includes many types of activities other than recording. These need to be grouped using a richer semantics related to sessions in a recording studio.

First, we introduce a broad top-level concept `studio:StudioSession` (4.35) to describe session types not necessarily related to recording sessions. For instance, AIR Studios mentioned in our earlier example, features dedicated rooms for song writing, or rehearsal sessions, without special recording equipment. Here however, we only detail those sessions related to engineering workflows.

$$\forall\, S(\mathsf{studio:StudioSession}(S) \rightarrow \mathsf{event:Event}(S)) \tag{4.34}$$
$$\forall\, S(\mathsf{studio:PostProductionSession}(S) \tag{4.35}$$
$$\rightarrow \mathsf{mo:RecordingSession}(S)) \wedge \mathsf{studio:StudioSession}(S)) \tag{4.36}$$
$$\forall\, S(\mathsf{studio:EditingSession}(S) \rightarrow \mathsf{studio:PostProductionSession}(S)) \tag{4.37}$$
$$\forall\, S(\mathsf{studio:MixingSession}(S) \rightarrow \mathsf{studio:PostProductionSession}(S)) \tag{4.38}$$
$$\forall\, S(\mathsf{studio:MasteringSession}(S) \rightarrow \mathsf{studio:PostProductionSession}(S)) \tag{4.39}$$

Using the above terms, we can group activities related to post-production. Being an event itself, `studio:PostProductionSession` (4.36) may be decomposed into editing, mixing and mastering sessions (4.37-4.39), which, in modern record production are often carried out by different studios, and involve different audio engineers.

Compared to the Music Ontology, we also refine the possible products of recording sessions. The concept `mo:SignalGroup` can be used to collect signals produced by recordings. Its semantics however are not clear enough to know whether a signal group refers to the multitrack master recordings of a song, or all signals ever recorded during a long recording session. In fact, the Music Ontology defines this concept in the latter sense, but it is used for both cases lacking a more precise definition. We provide two extensions shown in Figure 4.8 to refer to multitrack masters, and mixed and finalised release masters.

$$\forall\, S(\mathsf{studio:MasterSignal}(S) \rightarrow \mathsf{mo:Signal}(S)) \tag{4.40}$$
$$\forall\, S(\mathsf{studio:MultitrackMaster}(S) \rightarrow \mathsf{mo:SignalGroup}(S)) \tag{4.41}$$

A master signal represents a particular adaptation of the recording of a song to a given media (i.e. CD, vinyl, radio, download) which require different mastering procedures; e.g. different loudness compression and equalisation. The concept `studio:MultitrackMaster` (4.41) represents the edited and synchronised signals — strictly related to the performance of a single work — that can be mixed to produce a final release. We also define the sub-properties `studio:produced_master` and `studio:produced_multitrack_master`, to refer to these concepts directly. Using the facilities introduced so far, we can describe the recording and post-production of a song as shown in Listing 4.8.

```
:rs a mo:RecordingSession ;
  mo:engineer [ a mo:SoundEngineer ; foaf:name "Mike" ] ;
  event:sub_event :rec1, :rec2, :ps;
  mo:produced_signal_group :record_takes ;
  mo:produced_signal :master_signal .

:rec1 a mo:Recording ;
  rdfs:comment "voice" ;
  mo:produced_signal :sig1 .

:sig1 a mo:Signal .

:rec2 a mo:Recording ;
  rdfs:comment "guitar" ;
  mo:produced_signal :sig2 .

:sig2 a mo:Signal .

:record_takes a mo:SignalGroup ;
       mo:signal :sig1, :sig2 .

:ps a studio:PostProductionSession ;
       studio:mastering_engineer [
              a studio:SoundEngineer ; foaf:name "Chris" ] ;
       event:factor :sig1, sig2 ;
       studio:produced_master :master_signal .

:master_signal a studio:MasterSignal .
```

Listing 4.8: A recording session with post-production

As we can see from the example, this is not a big departure from the original Music Ontology workflow. We were able to say that the recording session involved post-production with a different mastering engineer. This session took the originally recorded signals as factors, and produced a (presumably mixed) master signal. We could not yet describe, however, what exactly happened during post-production, or precisely define how its factors are related to the main event. We now outline how audio engineering workflows can be described in detail.

Recall that the Music Ontology describes production workflows by using events which enable moving between the different layers of the FRBR model. For instance, a *performance event* of a musical work produces a sound. A *recording event* of this sound produces a signal (both sound and signal are musical expressions). The next event in this model is `mo:ReleaseEvent` which takes an expression (signal) and produces a manifestation, e.g. a published record or track. Most audio engineering work is related to the recording of a performance, and what happens between recording and publication. Therefore, our model has to deal with the transformation of expressions, sounds and signals. Here, we are most concerned with post-production workflows, and with signal transformations.

We define four new event types which come after `mo:Performance` and `mo:Recording` in the overall workflow, but before `mo:ReleaseEvent`.

$$\forall\, E(\mathsf{studio\!:\!Edit}(E) \vee \mathsf{studio\!:\!Transform}(E) \vee \mathsf{studio\!:\!Mixing}(E) \rightarrow \mathsf{event\!:\!Event}(E)) \tag{4.42}$$

$$\forall\, M(\mathsf{studio\!:\!Mastering}(M) \rightarrow \mathsf{studio\!:\!Transform}(M)) \tag{4.43}$$

The concept `studio:Edit` corresponds to actions such as select, cut, or move. These refinements are provided in an extension of the Studio Ontology in the context of multitrack editing (see Section 4.2.5.4). The term `studio:Transform` represents the application of audio processing tools, such as audio effects, while `studio:Mixing` refers to the combination of audio signals. In order to describe the relationships of these events to signals operated on, we need to extend the event model discussed in Section 4.1.2.2. The music ontology defines `mo:produced_signal` as a sub-property of `event:product`, to link a recording event to a signal it produced. Similarly, we introduce the properties `studio:consumed_signal` and `studio:consumed_signal_group` as follows:

$$\begin{aligned}&\forall\, E, S(\mathsf{studio\!:\!consumed\_signal}(E,S)\\ &\ \rightarrow \mathsf{event\!:\!Event}(E) \wedge \mathsf{mo\!:\!Signal}(S) \wedge \mathsf{event\!:\!factor}(E,S))\end{aligned} \tag{4.44}$$

$$\begin{aligned}&\forall\, E, S(\mathsf{studio\!:\!consumed\_signal\_group}(E,S)\\ &\ \rightarrow \mathsf{event\!:\!Event}(E) \wedge \mathsf{mo\!:\!SignalGroup}(S) \wedge \mathsf{event\!:\!factor}(E,S))\end{aligned} \tag{4.45}$$

The properties defined in Sentence 4.44 and 4.45 can be used to link events to signals that are used as factors in events. For example, a mixing event may consume a group of signals and produce a mixed signal. The definitions could be refined by constraining the domain of these properties, for instance, to a union of `studio:Edit`, `studio:Transform`, and `studio:Mixing`. However, this is currently not provided. We similarly define `studio:produced_signal`, because the domain of `mo:produced_signal` is limited to `mo:Recording`. Next, we outline how the above tools can be used to describe signal processing workflows in the studio. First outline how a set of mixing and transformation events form an *event flow*, and then we examine how these can be linked to actual devices, and describe the *signal flow* and device configuration.

4.2.4.3 Signal processing workflows

To describe how a piece of music is processed in the studio, it is insufficient in itself to describe a signal flow (i.e. flow chart) or a set of transformations. We need to consider a random set of mixing or transformation events, as in non-linear editing (i.e. with random or non-sequential access), as well as real-time, quasi-simultaneous transformations, such as a signal routed through several processing units for recording. (Apart from the small latency of signal processing units, these have the same duration as the recording event itself.) To fulfil both requirements, we consider parallel signal and event flows linked using signal entities that are instances of the `mo:Signal` concept. This is illustrated in Figure 4.9.

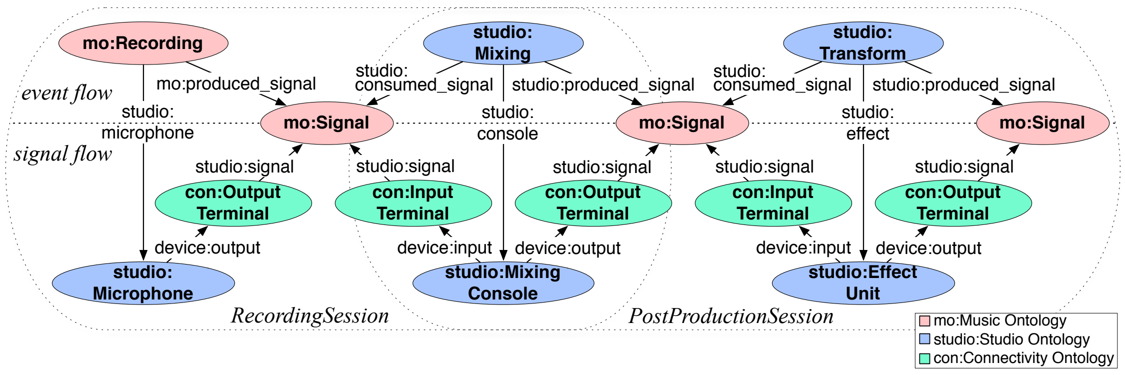


Figure 4.9: Recording, mixing and transformation events with an associated signal flow

The top part of the figure shows a recording, a mixing and a transformation event. Several signals (not shown for brevity), or a signal group can be attached to a mixing event and corresponding device. The mixed results may be processed further, for instance in a mastering session, producing a master signal.

We conceptualise the event flow as a series of events, where each processing event uses the product of another event as factor. The relationship between events is made explicit by the sub-properties `studio:produced_signal` and `studio:consumed_signal` we defined previously. The outlined set up signifies our ontological commitment to changing identities, a problem thoroughly discussed in philosophy [Strawson, 1959]. Once transformed, a signal receives new identity which alleviates difficult transaction management problems in our system regarding the changing attributes of signals.

```
:tr a studio:Transform ;
       studio:mastering_engineer <http://foaf.me/fazekasgy#me> ;
       studio:device [ a studio:EffectUnit ; device:model "TC M300" ] ;
       event:time [ a tl:Instant ;
              tl:timeline tl:universaltimeline ;
              tl:at "2011-03-20T09:00:00Z"^^xsd:dateTime ] ;
       studio:media_time [ a tl:Interval ;
              tl:timeline :sig_in_timeline ;
              tl:beginsAtInt "20000"^^xsd:int ;
              tl:durationInt "441000"^^xsd:int ] ;
       studio:consumed_signal :sig_in ;
       studio:produced_signal :sig_out ;

:sig_in_timeline a tl:DiscreteTimeLine ;
       rdfs:comment "The timeline of the source signal" .

:sig_out_timeline a tl:DiscreteTimeLine ;
       rdfs:comment "The timeline of the result signal" .

:sig_in a mo:Signal ;
       mo:sample_rate "44100"^^xsd:float ;
   mo:time [ a tl:Interval ;
       tl:timeline :sig_in_timeline ] .

:sig_out a mo:Signal
       mo:time [ a tl:Interval ;
       tl:timeline :sig_out_timeline ; ] .
```

Listing 4.9: Describing a signal transformation

Our use cases include the need for describing the event flow of audio transformations, as well as tracking the provenance of signals and edit decisions. Therefore, we have to consider linking processing events to the universal timeline in order to describe *exactly when* and *in what order* they have happened, and to a signal timeline to express that a transformation was applied, for instance to the chorus section of a song, or that a mix involves two signals with different time intervals at different positions related to the time extents of each signal.

This is achieved by introducing the property `studio:media_time` which is used to link transformation, mixing and edit events to signals. At the same time, the property `event:time` can be used to link processing events to the universal or an abstract timeline which can be used to simply order events without explicit timing. This is exemplified in listings 4.9. Here, we describe a transform event, represented by `studio:Transform` which was executed exactly at 9 AM on 2011-03-20, and provide a sample accurate description of how the transform was applied to a 10s portion of the signal starting at sample 20000. We also describe the resulting signal, and link the transform to an engineer and a device executing it. This description thus provides full provenance or the application of the transform. We can also link this description to our previous example of Listing 4.8, making it part of a post-production, recording or mastering session.

Using this information, an agent is able to render the audio by selecting the signal at each point in time, that is the result of the most recent or last transformation in the flow. This is in fact very similar to how an audio editor program — with an unlimited undo buffer — structures audio data and metadata related to user actions. Therefore, our ontologies are relatively easily linked to data structures, if implemented in a semantic audio editor.

Finally, we describe how the event flow is connected to a signal flow, illustrated in the bottom part of Figure 4.9. Essentially, we link events to devices using `studio:device`, a sub-property of `event:factor`. Some specialisations such as `studio:microphone`, `studio:console` or `studio:effect` can be used to make the relationship more explicit. We reify device connections using subclasses of the `con:Terminal` concept defined in the Connectivity Ontology. We already discussed a reason for this reification in Section 4.2.3.9 — that is, the need for matching signal and terminal characteristics. For example, we need to be able to describe that a device processes only the first two channels of a multichannel signal. Another reason for this reification is the need for describing exact signal connections in events which work with multiple signals. For instance, in case of mixing events, we may link several instruments to particular channels of a mixing console, with specific channel configurations. We discuss these problems in more detail in the context of the Audio Mixer Ontology in Section 4.2.5.2. The description of device configuration was discussed in the context of the Device Ontology, please see for instance listings 4.6 in Section 4.2.3.5.

The third level of the studio ontology defines highly specific terms related, for instance, to media adaptation, format transfer or sequence editing, a kind of mastering session where the best order of songs for an album is selected. This level is includes some very closely tied extensions which rely on both the fundamental elements of our framework, (e.g. the Microphone and Audio Mixer Ontologies depend mainly on the Device, Signal Processing Device and Connectivity ontologies) or provide extensions to the audio engineering workflow such as the Multitrack Ontology describing multitrack audio editors and software components of digital audio workstations. In the next section, we provide a brief outline of some extensions.

4.2.5 Extensions

Ontology extensions are useful to allow the user to choose a desired level of granularity, given some domain specific details provided by the modeller. In this section we describe some extensions of the Studio Ontology. We examine four core areas of the audio engineering workflow, where we need to provide more specific device descriptions, or need to cover specific know-how, such as microphone techniques. These areas are: Audio Recording, Audio Mixing, Audio Effects and Audio Editing.

4.2.5.1 Audio Recording

The Studio Ontology framework provides one extension to describe audio recording at present, the Microphone Ontology. This ontology concerns both microphones and microphone techniques. Its core concept `mic:Microphone` is plugged under the `studio:Microphone` concept, and serves as basis for a small taxonomy of microphones, organised by their transducer principle or diaphragm type. (The diaphragm of a microphone captures air pressure variation caused by sound and converts it to mechanical motion (in most cases) which may be captured as another form of energy.) This is the most common way audio engineers categorise microphones, besides their polar patterns, also known as directivity pattern which describes how sensitive a microphone is to sounds arriving from a particular direction.

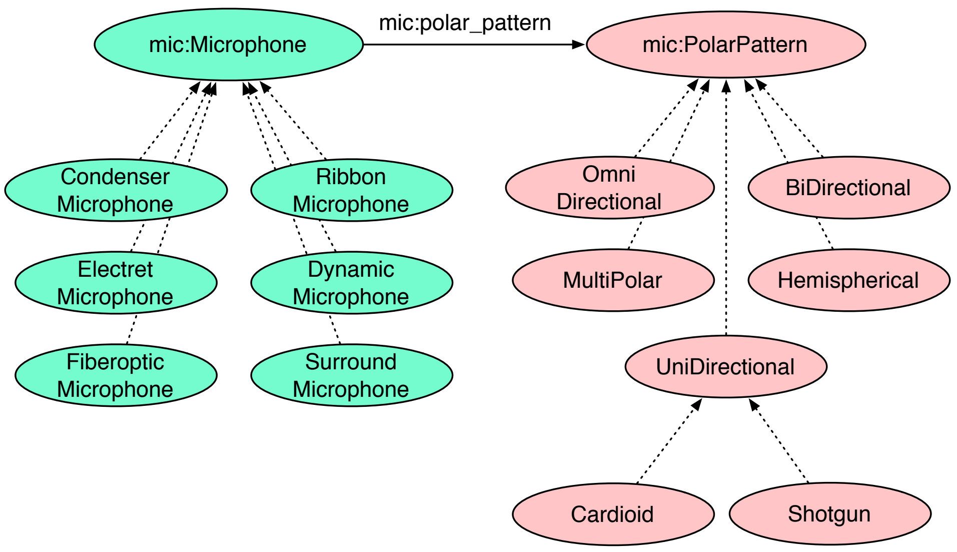


Figure 4.10: Microphone Ontology (partial extract - dashed lines represent subsumption relations)

Our ontology includes both taxonomies, and defines the predicate `mic:polar_pattern`, to link microphones to individual polar patterns, that are specific instances of the main types of polar patterns shown in Figure 4.10 (please note that concept names are abbreviated for brevity compared to the ontology definitions). Individuals of `mic:CardioidPolarPattern` includes for instance `mic:Cardioid`, `mic:WideCardioid`, `mic:SubCardioid`, `mic:HyperCardioid`, and `mic:SuperCardioid`. However, more esoteric polar patterns can also be described in individual cases. A further advantage of this configuration, is the ability to express class specific restrictions on polar pattern predicates, to represent that not all polar patterns can be realised by all capsule technologies, however this is not currently included in the ontology.

The Microphone Ontology also allows for describing most properties one may find in a microphone data sheet, for instance `mic:diaphragm_size`, `mic:high_frequency_rolloff` or `mic:output_impedance`. The `mic:Configuration` concept (subclass of `device:State`) can be used to describe variable parameters of microphones such as sensitivity, or variable polar pattern setting if these change during a particular recording event as described in the context of the Device Ontology (see Section 4.2.3.3).

We can also describe microphone placement, using properties for describing distance from the sound source, as well as azimuth and elevation relative to the longest axis or the principal radiating direction of the sounding body of the instrument. This is in contrast with the Music Ontology which only allows the localisation of recording events using geographical coordinates. This isn't very useful from an audio engineering point of view. However, our conceptualisation is still not clear of ambiguities. For instance, it may be the case that neither the radiation pattern nor the main axis is easily identified. There are no agreed upon standards for describing microphone placement, therefore these data may not be interpreted precisely without a natural language description. A possible solution to this problem is the development of a Musical Instrument Ontology (see Section 4.5) which enables the description of an instrument. We can then provide instrument specific terms to describe recording.

The Microphone Ontology includes the concept `mic:MicrophoneArrangement` and allows for describing stereo and spatial recording techniques, such as a `mic:BlumleinPair`, a `mic:MidSide`, `mic:ORTF`, or `mic:DeccaTree`, with their constituent microphones, their distances, angles and configurations. For instance, the description of the Mid/Side technique involves two microphones, one with a cardioid polar pattern placed on-axis, and one with a bidirectional polar pattern, arranged in a 90 or 270 degrees angle. In these cases, the placement predicates mentioned above should describe the placement of the whole array (i.e. how its central axis is related to the sound source).

Besides microphones, further ontology extensions are required to describe the details of recording devices or audio interfaces. This constitutes future work. In the next section we review our ontology for audio mixing.

4.2.5.2 Audio mixing

The Audio Mixer Ontology allows detailed description of mixing consoles both in terms of static characteristics and particular settings such as channel strip configuration in a recording event. The ontology is modelled after a generalised blueprint of mixing consoles shown in Figure 4.12. This was obtained from studying several commercial hardware designs[5], however, software implementations were also taken into account. Our ontology depends on terms defined in the Device (Section 4.2.3.3) and Connectivity (Section 4.2.3.9) ontologies.

The ontology itself defines concepts such as `mx:Channel` which represents a channel strip. This can be linked with parameters such as fader levels, panning, and equalisation using, for instance, an equaliser module defined in an Audio Effects Ontology (Section 4.2.5.3). The concept `mx:Bus` represents a mixing bus of which several can be defined and linked with channels using the `mx:bus` property. Such an association represents signal *routing* in audio engineering terms. The concept `mx:InsertTerminal` is defined as subclass of both `con:InputTerminal` and `con:OutputTerminal` as it serves both functionality.

The description of panning, that is, stereo and multichannel spatial positioning cannot be generalised apart from the simple stereo case. Therefore this is not shown in the diagram. Here, we adopt an approach which is similar to how surround panning is described in the AES31-3[6] standard. The predicate `mx:pan` and its equivalent `mx:left_right_position` represents traditional stereo panning, while using a combination of `mx:left_right_position` and `mx:front_rear_position` we can represent surround panning irrespectively of the number of target channels.

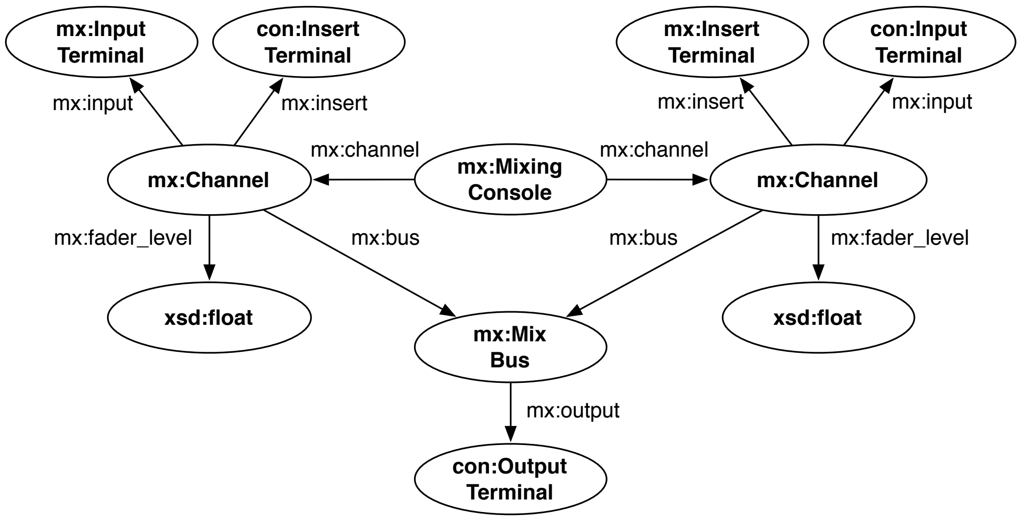


Figure 4.11: Using the Audio Mixer Ontology

[5]Circuit diagrams were obtained through an authorised service centre of studio equipment.
[6]AES31-3-2008 Standard: http://www.aes.org/publications/standards/search.cfm?docID=32

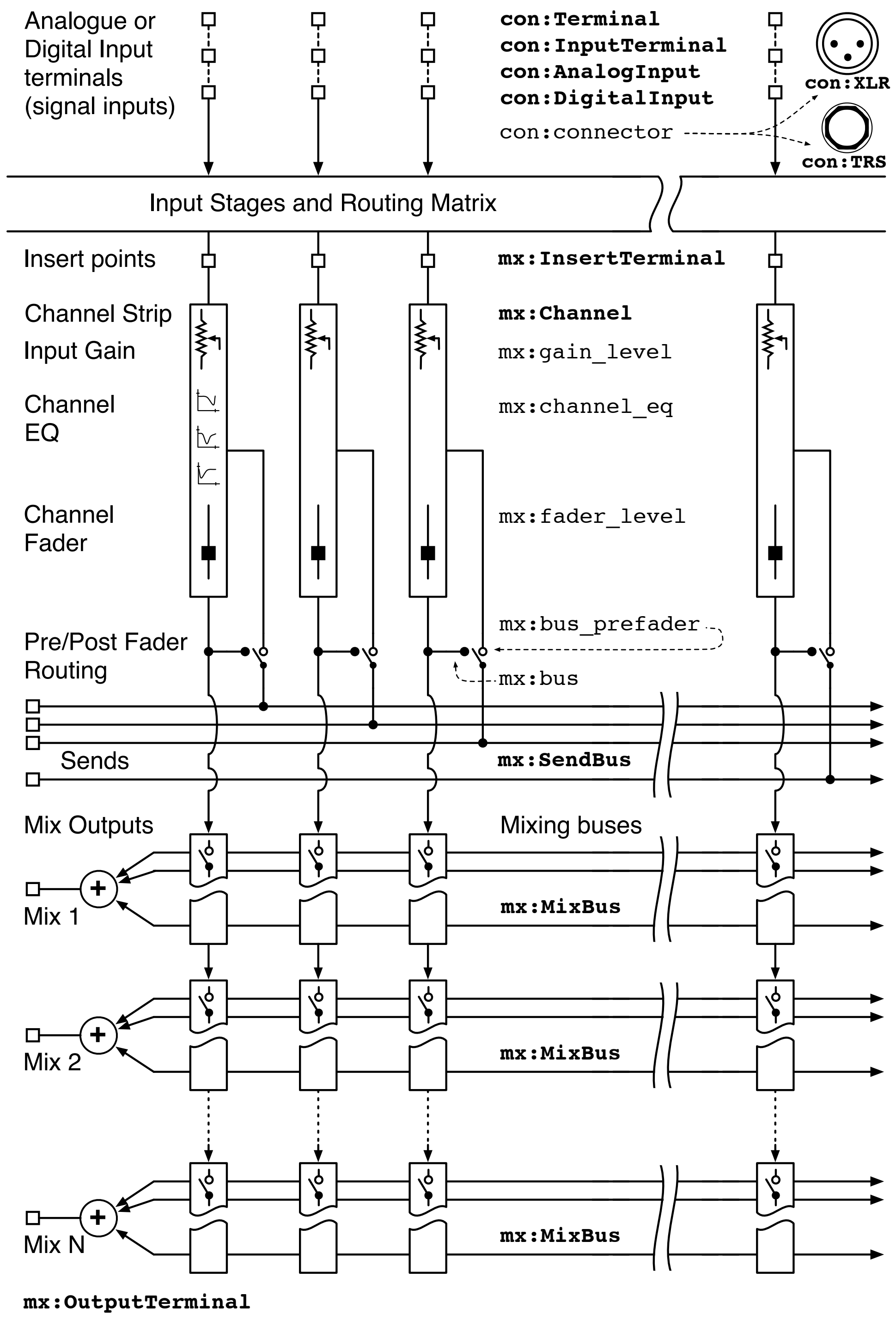


Figure 4.12: Overview of the Audio Mixer Ontology (simplified - `mx:` terms set in `bold` represents concepts, all other terms are predicates)

The simplest use of the ontology is depicted in Figure 4.11. We describe a static mixing situation, involving two channels and a single mixing bus. In more complex cases, the full configuration may be linked to a device state (recall, this is conceptualised as an abstraction of the changeable attributes of a device, see Section 4.2.3.3) instead of a mixing console instance directly, which allows the description of a sequence of routing decisions and other random parameter changes during a recording event. Some configuration changes however, such as fader automation, may be too frequent, and are too uniform to be efficiently represented by the full-fledged device state model. Therefore we provide object properties, such as `mx:fader_automation` which link a channel to an *automation signal*. An automation signal is a discrete time signal, whose timeline is linked to the audio signal timeline using the timeline map concept discussed in Section 4.1.2.1. This provides a mechanism for sample accurate representation of automated mixing parameters.

4.2.5.3 Audio effects

The core Studio Ontology includes concepts to refer to audio effect units and plugins that are particular hardware or software devices (see Table 4.2 in Section 4.2.4.1), and a small independent taxonomy of audio effects and processors based on their typical applications in audio engineering. This adheres to the common distinctions engineers make between non-linear audio transformations and audio effects based typically on whether they are used in the side-chain or send bus or at an insert terminal. This distinction however is not scientifically sound, nor can it be explained by implementation details or perceptual characteristics.

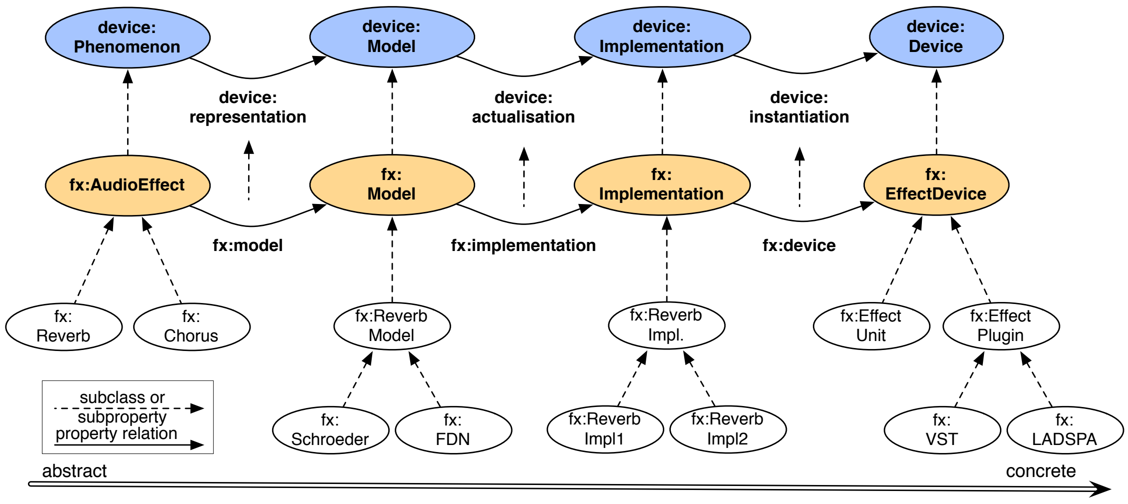


Figure 4.13: Harmonisation of an Audio Effects Ontology with the Device Ontology

Audio effects in essence are best conceptualised as physical *phenomena*, separated from their *models* (circuit designs or algorithms), particular *implementations* that bear the same characteristics (an algorithm implemented in Matlab, C++ or Python), and concrete *devices*, such as a plugin or an effect unit that one may own. This can be expressed using the device description model discussed in Section 4.2.3.7. The harmonisation of an Audio Effects Ontology to our model is shown in Figure 4.13.

The reason for defining four conceptual layers for describing audio effects can be justified by the need for expressing different attributes of entities existing on these levels. Similarly to the way one may explain the FRBR model, we can argue that a physical process producing an audio effect does not have the same characteristics as a computer model, of which several different types can be created. This is similar to how a musical work may be expressed differently in various performances.

The distinction between model, implementation and device is more subtle. However, we can argue that an implementation may not have the same parameters as the model, for example, some complex parameters of an algorithm may not be exposed to the user in a particular implementation. The difference between implementation and device is analogous to the difference between a *manifestation* and an *item* in FRBR terms. A record may be available on CD, Vinyl, or as a digital download, yet it is the same record. Similarly, an implementation of an effect may be available as a VST plugin, and extension module for a digital console, or a standalone effect unit, yet it is the same implementation (e.g. same C++ code compiled for different CPUs, and wrapped in different APIs). Albeit this model is accurate, using all layers can result in complex data which may be unnecessary in some applications. This issue is easily resolved however by defining short-cut properties between layers. For instance, we may link a plugin directly to an effect, if we don't care about the algorithmic model and implementation details.

Using the Studio Ontology, the application of audio effects to signals can be described using the concept `studio:Transform` which is an *event* that takes a signal as a factor, (expressed using `studio:consumed_signal`) and produces a transformed signal (`studio:produced_signal`). This concept is subsumed in a more specific effect ontology. The transform can be linked to two time objects, one relating it the the universal timeline, and another to the processed signal timeline.

The Studio Ontology sets the problem of audio effect classification aside. Creating Audio Effects ontologies based on multidisciplinary classification [Verfaille et al., 2006a] is our ongoing collaborative work [Fazekas et al., 2011], [Wimering et al., 2011]. The four organising principles we consider are as follows:

- **Perceptual attributes**: describe how an effect modifies *loudness, pitch, timbre*, or *spatial localisation*. This classification has mainly an aesthetic value.
- **Implementation**: A technical classification may be based on the type of algorithms involved, (e.g. *filter, delay, modulator*), or whether the effect uses time or frequency domain processing.

- **Parameters**: Another technical classification is possible by considering standard parameters, (e.g. *delay time, LFO frequency*) common in different underlying implementations.
- **Application**: Classify effects by their typical application in audio engineering. For instance, we can distinguish between *artistic effects* and *processors* that are used for signal conditioning, e.g. *mastering processors*.

All four organising principles result in different classification schemes which have specific applications in audio production. For instance, find audio effects which affect the same perceptual attribute, effects which have the same parameters, or match audio effects by different manufacturers. Since accommodating these different views are best achieved by designing different ontology modules, we consider moving all audio effect related terms to an extending ontology apart from the top level concepts related to the identification and application of effect devices.

4.2.5.4 Audio editing

The final set of extensions we provide deal with two specific domains of audio editing: complex tools, such as multitrack audio editing hosts and digital audio workstation software, and edit decisions in audio post-production workflow.

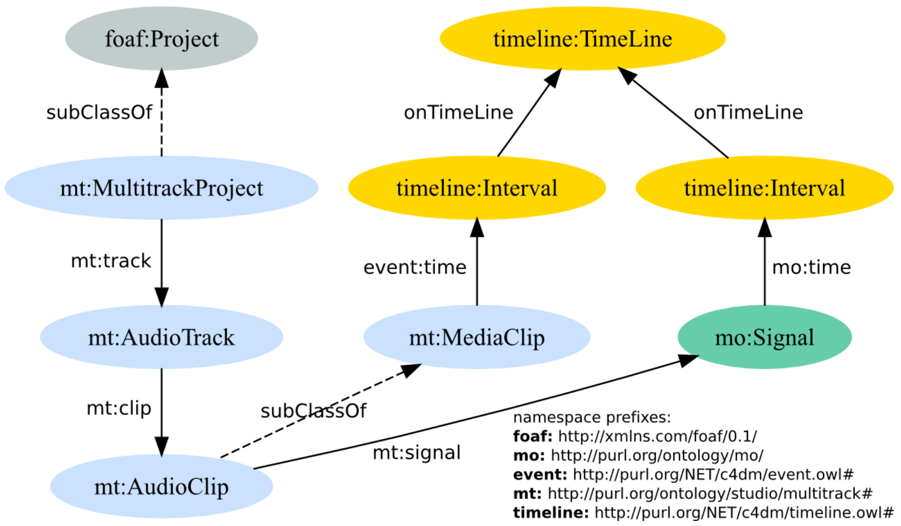


Figure 4.14: Using the Multitrack Ontotlogy

Modern digital audio workstations organise recording projects into a set of tracks — which may correspond to physical input channels or virtual channels created in an ad hoc way — and potentially overlapping clips contained in them corresponding to various takes during a recording session. This organisation is best represented independently from the signal processing workflow. The Multitrack Ontology serves this purpose. It relates the the hierarchy of *Clip*s and *Track*s to other concepts in the Music and Studio ontologies, and defines terms such as `mt:MultitrackProject`, `mt:MediaTrack`, `mt:AudioTrack`, and `mt:AudioClip` as follows:

$$\forall\, P(\mathsf{mt\!:\!MultitrackProject}(P) \rightarrow \mathsf{foaf\!:\!Project}(P)) \tag{4.46}$$
$$\forall\, T(\mathsf{mt\!:\!ProjectTrack}(T) \rightarrow \mathsf{owl\!:\!Thing}(T)) \tag{4.47}$$
$$\forall\, C(\mathsf{mt\!:\!MediaClip}(C) \rightarrow \mathsf{event\!:\!Event}(C)) \tag{4.48}$$
$$\forall\, M(\mathsf{mt\!:\!MediaTrack}(M) \rightarrow \mathsf{mt\!:\!ProjectTrack}(M)) \tag{4.49}$$
$$\forall\, U(\mathsf{mt\!:\!AnnotationTrack}(U) \rightarrow \mathsf{mt\!:\!ProjectTrack}(U)) \tag{4.50}$$
$$\forall\, A(\mathsf{mt\!:\!AudioTrack}(A) \rightarrow \mathsf{mt\!:\!MediaTrack}(A)) \tag{4.51}$$
$$\forall\, V(\mathsf{mt\!:\!VideoTrack}(V) \rightarrow \mathsf{mt\!:\!MediaTrack}(V)) \tag{4.52}$$
$$\forall\, AC(\mathsf{mt\!:\!AudioClip}(AC) \rightarrow \mathsf{mt\!:\!MediaClip}(AC)) \tag{4.53}$$
$$\forall\, VC(\mathsf{mt\!:\!VideoClip}(VC) \rightarrow \mathsf{mt\!:\!MediaClip}(VC)) \tag{4.54}$$

The most important properties are defined as:

$$\forall\, P,T(\mathsf{mt\!:\!track}(P,T) \rightarrow \mathsf{mt\!:\!MultitrackProject}(P) \wedge \mathsf{mt\!:\!ProjectTrack}(T)) \tag{4.55}$$
$$\forall\, T,C(\mathsf{mt\!:\!clip}(T,C) \rightarrow \mathsf{mt\!:\!ProjectTrack}(T) \wedge \mathsf{mt\!:\!MediaClip}(C)) \tag{4.56}$$
$$\forall\, C,S(\mathsf{mt\!:\!signal}(C,S) \rightarrow \mathsf{mt\!:\!AudioClip}(C) \wedge \mathsf{mo\!:\!Signal}(S)) \tag{4.57}$$
$$\begin{aligned}&\forall\, C,T(\mathsf{mt\!:\!project\_time}(C,T)\\ &\rightarrow \mathsf{mt\!:\!MediaTrack}(C) \vee \mathsf{mt\!:\!MediaClip}(C) \wedge \mathsf{time\!:\!TemporalEntity}(T))\end{aligned} \tag{4.58}$$

The use of these terms and their link to other ontologies in shown in Figure 4.14. Audio tracks and clips are conceptualised as container structures, as opposed to signals themselves. A track may represent several takes during a recording session, and contain a set of clips linked to signals, however they may also be empty. To express temporal relations between signals, clips, tracks and recording projects, these entities may be associated with time objects, defined on the timeline corresponding to a recording project. For instance, the relative temporal location of a track within a project can be described using `mt:project_time`. This conceptualisation closely corresponds to the object model of multitrack audio editors, where signals are represented as raw data object related to files on disk (which may be split to small blocks addressed separately), and clips refer to these object using pointers. For this similarity, our ontology is relatively easily linked with classes in a semantic audio editor.

A small Edit Ontology provides for describing successions of edit decisions. This ontology defines a handful of common operations in audio editors shown in Figure 4.15. For instance, we define a complex choice concept `edit:Choice` which operates on `mo:SignalGroup` entities. The result of such an operation is typically a signal group which contains only some elements of the original. This operation represents the act of choosing the takes corresponding to a single song, or the best takes of a song from a signal group holding all signals produced by a recording session.

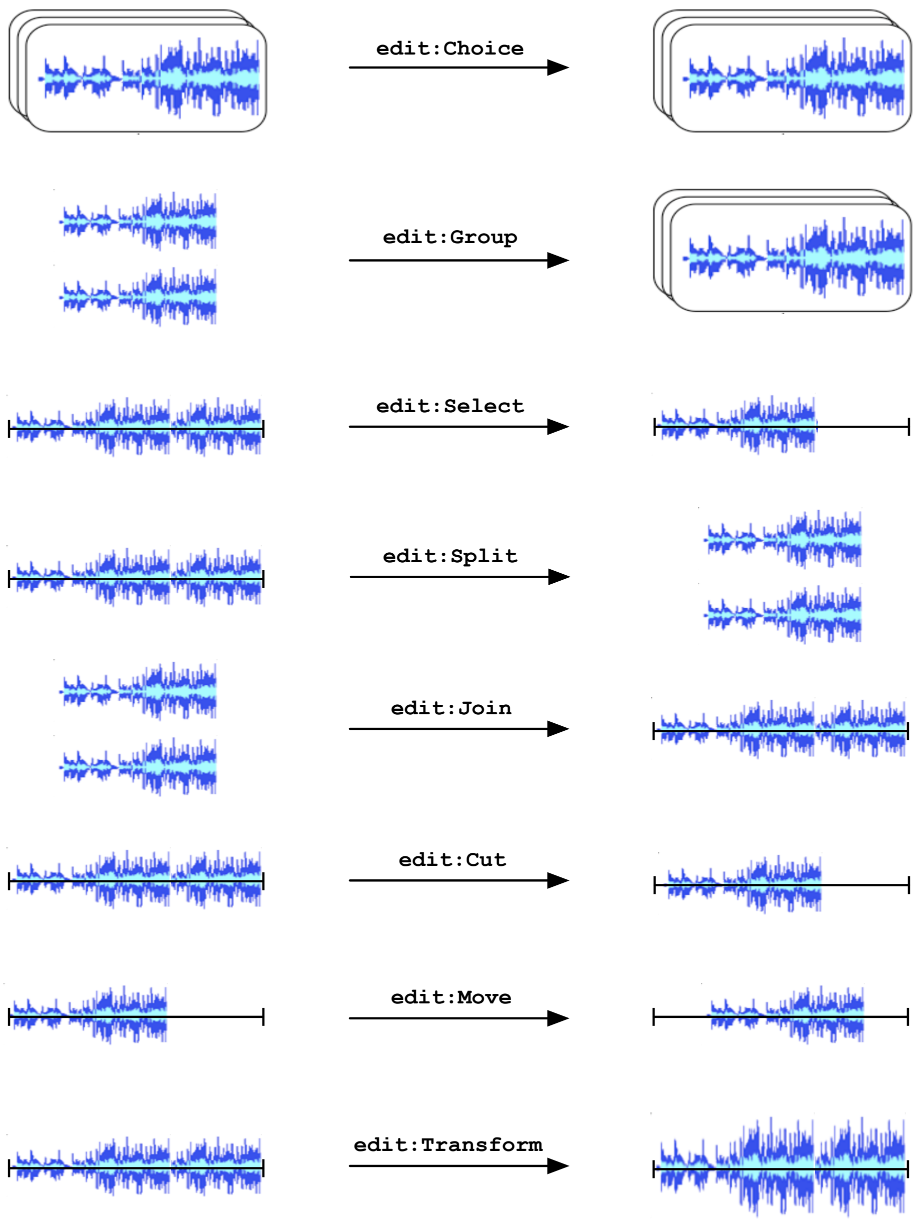


Figure 4.15: Some basic audio editing operations

Edit decisions are modelled as events linked to the universal timeline using `event:time` and the audio signal timeline using `studio:media_time`. Our ontology should support both sample editors which directly operate on audio signals, and multitrack workstations which use more complex abstractions such as the previously discussed clip and track composition. Therefore, edit operations are defined such that they can describe lower level signal manipulation as well as operation involving clips. Similarly to transformations, edit actions produce new entities in our framework, therefore the ontology can be used to capture full provenance related to the audio editing workflow encoding a series of modifications in a single RDF graph.

To support the need for providing application specific specialisation or links to implementation classes when used in a specific audio editor, is the reason for the modularisation of this ontology, as opposed to defining relevant terms in the core Studio Ontology. For each particular audio editor, we may provide a suitable Edit Ontology module which holds specific terms that subsume the basic operations defined in this ontology.

4.2.6 Summary

The Studio Ontology Framework provides a novel conceptualisation of the recording studio environment. Although it contains some knowledge representation elements that may be found elsewhere, for instance, in metadata standards or UML models of audio editors, its implementation as a Semantic Web ontology, and the fact that it provides an explicit knowledge representation without bounding it to specific software makes it unique and the first of its kind.

The ability to provide machine-processable representations of the information one may find on web pages of recording studios is a contribution to the Semantic Web in itself. Our framework facilitates finding studios with specific equipment or personnel using complex queries. However, a more significant benefit comes with the ability to denote how a piece of music was produced. We can argue that contributions form the producer or the sound engineer are just as important in modern music as composition, but we had no way to record his/her actions and choices with the transparency music is denoted using scores.

Collecting these data in production is a significant effort, however a lot can be done automatically if we can make ontology based models are available in digital mixing consoles, and post production tools. The Meta Object Facility Specification[7] enables source code generation from conceptual models. To take the continuously evolving nature of ontologies into account, we provide an alternative using run-time model generation. This is described in Section 5.1.

[7]http://www.omg.org/mof/

4.3 Audio Features Ontology

Representing content based features of audio is an important requirement both in a semantic audio applications, as well as knowledge based environment for Music Information Retrieval. The Audio Features Ontology published in [Raimond, 2007] is a collaborative effort designed to fulfil representation requirements in research projects, including OMRAS2 [Fazekas et al., 2010], as well as the requirements for distributed music information systems [Raimond, 2008], and online music analysis applications, for instance, the one we discuss in Section 5.3.

The Audio Features Ontology can be used to express both acoustical and musicological features. It allows publishing content-derived data about audio recordings and provides concepts such as `af:Note`, `af:Segment`, `af:Beat`, or `af:KeyChange` on top of the Event Ontology. It can therefore be used to classify temporal regions of audio signals. Figure 4.16 shows the basic structure of the ontology.

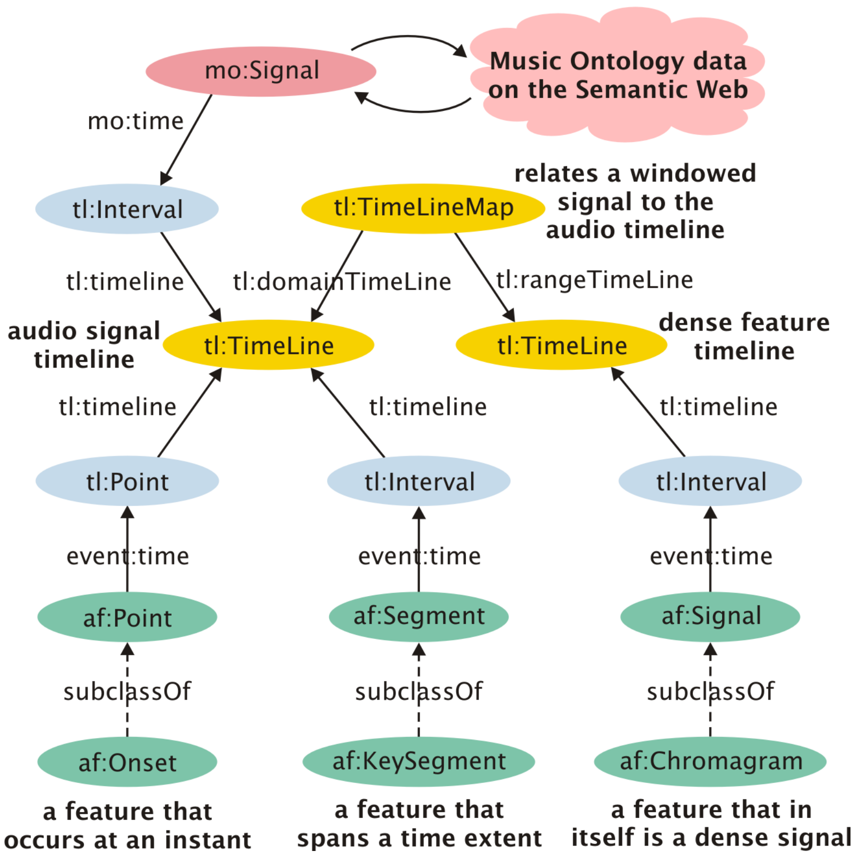


Figure 4.16: Audio Features Ontology

The main scope of the ontology is to provide a framework for communication, feature representation, and describe the association of features and audio signals. Therefore it is free from deep taxonomical organisation, and does not attempt to describe the interrelationships or computation of audio features. With regards to the different conceptualisations of feature representations presented in Table 3.2 (see Section 3.2.5.1), the Audio Features Ontology deals with *data density*, and *temporal characteristics*. It differentiates between dense signal-like features of various dimensionality, for instance chromagrams and detection functions, and sparse features that are scattered across the signal timeline, for instance notes or note onsets. The first group of features are commonly represented by the term `af:Signal`. Sparse features may be segments (`af:Segment`) or points (`af:Points`) linked to time intervals or time instants.

```
<http://isophonics.net/sawa/audiofile/temp/AU775621fe> a mo:AudioFile ;
    dc:title """music-test.wav""" ;
    mo:encodes :signal_1.

:signal_1 a mo:Signal ;
    mo:time [
        a tl:Interval ;
        tl:onTimeLine :signal_timeline_1
    ] .

:signal_timeline_1 a tl:Timeline .

:event_2 a <http://purl.org/ontology/af/StructuralSegment> ;
    event:time [
        a tl:Interval ;
        tl:onTimeLine :signal_timeline_1 ;
        tl:at "PT19.600000000S"^^xsd:duration ;
        tl:duration "PT10.500000000S"^^xsd:duration ;
    ] ;
    af:feature "9" .
```

Listing 4.10: Segmentation data expressed using the Audio Features Ontology

Listings 4.10 shows an example of describing a structural segment within an audio file, extracted using the algorithm detailed in [Levy and Sandler, 2006b]. First, we identify an audio file then describe a timeline instance. This timeline is used to link the segment feature description with the time extent of a signal entity representing the audio file.

The Audio Features Ontology currently does not cover all of the commonly used audio features. However, if we need to publish features that have no predefined term in this ontology, we can synthesise a new class within an RDF document as a subclass of an appropriate class in the Event ontology. This ensures that the features can be interpreted as time-based events, even where further semantic associations are unavailable. The present version of the ontology was created to fulfil some case specific design goals, and therefore its domain boundaries are fuzzy, while its vocabulary is incomplete with regards to user needs within the audio research communities. Further work will be required to provide a better coverage of features commonly used by researchers, as well as to enable better generalisation of its model, and harmonisation with existing research tools supporting a wider set of use cases and existing research data sets. This work should increase community involvement and should be completed with a view of extending the vocabulary to cover most state of the art feature extraction techniques.

4.4 Audio Plugin Ontologies

Plugin interfaces provide a standard way to extend an application with additional functionality without the need for changing the application. In this context, the program extended by a plugin is called the *host application*. The host works independently of plugins and provides services — typically through an application programming interface (API) — that plugins can use. The two main services a host API has to provide are *i)* a way for plugins to register their functionality within the host application and present this functionality to the user, and *ii)* a standard way to exchange information with the host.

The use of plugin architectures is a widely adopted way of creating extensible software applications. From a software engineering point of view, this fulfils most requirements of semantic audio applications as well. In order to effectively interact with semantic audio however, we need to be able to manage audio analysis components within an application together with the configuration parameters and the resulting data. Since different applications, use cases and signal types require different algorithms and configurations, a modular plugin architecture is highly desirable. An problem exists however with APIs that are typically language specific, while the semantics of communication between a host and the plugin is bound to specific implementations. These problems make it difficult for instance to collect provenance information, that is, track the use of a plugin, or associate an application context (e.g. plugin parameters) with the produced output. Here we briefly review two plugin APIs which use RDF to address some of these issues.

The LV2 plugin standard [Wilms et al., 2007] is an incarnation of the Linux Audio Developers Simple Plugin Interface (LADSPA) format, which uses RDF as a crucial part of its API. It comes with a simple OWL ontology which enables describing plugins and plugin ports, and contains a vocabulary of plugin types. The RDF data expressed using this ontology may contain information such as the index and name the ports a plugin may use. The purpose of the LV2 plugin ontology essentially is to move the problem of input and output descriptions outside of the C++ API specification.

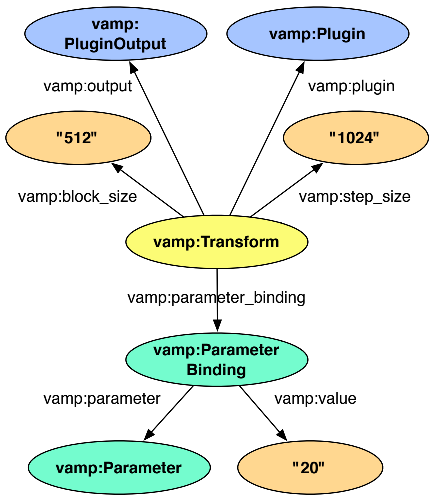


Figure 4.17: Vamp Transform Ontology

For the ontological representation of audio feature extraction algorithms, the Vamp plugin ontology[8] corresponding to the Vamp plugin API [Cannam, 2009] provides an example. This ontology features similar capabilities in describing plugins as the afore mentioned LV2 ontology, however Vamp hosts can also query the plugin binary directly for the information necessary to run the plugin, therefore host environments can run Vamp plugins with or without RDF descriptions. A related transform ontology was developed for the automatic configuration of feature extractors available as Vamp plugins. The basic entities defined in this ontology are shown in Figure 4.17.

The term `vamp:Transform` is used to conceptualise a feature extractor algorithm together with a set of parameters expressed as generic parameter bindings. This model allows for the representation of zero or more general purpose parameters, with the `vamp:step_size` and `vamp:block_size` parameters — common across short-time windowed feature extraction algorithms — liked to the transform using mandatory predicates. Listing 4.11 shows an example of using the Vamp Transform Ontology to describe the segmentation algorithm whose output is shown in Listing 4.10 of the previous section. We make use of this ontology for communication between a Web-based audio analysis tool and its computation engine as described in Section 5.3. While this is already a good starting point, further refinement is necessary to describe complete workflows within music analysis algorithms, in order to allow

[8]vamp-plugins.org/ontology/vamp

inference over contextual data and use the most suitable components for a specific analysis case.

```
:transform a vamp:Transform ;
   vamp:plugin <http://vamp-plugins.org/rdf/plugins/qm-vamp-plugins#qm-
       segmenter> ;
   vamp:step_size "8820"^^xsd:int ;
   vamp:block_size "26460"^^xsd:int ;

vamp:parameter_binding [
   vamp:parameter [ vamp:identifier "featureType" ] ;
   vamp:value "1"^^xsd:float ] ;

vamp:parameter_binding [
   vamp:parameter [ vamp:identifier "neighbourhoodLimit" ] ;
   vamp:value "4"^^xsd:float ] ;

vamp:parameter_binding [
   vamp:parameter [ vamp:identifier "nSegmentTypes" ] ;
   vamp:value "10"^^xsd:float ] ;

vamp:output
   <http://vamp-plugins.org/rdf/plugins/qm-vamp-plugins#qm-
       segmenter_output_segmentation> .
```

Listing 4.11: Description of algorithm parameters using the Vamp Transform Ontology

4.5 Instrument Ontology

An important use cases for developing ontologies describing musical instruments from a recording point of view was already outlined in Section 4.2.5.1. In this section we describe our ongoing collaborative work in musical instrument ontology design.

4.5.1 Motivation

A particularly interesting application of an instrument ontology is in describing audio engineering workflows, in particular, describing microphone techniques. Currently, there is no standard to describe the placement of a microphone relative to an instrument, even though it has a profound effect on the recorded sound due to the diverse radiation patterns of musical instruments, and the fact that the frequency characteristics of the recorded sound is dependent of this relation.

The Music Ontology provides a way for spatial localisation of recording events including the position of microphones. However, the use of geographical coordinates for this purpose is insufficient in audio engineering applications. We introduced more precise methods to describe the relation of a microphone (or a microphone arrangement) to the instrument in Section 4.2.5.1. This is based on describing the distance and angles between the microphone pickup and its main axis, and the sound source. However, we found that in many cases, identifying reference points for this relation requires the description of the instrument in question. Other important use cases include instrument identification in a knowledge based environment, and semantic labelling of audio recordings using shared instrument resources, as well as the ability to identify instrument families. Instrument classification is not an easy task however. Several classification systems have been proposed by musicologists and etno-musicologists, but no universally accepted system has emerged.

4.5.2 Instrument classification systems

The Music Ontology relies on the instrument taxonomy published by Herman[9] and defines only the top level concept `mo:Instruemnt`. This taxonomy is based on the MusicBrainz instrument tree, and uses the Simple Knowledge Organisation Systems (SKOS)[10], a model for expressing controlled vocabularies, thesauri, and taxonomies in RDF.

SKOS defines `skos:Concept`, whose individuals may be associated with one or more lexical labels, using `skos:prefLabel, skos:altLabel`, and placed within a hierarchy using `skos:broader`, `skos:narrower`, or `skos:related` properties. While this is well suited for hierarchical classification schemes, it provides limited support for other types of relationships; `skos:related` for example, may be used to describe associative relations, but only in a semi-formal way, without a more explicit definition of the semantics of this relation. Moreover, the transitivity of broader and narrower relations are not guaranteed, therefore it is difficult to infer, for instance, the instrument family of a given instrument without additional knowledge not expressed in the model. This taxonomy is therefore sufficient for applications requiring only a semantic label to represent instruments associated with audio items, it cannot represent the heterogeneity of instrument relations, or express detailed information, such as shape, size and the various parts of instruments.

The most widely used instrument classification scheme in the museum community as well as by musicologists is the Hornobostel and Sachs [von Hornbostel and Sachs, 1914] system. This exhibits a downward taxonomy by logical division. Many attempts have been made to

[9] http://purl.org/ontology/mo/mit
[10] http://www.w3.org/TR/skos-reference

improve this [Elschek, 1969], [Lysloff and Matson, 1985], but very few of these departed from taxonomical organisation.

4.5.3 Instrument Ontology

Although taxonomies allow us to organise data in a hierarchical structure very efficiently, they encode a strict relationship between a parent node and a child node without defining the detailed relationships among instruments or their parts. Musical instruments however have a multi-relational model, an instrument may belong to more than one family or sub-family. Our recent collaborative research [Kolozali et al., 2011, 2010] showed that ontologies based purely on the above mentioned system are insufficient for rich knowledge representation of instruments. They cannot support complex query answering, or encode the instrument characteristics required for the use cases mentioned in Section 4.5.1 and 4.2.5.1.

Our current work includes the design of a Musical Instrument Ontology, which addresses the issues discussed above. We consider two alternative designs. The first utilises the most agreed upon classification scheme as a starting point, and expresses multi-relational instrument family memberships which can be encoded in OWL. This can be extended with a vocabulary of instrument parts, and linked with the hierarchical organisation to describe individual instruments using our device decomposition model discussed in Section 4.2.3.3, A foundational ontology (see Section 3.3.2) may also be extended for this purpose. The other design can be based on a flat vocabulary of instruments and parts as a starting point, and develop methods to transform this into specific domain ontologies using additional data supplied to a reasoning engine. The benefit of this is that several applications which require alternative instrument classification can rely on the same shared vocabulary of instrument terms.

4.6 Temperament Ontology

Knowledge about musical temperament — that is, the tuning of an instrument using a particular tuning system — is very important in recording pieces composed before the predominant use of equal temperament staring from the 18th century. This tuning system exhibits a uniform frequency ratio between the fundamental frequencies of adjacent notes. With historical temperaments, each musical key has its own character, therefore it is useful to capture information about temperament. It is useful to provide a semantic label to identify temperament, as well as detailed information about the specifics of tuning used for an instrument in a recording. We developed a Temperament Ontology for this purpose as an extension of the Music and the Studio Ontology frameworks, and utilised it in the Web-based temperament recognition system discussed in Section 5.4.3.

4.6.1 Instrument tuning systems

Tuning an instrument consists of choosing the frequency values and spacing or ratio of pitches that are used. Pure (just) intervals of pitches correspond to whole number ratios of their frequencies, however these ratios are not compatible with each other as they arranged in scales (the way octaves are divided into discrete pitch classes) in Western music. It is not possible to fit for instance twelve pure fifths into seven octaves, i.e. $(\frac{3}{2})^{12} \neq 2^7$. The difference is called the Pythagorean or Ditonic comma equivalent to 23.5 cents. This difference has to be tempered out — that is, some or all fifths has to be mistuned slightly in order to fit them. There are many tuning systems. Most commonly, they differ in the way they compromise pure intervals to solve this problem.

4.6.2 An open-ended temperament description model

There is no mutual agreement in the literature on the description or classification of temperaments. Therefore, in this ontology we do not impose a hierarchy between types of temperaments.

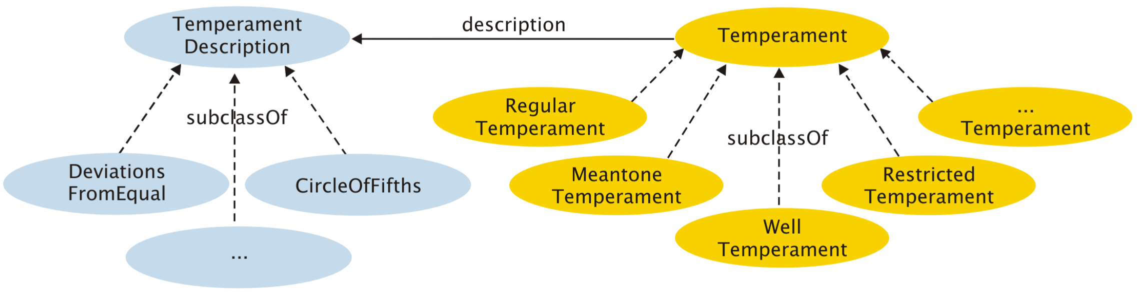


Figure 4.18: Overview of the Temperament Ontology

We define an opaque top-level temperament concept illustrated in Figure 4.18. Subclasses of this concept can be used in describing individual temperaments, if necessary, using multiple class memberships. Since there is more than one way to associate tuning systems with their properties, we treat temperament descriptions as concepts as well, and use reification to keep the model open and extensible.

4.6.3 Temperament descriptions

Temperaments can be characterised in several different ways. The most common methods are using either the circle of fifths or give the pitch deviations from equal temperament. We define these descriptions as concepts in the ontology, however, other descriptions may be used and defined in the future. For example, one might find it convenient to express the same information using the circle of fourths.

4.6.3.1 Deviations from Equal Temperament

In equal temperament, an octave is divided into twelve equal intervals. As a result, only octaves are pure. All other intervals are impure, and the deviation from pure is different in case of each interval. Since equal temperament has become very common, other temperaments are often described by the frequency deviations, in cents, of each pitch class from the corresponding pitch class in equal temperament.

```
@prefix : </> .
@prefix pc: <http://purl.org/ontology/temperament/pitchclass/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix tm: <http://purl.org/ontology/temperament/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

:temperament_0 a tm:Temperament ,
        tm:Valotti ;
    tm:description :description_00 ,
        :description_01 .

:description_00 a tm:DeviationsFromEqual ;
    rdfs:label "Deviations from equal temperament in cents." ;
    tm:deviation_from_equal
        [ a tm:PitchClassDeviation ;
           tm:pitch_class pc:G ;
           tm:value "3.90984723"^^xsd:float ] ,
        [ a tm:PitchClassDeviation ;
           tm:pitch_class pc:Gs ;
           tm:value "1.95488264"^^xsd:float ] .
           # ... up to 12 pitch classes
```

Listing 4.12: Temperament Ontology example using deviations from equal temperament

4.6.3.2 The Circle of Fifths

The circle of fifths has several uses in music theory. It shows the harmonic relationships of the twelve major and minor keys, and it can be seen as a circle of the corresponding pitch class intervals, such as (C-G), (G-D), (D-A) and so on. If we go around the circle using pure fifth intervals, it wouldn't close. What remains is the Pythagorean comma. Hence, it is often used to describe temperaments by showing how the comma is distributed among the intervals to close the circle. Note that there are several other types of commas related to different tuning problems. These are defined in the ontology.

```
:description_01 a tm:CircleOfFifths ;
    rdfs:label "Deviations from pure fifth given by a fraction of
        Pythagorean comma." ;
    tm:interval
        [ a tm:FifthInterval ;
           tm:deviation [ a tm:IntervalDeviation ;
                    rdfs:label "-1/6" ;
                    tm:comma tm:PythagoreanComma ;
                    tm:value "-0.166376139378"^^xsd:float ] ;
           tm:lower pc:D ;
           tm:upper pc:A ] ,
        [ a tm:FifthInterval ;
           tm:deviation [ a tm:IntervalDeviation ;
                    rdfs:label "0" ;
                    tm:comma tm:PythagoreanComma ;
                    tm:value "0.0"^^xsd:float ] ;
           tm:lower pc:B ;
           tm:upper pc:Fs ] .
           # ... up to 12 fifths
```

Listing 4.13: Temperament Ontology example using the Circle of Fifths

4.6.4 Using the Temperament Ontology

This ontology was designed to describe the results of automatic temperament classification [Tidhar et al., 2010a], as well as standard temperament profiles. The RDF data expressed using this ontology may contain the information shown in listings 4.12 and 4.13

The examples describe the temperament of a real-world recording classified as Valotti. Here, we provide two types of temperament descriptions. The first, `:description_00` uses pitch class deviations from equal temperament to specify how the extracted temperament is related to equal temperament. The values for each pitch class (only two are shown for brevity) are given in cents. The second description type, `:description_01` is inferred from the first. It characterises the same temperament using the Circle of Fifths such that the deviations from pure intervals are given for 12 fifths on the circle in fractions of Pythagorean comma. The human readable `rdfs:label` is computed using a small tolerance around the actual deviation derived from measured data.

4.7 Summary

In this chapter we first reviewed ontology design principles, and how they may be applied in developing ontologies for audio applications. We discussed the state of the art in semantic audio information management, and outlined how ontologies may be used to enhance present the state of the art.

As main contribution, we introduced novel ontologies for this purpose. Notably, this includes the Studio Ontology Framework (Section 4.2), which extends the Music Ontology to capture detailed information about music production in the studio, and the Temperament Ontology (Section 4.6), which can be used to identify tuning systems, and describe specific instrument tuning characteristics. The Studio Ontology framework covers the following domains:

- **Technological Artefacts**: see the Device and Signal Processing Device Ontologies (Section 4.2.3.3, and Section 4.2.3.8).
- **Modelling change** of variable attributes of devices (Section 4.2.3.4)
- **Describing recording studios** on the Semantic Web (Section 4.2.4.1)
- **Modelling the audio engineering workflow** by describing recording session types and audio transformations (Section 4.2.4.2).
- **Signal processing workflow provenance**: Provide detailed description of events and device connections in audio signal processing (Section 4.2.4.3).
- **A model of describing the life cycle of complex audio effects**: A model that parallels FRBR in describing signal processing devices, in particular, audio effects (Section 4.2.5.3).
- **Recording techniques**: Ontologies for microphones and microphone techniques (Section 4.2.5.1), audio mixing (Section 4.2.5.2) and audio editing (Section 4.2.5.4).

We also reviewed our current collaborations in developing ontologies for musical instruments and multidisciplinary classification of audio effects. These ontologies will extend the two main ontology frameworks discussed in this chapter, the Music Ontology and the Studio Ontology.

While ontology development on one hand serves the purpose of shared knowledge representation across several domains or communities, as well as facilitate data communication between computational agent and tools using the Semantic Web in itself, on the other hand, our use cases in semantic audio information management require novel software architectures to reach the full potential of the developed ontologies. In the next chapter we outline how these ontologies are utilised in audio production tools and Web based music information environments.

Chapter 5

Software Tools and Semantic Audio Applications

The ontologies described in the previous chapter provide the means for communicating information about many important aspects of music in the context of semantic audio tools. However, the use of ontologies in music software presents difficulties. Ontologies are structurally complex, therefore generating and interpreting data adhering to an ontology is far from trivial. This problem can be mitigated by creating software libraries and high-level tools that hide complexities from the end user, who is, in our case, a developer or a researcher using an ontology-based system.

Most existing tools for this purpose are either too low-level, e.g. require the manipulation of individual RDF statements, or not particularly well suited for audio related use cases. These problems are discussed in more detail and addressed in this chapter. First, a software library for creating ontology-aware adaptive data structures will be introduced with applications in audio, then a Web-based demonstrator framework will be described that dynamically computes features of uploaded audio files and stores the results in RDF. Finally, we outline some applications of this framework, including an acoustic similarity-based recommender, and a service for musical temperament estimation.

5.1 RDF data binding with Meta-Object Protocol

Relying on a flexible, ontology-based information management and knowledge representation framework for audio analysis and intelligent, semantic audio tools turned out to be an important requirement in our research. Our primary motivation is in the use of modular ontology schema instead of existing disharmonious metadata formats. In order to advocate uses of metadata in audio processing, we developed a unified information framework and data collection tool. This tool can be easily integrated in existing audio production software.

5.1.1 Design issues of ontology-based information systems

Although ontologies provide modularity in specifying metadata schema and flexible knowledge management, often, static software implementations limit the extensibility of a system. Among the most common problems we find is the use of external database software accessed through hard-coded query templates. This limits the ability to adapt to changes in metadata schema, besides, accessing the database involves expensive query processing. It is not uncommon that an object-based or even RDF-based information system is used together with a relational database back-end. This incurs complicated mapping to relational schema which would otherwise be unnecessary. The RDF-MOP library described in this section addresses this issue.

Our system avoids the above problems by relying on an efficient local hash database implementation providing a native RDF store and low-level manipulation of statements in an RDF graph. In order to abstract these low-level calls, we develop an automatic mapping mechanism between application objects and RDF statements. This system is able to persist application data stored in an existing object hierarchy, together with semantic associations obtained from ontology definitions. The mechanism also provides atomic transaction management for groups of RDF statements associated with data from a single application object.

Since our framework is to be used as an efficient semantic metadata store, we avoid the overhead of network communication between the database and the host application. Therefore, we implement this system as a shared dynamically loaded library, compiled together with a database implementation.

Finally, the novelty of the system is the ability to extend an application with dynamically generated storage containers representing metadata terms defined using the Semantic Web ontologies described in the previous chapters. This is achieved using a meta-object protocol and an associated type system outlined in Sections 5.1.3 and 5.1.4.

5.1.1.1 Requirements

The most important requirements of the system can be summarised as follows:

- *Extension:* The system is be able to extend the application with metadata storage dynamically.
- *Mapping:* The system is able to translate and store application data represented in an existing object hierarchy.
- *Consistency:* The system maintains metadata and database consistency during interaction with a user.
- *Integration:* The system can be appended to an existing audio application with the least possible interference with original code.

5.1.1.2 Dependencies and configuration

Our aim is to build a reference implementation for existing audio applications written in C++. This includes the open-source audio editor Audacity[1]. This confines our choice of RDF tools to those with C/C++ support. As basis for our triple store implementation, we use the Redland RDF libraries [Beckett, 2008]. This library permits in-memory or persistent storage. We configure the library to use an efficient hash database. For this purpose, we use BerkleyDB[2], a high-performance open-source embedded database solution distributed by Oracle. Our current implementation makes use of the cross-platform framework wxWidgets[3] for greater compatibility with Audacity. Ideally, the system should rely on the Standard Template Library (STL), however, wxWidgets provides compatible classes which makes such transition relatively easy.

5.1.2 Data binding

If we wish to use RDF to capture information resulting from audio analysis or user interaction in semantic audio tools, we need to be able to match the information sources within the tool to the RDF data model and/or a corresponding data store. This corresponds to the process commonly termed as *data binding* in software engineering. It is a technique that binds two data sources together and maintains synchronisation.

Finding an efficient way of accessing from or updating information to a triple store within a typical audio application written in an object orientated language is a primary challenge. Using a local database which is dynamically loaded into the application, this can be achieved in two ways: using in-process API calls, or using SPARQL. Because of the computational expense associated with query processing, we base our implementation on low-level API calls. However, besides the burden of manipulating the RDF store at a fairly low level, an important problem arises form conflicting data models: the class hierarchy of a typical object-orientated application, and the RDF graph. As a solution to this problem, we use a software engineering technique called metadata mapping. This is similar to object-relational mapping used in the context of relational databases. We provide a mechanism which allows persisting application objects in an RDF database automatically. However, this requires the data in the application to be associated with ontological semantics (see Section 2.4.3). A further requirement is the ability to dynamically instantiate objects representing arbitrary metadata terms defined in an ontology, and accommodate changes in the ontology without significant reengineering, or the need for recompiling the application. In our system, a run-time Meta-object Protocol (MOP) [Kiczales et al., 1991] provides the basis for the solution to these problems.

[1]http://www.audacityteam.org/
[2]http://www.oracle.com/technology/products/berkeley-db/index.html
[3]http://www.wxwidgets.org

5.1.3 Meta-object Protocol

A Meta-object protocol (MOP), in simple terms, provide a way to interact between a set of functional objects, which are at the base-level in typical object orientated systems, and a set of meta-objects which describe base-level objects and their behaviour at the meta-level.

Meta-object protocols can be used to implement reflective systems which can reason about and act upon themselves. Such systems are composed of a base-level and a meta-level, whereby objects at the meta-level have access to representations of the base-level being reasoned about. The meta-level is causally connected to the base-level, such that changes at the meta-level cause changes to the behaviour of the base-level [Welch and Stroud, 2001]. From our perspective, the most important property of a meta-object system is the ability to associate objects with descriptions, that is, the ability to move from simple data within a software system towards the representation of information and knowledge (see Section 2.1).

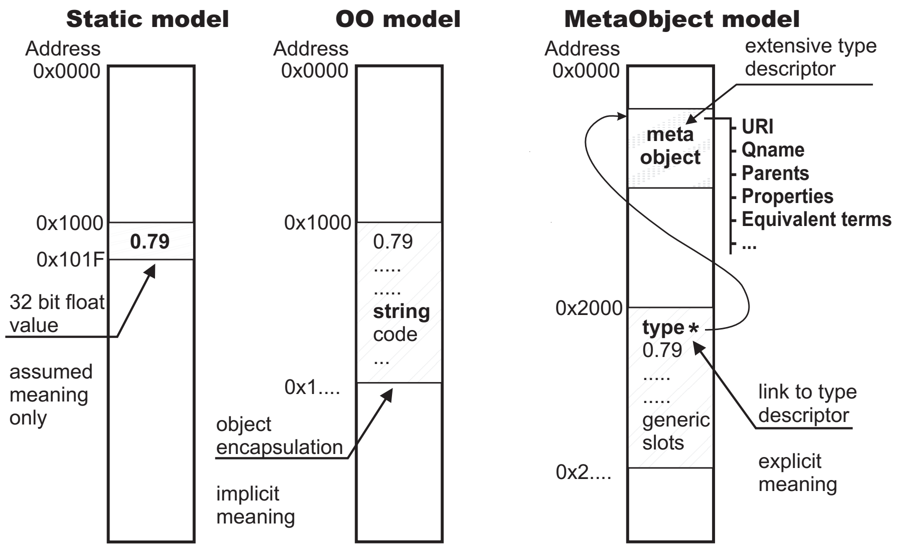


Figure 5.1: Memory Model for Meta-object Protocol

Meta-object protocols were originally developed for the Common Lisp Object System (CLOS) [Levine, 2003]. They can also be found in the context of more recent dynamic languages interpreters such as that of Python's[4], and have been utilised in numerous other applications. For instance in the Python language interpreter, objects representing data elements are always associated with a *type object* describing the base object and controlling its behaviour. Other examples include using a MOP for facilitating atomicity in database transactions [Stroud and Wu, 1995] to support maintaining data consistency in unreliable, concurrent, or multiuser environments, providing language extensions for C++ [Chiba, 1995], developing distributed object systems [Lee et al., 1999], implementing behavioural reflection

[4]http://www.python.org/

in byte code interpreters [Welch and Stroud, 2001], and finally the development of fault-tolerant systems [Taiani et al., 2005]. Using meta-objects allows for extensible association of data with semantics within an application, therefore, the application is able to inspect the relations, associations or state of its objects, and dynamically change their meaning and behaviour. For our particular use case however, we do not need to implement the full protocol required for instance for a dynamic interpreter. Yet, its use is beneficial for blending functional and logic programming paradigms when managing metadata in an efficient but static programming environment.

From an implementation point of view, there are two main types of meta-object protocols, *run-time* MOPs, and *compile-time* MOPs. Run-time protocols use meta-objects, such as extensive type descriptors in the run-time environment, while compile-time MOPs provide control over the compilation of a program, therefore the meta-level remains static.

In our library, we implement a run-time meta-object protocol to facilitate metadata mapping between application data and RDF data. In this scheme, each object representing some arbitrary data is linked with a meta-object, which associates the object with an URI corresponding to an ontological definition. Some inferred characteristics, such as the object's place in the taxonomy part of the ontology, its permitted relationships, or links to equivalent terms are also included. This is illustrated in Figure 5.1 with the comparison of simplified memory models of various strategies for storing application data.

Two specific types of meta-objects are used in our system to represent RDF classes and properties. These meta-objects are statically designed to represent information defined by ontology schema, however, they are dynamically instantiated by a generator via an inference mechanism when loading schema documents into memory. This is achieved using the following protocol: Ontology schema are parsed into a model using a suitable Redland syntax parser. Obeying RDF and OWL language rules, we build meta-objects for each class and property defined in the ontologies in question. First, we enumerate class and property declarations in the conjunctive model and instantiate a skeleton object for each. Next, we separately infer the inheritance hierarchy within the disjoint hierarchies of ontological terms and relationships. This information is appropriately used to model the same hierarchy within the set of previously created meta-objects. This is followed by assessing equivalence relationships and update the object model accordingly. Finally, the assignment of properties to classes in the model can be made. The meta-objects resulting from this process are stored in hash maps with keys corresponding to the resource URIs used for identifying them. These objects are available in the application and can be used to link data with semantics, and to create metadata containers according to their descriptions. In practice, this is achieved by constructing objects of a specialised type system we describe in Section 5.1.4.

5.1.4 Type system

For the purpose of instantiating metadata containers as needed in the editing workflow, we develop a type system associated with the meta-object protocol and the basic node types appearing in the RDF model. Elements of this system can be dynamically created and used to store metadata in a generic way.

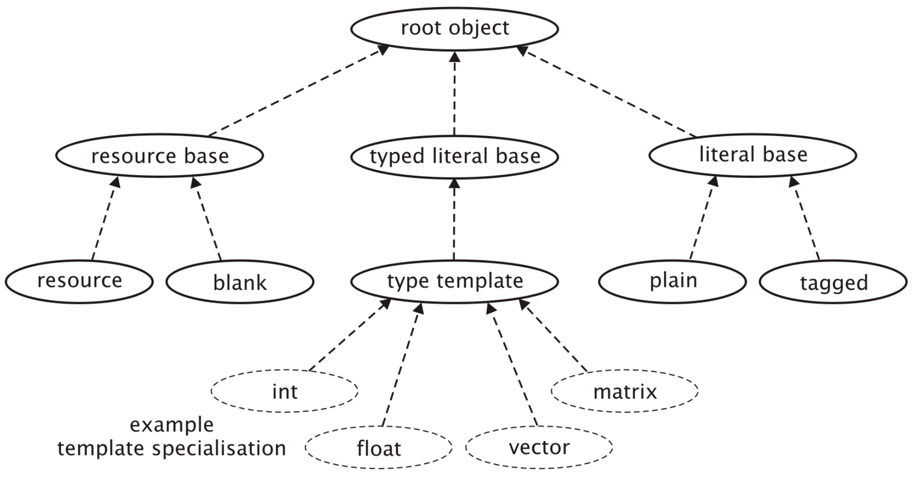


Figure 5.2: Type System

The system addresses the various data representation needs in our software. Generic resource types are assigned a corresponding meta-object, linking the resource to its ontological class definition. These objects contain a map in order to model the properties associated with the class. Literals are modelled after the types permitted in RDF and XML Schema. We can represent both plain string literals, and string literals coupled with a language tag this way. Numerical types however require a more complex representation. Our solution is based on C++ template specialisation. We map permitted XSD types (which are used for identifying data types in RDF) to corresponding simple or complex C++ data types wrapped into generic container templates. For instance, vectors and matrix classes can be mapped as plain strings or suitable XML literals in the RDF representation. As an example, a C++ meta-program for representing typed RDF literals can be found in Section C.2 of the appendix.

The classes of this subsystem can be configured in three different ways. They can act as references simply associating semantics with data stored elsewhere. This is similar to the implementation of logic references in the Castor logic programming library [Naik, 2006]. The objects can be added to existing data structures, wrapping existing functionality. Finally, they can be used independently, within a separate hierarchy, for fulfilling more complex metadata management needs. For example, this shall be used for storing the wide range of audio features associated with a track. In all modes of operation, a set of overloaded constructors are used for creating appropriately configured objects depending on their use.

5.1.5 Architecture

The architecture of the library consists of several classes with a complex interaction model. A simplified diagram showing the main building blocks can be seen in Figure 5.3.

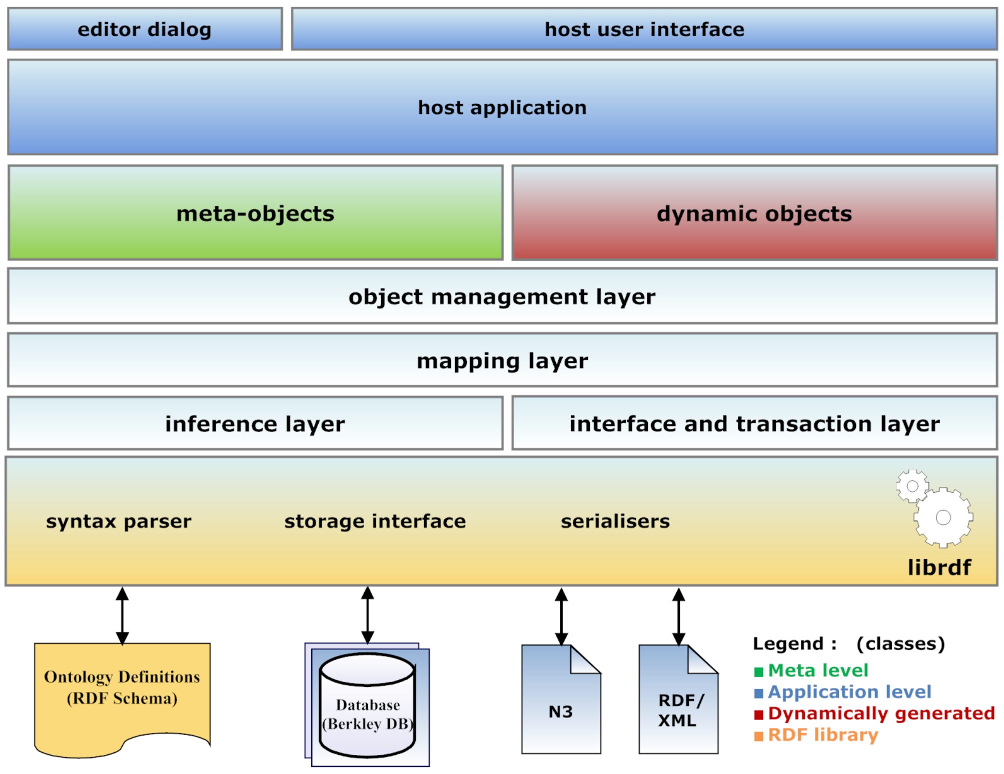


Figure 5.3: Architecture of the RDF-MOP library

The *inference layer* is used to extract the class and property hierarchy of ontology definitions to build meta-objects. Logic programming functionalities can be added at this level in the future.

The *interface and transaction layer* is responsible for persisting the data stored in the object system, wrapping low-level graph manipulation calls and coordinating the addition of triples that must be stored atomically.

The *mapping layer* consists of a name space manager, a meta-object map, an object registry and a type mapper class. It is mainly responsible for linking meta-objects and dynamic-objects used in the system. The *TypeMapper* class maps XSD data type URI's used for identifying RDF typed literals onto function objects. These function objects (or functors) are used for creating dynamic objects, appropriate instances of dynamic class templates as described in the previous section.

The *object management layer* consists of the meta object and dynamic object managers.

Their primary function is storing and maintaining the objects created during the interaction with the model. Some additional functionality, common across dynamic objects, is implemented here using external polymorphism.

Meta-objects contain information corresponding to RDF and OWL classes and properties. For example, a *classInfo* object holds the inheritance hierarchy of a particular class. These instances are dynamically generated when parsing ontology schema.

Dynamic objects represent terms and relationships as objects in the application, string literals, simple XSD types as well as more complex numerical types. The previously described type system is implemented here.

The software library and core architecture described in this section provides the basis for a *semantic audio desktop*, which builds on and extends the idea of using RDF and Semantic Web technologies for personal information management in typical desktop environments.

5.2 The Semantic Audio Desktop

Among the first steps towards a complex and interactive environment — where human and machine intelligence are working together — is a suitable information management system. The system proposed here consists of two main parts, one of which is a knowledge representation model. To this end, recognising the parallelism between the unbounded nature of musical information needs in our application, and the general information needs on the Web, we borrow the data and knowledge representation model of the Semantic Web. A software framework constitutes the second part of the system. This framework implements the afore-mentioned data model, provides connectivity, representational mapping, and grounds for interaction, for example, the ability of tracking user actions. In the rest of this section, we outline some applications using these principles, as well as applications of the software components and ontologies in music production.

The proposed framework is conceptually related to the *Semantic Desktop* paradigm; the idea of an intelligent desktop environment [Decker and Frank, 2004]. By definition, the Semantic Desktop [Sauermann et al., 2005] is a device in which an individual stores digital information — that is, data which are otherwise stored on a personal computer by means of conventional techniques such as binary files, spread sheets, and so on. Its most prominent aim is the advancement of personal information management through the application of Semantic Web technologies to the desktop environment. In our work, we extend this idea to the audio production environment, and in particular, audio editors used in post production. Besides seeing the *Semantic Audio Desktop* as requisite for building intelligent audio tools, there is additional benefit from using this system. We shall be able to collect high-quality metadata during the music production process, and open up the creative environment to *social media* or other means that allow the creation and exchange of user-generated content [Kaplan and

Haenlein, 2009]. The metadata — being in the same format as other data on the Semantic Web — can be fed back into the public domain, for example as a Linked Data resource. Thus, these data can be used in advanced cataloguing, content-based music recommendation and search services, as well as music education, collaborative music making and other future Web applications.

5.2.1 Metadata management in music production

Consider the user interface shown in Figure 5.4. If we were to express this information in RDF, we have several choices: A programmer may manually pick the relevant terms from ontologies and write some static code for each individual tag that translates the values entered by the user. Alternatively, we can rely on an API or SPARQL. In the first case, we have to use API commands to compose RDF statements from terms and values, then serialise the statements or store them in a database. In the second case, we may use SPARQL update protocol[5] to store the information in a database. Yet again, we have to hard code either a SPARQL update or query for each item we wish to publish.

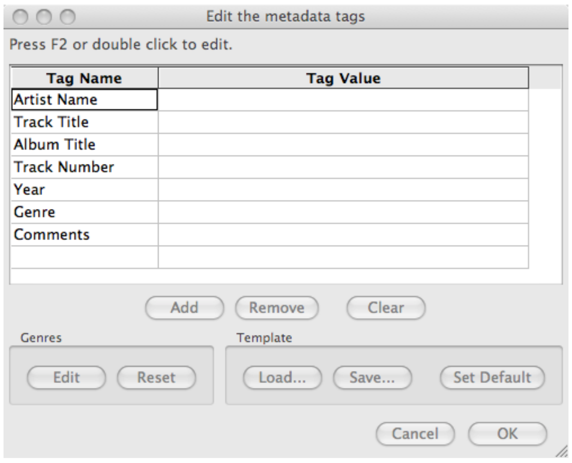
In each of the above cases we lose two of the most important advantages of using RDF: the greater expressive power and its open-ended nature. First, our expressiveness is limited *not* by the data model or the richness of the ontologies, but by the interface implementation. Second, as ontologies evolve, both programming and user interfaces become deprecated, requiring a software update. We address these issues by building both our internal data representation and the user interface directly from schema expressing ontologies. It has to be noted that we neither assume a persistent network connection, nor we rely on an external database for most proposed functionality to work. This application relies on a built in database back-end and RDF library described in Section 5.1.

5.2.2 Data collection in the studio

Developing a dynamic user interface for RDF data poses similar challenges to the mapping problem detailed in Section 5.2.1 In fact, in order to generate a user interface we use the same process to collect information about terms and relationships. This allows us to load hierarchies into dynamic *list controls* or *tree controls* provided by application programming frameworks. Such an interface is shown in Figure 5.5 with a possible view on the data. It exemplifies loading terms from the Multitrack Ontology allowing to describe a recording project in detail.

The three panels of the interface are divided between a listing of classes or *categories* available in a domain, *instances* or individuals of a selected class, and lastly, the *properties* of a selected individual as shown in the third pane. We generate the content of the *Hints and Help* box from `rdfs:comment` predicates.

[5] http://www.w3.org/TR/sparql11-update/

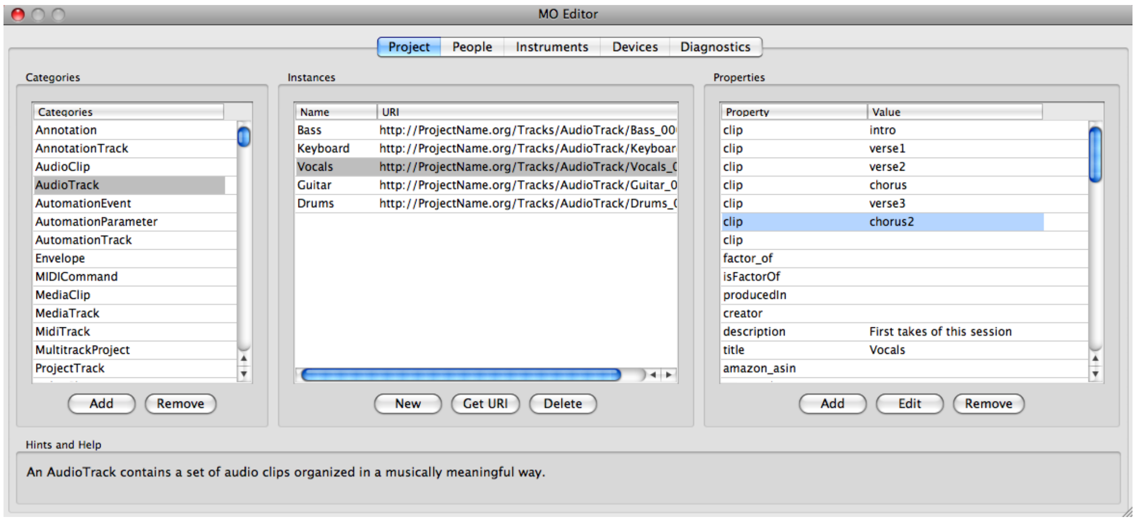


Press F2 or double click to edit.

Tag Name	Tag Value
Artist Name	
Track Title	
Album Title	
Track Number	
Year	
Genre	
Comments	

Figure 5.4: ID3 Metadata Editor Interface

If we were to add a new *track type* to the editor this interface does not have to be changed. New terms added to the ontologies will show up in the correct place. Admittedly, there are difficulties in generating a user interface for ontologies expressed in RDF. Most notably, allowing a large number of terms to be seen directly is confusing to the user. We address this issue by using a tabbed interface such that each page has a limited focus. For example, the interface of Figure 5.5, separate pages are designated for *project* details, *people* or *devices*. The selection of focus is semi-automatic since we choose top-level classes and display both derived types and the ones which can be taken as *values for some properties* declared in the same ontology. Additionally, the terms may be grouped using expandable *tree controls* to provide an even clearer user interface.



Categories
Annotation
AnnotationTrack
AudioClip
AudioTrack
AutomationEvent
AutomationParameter
AutomationTrack
Envelope
MIDICommand
MediaClip
MediaTrack
MidiTrack
MultitrackProject
ProjectTrack

Name	URI
Bass	http://ProjectName.org/Tracks/AudioTrack/Bass_00
Keyboard	http://ProjectName.org/Tracks/AudioTrack/Keyboar
Vocals	http://ProjectName.org/Tracks/AudioTrack/Vocals_(
Guitar	http://ProjectName.org/Tracks/AudioTrack/Guitar_0
Drums	http://ProjectName.org/Tracks/AudioTrack/Drums_(

Property	Value
clip	intro
clip	verse1
clip	verse2
clip	chorus
clip	verse3
clip	chorus2
clip	
factor_of	
isFactorOf	
producedIn	
creator	
description	First takes of this session
title	Vocals
amazon_asin	

An AudioTrack contains a set of audio clips organized in a musically meaningful way.

Figure 5.5: Music Ontology Interface in Audacity

A lot of information that would normally require manual data entry can also be generated during the normal working procedure. A track title for example can be obtained from a relevant widget of the audio editor interface. The clip sequence names might be obtained from labels naturally assigned to recording takes. This releases the user from entering trivial information and allows to concentrate on interesting details. User defined terms can also be added or removed using the appropriate buttons. However, this may incur dynamic ontology expansion which is problematic if such data is published online.

Instances can be added in the middle pane using the *New* button. A necessary URI is generated automatically, however if a specific URI is to be assigned, for example a FOAF link to a person, we may also use that here. A lookup mechanism that allows access to more information about a resource in this interface constitutes future work.

5.2.3 Workflow tracking

In this section, we describe a subsystem for logging user actions in an audio editor. The primary motivation behind developing such a system is the requirement for tracking the application's state. This is vital if we wish to keep metadata consistent with the audio data during an editing session. Example use cases include running a processing algorithm destructively, in which case new metadata has to be extracted, or splitting an audio track, in which case references have to be updated. A useful side effect of this system is the ability to describe the edit history in RDF. In future applications, this may also be applied for establishing a user context. In a personalisation framework, this can be used to aid the editing process itself.

Human-Computer interaction in a typical graphical environment is centred around commands corresponding to mouse clicks or key-strokes. In response to these commands, the machine performs some of its functionality which results in some changes of data values in the software. This general model applies to audio editing procedures such as moving an audio clip on a timeline, or running a processor plugin. If we choose to have metadata representing some aspects of the data, it has to be kept consistent both in memory and in a database.

We developed an Edit Ontology described in Section 4.2.5.4. This provides the necessary semantic framework for recording user events. The ontology is able to represent user actions operating on data. In order to fulfil our requirement for consistency, these changes need to be reflected in the metadata, thus reflected in the database. In practice this means that we have to deal with conflicting statements, as a result of edit commands. Unfortunately, this cannot trivially be expressed in a single RDF graph or document. Several approaches were suggested to resolve similar problems, such as the RDF reification syntax or the introduction of *contexts*, [Guha et al., 2004] turning triple statements into quads for representing a context in which a statement is trusted. The use of named graphs was suggested in [Carrolla et al., 2005]. This allows the representation of possibly conflicting information in separate graphs.

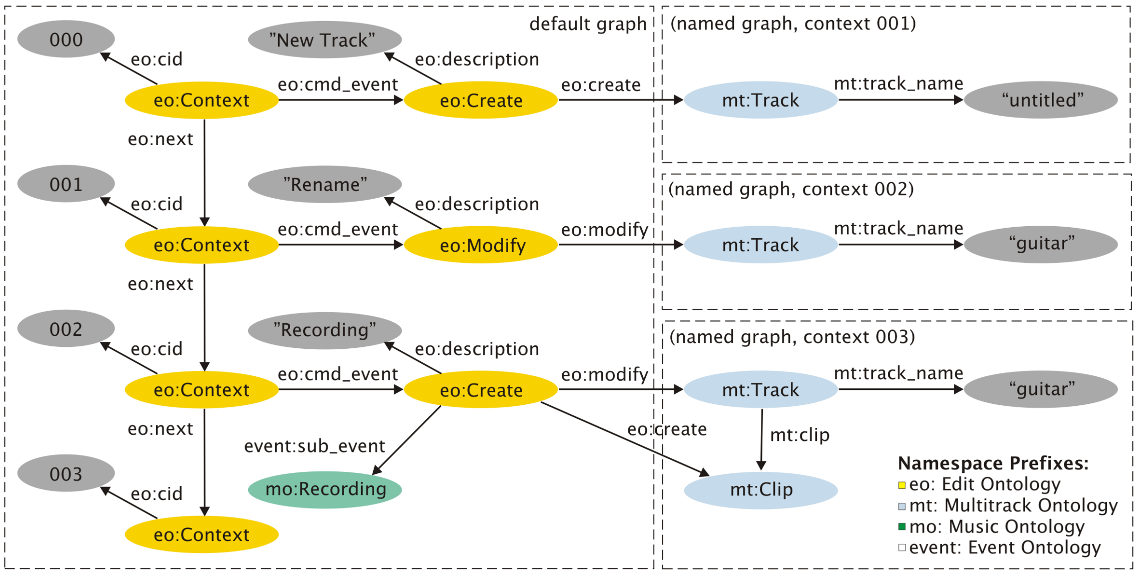


Figure 5.6: Workflow tracking using Named Graphs

In one experimental setup we used named graphs to represent changes to the metadata, as a result of a command events. This technique is supported by both the Redland library and the SPARQL specification for issuing queries on a set of graphs called RDF data sets. An RDF Dataset comprises one graph, the default graph, which does not have a name, and zero or more named graphs [Prud'hommeaux and Seaborne, 2008]. A named graph is an RDF graph associated with a URI reference. This reference may appear in another graph, typically a *default graph*. Following the Redland naming convention, we call this reference a *context node* and associate it with the concept of context in the Edit Ontology. We consider a set of modifications as a result of a command atomic and immutable. We start from an empty graph associated with the zeroth context. Each command event produces a new pair (context,named graph). Then, the new named graph is used to store the statements related to the metadata objects, changed as a result of executing the command. Further, we consider a group of statements about a single metadata object immutable. Any change to the state of the object is stored into a new named graph associated with the context. An example of using this system is depicted in Figure 5.6. It describes a typical sequence of commands: creating a new audio track, changing its name and finally creating a recording. The context nodes in the default graph are used to identify the named graphs, which record the changes to particular statements about subjects.

This methodology has been inspired by, and in some respect similar to other workflow modelling frameworks including PML [da Silva et al., 2006], and the music processing workflow model N3-Tr [Raimond, 2008] discussed in Section 3.2.6.3. Our main objective however is workflow tracking as opposed to planning or execution. Albeit our method provides a working solution, it has considerable drawbacks. Most importantly, the use of named graphs presents difficulties in query formulation (see Section 5.2.4) and data serialisation. Other concerns associated with this representation of change are discussed in Section 4.2.3.4. For this reason, we have moved away from this model in favour of the consolidated reification mechanism discussed in Section 4.2.3.5. However this has not been extended to the whole set of entities that need to be considered in a semantic audio desktop environment. This constitutes future work. In the next section, we discuss a SPARQL query interface which provides access to data collected using the data entry interface, as well as workflow data collected using the named graph model.

5.2.4 Query interface

A SPARQL query interface was built to retrieved the information collected by the interface described in Section 5.2.2. This interface also supports the workflow tracking model described in the previous section. The following simple example shows how one could find the working *project title* using a SPARQL query:

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>

SELECT ?projectTitle
WHERE {
   <urn:project:0001> dc:title ?projectTitle .
}
```

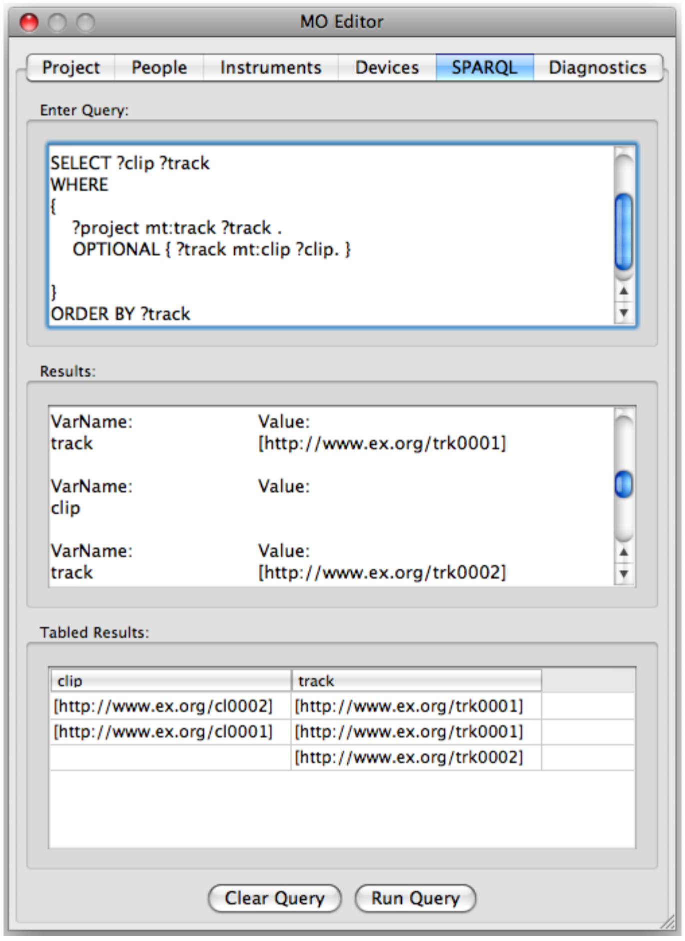
Listing 5.1: Simple SPARQL query in an semantic audio editor

```
PREFIX eo: <http://purl.org/ontology/studio/edit#>
PREFIX mt: <http://purl.org/ontology/studio/multitrack#>

SELECT ?description ?trackName
WHERE {
        ?gr eo:cmd_event ?commandEvent.
        ?commandEvent eo:description ?description.
        GRAPH ?gr
        {
           ?track mt:track_name ?trackName.
        }
}
```

Listing 5.2: SPARQL query for retrieving a set of command events

The example in Listing 5.2 demonstrates how to query the database containing the workflow data described in accordance with the named graph model detailed in the last section. Assume we would like to list the sequence of modifications to the editor so far. Here, the query is matched against all named graphs in the database, containing various states of metadata as a result of a user operating on the editor interface. Using the model in Figure 5.6, this query provides the following results: Besides spreadsheet like browsing of metadata (see fig. 5.5), we extend the previously described interface to enable the evaluation of SPARQL queries on the local database. This is illustrated in Figure 5.7.

```
description  trackName

"New Track"  "untitled"
"Rename"     "guitar"
"Recording"  "guitar"
```



```
SELECT ?clip ?track
WHERE
{
    ?project mt:track ?track .
    OPTIONAL { ?track mt:clip ?clip. }

}
ORDER BY ?track
```

```
VarName:            Value:
track               [http://www.ex.org/trk0001]

VarName:            Value:
clip

VarName:            Value:
track               [http://www.ex.org/trk0002]
```

clip	track
[http://www.ex.org/cl0002]	[http://www.ex.org/trk0001]
[http://www.ex.org/cl0001]	[http://www.ex.org/trk0001]
	[http://www.ex.org/trk0002]

Figure 5.7: SPARQL query interface in Audacity

5.3 SAWA: A Web architecture for semantic audio analysis

Sonic Annotator Web Application[6] (SAWA) was developed with the intention of providing an easy to use graphical user interface (GUI) for automated audio analysis on the Web. This is in contrast with systems providing a Web services API only such as the EchoNest service. It was designed specifically for demonstrating audio feature extraction technologies, and the use of RDF and Semantic Web technologies in audio analysis. The demonstrator was gradually developed into a harmonised framework of support libraries which facilitate the rapid development of Web-based applications involving audio analysis.

The Web applications built on this framework currently include SAWA-Feature Extractor, an audio analysis tool, SAWA-Recommender, a system using content-based similarity as basis for recommendation, as well as applications built collaboratively, such as SAWA-TempEst, an online system for Harpsichord tuning estimation [Tidhar et al., 2010a], and Hotttabs, a multimedia guitar tutor [Barthet et al., 2011]. These applications are detailed in Section 5.4. Further applications are currently in development including SAWA-Experimenter which supports the easy setup an execution of experiments such as the ones discussed in Section 1.5.3. SAWA ties together many of technologies developed at the Centre for Digital Music during the OMRAS2 project[7] [Dixon et al., 2010; Fazekas et al., 2010; Cannam et al., 2010a]. It utilises Semantic Web technologies and it uses Semantic Web ontologies which provide its internal model. In this section, we outline the software architecture of this framework.

5.3.1 Objectives

We can summarise the main objectives of the SAWA framework as follows:

- Facilitate the use of semantic audio analysis on the Web.
- Support a wide range of different algorithms.
- Allow the use of stable released code together with research prototypes.
- Use a flexible, ontology-based common data model for representing information about audio analysis algorithms, configurations and results.
- Be able to adopt to changes in configuration parameters.
- Automatically generate user interfaces for different algorithms and configurations.
- Store feature extraction results, and avoid repeated computation of features given the same audio content and algorithm, with the same parameter configuration.

[6] http://isophonics.net/sawa/
[7] http://www.omras2.org/

5.3.2 Components

The SAWA framework is built on a number of modular components combining software developed collaboratively, as well as using third-party components. The author's contribution include the development of VamPy and VamPy plugins (see Section 5.3.2.3), contribution to the design of ontologies used by the system, contribution to open source projects such as MoPy (see Section 5.3.2.6) that facilitate using ontologies in software, and finally writing the SAWA system that ties these components together. SAWA itself consists of a set of Python libraries written using the components detailed below, as well as standard application wrappers around these components that other Python libraries can use.

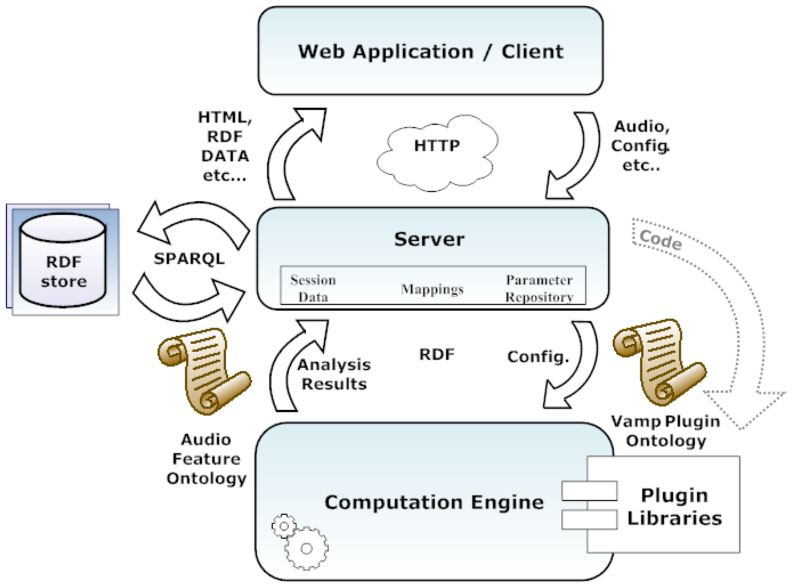


Figure 5.8: An illustration of how various software components and ontologies are used in SAWA

Figure 5.8 shows how various components and ontologies interact in a client-server architecture for building browser-based applications. The server manages multiple concurrent user sessions, maps user queries to computational or database requests, and holds parameter repositories describing audio analysis plugins. It communicates with the computation engine and the end-user application using data expressed in RDF, and with a database back-end using SPARQL. In the following section, we first briefly outline the most important components that are utilised in the SAWA system. The architecture and implementation is then discussed in Section 5.3.3.

5.3.2.1 Vamp plugins

Vamp [Cannam, 2009] is an plugin system designed for audio feature extraction. It consists of a binary plugin interface with C linkage, and a C++ SDK for plugin and host development.

Vamp plugins accept audio data as input and produce structured data as output.

The use of Vamp plugins enable SAWA to support a wide range of different algorithms. A large library of Vamp plugins are available[8], while the plugins are supported in host applications such as Sonic Visualiser (see Section 5.3.2.4), Sonic Annotator (Section 5.3.2.5), and the Audacity[9] audio editor. The range of existing Vamp plugins include:

- Estimators for *musically meaningful features*, including note onset detectors, beat and tempo estimators, structural segmentation, and key estimators;
- *Low-level audio feature extractors*, such as amplitude, measures of spectral shape;
- *Metadata annotators* such as audio fingerprinting and identification;
- Calculators for *dense features* that are often used in visualisations, such as chromagram and harmonic spectrum.

The features represented in C++ data structures returned by a Vamp plugin are rich enough to contain the necessary data to represent musically meaningful features such as beat or key-change information. However, they are not rich enough to describe what those features represent. They do not provide enough information for a host program to place them in context among other musical features.

While it may be evident to a human that a plugin named "key change detector" detects key changes, a program cannot know that the outputs of this plugin will be comparable to other sources of key information [Fazekas et al., 2009]. In order to establish this relationship and ensure that the returned features are correctly understood, the plugin's output needs to be associated with relevant concepts in the Audio Features Ontology (see Section 4.3). The terms used to make this connection are found in the Vamp Plugin Ontology (see Section 4.4).

5.3.2.2 Vamp ontologies

The Vamp Plugin Ontology (see Section 4.4 and [Cannam et al., 2010a]) consists of two conceptually different sub-ontologies: The main part of the ontology describes Vamp Plugins as separate entities and allows linking them to individual plugin outputs, that is a type of output a plugin is capable to produce. It also allows grouping plugins into libraries similarly to their binary distribution.

The most useful aspect of this for the SAWA system is the association of a plugin outputs with terms in the Audio Features Ontology to express what the output describes. These may be distinct event types like note onsets, features describing aspects of the whole recording such as an audio fingerprint or dense signal data such as a chromagram. SAWA uses this

[8] `http://vamp-plugins.org/download.html`

[9] Vamp Plugins in Audacity: `http://audacityteam.org/wiki/index.php?title=Vamp_Plug-ins`

ontology to identify the exact source of data and to annotate the result so that they can be used more meaningfully. The Vamp Transform Ontology is published as part of the Vamp Plugin Ontology, though it is considered conceptually separate. It contains terms to describe how a plugin may be configured and run. SAWA uses this to drive its processing, identifying parameter values and other details such as audio block and step sizes. This information is expressed and stored using the same RDF format as the results without imposing any additional encoding requirements. Any agent reading these results will therefore have certainty about how they were computed. This type of information is a very valuable detail in the context of any data-centric field of research.

5.3.2.3 VamPy

VamPy, a Python wrapper for Vamp plugins is a crucial component in several applications of SAWA. This wrapper plugin acts as an ordinary Vamp plugin which may be installed in the usual manner. When it is installed, any appropriately structured Python scripts found in its script directory will be presented as if they were individual Vamp plugins for any Vamp host to use. VamPy permits Vamp plugins to be rapidly and dynamically developed using Python libraries for numerical and scientific computation such as NumPy and SciPy. Here, we only outline how VamPy is used in SAWA. Its architecture and services, such as the type inference mechanism which enable the use of dynamic typing in VamPy plugins is briefly described in Appendix B.2.

The most useful aspect of VamPy for the SAWA system is that it permits the use of audio analysis research prototypes written in Python, without the need for implementing the plugin in C++ first. We utilise this feature of VamPy in the TempEst system discussed in Section 5.4.3, where the audio analysis as well as the resulting RDF data is produced by a VamPy plugin. Another possible application of VamPy is illustrated in Figure 5.8, where it allows for the extension of the system using experimental scripts. However, for security, this requires sophisticated sandboxing facilities which are currently not provided.

5.3.2.4 Sonic Visualiser

Sonic Visualiser[10] [Cannam et al., 2010b], is an audio analysis application and Vamp plugin host. It is also capable of loading and viewing features calculated by other programs. In particular, it can load multiple sets of features associated with a single audio file from an RDF document generated by SAWA or Sonic Annotator. It therefore serves as a possible "offline" visualisation interface for the SAWA system [Fazekas et al., 2009].

[10]http://sonicvisualiser.org/

5.3.2.5 Sonic Annotator

Sonic Annotator[11] is an audio analysis application which applies Vamp feature extraction plugins to audio data in a batch. It is built using Sonic Visualiser libraries, the two applications therefore share capabilities such as broad support for different audio file formats, network retrieval of audio files, and the interpretation of feature extraction specifications (also called "transform" in the context of Vamp related ontologies) in RDF using the Vamp Transform Ontology (see Section 5.3.2.2).

Sonic Annotator currently serves as the main computation engine and Vamp plugin host within SAWA, albeit other solutions, such as Python Vamp host SDK (a low-level Python wrapper around the Vamp C++ host SDK) is under development. SAWA includes a Python wrapper library around the functionality of Sonic Annotator, which communicates with the application using Unix pipes. This library provides a simple API for other parts of the system to use, including methods that are similar to those of the Vamp host API for enumerating, configuring and calling Vamp plugins. The communication between the Python wrapper and Sonic Annotator is based on sending and receiving data encoded using RDF.

5.3.2.6 MoPy

MoPy[12] is a Python interface for the Music Ontology and related ontologies. It generates Python classes associated with ontology terms which simplifies interaction with RDF data. SAWA uses an adapted version of MoPy in generating Vamp plugin configuration pages, and in general to complement its object model using terms defined in Semantic Web ontologies.

5.3.2.7 CherryPy

SAWA builds on its own Python-based Web server using the third-party open-source library CherryPy[13]. This library allows for writing HTTP request handlers as ordinary methods defined within a web application class. This way it is straightforward to serve dynamically generated web pages, as well as publishing data received from other system components. CherryPy supports the Web Server Gateway Interface (WSGI) interface, making SAWA portable to other server environments and implemented, for instance, as a traditional Common Gateway Interface (CGI) application behind an Apache server.

5.3.3 Architecture

SAWA combines the previously described components into a framework that facilitates the development of Web-based applications involving semantic audio analysis. SAWA supports

[11]http://omras2.org/SonicAnnotator
[12]http://sourceforge.net/projects/motools/mopy
[13]http://www.cherrypy.org/

applications that follow the Representational State Transfer (REST) style Web application design, where Web pages form a virtual state machine, allowing a user to progress through the application by selecting links, with each action resulting in a transition to the next state of the application by transferring a representation of that state to the user [Fielding and Taylor, 2002]. It also supports applications that follow linked-data principles [Berners-Lee, 2006], and Semantic Web user agents, that connect to an application using a SPARQL-endpoint. Note that currently the SPARQL interface is in experimental status, and supports only the retrieval of previously computed transforms from SAWA's database.

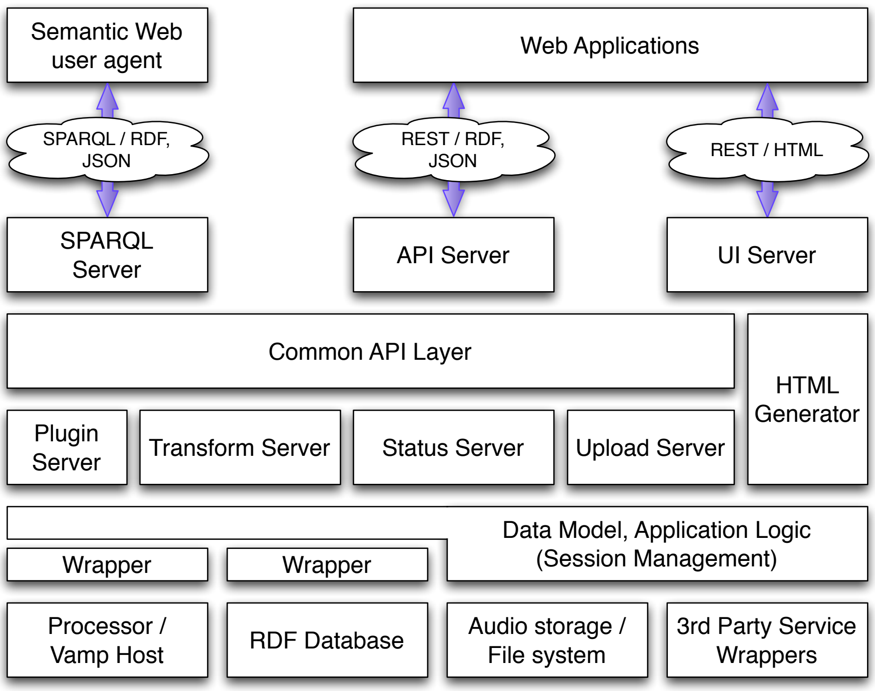


Figure 5.9: Simplified software architecture of SAWA

A somewhat simplified architecture of the SAWA framework is shown in Figure 5.9. These building blocks are implemented as one or more classes in the actual application. Different types of applications are serviced by conceptually different server components available at different ports, or using different URIs. The API and UI servers together provide for building web applications that use interfaces generated on the server side. This supports building light weight clients using a combination of standard Web technologies (HTML, CSS, Javascript). Web applications that use client side interface generation can makes use of, for instance, the JQuery[14] library to dynamically load content from the server via XMLHttp requests. This communication can relay on the the Web API server alone, but the client has to be able to process the returned RDF or Javascript Object Notation (JSON) data. An experimental

[14]http://jquery.com/

SPARQL server provides access to information stored in SAWA's database, such as previously computed features of audio files. The various functions provided by SAWA are bound together in a common layer, a simple Python API provided by a set of underlying classes, which can be used to implement different types of servers.

5.3.3.1 Domain model and control logic

The communication between the clients and the server is co-ordinated using the Model View Controller (MVC) architectural pattern [Evans, 2003]. Some important domain objects in the MVC *model*, as well as the database schemata for the application framework are provided by the Music Ontology, its extensions and the Vamp plugin ontologies. The corresponding data structures are generated from the ontology specification using the previously mentioned MoPy library (see 5.3.2.6).

The *controllers* in the MVC model are implemented by the classes managing information about feature extraction plugins (plugin server), caching and previously executed transforms (transform server), file uploads (upload server) and the current status of the application (status server). These components are implemented by multiple classes, each providing a simple API with relevant functionality. For instance, the Transform Server consists of two classes: the `TransformManager` is responsible for maintaining information about previously computed transforms accessing the database for storing or retrieving results and notifying the domain logic to perform feature extraction if the transform is not stored. The `TransformServer` class is responsible for content negotiation and accessing different *views* of the data according to the requested output type. It provides for returning data in different formats, for instance, RDF data in Turtle syntax, RDF/XML embedded in HTML (to be displayed to a user who wish to learn about the use of RDF in audio analysis), or a file download. The necessary data conversions may be performed by different classes such as the underlying `ResultsController` or the HTML user interface generator.

The status server is responsible for informing clients about the status of long processes such as feature extraction or data conversion in response to API calls. This is important in cases where numerous audio files or computationally expensive transforms are part of a request. The upload server accepts and manages file uploads, and perform tasks such as verification of the uploaded file (i.e. check if a valid audio file was indeed uploaded by the user) and generating responses necessary to support multipart HTML5 uploads. The plugin server maintains and provides information about the available feature extractors in the system.

These functionalities are tied together by additional application logic for accessing the feature extractor engine, the underlying database, the file system for temporarily audio file storage, and third-party Web services such as MusicBrainz, the EchoNest[15] or YouTube[16].

[15] http://www.echonest.com/
[16] http://www.youtube.com/

5.3.3.2 User session management

SAWA operates in a multi-user environment. It is able to handle multiple concurrent user sessions and maintain information such as uploaded audio files, executed transforms, and saved transform configurations in several isolated execution contexts. User session management is implemented as an extension plugin hosted by CherryPy's multi-threaded Web server environment. The extension called `DirectorySession` isolates user sessions by creating a separate file system directory for each active session where audio files, configuration data and temporary files offered by the transform server can be stored. It maintains status information in memory for the duration of the user session and notifies the relevant logic to delete temporary files and any uploaded audio material (assumed to be copyrighted) when the session expires.

5.3.3.3 Dynamic user interface generation

SAWA provides for generating a configuration interface for audio feature extractors which light weight clients can request and use. These interfaces are automatically built by interpreting RDF data describing Vamp plugins using the Vamp plugin ontology (see Section 5.3.2.2). The interfaces are expressed in HTML, which is generated by mapping plugin parameter types to HTML form elements. Similarly, RDF data describing plugin libraries can be used to generate high-level descriptions of Vamp plugins such as the main interface of SAWA-FeatureExtractor discussed in Section 5.4.1. The automatically generated user interfaces can accommodate different feature extractors as long as they are described in RDF. Therefore it is possible to extend SAWA with new functionality (i.e. adding new plugins) without changing or updating its codebase.

5.3.3.4 RDF caching and pagination

SAWA identifies uploaded audio files by computing a fingerprint using the third-party MusicDNS[17] system and retrieves an associated identifier. This identifier is matched against the MusicBrainz database to obtain editorial metadata which can be added to the results of audio analysis. The fingerprints are also stored in an RDF database together with these results. SAWA caches output of feature extractors at the level of audio items. All data related to a unique fingerprint is stored in a named RDF graph [Carrolla et al., 2005] together with the provenance of the transform execution. This includes the configuration parameters of the algorithm computing the features. Caching can be enabled or disabled by the user or a description associated with the algorithm stating whether the transform is *deterministic* producing the same result at each execution, or *non-deterministic* producing different results at each execution.

[17] http://www.musicdns.com/

Additionally, SAWA can divide large RDF descriptions of audio feature extraction results for efficient communication and storage. This is especially useful when retrieving results of low-level transformations such as spectrograms which tend to be very large when expressed in RDF. This process is called pagination. We utilise the RDF vocabulary associated with the link-data API[18] for describing how result sets belong together and how they may be recombined on the client side.

5.3.4 SAWA and linked data

Using RDF and the linked data concept is key in creating the modular architecture described above. The server side application uses RDF data to communicate with other components such as Sonic Annotator and interprets RDF data to generate its user interface dynamically. It also accesses third-party data sources to obtain metadata, and incorporates these in the output. An additional benefit may come from the fact that the data collected by this application may be released and linked with additional linked open data repositories discussed in Section 2.4.7. The advantages of using the RDF format in the context of SAWA are manifold:

- **Generic interface design:** Interfaces can be generated for any number and type of plugins and their output configuration.
- **Generic implementation:** The system may rely on any suitable Vamp plugin host or computation engine.
- **Efficiency:** Cached results can be returned from a suitable RDF store instead of repeating computation.
- **Extensibility:** The system can access other linked data services to augment the results with different types of metadata.

So far we have discussed the core components and principles that are utilised in building the framework of libraries that constitute the SAWA system. In the next section we describe several applications that were built using this framework. This includes Web applications for batch audio feature extraction, music recommendation, music tuition, and the analysis and estimation of musical temperament.

5.4 Applications of SAWA

In this section we outline some Web-based tools built using SAWA and Semantic Web technologies. These tools support primarily music informatics use cases. The SAWA framework introduced in Section 5.3 provides for flexible Web-based access to music analysis algorithms

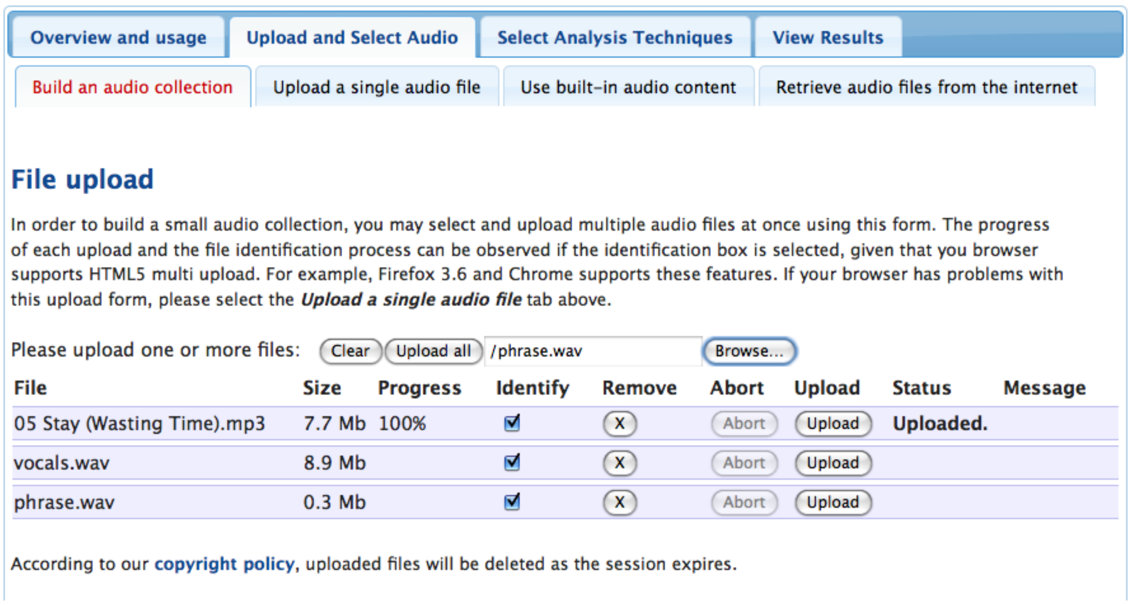
[18]http://purl.org/linked-data/api/vocab

wrapped as Vamp or VamPy plugins and serves as the basis for a set of Web applications with similar underlying needs. Some of the prime goals of the applications descried in this section can be summarised as follows:

- Demonstrate the utilities of dynamic interpretation and generation of RDF data in end-user applications.
- Prototype some data modelling concepts to be adapted in information management for semantic audio applications.
- Promote the use of the RDF data model for interoperability in music information retrieval.

5.4.1 SAWA Feature Extractor

The main application of SAWA is a Web-based audio analysis system[19]. Contrary to systems using a Web Services API, this program is developed with the intention of providing an easy to use human interface as well as a SPARQL query interface. SAWA Feature Extractor works with a collection of audio files uploaded by the user. A collection can be built using the interface shown in Figure 5.10.


Overview and usage
Upload and Select Audio
Select Analysis Techniques
View Results
Build an audio collection
Upload a single audio file
Use built-in audio content
Retrieve audio files from the internet

File upload

In order to build a small audio collection, you may select and upload multiple audio files at once using this form. The progress of each upload and the file identification process can be observed if the identification box is selected, given that you browser supports HTML5 multi upload. For example, Firefox 3.6 and Chrome supports these features. If your browser has problems with this upload form, please select the ***Upload a single audio file*** tab above.

Please upload one or more files: Clear Upload all /phrase.wav Browse...

File	Size	Progress	Identify	Remove	Abort	Upload	Status	Message
05 Stay (Wasting Time).mp3	7.7 Mb	100%	✓	X	Abort	Upload	**Uploaded.**	
vocals.wav	8.9 Mb		✓	X	Abort	Upload		
phrase.wav	0.3 Mb		✓	X	Abort	Upload		

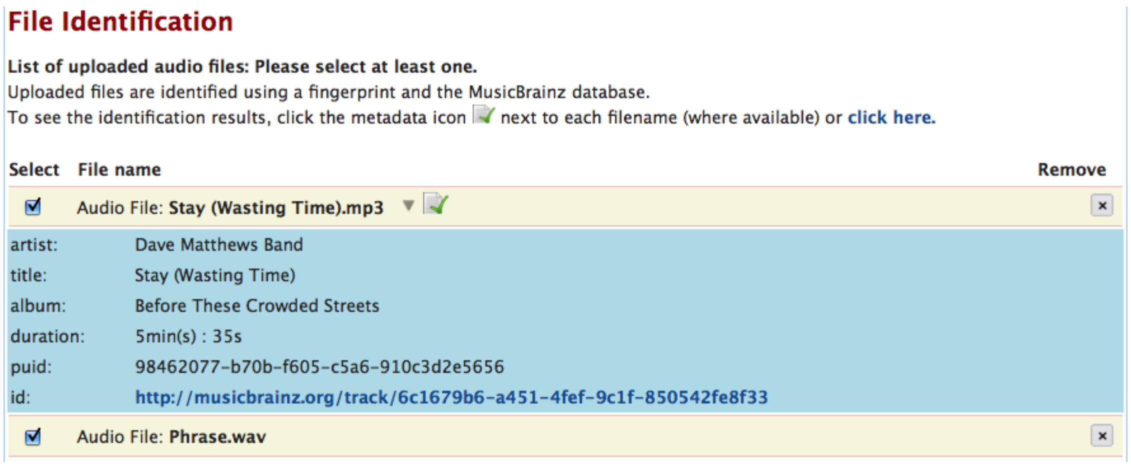
According to our **copyright policy**, uploaded files will be deleted as the session expires.

Figure 5.10: Audio file upload and collection builder interface in SAWA

The system supports multipart HTML5 form uploads, therefore a set of audio files to be analysed can be compiled locally and several files can be uploaded in parallel. An interface

[19]SAWA Feature Extractor: http://isophonics.net/sawa/

for creating a collection from content available on the web is also in development. The various interfaces can be selected using the tabbed interface shown in the above figure.


File Identification

List of uploaded audio files: Please select at least one.
Uploaded files are identified using a fingerprint and the MusicBrainz database.
To see the identification results, click the metadata icon next to each filename (where available) or **click here.**

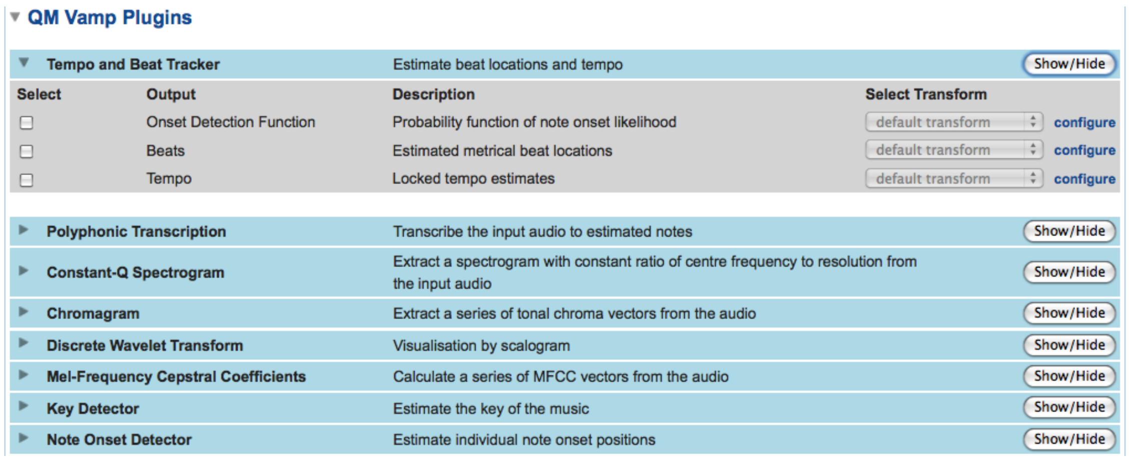
Select	File name	Remove
✓	Audio File: **Stay (Wasting Time).mp3**	×

artist:	Dave Matthews Band
title:	Stay (Wasting Time)
album:	Before These Crowded Streets
duration:	5min(s) : 35s
puid:	98462077-b70b-f605-c5a6-910c3d2e5656
id:	http://musicbrainz.org/track/6c1679b6-a451-4fef-9c1f-850542fe8f33

Select	File name	Remove
✓	Audio File: **Phrase.wav**	×

Figure 5.11: Audio file identification in SAWA

Optionally, each audio file is identified using a fingerprint, and basic bibliographic metadata given an associated identifier (PUID) is retrieved from the MusicBrainz service. Figure 5.11 exemplifies the result of identification. The fingerprint serves as the identifier that is used for storing and later identifying provenance information and to facilitate RDF caching.


QM Vamp Plugins

Tempo and Beat Tracker — Estimate beat locations and tempo — Show/Hide

Select	Output	Description	Select Transform
☐	Onset Detection Function	Probability function of note onset likelihood	default transform configure
☐	Beats	Estimated metrical beat locations	default transform configure
☐	Tempo	Locked tempo estimates	default transform configure

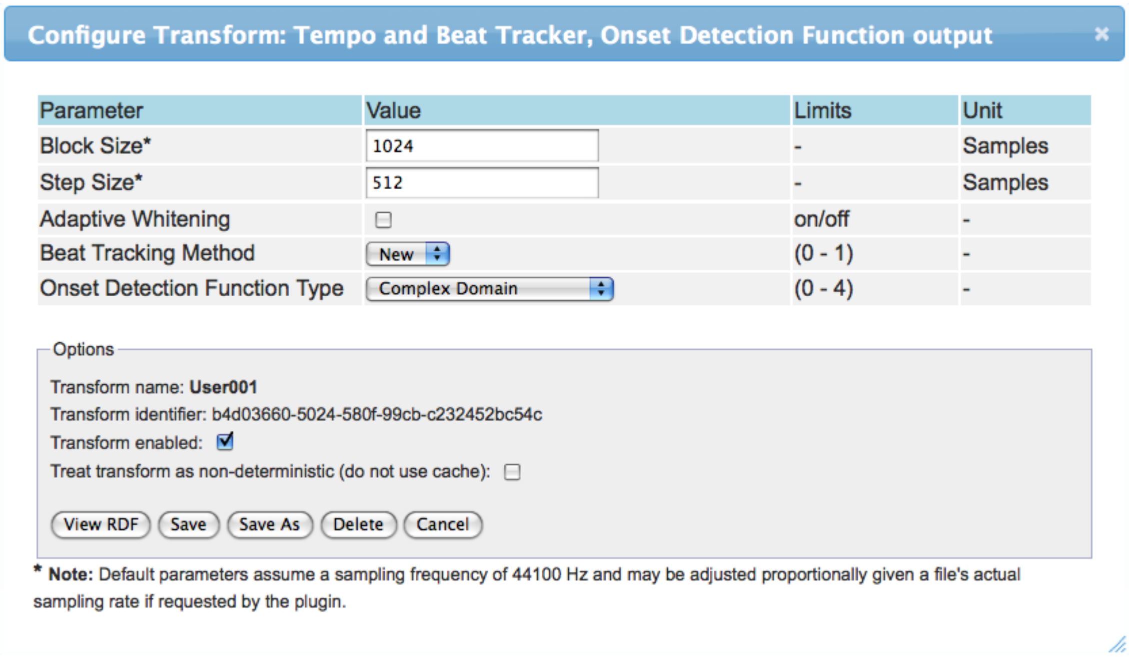
Polyphonic Transcription	Transcribe the input audio to estimated notes	Show/Hide
Constant-Q Spectrogram	Extract a spectrogram with constant ratio of centre frequency to resolution from the input audio	Show/Hide
Chromagram	Extract a series of tonal chroma vectors from the audio	Show/Hide
Discrete Wavelet Transform	Visualisation by scalogram	Show/Hide
Mel-Frequency Cepstral Coefficients	Calculate a series of MFCC vectors from the audio	Show/Hide
Key Detector	Estimate the key of the music	Show/Hide
Note Onset Detector	Estimate individual note onset positions	Show/Hide

Figure 5.12: Selecting a feature extractor in SAWA

SAWA Feature Extractor allows for the selection and configuration of one or more Vamp outputs and execute transforms on previously uploaded files. Here, a transform is seen as an algorithm associated with a Vamp plugin output and a specific set of configured parameters used during the execution. That is, a single algorithm configured differently is seen as distinct transforms. Results are returned as RDF data. This can be examined using an RDF browser such as Tabulator, imported in Sonic Visualiser and viewed in context of the audio or published on the Semantic Web. SAWA's plugin selection interface can be seen in Figure 5.12.

5.4.1.1 Configuration interface

The interface is designed using a combination of faceted-browser and RDF-browser concepts to accommodate the specialised nature of the application. This means that some constraints are imposed on what is displayed to the user, however, when available, individual Vamp plugin descriptions can be browsed freely. Plugin libraries such as *qm-vamp-plugins* in Figure 5.12 are displayed as top level elements. The user may click on the triangular symbols associated with libraries or plugins to expand the content such as the description of each output of a plugin. One or more transforms can be configured for a selected plugin output. These transforms are saved temporarily and applied individually on each audio file after submitting a query. It is required to select at least one transform, however configuration is optional. By default, the system computes a predefined transform associated with a plugin output. Feature extractors may be configured using an automatically generated interface. This is exemplified in Figure 5.13 showing the parameters for a tempo and beat tracker.


Configure Transform: Tempo and Beat Tracker, Onset Detection Function output

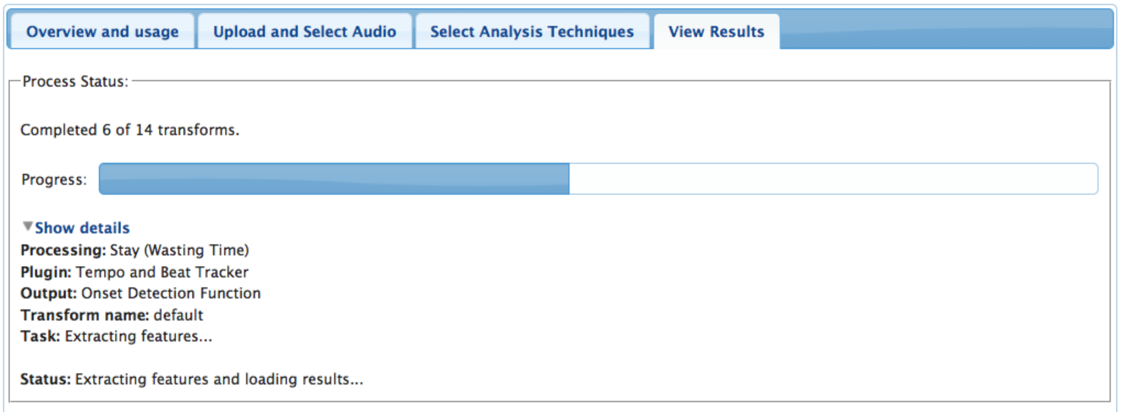
Parameter	Value	Limits	Unit
Block Size*	1024	-	Samples
Step Size*	512	-	Samples
Adaptive Whitening	☐	on/off	-
Beat Tracking Method	New	(0 - 1)	-
Onset Detection Function Type	Complex Domain	(0 - 4)	-

Options
Transform name: **User001**
Transform identifier: b4d03660-5024-580f-99cb-c232452bc54c
Transform enabled: ☑
Treat transform as non-deterministic (do not use cache): ☐

* **Note:** Default parameters assume a sampling frequency of 44100 Hz and may be adjusted proportionally given a file's actual sampling rate if requested by the plugin.

Figure 5.13: Feature extractor configuration in SAWA

Once a set of feature extractors have been selected and configured, a query can be submitted to perform the analysis of all or a user defined subset of the uploaded audio files. The progress of the analysis can be monitored using the status display shown in Figure 5.14. While the feature extraction process is running, the results are loaded by the Web application as they become available, and a table of results shown in Figure 5.15 is dynamically expanded.



Process Status:

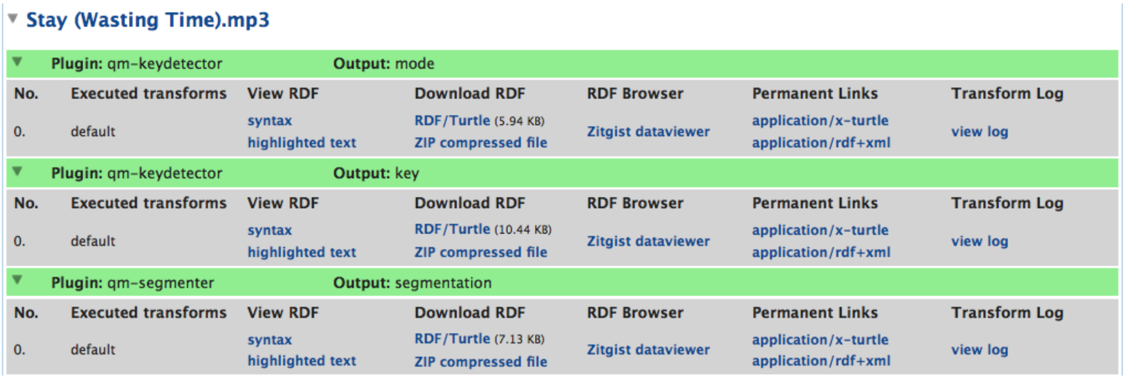
Completed 6 of 14 transforms.

Progress:

▼Show details
Processing: Stay (Wasting Time)
Plugin: Tempo and Beat Tracker
Output: Onset Detection Function
Transform name: default
Task: Extracting features...

Status: Extracting features and loading results...

Figure 5.14: Feature extraction status display in SAWA



▼ **Stay (Wasting Time).mp3**

Plugin: qm-keydetector **Output:** mode

No.	Executed transforms	View RDF	Download RDF	RDF Browser	Permanent Links	Transform Log
0.	default	syntax highlighted text	RDF/Turtle (5.94 KB) ZIP compressed file	Zitgist dataviewer	application/x-turtle application/rdf+xml	view log

Plugin: qm-keydetector **Output:** key

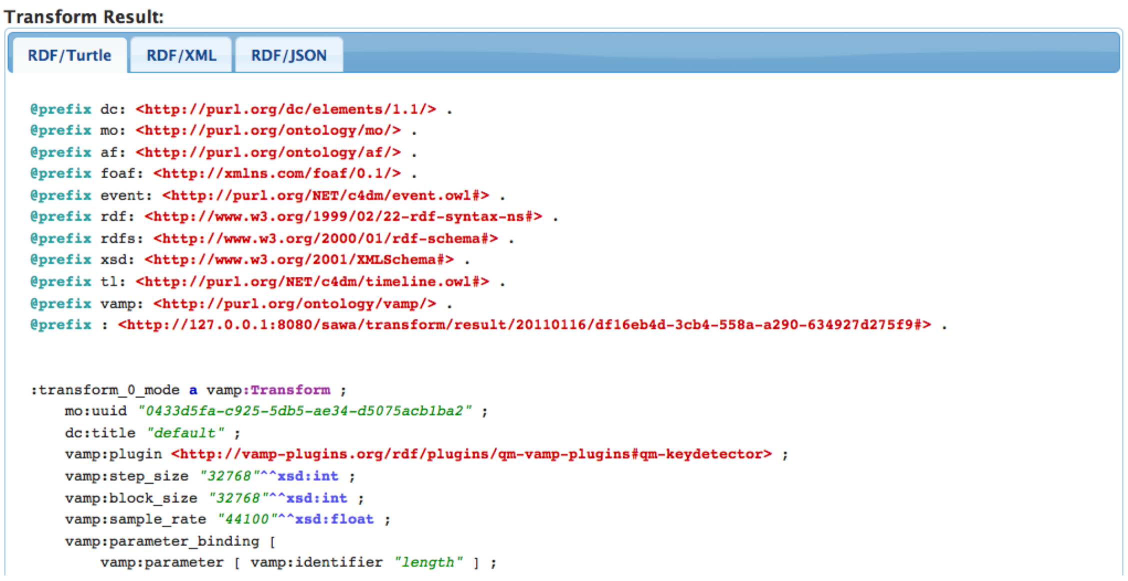
No.	Executed transforms	View RDF	Download RDF	RDF Browser	Permanent Links	Transform Log
0.	default	syntax highlighted text	RDF/Turtle (10.44 KB) ZIP compressed file	Zitgist dataviewer	application/x-turtle application/rdf+xml	view log

Plugin: qm-segmenter **Output:** segmentation

No.	Executed transforms	View RDF	Download RDF	RDF Browser	Permanent Links	Transform Log
0.	default	syntax highlighted text	RDF/Turtle (7.13 KB) ZIP compressed file	Zitgist dataviewer	application/x-turtle application/rdf+xml	view log

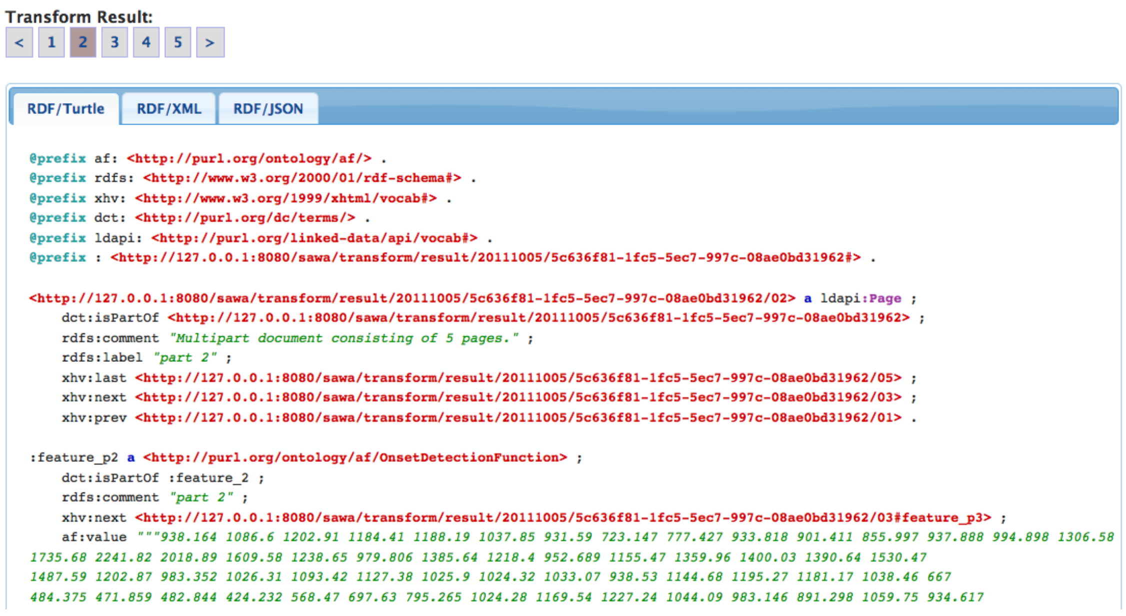
Figure 5.15: Table of feature extraction results in SAWA

The feature extraction results are available in several different formats generated on demand from the default RDF/Turtle syntax. These results can be examined within the application as shown in Figure 5.16, downloaded as uncompressed RDF or as a ZIP compressed file, or examined in a data viewer. Additionally, each result receives a permanent URI. This allows all transform results to be accessed later, even after the user session has been expired. For instance, the permanent URI: `http://isophonics.net/sawa/transform/result/20110116/df16eb4d-3cb4-558a-a290-634927d275f9` is associated with the result shown in the example. The URI is designed to include the date of feature extraction and a universally unique identifier (UUID) generated given the audio fingerprint and the transform configuration in order to avoid potential URI collisions. These permanent links are available in RDF/Turtle and RDF/XML formats. Results consisting of a large number of RDF triples and those that include large matrices are paginated as shown in Figure 5.17.



```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix mo: <http://purl.org/ontology/mo/> .
@prefix af: <http://purl.org/ontology/af/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix event: <http://purl.org/NET/c4dm/event.owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix tl: <http://purl.org/NET/c4dm/timeline.owl#> .
@prefix vamp: <http://purl.org/ontology/vamp/> .
@prefix : <http://127.0.0.1:8080/sawa/transform/result/20110116/df16eb4d-3cb4-558a-a290-634927d275f9#> .


:transform_0_mode a vamp:Transform ;
    mo:uuid "0433d5fa-c925-5db5-ae34-d5075acb1ba2" ;
    dc:title "default" ;
    vamp:plugin <http://vamp-plugins.org/rdf/plugins/qm-vamp-plugins#qm-keydetector> ;
    vamp:step_size "32768"^^xsd:int ;
    vamp:block_size "32768"^^xsd:int ;
    vamp:sample_rate "44100"^^xsd:float ;
    vamp:parameter_binding [
        vamp:parameter [ vamp:identifier "length" ] ;
```

Figure 5.16: Feature extraction results in SAWA

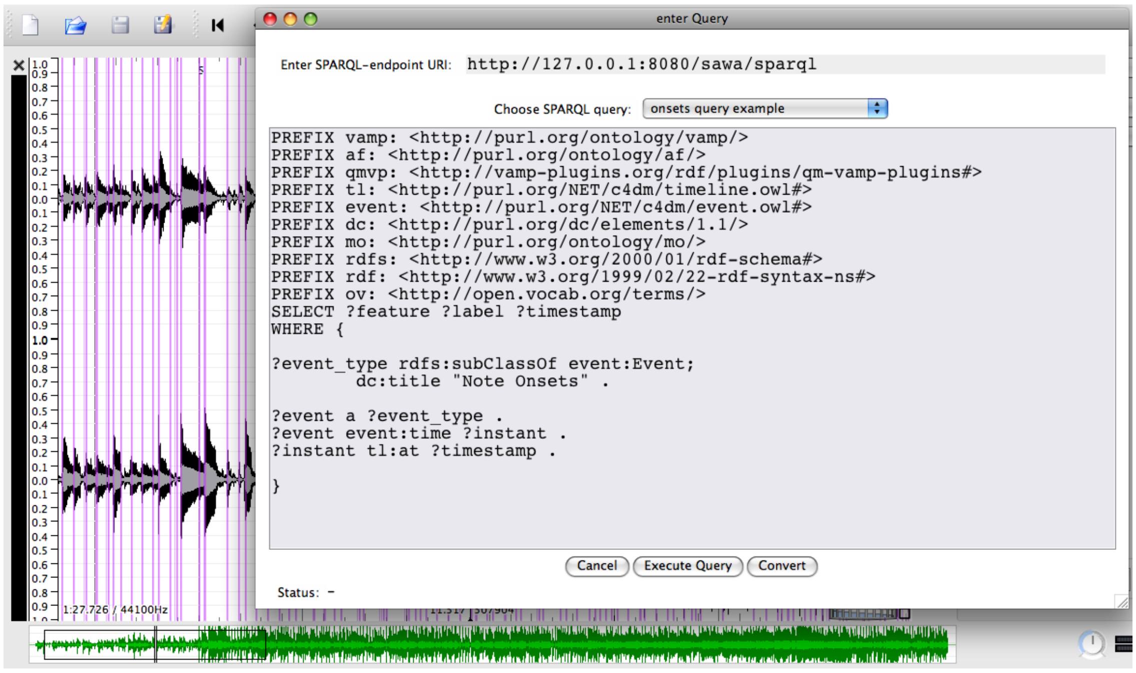


```
@prefix af: <http://purl.org/ontology/af/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xhv: <http://www.w3.org/1999/xhtml/vocab#> .
@prefix dct: <http://purl.org/dc/terms/> .
@prefix ldapi: <http://purl.org/linked-data/api/vocab#> .
@prefix : <http://127.0.0.1:8080/sawa/transform/result/20111005/5c636f81-1fc5-5ec7-997c-08ae0bd31962#> .

<http://127.0.0.1:8080/sawa/transform/result/20111005/5c636f81-1fc5-5ec7-997c-08ae0bd31962/02> a ldapi:Page ;
    dct:isPartOf <http://127.0.0.1:8080/sawa/transform/result/20111005/5c636f81-1fc5-5ec7-997c-08ae0bd31962> ;
    rdfs:comment "Multipart document consisting of 5 pages." ;
    rdfs:label "part 2" ;
    xhv:last <http://127.0.0.1:8080/sawa/transform/result/20111005/5c636f81-1fc5-5ec7-997c-08ae0bd31962/05> ;
    xhv:next <http://127.0.0.1:8080/sawa/transform/result/20111005/5c636f81-1fc5-5ec7-997c-08ae0bd31962/03> ;
    xhv:prev <http://127.0.0.1:8080/sawa/transform/result/20111005/5c636f81-1fc5-5ec7-997c-08ae0bd31962/01> .

:feature_p2 a <http://purl.org/ontology/af/OnsetDetectionFunction> ;
    dct:isPartOf :feature_2 ;
    rdfs:comment "part 2" ;
    xhv:next <http://127.0.0.1:8080/sawa/transform/result/20111005/5c636f81-1fc5-5ec7-997c-08ae0bd31962/03#feature_p3> ;
    af:value """938.164 1086.6 1202.91 1184.41 1188.19 1037.85 931.59 723.147 777.427 933.818 901.411 855.997 937.888 994.898 1306.58
1735.68 2241.82 2018.89 1609.58 1238.65 979.806 1385.64 1218.4 952.689 1155.47 1359.96 1400.03 1390.64 1530.47
1487.59 1202.87 983.352 1026.31 1093.42 1127.38 1025.9 1024.32 1033.07 938.53 1144.68 1195.27 1181.17 1038.46 667
484.375 471.859 482.844 424.232 568.47 697.63 795.265 1024.28 1169.54 1227.24 1044.09 983.146 891.298 1059.75 934.617
```

Figure 5.17: Paginated results in SAWA

5.4.1.2 SPARQL client

In order to test the experimental SPARQL-endpoint of SAWA Feature Extractor, we built a SPARQL client prototype for Sonic Visualiser. This client is wrapped in a VamPy plugin which accesses a user defined SPARQL-endpoint and converts the returned data into a format a Vamp host can understand. A drawback of this solution is that the plugin protocol still requires the audio content to be transferred to the plugin instead of simply identifying the file we wish to retrieve the data for. Nevertheless, it is suitable for querying the database of SAWA and displaying the results in a visual host like Sonic Visualiser or Audacity. The simple query interface of this client is shown in Figure 5.18.



Enter SPARQL-endpoint URI: http://127.0.0.1:8080/sawa/sparql

Choose SPARQL query: onsets query example

```
PREFIX vamp: <http://purl.org/ontology/vamp/>
PREFIX af: <http://purl.org/ontology/af/>
PREFIX qmvp: <http://vamp-plugins.org/rdf/plugins/qm-vamp-plugins#>
PREFIX tl: <http://purl.org/NET/c4dm/timeline.owl#>
PREFIX event: <http://purl.org/NET/c4dm/event.owl#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX mo: <http://purl.org/ontology/mo/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ov: <http://open.vocab.org/terms/>
SELECT ?feature ?label ?timestamp
WHERE {

?event_type rdfs:subClassOf event:Event;
        dc:title "Note Onsets" .

?event a ?event_type .
?event event:time ?instant .
?instant tl:at ?timestamp .

}
```

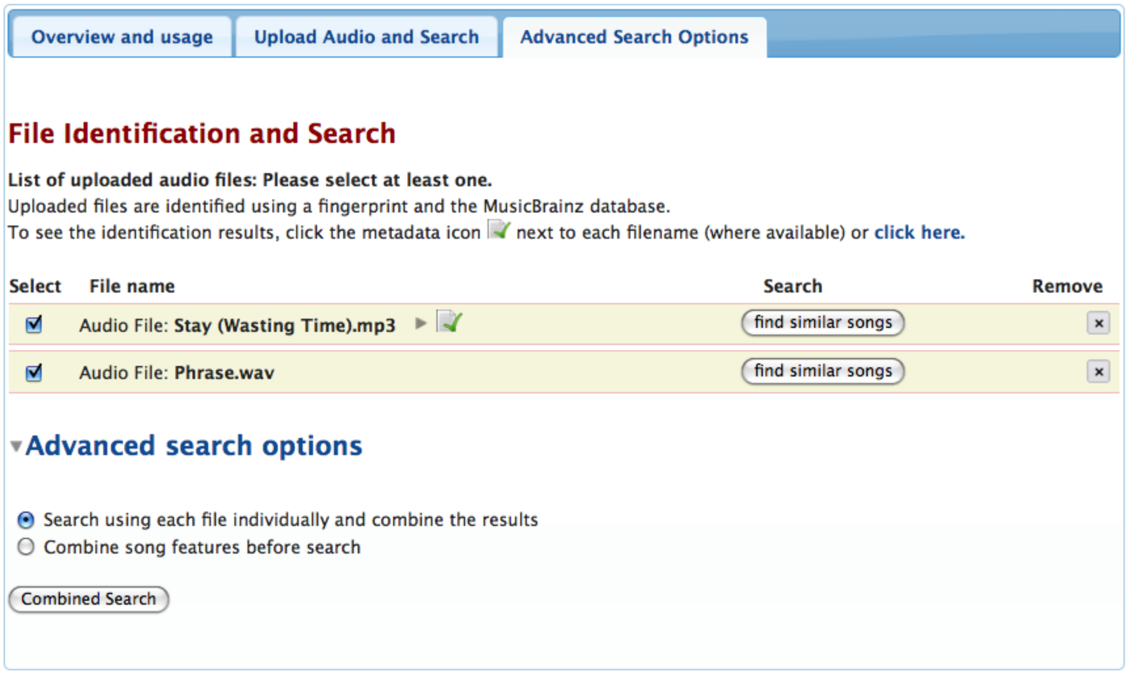
Figure 5.18: SPARQL client in Sonic Visualiser. Note that only onset information available about a song is assumed to be present in the database in the experimental query shown above. The query is executed in a VamPy plugin which converts JSON results from the SAWA end-point into *Vamp::Feature* data structures.

5.4.2 SAWA Recommender

Assessing the similarity between sounds including short audio segments such as recording takes as well as complete songs have many potential uses in semantic audio applications. A measure of similarity can form the basis of music browsing and recommendation systems [Logan and Salomon, 2001; Levy and Sandler, 2006a], as well as intelligent tools in the studio [Fazekas and Sandler, 2007b], for instance, for finding similarities across multiple recording takes. An important aspect of similarity is *timbre* which describes a perceptual characteristics of sound independent of its pitch (see Section 1.4.1.2). In this section, we describe SAWA-Recommender[20], a *query by example* type content-based music recommendation service. Its main goal is to demonstrate an application of the SAWA system in a simple recommendation tool using content-based similarity, and the use of the music similarity database and search technique briefly outlined in Appendix B.1.

[20]SAWA-Recommender: http://isophonics.net/sawa/rec/

SAWA-Recommender finds songs in its database which sound similar in some sense to one or more uploaded music files. Most notably, it finds songs which have similar instrumentation or similarly sounding dominant instruments such as lead vocals, saxophones, or violins. SAWA-Recommender uses a database of timbre features extracted from over 150,000 tracks in a distributed way by users of SoundBite[21], a playlist generator plugin introduced in [Sandler and Levy, 2007] for the iTunes[22] and SongBird[23] media players. This software analyses each user's music collection and reports the features alongside some metadata to a central server. The returned data is the basis for the Isophone database [Tidhar et al., 2009] and our online recommender system. This database mainly consists of popular music, however the similarity algorithm itself proved valuable in informal testing also in the context of classical recordings.



File Identification and Search

List of uploaded audio files: Please select at least one.
Uploaded files are identified using a fingerprint and the MusicBrainz database.
To see the identification results, click the metadata icon ✓ next to each filename (where available) or **click here.**

Select	File name	Search	Remove
✓	Audio File: **Stay (Wasting Time).mp3**	find similar songs	×
✓	Audio File: **Phrase.wav**	find similar songs	×

Advanced search options

[✓ Search using each file individually and combine the results]
Combine song features before search

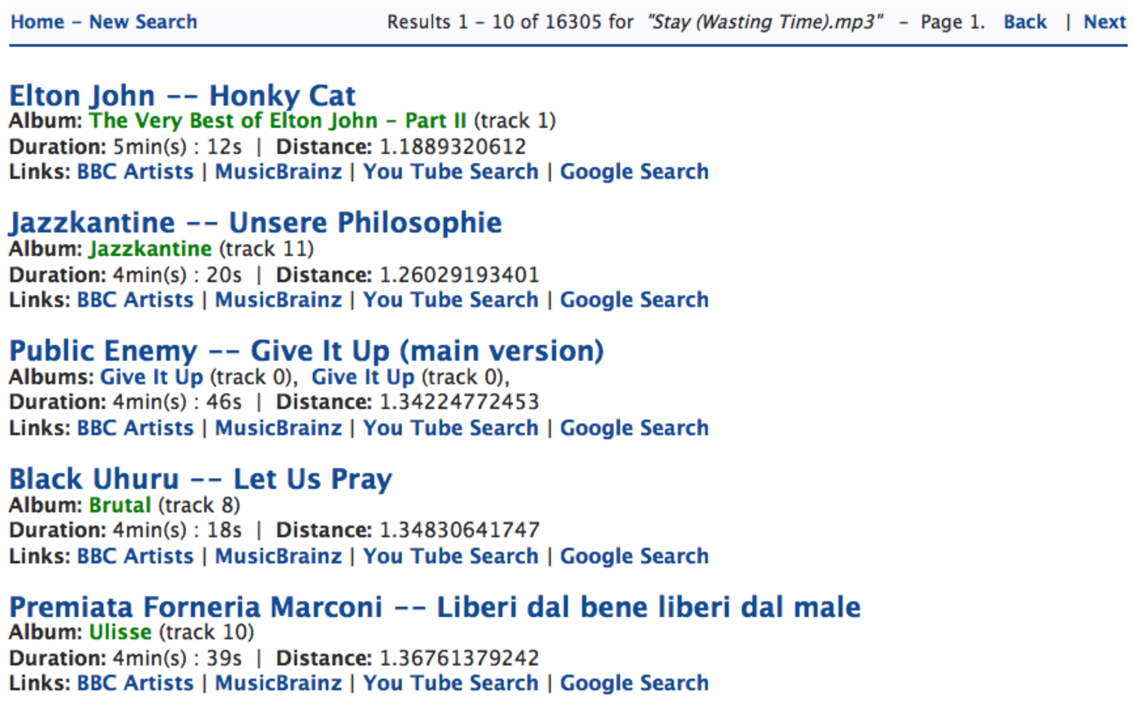
Figure 5.19: SAWA-Recommender audio collection and search interface

In SAWA-recommender, a query is formed by one or more audio files uploaded by the user. It is typically based on single file, however, uploading multiple audio files is also allowed. In the latter case, a small set of songs forms the basis of the query, either by considering similarity to any of the uploaded songs (and ranking the results appropriately), or formulating a single common query by jointly calculating the features of the query songs. The file upload and audio collection builder interfaces are shared between various SAWA applications. The search interface of SAWA-Recommender is shown in Figure 5.19.

Query processing is performed in three steps. First, we extract features from the uploaded

[21]http://www.isophonics.net/content/soundbite
[22]http://www.apple.com/itunes/
[23]http://getsongbird.com/

audio files. The calculated query is matched against the Isophone database holding timbre similarity features and MusicBrainz identifiers associated with each song in this database. For optimised search, the query features are matched against a self-organising similarity model. This model is a modified Self Organising Map (SOM) discussed in Appendix B.1. Each node of this map is associated with a Gaussian distribution as opposed to a simple weight vector as in the original algorithm [Kohonen, 1995]. Finally, a selected group of songs are ranked based on their similarity to the query. The MusicBrainz database is accessed to obtain metadata about songs in the result set, and these are displayed to the user. The metadata consist of basic information such as song title, album title and the main artist's name associated with each song. We also provide direct links to MusicBrainz, as well as Linked Data services such as BBC Music[24] artist pages.


Home – New Search   Results 1 – 10 of 16305 for *"Stay (Wasting Time).mp3"* – Page 1. Back | Next

Elton John -- Honky Cat
Album: The Very Best of Elton John – Part II (track 1)
Duration: 5min(s) : 12s | **Distance:** 1.1889320612
Links: BBC Artists | MusicBrainz | You Tube Search | Google Search

Jazzkantine -- Unsere Philosophie
Album: Jazzkantine (track 11)
Duration: 4min(s) : 20s | **Distance:** 1.26029193401
Links: BBC Artists | MusicBrainz | You Tube Search | Google Search

Public Enemy -- Give It Up (main version)
Albums: Give It Up (track 0), Give It Up (track 0),
Duration: 4min(s) : 46s | **Distance:** 1.34224772453
Links: BBC Artists | MusicBrainz | You Tube Search | Google Search

Black Uhuru -- Let Us Pray
Album: Brutal (track 8)
Duration: 4min(s) : 18s | **Distance:** 1.34830641747
Links: BBC Artists | MusicBrainz | You Tube Search | Google Search

Premiata Forneria Marconi -- Liberi dal bene liberi dal male
Album: Ulisse (track 10)
Duration: 4min(s) : 39s | **Distance:** 1.36761379242
Links: BBC Artists | MusicBrainz | You Tube Search | Google Search

Figure 5.20: SAWA-Recommender results display

Since similarity assessment in this system follows the same principles applied in SoundBite, these results can be seen as content-based recommendations. However, given the size of the database they might be useful for identifying unknown songs or song segments. In a commercial application, the service might be useful for finding an alternative for a song where, for instance, a copyright agreement for its use cannot be obtained.

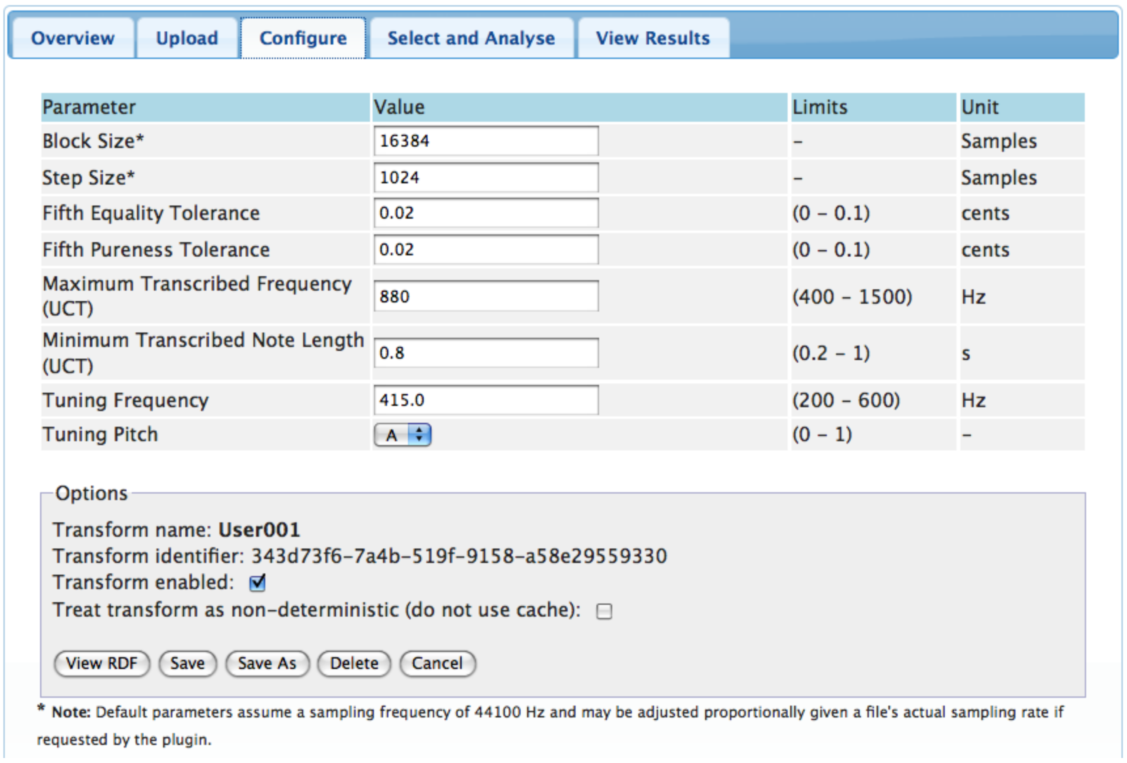
[24]http://www.bbc.co.uk/music/

5.4.3 SAWA TempEst

SAWA-TempEst[25] is a content based instrument tuning recognition system for harpsichord recordings. It uses the temperament estimation algorithm described in [Tidhar et al., 2010b].

In order to expose the algorithm as a Web application, we implemented it as VamPy plugin (see Appendix B.2) which can be executed in the SAWA environment. Using TempEst, one may upload a set of harpsichord recordings for analysis, and receive a detailed description of the temperament used in these recordings. This includes the temperament identified by the classifier, some known properties of the temperament resulting from inference on measurement data, and the measurements themselves on which the classification and inference are based on. The signal processing component of TempEst produces *deviations from equal* temperament. This description format is converted to the *circle of fifths* before inferring additional properties of the temperament such as regularity. These results are returned as RDF documents and expressed using the Temperament Ontology discussed in Section 4.6.

SAWA-TempEst shares the file upload interfaces with other SAWA applications, but provides a different configuration interface for the temperament estimation algorithm shown in Figure 5.21.



Parameter	Value	Limits	Unit
Block Size*	16384	–	Samples
Step Size*	1024	–	Samples
Fifth Equality Tolerance	0.02	(0 – 0.1)	cents
Fifth Pureness Tolerance	0.02	(0 – 0.1)	cents
Maximum Transcribed Frequency (UCT)	880	(400 – 1500)	Hz
Minimum Transcribed Note Length (UCT)	0.8	(0.2 – 1)	s
Tuning Frequency	415.0	(200 – 600)	Hz
Tuning Pitch	A	(0 – 1)	–

Options

Transform name: **User001**
Transform identifier: 343d73f6-7a4b-519f-9158-a58e29559330
Transform enabled: ☑
Treat transform as non-deterministic (do not use cache): ☐

* **Note:** Default parameters assume a sampling frequency of 44100 Hz and may be adjusted proportionally given a file's actual sampling rate if requested by the plugin.

Figure 5.21: SAWA-TempEst configuration interface

[25]SAWA-TempEst: http://isophonics.net/sawa/tempest

In order to find properties of the analysed temperament (i.e. its position in the hierarchy of temperaments), TempEst utilises the Closed World Machine (CWM) [Berners-Lee et al., 2006], a Python based reasoning engine developed as part of the Semantic Web Application Platform (SWAP). The description of related rules can be find in [Tidhar et al., 2010a]. The RDF data returned by the TempEst SAWA application may contain the information detailed in Listing 4.12 and 4.13 of Section 4.6. This provides a description of the recognised temperament as a set of deviations from equal temperament for each note, and the circle of fifths description type derived from this, which is a set of deviations from pure intervals on the circle of fifths given in Pythagorean commas. The output also contains all provenance information regarding the feature extraction and the temperament classification results.

5.4.4 SAWA Experimenter

SAWA Experimenter is an extension of SAWA Feature Extractor with functionality to accept ground truth data in RDF such as the data described in [Mauch et al., 2009], and compare these data with results. The service facilitates MIR experiments, for instance, the onset detection evaluation described in Section 1.5.3. It can be used to automatically determine the most suitable set of parameters to be used for a Vamp plugin and a class of signals. This service is currently in development and will soon be available online.

5.4.5 Hotttabs

Hotttabs [Barthet et al., 2011] is a multimedia guitar tutor Web application which uses video tutorials and guitar tablatures commonly referred as "tabs". It takes a multimodal approach to tuition and tab recommendation, and successfully integrates several data sources by aggregating and internally correlating information in a process whereby the Music Ontology provides the common ground. It also demonstrates the use of SAWA for rapid development of semantic audio applications. The application was built collaboratively[26] based on the idea of Amélie Anglade during the Music Hack Day London and Barcelona 2010 events.

Hotttabs recommends the user a list of songs for practice and rehearsal consisting of the twenty most popular songs at the time of the query. These songs are obtained using the "hotttness" measure of The EchoNest service[27]. To retrieve relevant guitar video tutorials for a selected song, Hotttabs accesses the YouTube video database. The audio track of the YouTube video is extracted and processed using the chord recognition algorithm described in [Mauch and Dixon, 2010a], and the identified chords are displayed synchronously with the video. The application includes a crawler [Macrae and Dixon, 2011] for mining the web for guitar tabs as well as parsing the data before classification and display. To help the user to

[26]For full credits, see http://isophonics.net/hotttabs/about.
[27]http://the.echonest.com/

choose between tabs, they are clustered into three categories based on the size of their chord vocabulary; easy, medium, and difficult (see fig. 5.23 and [Barthet et al., 2011]).

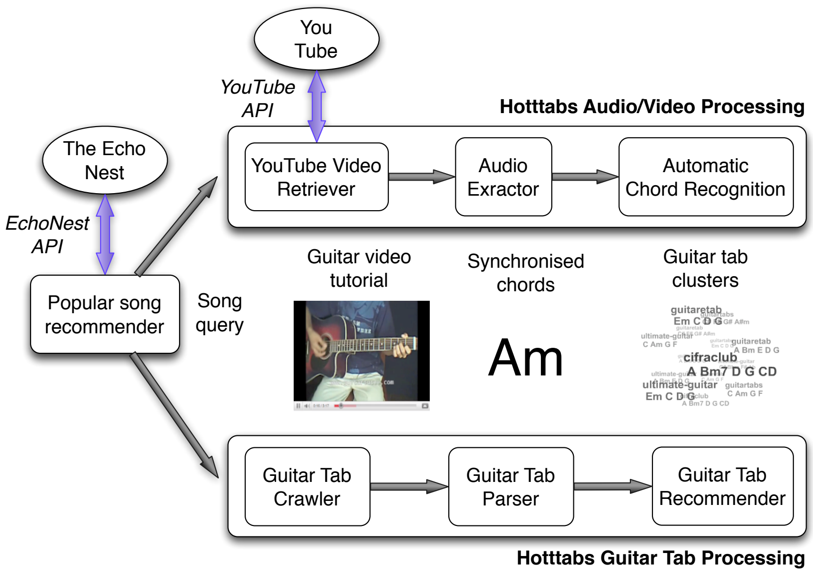


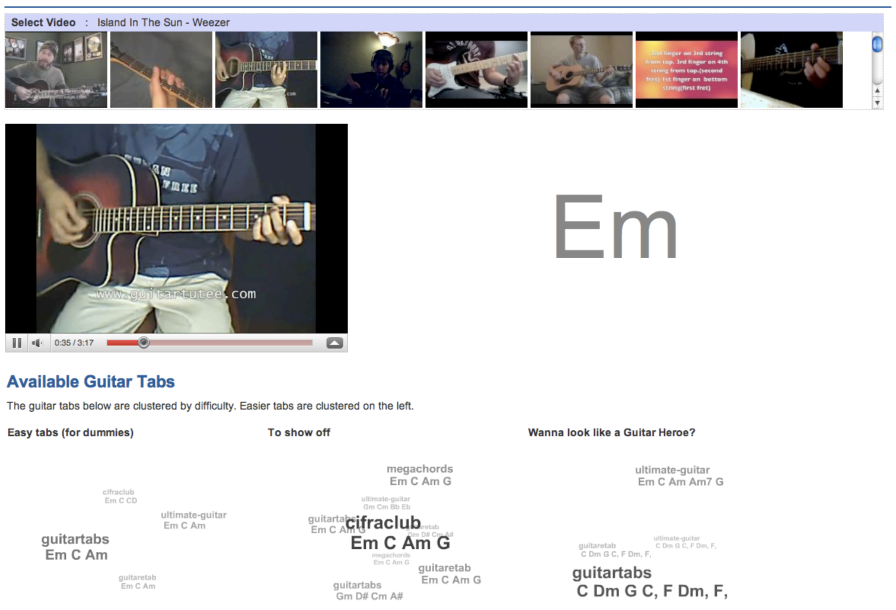
Figure 5.22: Overview of Hotttabs [Barthet et al., 2011]

The Hotttabs application integrates the functionality described so far in a Web-based client-server architecture. The client runs in most popular web browsers and provides an easy to use interface. It allows the user to interact with the application and perform the following actions: *i)* query for popular songs, *ii)* select from the results of the popular song query, and retrieve a list of video tutorials and three sets of tab clusters, *iii)* select from the list of video thumbnails and play the clip in an embedded video player synchronised with automatically extracted chords, and finally *iv)* select a tab from the tab clusters. Similarly to a search engine, this links to the site where it was originally published.

The light weight client uses a combination of standard web technologies (HTML, CSS, JavaScript) and makes use of the JQuery[28] library to dynamically load content from the server via XMLHttp requests. This content includes the list of popular songs and the a list of video thumbnails for a selected song. We developed client-side JavaScript code which interacts with the embedded YouTube player to display chord names next to the video. The chord sequence is requested when the user starts the video, and returned with timing information which is used to synchronise the chords with the video. The tab clusters are displayed using an adapted version of the WP-Cumulus Flash-based tag cloud plugin[29]. This plugin utilises XML data generated on the server side from the results of the tab search and tab clustering algorithm.

[28]http://jquery.com/
[29]http://wordpress.org/extend/plugins/wp-cumulus/

The server side of the Hotttabs application builds on semantic audio and Web technologies. SAWA is used as the basis for Hotttabs, extended with modules to access The Echo Nest and YouTube, and perform additional application specific functionality as shown in Figure 5.22. The communication between the client and server is co-ordinated using the MVC model of SAWA (see Section 5.3.3.1). The domain objects of this model as well as the Hotttabs database are provided by the Music Ontology framework such that corresponding data structures are generated from the ontology specification using MoPy. For instance, information about popular artists and their songs are stored in objects and database entries corresponding to the `mo:Track` and `mo:MusicArtist` concepts. Besides user interaction, the server also performs scheduled queries for popular songs to bootstrap the database. This is necessary since crawling for guitar tabs and the feature extraction process for chord analysis are too computationally expensive to be performed in real-time. This process uses the crawler mentioned previously, as well as the chord extraction algorithm implemented as a Vamp audio analysis plugin which can be loaded by the processing engine of SAWA.



The guitar tabs below are clustered by difficulty. Easier tabs are clustered on the left.

Figure 5.23: Video tutorials in Hotttabs

Hotttabs integrates several data sources, however, due to the lack of machine-processable information in guitar tab sharing sites and the disharmonious nature of Web APIs and their underlying data models, building applications using multiple modalities presents significant challenges. We believe that the use of dynamic query interfaces, for instance, SPARQL end-points relying on shared ontology-based schemata instead of proprietary Web APIs, as well as other principles advocated by the Linked Data community (see Section 2.4.7) would substantially facilitate the making of applications like Hotttabs.

5.5 Summary

In this chapter we described several software components for semantic audio analysis. The RDF-MOP library is designed to facilitate building ontology-based information management systems in desktop audio applications such as audio editors. This library is based on the concept of Meta-object protocol commonly used in dynamic language interpreters. This protocol allows us to associate domain objects within an application with ontological semantics by generating meta-objects from ontology specifications, and binding these objects with the relevant objects in the application domain. Using the RDF-MOP library, we demonstrated the first prototype of the *Semantic Audio Desktop* in Section 5.2.

Extending the Semantic Desktop paradigm, the idea of an intelligent desktop environment in which individuals store digital information which are otherwise stored on a computer by means of conventional techniques, our tool adapts Semantic Web technologies to the audio production environment. This allows the collection and sharing of metadata about the music production process, and serves the basis for an intelligent audio editing environment.

Sonic Annotator Web Application (SAWA) discussed in Section 5.3 is a framework which enables the rapid development of Web-based semantic audio applications. SAWA utilises a number of software components which were developed collaboratively during the OMRAS2 project, and ties these components into a coherent Web-application framework. These components also include unique contributions by the author such as VamPy, a wrapper for the Vamp audio analysis plugin system provides for using Python, a high productivity interpreted language for prototyping and implementing audio feature extraction algorithms. This permits the use of Vamp plugin hosts such as Sonic Visualiser and SAWA to be used with research prototypes written in this language. It also enables writing complex feature extractors which require a dynamically typed functional language, which supports language features such as introspection and labmda calculus. We also devised a method for efficiently indexing and searching a database of music similarity features based on models of musical timbre. This provides an extendible model for indexing similarity features using parametrised Gaussians of MFCC vectors and potentially scales up to a database containing millions of songs. This architecture is utilised in the music recommendation system described in Section 5.4.2. Several other applications of the SAWA framework were presented in Section 5.4. These include a feature extraction tool with a Web-based graphical interface and a SPARQL interface, a tuning recognition system, and a multimedia guitar tutor.

Building the Hotttabs system enabled us to identify some important challenges in creating multimodal applications that use the social Web, for instance, video and guitar tab sharing sites. We argued that building applications like Hotttabs could be made easier if Linked Data principles were routinely followed, and proprietary Web services were replaced by standardised service end-points grounded in shared ontology-based schemata.

Chapter 6

Case Studies and Evaluation

In previous chapters we discussed knowledge representation models for semantic audio applications, and outlined some software technologies to facilitate the use of explicitly conceptualised knowledge models in audio engineering and music information retrieval. In this chapter we evaluate important aspects of our framework. First, we discuss methodological problems, and outline various options available for ontology evaluation. Then, we present the evaluation of the Studio Ontology including data-driven automated tests, as well as case studies which show how our software and ontology frameworks may be used in real-world applications.

6.1 Evaluation methodology

The reasons why we follow a largely *design based* instead of empirical research methodology was briefly discussed in Section 1.6. Our evaluation is adopted to this method. Most components discussed in this work are intended to be part of larger systems, while very few are aimed directly towards end users. Therefore user evaluation requires domain experts who are able to adopt to and assess new technologies and able to devote the time necessary. We developed a set of closely coupled ontologies and tools which may only be assessed in the context of multiple disciplines, for instance, audio engineering, music information retrieval, and ontology engineering. The unavailability of experts knowledgeable in these disciplines and the potential associated cost renders user evaluation unfeasible. Our evaluation methodology is therefore centred around the following techniques:

- data-driven automated testing,
- task-based evaluation examining how components may be utilised,
- objective self-assessment based on domain-specific case studies.

6.2 Ontology evaluation

Ontology is an emerging field in engineering sciences. Although it has long been discussed in philosophy and artificial intelligence, ontologist in these fields are less concerned with the application or comparison of existing ontologies, than the underlying philosophical theories and knowledge representation languages. Therefore rigorous methods for ontology evaluation from an engineering perspective are yet to be developed.

The most common reason for creating an engineering ontology is to provide an explicit conceptualisation or model of a specific domain, or a wider body of knowledge. However, from a purely philosophical point of view, knowledge can not be managed or represented[1]. When knowledge is verbalised, it becomes information. Ontology languages then provide meta-level tools to structure this information, so that it can be shared, or subjected to automated reasoning procedures. The ultimate evaluation would then be to ask the question: *How well an ontology represents someone's innate knowledge, or some shared knowledge amongst members of a community?*

This depends on many implicit factors however, — for instance, how well a body of knowledge can be verbalised in the first place? — rendering most evaluation strategies common in engineering disciplines inappropriate. As pointed out in [Brewster et al., 2004], precision/recall based measures would ideally reflect the amount of knowledge correctly identified with respect to the whole knowledge available in the ontology, and the amount of knowledge correctly identified with respect to all the knowledge that should be captured by the ontology. However, what this means remains rather unclear. This is in part due to the difficulty in quantifying knowledge the same way *false* or *true* outcomes or data items may be enumerated.

6.2.1 Purpose of ontology evaluation

The most comprehensive reviews [Vrandečić, 2009], [Obrst et al., 2007] of ontology evaluation to date, albeit useful resources, do not provide a very clear picture of the field. For instance, evaluation methodologies and concrete techniques are intermingled. While Vrandečić [2009] and Gómez-Pérez [2004] attempt to separate the notions of ontology *validation* from ontology *verification* — that is, the question of whether the *right ontology was built* for a domain (validation), from the question of whether the ontology was built in the *right way* (verification), the techniques used in these aspects of ontology evaluation are not clearly separable. Setting aside a detailed review of methodologies available in the literature, here we provide only a high-level, but more ordered alternative view, which help to choose the right evaluation methodologies and techniques for our case.

An important question to discuss is related to the primary *purpose* of ontology evaluation with regards to a particular ontology-based application. Many existing methods assume a

[1]See the epistemological definitions of knowledge for instance in the Stanford Encyclopedia of Philosophy available at `http://plato.stanford.edu/entries/epistemology/`

so-called *gold-standard*, or the existence of multiple ontologies covering the same domain. Neither of these are available in every situation. The most common use cases for ontology evaluation are:

- **Suitability testing**: Evaluate if a given ontology is appropriate in a particular application, i.e. how adaptable or extensible it is.
- **Choosing an ontology**: Given multiple ontologies covering the same domain, and preferably developed following the same principles, select the best ontology.
- **Scientific and experimental reasons**: Evaluate automatically extracted ontologies against a manually created one, which is usually considered a gold-standard.

In each of the above cases, there are multiple features of an ontology that may be subjected to evaluation. For instance, when examining suitability in the context of an application, we may ask if an ontology is extensible enough, how difficult it is to adopt it to the application, how clear it is, or how well it covers a domain. In choosing an ontology, the first thing one may want to do is some automated comparison using quantifiable features. When evaluating against a gold-standard, one may be interested to compare the richness and structure of a generated ontology with a manually created one.

6.2.2 Evaluating ontology features and ontology design aspects

When dealing with hand-built ontologies, their features are ultimately a result of a number of design decisions, related to the principles discussed in Section 3.1.4. Therefore evaluation can be considered from at least two distinct perspectives: evaluate whether the right decisions were made in the first place, or evaluate the resulting ontology features. This is somewhat related, but not identical to a distinction between validation and verification, providing a more straightforward designation. Here, we outline the features and design aspects of ontologies that are commonly evaluated.

- **Domain coverage**: Richness, granularity, and completeness (also competency) of the ontology; that is, how much of an expert's domain knowledge is acquired.
- **Logical properties**: Consistency, coherence and integrity of an ontology. For instance, is the ontology free of contradictory axioms? Does it describe an unsatisfiable logical theory? Does it comply with meta-level principles (i.e. rigidity or unity discussed in the context of ontoClean [Guarino and Welty, 2002])? Does it allow only for the inferences consistent with its specification?
- **Formal properties**: Documentation and conventions used by the ontology; i.e. Does the ontology have a documentation matching its axioms? Does it follow a uniform naming scheme for its vocabulary terms?

- **Accuracy**: Correctness and precision of the knowledge representation; i.e. How well the ontology represents the real world, how well it complies with the modeller's expertise.
- **Clarity and Conciseness**: Effectiveness of the knowledge representation; i.e. Does the ontology comply with the principles of clarity, minimal ontological commitment and minimal encoding bias? Does it contain redundancies? (Does it specify the weakest theory possible and define only essential terms?) Does it provide a complete definition of its terms? Does it make representation choices for notational convenience?
- **Extensibility, Reusability, Adaptability**: These features are mainly related to the modularity and extensibility principles; i.e. Does the ontology have the right scope and granularity of knowledge representation? Does it anticipate uses outside of its primary domain? Can it be extended monotonically? Does it feature a set of harmonised modules, separating domain specific and domain independent components?

From a substantially different point of view, we may want to evaluate decisions related to design choices in the development process of a *particular* ontology, rather than evaluating how its features adhere to general design principles. The following list of design aspects is based on [Vrandečić, 2009], although we argue some of the definitions given there.

- **Vocabulary**: The choice of terms (concepts, properties and individuals) included in the ontology, and choices related to their representation, e.g. notational conventions.
- **Syntax**: The serialisation syntax used for encoding the ontology. Some syntaxes support a wide range of different knowledge representations, others may be equivalent, or transformable one or both ways. For instance, all RDX/XML may be written in N3, but not all N3 can be written in RDF/XML. Documentation given in syntax specific formats however are generally not transformable.
- **Structure**: The concrete structure of the ontology given by the underlying RDF graph. The same semantics may be encoded using different structures.
- **Semantics**: The actual set of logical models described by the ontology. For instance, a specific cardinality constrain may be expressed using different language features, resulting in different graph structures, but all these structures express the same constraints on the resulting data.
- **Representation**: Related to both structure and semantics, representation may be analysed to find differences between the formal specification and the shared conceptualisation of an ontology. For instance, we can compare graph-based measures between an ontology and a simplified, but semantically equivalent version of it.

- **Context**: The notion of context captures many of the design principles and requirements mentioned previously. For instance, the suitability of the ontology in an application with regards to its design requirements, how well it can be used for representing information in a concrete data source, or how suitable it is for answering specific queries, often formalised using *competency questions*.

6.2.3 Methodologies and techniques for ontology evaluation

Each of the ontology features and design choices discussed above may be evaluated from several different perspectives, using a variety of *methodologies* and concrete *techniques*. In fact, it is useful to discuss these aspects of evaluation separately, since different methodologies are not equally suitable for evaluating independent aspects and features of an ontology. Methodologies in general may fulfil distinct evaluation goals, and may be used at various stages of the ontology engineering process. Meanwhile, concrete evaluation techniques can be used in the context of several different evaluation methodologies. In Table 6.1, we summarise some notable categories of evaluation methodologies and the techniques related to each category. We also indicate the ontology design aspects and features that may be evaluated by each method.

Rule-based methodologies are primarily used for evaluating the logical properties of ontologies, such as consistency and integrity. OntoClean [Guarino and Welty, 2002] is the most widely used technique for this purpose. The main goal of OntolClean is to expose misuses of knowledge representation options like the subsumption relation. It specifies certain meta-properties (essence, rigidity, identity, unity, dependence) that characterise the intended meaning of classes and relationships. The user then labels ontology terms using these properties, and re-examines the ontology based on the rules specified by ontoClean. For instance, an anti-rigid term, that is not essential for all instances, cannot subsume a rigid term, that is essential in all circumstances. The principles related to identity and unity can be used to discover instantiation problems (i.e. help to decide if an entity should be a sub-class or an instance of a class), or common confusions of different taxonomic relations, such as hyponomy (is-a) and meronomy (part-of). Rule-based techniques can also be used, for instance, to discover if an ontology specifies unsatisfiable classes (i.e. a subclass of two disjoint classes), or to find circular class definitions. The ODEval tool [Corcho et al., 2004] can be used for this purpose. However, the techniques mentioned so far are more related to ontology design, rather than evaluation. With relatively small domain ontologies, they can be easily performed by manual inspection during the design process, as we did in case of the Studio Ontology.

Metrics-based methods focus on quantifiable features, for example, graph-based structural parameters of ontologies. Most of these features however express relatively simple aspects of ontologies, therefore they are most commonly used in ontology selection as a preliminary inspection step, or to measure the contribution of a module to a framework of ontologies. OntoQA [Tartir et al., 2005] for instance, specifies metrics such as *relationship richness* or *inheritance richness* (see Section 6.3.1.1 where we utilised these measures) which may be directly extracted from the RDF graph representing the ontology schema. It also specifies more complex features which can be used if a sufficiently populated knowledge base is available, using the examined ontology. oQual [Gangemi et al., 2006] also specifies similar graph based measures, but it goes beyond the structural level by prescribing a set of functional metrics, that may be used in the context of expert user evaluation, or Data-driven evaluation using Natural Language Processing (NLP) techniques. In fact, more complex metrics-based methods are used as part of data-driven, query-driven or cost-based evaluation.

Evolution-based methods attempt to characterise ontologies based on how they change over time. For instance, Haase et al [2005b] describe a technique which detects inconsistencies in evolving ontologies, and repairs them across different versions by eliminating the statements that cause inconsistency. An ontology can also be evaluated against the presence of certain patterns that are typical in an application. These patterns essentially act as unit tests [Vrandečić and Gangemi, 2006] for ontologies. For instance, we can test if certain axioms can or cannot be derived from the ontology. In the context of evolving or dynamic ontologies, we can test certain assumptions with regards to the ontology. These techniques are more related to ontology engineering (and change engineering) rather than final evaluation at the end of an initial development cycle. They are most beneficial with frequently changing and very large ontologies.

Cost-based evaluation tries to characterise either the cost of application within an organisation, or the performance in terms of the cost of the errors in information extraction, or other NLP-based techniques. The former is most useful in ontology selection, to evaluate formal properties such as the quality of documentation, or the ease (or difficulty) of use related to the clarity and adaptability of the ontology. The latter technique is substantially different and can also be seen as a form of data-driven evaluation. It defines a relevant cost model for an application, using some expected prior target statistics. A cost would typically be associated with a miss or a spurious answer within each category of result, while the expected costs of error would typically be based on probability, estimated using an expert annotated test corpus and manual mapping. See [Paslaru et al., 2006] for examples of cost-based evaluation.

Data-driven ontology evaluation was introduced in [Brewster et al., 2004]. It focuses on the comparison of ontologies with a text corpus from a relevant domain. Brewster introduces a basic method for measuring lexical coverage, using a vector-space model of terms built from a corpus and an ontology. This provides a measure of the agreement between the ontology and the corpus (representing domain knowledge), similarly to how documents are ranked relative to a query in a search engine. A more complex *'tennis measure'* is based on a method consisting of feature extraction (identification of key terms), query expansion (using for instance hypernyms from WordNet), and mapping the key terms to the ontology. This method provides an *ontology fit* seen as a structural fitness measure. We utilise this techniques in our evaluation, and provide more details in Section 6.3.1.2. Other data-driven techniques primarily focus on human assessment instead of NLP-based automated or semi-automated evaluation. Semantic agreement, can be based on the measurement of human agreement on classification tasks, for instance, the problem of classifying instances using an ontology, or mapping concepts to candidate classes in one or more ontologies [Obrst et al., 2007]. This technique however is highly influenced by how well humans are trained in a set of guidelines for labelling examples in terms of ontological categories, as well as the quality of the guidelines themselves.

Task-based methodologies focus on evaluating the suitability of an ontology in a given application. Similarly to metrics-based evaluation, task-based evaluation is an *umbrella term*, where having a set of pre-defined *requirements for a task* is the common element across methodologies. Task-based evaluation may offer a measure of practical aspects, such as the human ability to formulate queries using an ontology, or the accuracy of responses provided by the system's inferential component [Obrst et al., 2007]. It can take the form of data-driven evaluation using a corpus, an evaluate the agreement between typical features of the corpus (such as commonly occurring scenarios) and the ontology involving human assessment, or some metrics discussed earlier. Task-based evaluation may also be based on the applicability of an ontology in a given tool or system, for instance, by examining whether we can fulfil formalised communication requirements, or represent data stored in a legacy relational database. Generally, these methods evaluate whether an ontology can fulfil requirements in specific use cases.

Query-driven methodologies focus on query answering tasks, and may be seen as a combination of data-driven and task-bask based evaluation. For instance, the *'ontology fit'* described in [Raimond, 2008] measures how well certain features extracted from a set of verbalised user queries can be represented by an ontology, using techniques mainly based on the tennis measure describes in the context of data-driven evaluation. Other methods are based on formulating hypothetical queries given one or more ontologies, and a data source that can be mapped to these ontologies. The evaluation is then based on how difficult it is to write actual queries (e.g. using the SPARQL language) to get relevant answers. We may also evaluate based on the complexity of required queries, or the cost of query execution, in order to get the same answers, using different ontologies. In this context, the best ontology is the one which supports the most queries, which are also easy to write, and cheap to evaluate. In [Kolozali et al., 2011], we used similar query-driven techniques to evaluate different musical instrument ontologies.

Evaluation Methodology	**Evaluation Technique**	**Ontology Design Aspect**	**Ontology Feature**
Rule-based (logical)	consistency checking OntoClean[1] ODEval[2]	semantics structure representation	logical properties accuracy conciseness
Metrics-based	OntoMetric[3] OntoQA[4] oQual[5] ontology fit[6]	vocabulary structure representation context	domain coverage accuracy
Evolution-based	application change evaluation[8] unit testing[9]	semantics vocabulary representation context	adaptability extensibility reusability logical properties
Cost-based	cost of error cost of application Ontocom[10]	semantics representation context	formal properties domain coverage clarity conciseness
Data-driven	lexical coverage (NLP)[7] tennis measure[6] human assessment	vocabulary semantics context	domain coverage accuracy clarity
Task-based	application human assessment	vocabulary syntax semantics context	formal properties adaptability domain coverage extensibility
Query-driven (task-based)	ontology fit[6] query formulation query complexity human assessment	vocabulary semantics context	domain coverage accuracy logical properties

[1] OntoClean: see [Guarino and Welty, 2002].
[2] ODEval: see [Corcho et al., 2004].
[3] OntoMetric: see [Lozano-Tello and Gómez-Pérez, 2004].
[4] OntoQA: see [Tartir et al., 2005].
[5] oQual: see [Gangemi et al., 2006].
[6] tennis measure and ontology fit: see [Brewster et al., 2004] and [Raimond, 2008].
[7] Natural Language Processing (NLP): see for instance [Brewster et al., 2004].
[8] Change evaluation: see [Haase et al., 2005b].
[9] Unit testing: see [Vrandečić and Gangemi, 2006], [Vrandečić, 2009].
[10] Ontocom: see [Paslaru et al., 2006].

Table 6.1: Summary of ontology evaluation methodologies, applicable techniques and the design choices and ontology features these techniques may evaluate

The assignment of features and design aspects to techniques and methodologies in Table 6.1 is based on common uses in the literature, yet it may be arguable and provides only a coarse indication of what method to use for testing certain aspects of an ontology.

Before choosing a particular set of techniques, we also face the more general decision of whether we need a primarily qualitative or a quantitative (metrics-based) method. Quantitative techniques may tell us about the structure or richness of the ontology, but they are typically not sufficient for judging the overall quality of an ontology. Qualitative techniques, for instance, task-based evaluation with human assessment, or testing an ontology in a description or query answering task, tell us more about how well an ontology may perform in a certain application, and in a certain environment, however this requires human level intelligence. Choosing the right users for qualitative evaluation could be a challenging task.

6.3 Evaluation of the Studio Ontology framework

In the light of the above discussion, we argue for the need to perform both quantitative and qualitative evaluation of the Studio Ontology. Due to the facts that this ontology is the first of its kind, and expert users in this domain are unavailable, some methods, such as evolution and cost-based evaluation are not applicable, while others, such as rule-based evaluation were applied during development. Therefore we opt primarily for metrics-based as well as data-driven quantitative evaluation, followed by task-based qualitative evaluation with self-assessment.

6.3.1 Quantitative evaluation

First, we perform quantitative evaluation to measure the contribution of the Studio Ontology to an ontology framework of music production, consisting of the Music Ontology and its core components, and the Studio Ontology frameworks. This evaluation includes simple structural features computed from ontology schemata. Then we measure lexical coverage and conceptual ontology fit using a relevant text corpus.

6.3.1.1 Structural schema metrics

The purpose of using structural ontology metrics can be two-fold. On one hand, it is used to measure the difference between certain aspects of ontologies designed to cover the same domain. This is the most common use, for instance, in ontology selection. On the other hand, simple quantifiable features of ontology schema can provide an indication of how a set of modules contribute to a framework of ontologies.

The Studio Ontology framework was designed to extend the Music Ontology framework. In order to measure its contribution, we rely on three schema metrics defined in the ontoQA metric-based ontology quality analysis system [Tartir et al., 2005]. We updated the terminology to better suit OWL ontologies, but the method of computation is essentially the same. We extract the following features from ontologies constituting both frameworks:

Inheritance richness (IR) is defined in Equation 6.1 as the average number of subclasses per class, where $|SC| = \sum_{C_i \in C} |H^C(C_l, C_i)|$, the sum of classes defined as a subclass for each class C_i in the concept taxonomy $H^C \subseteq C \times C$ of the ontology.

$$IR = \frac{|SC|}{|C|} \tag{6.1}$$

Relationship richness (RR) is defined in Equation 6.2 as the ratio of the number of relationships defined between classes, divided by the number of inheritance relationships plus the number of relationships. In OWL terms, the number of relationships is equivalent to the number of object properties ($|OP|$), where OP is a set of relations and the function **op**: $OP \rightarrow C \times C$ relates concepts non-taxonomically. The number of inheritance relationships is equivalent to the sum of classes defined as a subclass for each class in the concept taxonomy ($|SC|$), defined as before.

$$RR = \frac{|OP|}{|SC| + |OP|} \tag{6.2}$$

Attribute richness (AR) is defined in Equation 6.3 as the average number of attributes per class. It is computed as the number of data type properties ($|DP|$) for all classes divided by the number of classes, where DP is a set of relations and the function **dp**: $DP \rightarrow C$ relates concepts with literal values.

$$AR = \frac{|DP|}{|C|} \tag{6.3}$$

The results of extracting features from three groups of ontology modules is shown in Table 6.2. The first group (FFDET) consists of the core parts of the Music Ontology (FRBR, FOAF, Dublin Core, Event and Timeline ontologies), the second group (MO) represents the Music Ontology framework without extensions, while the third group (MOS) includes the Music and Studio Ontology frameworks. The values stand for the number of classes $|C|$, number of inheritance relationships $|SC|$, the number of object property relationships $|OP|$, the number of data type properties $|DP|$, and the three metrics defined above.

Ontology	$\|C\|$	$\|SC\|$	$\|OP\|$	$\|DP\|$	IR	RR	AR
FFDET	74	76	183	68	1.0270	0.7066	0.9189
MO	133	201	412	115	1.5113	0.6721	0.8647
MOS	369	745	542	201	2.0190	0.4211	0.5447

Table 6.2: Schema-based structural ontology metrics using ontoQA

The calculations take the logical ontology model into account. The values $|SC|, |OP|, |DP|$ therefore include not only the number of relationships directly asserted by the ontology schema, but inferred relations resulting from the transitivity of *is-a* relations for example, and the inheritance of object property and attribute relations. In case a property is not fully defined in an ontology, i.e. no domain and range are declared, we only considered it as one relation to avoid distorting the measures when the intended use of a property is not clear.

The features IR, RR and AR quantify the following aspects of our ontology modules: The increase in inheritance richness indicates that we move from small domain ontologies, defining few classes with a rich set of properties, towards ontologies representing a wider range of general knowledge. For instance, the Event Ontology defines only three taxonomically unrelated classes and seven properties, while the Music and Studio ontology frameworks add a large number of classes subsuming core concepts.

The decrease in relationship and attribute richness is also related to the above change in the nature of the measured ontologies. We observe however a rather sharp decrease in both RR and AR when our Studio Ontology framework is considered (MOS). This is due to the fact that it defines a large taxonomy of music production tools with a similar set of properties. Only microphones and audio mixers are currently covered in detail by providing a rich set of properties in two domain specific extensions. Other tools, such as recording devices or audio effects may be included in the workflow model and description, but their class specific attributes and relationships are not included in the ontology. These measures therefore indicate an area where our ontology framework can be improved.

In addition to schema-based features, OntoQA defines a set of knowledge base (KB) metrics. Class Richness for instance measures how instances are distributed across classes, while Class Connectivity indicates what classes are central in the ontology based on the instance relationship graph. These metrics however require a sufficiently large and correctly populated knowledge base, which is not available for us. A possible option is populating a knowledge base from a text corpus automatically, using natural language processing (NLP) and ontology learning methodologies. However state of the art techniques [Cimiano et al., 2009] are not guaranteed to produce a knowledge base, precise enough to compute ontology metrics reliably. Therefore we consider the use of KB metrics in future work only. In the next section, we apply two corpus-driven evaluation techniques which do not require sophisticated natural language understanding.

6.3.1.2 Data-driven evaluation

The schema based structural ontology metrics discussed in the previous section do not provide information on how well an ontology represents a specific domain, and how well it supports describing information in this domain. In the following sections, we perform data-driven evaluation of the Studio Ontology, using a text corpus specific to music production. We first describe the data set, then we measure the lexical coverage of of this data set given different combinations of ontology modules. Finally, we measure how well some characteristic features extracted from this data set can be expressed by our ontologies.

6.3.1.3 Text corpus

For our data-driven evaluation, we consider extracting music production specific text from the online archives of *Sound On Sound*, a periodical popular with music industry professionals[2]. We downloaded over 8000 articles comprising issues of the magazine between January 1994 and March 2011. We processed the files using the BeautifulSoup[3] HTML parser library, in order to extract plain text from non well-formed HTML. This process produced 7757 text files, each containing an article with some metadata, such as the title and publication date of the article, and the column it originally appeared in.

Given the large size of this corpus and the varying scope of the articles, we partitioned the data set according to the following principles: *i)* We split the data by date into two parts. Part *A* contains 3686 articles between 1994-2001, while part *B* contains 4071 articles form January 2002 to March 2011. *ii)* Articles were then grouped by column, resulting in the following groups in each part of the data set:

- **Opinion**: editorials, relationships between artists, producers and engineers
- **Reviews**: predominantly reviews of music production tools
- **Technique**: descriptions of audio production and audio engineering techniques
- **All others**: articles related to music business and other small columns

In both parts, the first three groups cover approx. 95% of the total content. A typical article from the *Opinion* column includes for instance:

> *Wayne Bennett & Speech Debelle*: "It took rapper Speech Debelle years to find a producer who could bring to life the sounds in her head. The answer, as Wayne Bennett discovered, was to record her music as if it were a folk album."

[2]Sound On Sound archive: http://www.soundonsound.com/AllIssues.php
[3]BeautifulSoup: http://www.crummy.com/software/BeautifulSoup/

The *Reviews* column typically contains articles such as:

> *SSL X-Patch Studio Router*: "Creating complex routing chains in a hybrid DAW/outboard setup presents its own set of problems to which Solid State Logic offer this solution..."

A representative example from the *Technique* column would be:

> *Secrets Of The Mix Engineers: Veronica Ferraro*: "Veronica Ferraro mixed 'When Love Takes Over' at her Super Sonic Scale studio near Paris, in which pride of place goes to her unique 56-channel Amek DMS desk, which contains Neve mic preamps, EQ, compression and gates. The presence of the desk is an outcome of her training in the late '80s at one of France's major studios, Studios Ferber."

Finally the articles are pre-processed for the evaluation techniques discussed in the following sections. Each article is tokenised, common stop words[4] (e.g. articles, prepositions) are removed and the tokens are stemmed using the Porter Stemmer algorithm[5] for regularising morphological and inflexional endings of English words. Stop words are either insignificant or too frequent, therefore they are typically removed from search queries and database indexes as well. Table 6.3 shows the 50 most frequently occurring stemmed terms in each part and in each group of the data set.

6.3.1.4 Lexical coverage

Similarly to our structural evaluation, here, we consider the Studio Ontology framework as an extension of the Music Ontology, and aim to measure its contribution by comparing the music production specific corpus discussed above, and different sets of ontology modules. In order to measure the lexical coverage of different ontology modules given our data set, we utilise a vector space representation [Salton et al., 1975] with term frequency - inverse document frequency (TF-IDF) weighting, and use the cosine distance to determine the similarity between the corpus and the ontologies. This method was first proposed in [Brewster et al., 2004] in ontology evaluation. It was also applied in [Raimond, 2008] for evaluating the Music Ontology.

We first create a vector space representation of each group in the data set, and each group of ontology modules, by extracting literal terms corresponding to `rdfs:label` and `rdfs:comment` (the inclusion of comments is seen as a form of query expansion) predicates in the schemata. This forms a combined document set D. In this model, each document $d \in D$ is represented by a vector, such that each vector dimension corresponds to a unique term $t \in d \in D$, and the values are the TF-IDF weights $w_{t,d} = tf_{t,d} \times idf_t$, where $tf_{t,d}$ is the number of times term t occurs in document d, divided by the number of words in the document, and idf_t is given by Equation 6.4.

[4] http://jmlr.csail.mit.edu/papers/volume5/lewis04a/a11-smart-stop-list/english.stop
[5] http://tartarus.org/~martin/PorterStemmer/

Part A								Part B							
Opinion		Reviews		Technique		All others		Opinion		Reviews		Technique		All others	
record	824	sound	1843	record	1000	make	109	record	707	sound	1460	record	921	make	186
music	760	make	1673	work	980	work	97	music	621	make	1306	audio	868	sound	169
work	684	set	1666	sound	922	time	97	work	524	set	1301	make	836	music	145
sound	651	work	1637	music	919	record	94	sound	519	work	1281	work	835	time	144
produc	588	control	1604	time	916	sound	93	produc	436	includ	1276	sound	782	work	134
time	584	record	1571	make	893	music	91	make	427	control	1261	time	738	record	131
make	553	includ	1492	audio	865	good	85	time	421	audio	1260	track	735	includ	123
studio	549	time	1488	set	844	end	82	studio	409	record	1233	set	730	great	117
track	533	provid	1474	back	747	produc	80	track	382	offer	1177	music	693	audio	110
thing	505	good	1458	start	724	start	80	signal	354	rang	1082	creat	602	offer	105
set	498	output	1444	number	718	find	79	thing	351	time	1080	includ	585	made	104
plai	486	power	1435	find	717	thing	79	mix	343	provid	1061	signal	572	set	102
peopl	480	rang	1427	signal	715	give	79	microphon	325	power	1049	mix	570	studio	102
digit	478	effect	1416	control	709	plai	79	set	318	design	1038	version	567	find	102
good	471	audio	1401	part	708	back	78	plai	318	output	1027	back	558	good	100
year	470	signal	1389	includ	699	set	78	devic	316	signal	1010	part	545	back	100
audio	470	level	1347	gener	698	put	77	audio	313	featur	1003	control	544	thing	99
part	469	offer	1337	midi	694	point	76	good	305	effect	992	start	539	product	99
back	465	input	1317	track	693	part	74	back	296	level	949	perform	528	plai	99
start	461	featur	1313	good	688	gener	72	lot	295	version	931	effect	525	part	98
mix	457	design	1305	system	688	provid	70	engin	286	music	928	plai	505	version	97
put	449	switch	1287	produc	687	mean	69	made	280	good	924	process	503	start	92
made	444	select	1287	thing	672	number	69	digit	280	input	921	devic	501	produc	90
lot	443	stereo	1246	provid	671	track	68	part	279	select	894	produc	489	requir	90
includ	428	user	1245	plai	668	lot	67	year	278	switch	873	offer	473	creat	89
signal	424	gener	1242	creat	663	peopl	67	start	275	instrument	859	featur	471	effect	89
end	423	number	1231	effect	661	turn	65	put	273	back	859	number	461	put	88
find	413	system	1230	version	645	includ	65	peopl	273	option	857	thing	458	track	87
call	410	produc	1224	digit	644	effect	65	listen	263	main	857	good	452	live	87
microphon	407	music	1200	devic	638	creat	64	includ	262	number	844	select	450	power	87
product	407	back	1187	run	636	control	63	product	262	mix	840	show	446	top	85
idea	406	main	1172	chang	628	import	62	instrument	253	addit	830	instrument	446	number	84
run	397	find	1168	point	625	take	62	end	252	gener	829	find	445	end	83
bit	394	midi	1154	end	617	call	62	album	243	produc	819	gener	441	featur	83
devic	394	plai	1145	perform	610	made	61	room	241	user	819	end	439	give	83
point	392	perform	1144	result	598	differ	61	perform	241	review	816	file	438	lot	82
interest	389	digit	1133	note	597	amount	61	song	239	stereo	815	note	431	bit	81
effect	385	mix	1121	softwar	594	problem	61	find	238	perform	801	run	430	perform	80
live	383	oper	1105	lot	592	mix	60	band	238	microphon	798	main	426	system	79
sampl	382	singl	1098	hard	581	singl	60	great	237	sampl	792	studio	410	bui	78
number	380	found	1096	process	580	run	60	guitar	233	studio	789	open	410	point	78
project	379	option	1094	requir	580	small	60	bit	231	singl	757	chang	409	option	77
long	377	channel	1090	type	578	result	60	control	231	creat	756	level	404	cost	77
move	376	button	1084	problem	577	level	59	live	230	end	749	move	402	provid	77
great	376	qualiti	1081	simpli	576	complet	59	releas	226	find	744	digit	399	select	77
process	375	addit	1069	output	575	type	59	gener	225	combin	743	add	398	small	77
chang	374	end	1069	give	563	studio	58	drum	225	found	740	midi	397	design	76
dai	373	start	1067	move	563	big	58	creat	225	digit	737	singl	397	so	76
give	372	studio	1064	offer	563	offer	58	process	224	midi	732	origin	396	gener	75
control	372	point	1061	mix	561	bit	57	point	223	softwar	732	lot	396	mix	73

Table 6.3: Top 50 words in each part of the data set

$$\mathrm{idf_t} = \log \frac{|D|}{1 + |\{d' \in D | t' \in d'\}|} \tag{6.4}$$

In the equation above, $|D|$ is the total number of documents, and $|\{d' \in D|t' \in d'\}|$ is the number of documents that contains the term t. Each ontology is then compared with a group in the corpus using the cosine distance given in Equation 6.5. This provides a measure of similarity that is invariant of document lengths, by taking only the cosine of the angle θ between document vectors X and Y into account. The result ranges between 0 and 1, as the TF-IDF weights are always positive, thus θ is bound at 90 degrees.

$$d_{Cos}(X,Y) = \cos(\theta_{X,Y}) = \frac{X \cdot Y}{||X|| \times ||Y||} \tag{6.5}$$

Part A								
Ontology	Opinion		Reviews		Technique		All Others	
	similarity	change	similarity	change	similarity	change	similarity	change
FFDET	0.1020		0.1152		0.1252		0.1048	
MO	0.1501	47.15%	0.1497	29.94%	0.1424	13.73%	0.1892	**80.53%**
MOS	0.1634	8.86%	0.1990	32.93%	0.2114	**48.45%**	0.2012	6.34%

Part B								
Ontology	Opinion		Reviews		Technique		All Others	
	similarity	change	similarity	change	similarity	change	similarity	change
FFDET	0.1137		0.0819		0.1102		0.1207	
MO	0.1540	35.44%	0.1230	50.18%	0.1205	9.34%	0.1764	46.14%
MOS	0.1747	13.44%	0.1879	**52.76%**	0.2135	**77.17%**	0.1848	4.76%

Table 6.4: Cosine similarity of ontology modules and different groups in the data set, and relative change in similarity when adding the Music Ontology (MO) and the Studio Ontology (MOS) frameworks to the combined set of modules compared.

Table 6.4 shows the results of computing the inter-document similarity between the same set of combined ontology documents discussed in Section 6.3.1.1, and different groups in our data set. The benefit of using the Studio Ontology (MOS group) is most apparent in the technology orientated columns of the magazine: Reviews and Technique. The Music Ontology framework alone covers many aspects of other types articles. In the Opinion column for instance, while it also contains technical articles, information about artists, albums, engineers, or music venues dominate.

It is interesting to observe that the improvement as a result of using the Studio Ontology is greater in part B of the data set, which contains articles between 2002-2011. We speculate that this is due to the dominance of computer-based production in the '00s compared to the previous decade, which caused an increase in both the size and use of a technical vocabulary in music production. This also explains the slight decrease in similarities overall. However, it can also be interpreted as an indication of the bias towards traditional music production procedures our ontology framework currently exhibits.

Although the measures computed so far provide an indication of how the Studio Ontology contributes towards a system for describing studio production, they are by no means sufficient to evaluate our framework against a data set or other ontologies. In the next section, we address this issue by applying similar ontology fit measures proposed in [Brewster et al., 2004] or [Raimond, 2008].

6.3.1.5 Structural and conceptual ontology fit

Simple lexical overlap based evaluation has two main limitations: *i)* it does not take the logical model of the ontology into account, *ii)* it requires direct correspondence between terms, i.e. hypernyms and synonyms are not considered similar, and the context of the use of terms is ignored. In order to overcome these limitations, we need to allow for some subjectivity in a more sophisticated evaluation architecture.

An alternative corpus-driven method is proposed in [Brewster et al., 2004], with the identification of key terms using probabilistic Latent Semantic Analysis [Hofmann, 1999], term clustering, query expansion, and finally measuring a structural *ontology fit*, by mapping the resulting term clusters to an ontology. Such a method however is largely dependent on the principles used in term clustering. For instance, it may be useful to evaluate a concept taxonomy, but there is no guarantee in general, that terms close in a corpus should also be close in a well-designed ontology. In a more flexible approach, we follow the semi-automatic evaluation technique proposed in [Raimond, 2008], where term clusters are mapped to an ontology manually. In order for this to be feasible, we need to obtain a representative reduction of the corpus.

6.3.1.6 Topic modelling

Observing the consistency of commonly occurring terms in our data set (see Table 6.3), we hypothesise that topic modelling [Blei, 2011] would work well for extracting important features (topics) from the articles. Topic modelling does not require prior annotations or labelling of the documents. The features emerge from the analysis of the original texts. We can then manually map these features to the ontologies, and measure the agreement between the mapping and the reduced representation of the corpus.

The fundamental idea behind topic modelling is that a document collection covers a fixed number of topics, and these topics contribute to documents in different proportions. For example, an article describing a recording session may contain terms related to *artists* and *sound engineers*, *microphones*, *instruments* and *recording devices*. Therefore terms related to these topics are more likely to occur in such articles. The overall probability of a term will depend on the topic distribution and the probability of a word within a particular topic. One of the simplest topic models, Latent Dirichlet Allocation (LDA) [Blei et al., 2003] tries to capture this intuition in a statistical modelling framework.

In LDA, each document is assumed to exhibit a distribution over a set of pre-defined topics, where each topic is seen as a distribution over a fixed vocabulary of terms, i.e. each term in a topic has an associated probability. Given this assumption, a document may be generated by first picking a per-document topic distribution, which is a Dirichlet distribution, and then for each word in the document randomly choosing a topic first, and then choosing the word from that topic. This generative process is characterised by the joint probability distribution shown in Equation 6.6.

The latent topic structure in LDA consists of the term distributions $\beta_{1:K}$ for K topics, as well as the topic proportions $\theta_{1:D}$ and word to topic assignments $z_{1:D}$ for D documents, such that $z_{d,n}$ corresponds to the assignment of the *n-th* word in document d to topic β_k. Given some observed words $w_{1:D}$ in D documents, (such that $w_{d,n}$ is the *n-th* word in document d), the problem of LDA is computing the conditional (posterior) probability of the hidden variables corresponding to the topic structure, see Equation 6.7. The probability of each word in a document is then given by Equation 6.8, where the *n-th* word in document d is associated with a probability that depends on the prevalence of the word in topic k, $P(w_{d,n}|z_{d,n} = k)$, and the probability of that topic within the document $P(z_{d,n} = k)$, summed over all K topics.

$$P(\beta_{1:K}, \theta_{1:D}, z_{1:D}, w_{1:D}) = \prod_{k=1}^{K} P(\beta_k) \prod_{d=1}^{D} P(\theta_d) \left(\prod_{n=1}^{N} P(z_{d,n}|\theta_d) P(w_{d,n}|\beta_{1:K}, z_{d,n}) \right) \tag{6.6}$$

$$P(\beta_{1:K}, \theta_{1:D}, z_{1:D}|w_{1:D}) = \frac{P(\beta_{1:K}, \theta_{1:D}, z_{1:D}, w_{1:D})}{P(w_{1:D})} \tag{6.7}$$

$$P(w_{d,n}) = \sum_{k=1}^{K} P(w_{d,n}|z_{d,n} = k) P(z_{d,n} = k) \tag{6.8}$$

To estimate the topic structure, it is necessary to compute the posterior probability (i.e. the conditional distribution shown in Equation 6.7), given the sequence of words $w_{1:D}$ in D documents. Unfortunately, computing $P(w_{1:D})$ would require summing over all possible ways of assigning each observed word to one of the topics (i.e. the marginal probability of seeing the observed corpus under any topic model [Blei, 2011]), which is exponential in complexity.

To solve this problem, topic modelling algorithms either approximate this conditional distribution using, for instance, Gibbs sampling [Griffiths and Steyvers, 2004], or use optimisation, and try to find the latent topic structure by minimising the Kullback-Leibler divergence between the posterior and a member of an assumed family of parameterised distributions over the hidden structure. In probabilistic modelling, this alternative method is called *variational* Bayesian inference [Wainwright and Jordan, 2008]. A comparison of these two main techniques is available in [Asuncion et al., 2009]. Here, we will use the online variational algorithm presented in [Hoffman et al., 2010].

6.3.1.7 Evaluation against topic models

We now evaluate our ontology framework using a reduced representation of the text corpus obtained by extracting dominant topics, and examine how well different groups of ontologies can represent these topics. As in previous evaluations, we consider the Studio Ontology an extension of the Music Ontology, and quantify its contribution in describing important topics in music production. We use the corpus of 7757 *Sound on Sound* articles described in Section 6.3.1.3. Here, we consider the whole body of the corpus, without partitioning into different columns and time periods, and use a topic modelling algorithm to infer the hidden topic structure[6]. Topic modelling should automatically yield suitable structures, without taking different columns of the magazine into account. We consider future work to check if the partitioning as descried in Section 6.3.1.3 provides more accurate measures, since the manual mapping of ontologies to a large number of topics is a considerable effort.

The number of topics K is an important input parameter of topic modelling algorithms. In a series of experiments using values $K = 20, 30, 50, 100, 200$, we found that we obtain the most useful topics with $K = 100$, however some inconsistent topics remain. We use a subjective and an objective criterion to verify the consistency of the inferred topics: *i)* check if we can identify the topic using a simple topic name (e.g. *recording*), and *ii)*, count the number of articles where the 10 most likely words from a topic co-occur. Using these criteria, we drop the topics that occur in less than 1% of the articles, and cannot be easily named at the same time. This leaves us with 69 topics, exemplified in Table 6.5 and 6.6.

Instrument Sampling		**Audio Effects**		**Studio Production**		**Microphones**		**Computer-based recording**	
term	**p**	**term**	**p**	**term**	**p**	**term**	**p**	**term**	**p**
sample	0.073	effect	0.031	record	0.018	microphone	0.041	firewire	0.012
sound	0.032	reverb	0.024	work	0.016	cardioid	0.028	pci	0.009
sampler	0.017	delay	0.023	studio	0.015	capsule	0.027	interface	0.009
effect	0.009	set	0.014	song	0.011	mount	0.025	channel	0.008
envelope	0.009	audio	0.014	album	0.010	hypercardioid	0.017	motherboard	0.007
instrument	0.008	control	0.013	producer	0.010	polar	0.016	aux	0.007
cut	0.007	parameter	0.009	band	0.008	diaphragm	0.016	daw	0.006
loop	0.007	filter	0.009	engineer	0.008	pattern	0.013	compressor	0.006
play	0.006	chorus	0.008	mixed	0.007	shotgun	0.012	microphone	0.006
waveform	0.005	process	0.008	master	0.007	omni	0.012	cpu	0.005

Table 6.5: Some dominant topics in the Sound on Sound data set, showing the top 10 terms with highest probability within the topic.

Ignoring the context of terms within the documents is a serious limitation of our previous evaluation using a vector space model. The use of topic models to derive a more accurate metrics solves this problem by introducing two levels of subjectivity in our evaluation method.

[6]Python implementation of online inference for LDA is available from:
http://www.cs.princeton.edu/~blei/topicmodeling.html

Electric Organ		Drum loops		Mixing		Reverberation		Equalisation	
term	**p**	**term**	**p**	**term**	**p**	**term**	**p**	**term**	**p**
organ	0.071	drum	0.069	sound	0.028	reverb	0.138	filter	0.063
hammond	0.056	loop	0.035	mix	0.017	convolution	0.038	equaliser	0.052
leslie	0.047	groove	0.019	drum	0.017	reflection	0.038	boost	0.033
keyboard	0.022	snare	0.016	vocal	0.016	early	0.021	frequency	0.027
drawbar	0.021	sound	0.015	track	0.013	room	0.018	shelve	0.022
tonewheel	0.019	beat	0.014	bass	0.010	diffuse	0.017	low	0.021
rotary	0.012	kit	0.014	pan	0.009	impulse	0.015	band	0.018
filter	0.012	pattern	0.013	instrument	0.008	time	0.014	parametric	0.018
play	0.011	tempo	0.012	record	0.008	wall	0.014	cut	0.018
speaker	0.011	sample	0.012	punch	0.007	response	0.013	high	0.013

Table 6.6: Dominant topics in the data set, showing more domain specific topics.

Firstly, the identification of the topics requires audio engineering knowledge. The topic *'Instrument Sampling'* for instance refers to creating a library of samples from an instrument's sound which can be used in wave-table synthesis. This task is relatively common in audio engineering and sound design. The topic is easily recognised by observing the high probability of terms like *waveform*, *envelope*, *loop*, *cut*. Secondly, given the identified topics, the meaning of polysemous words becomes clearer, thus they can be mapped more precisely. We know for example that the word *chorus* in the *'Audio Effects'* topic refers to an effect as opposed to a group of singers, and the term *shotgun* in the *'Microphones'* topic stands for a common metaphor (jargon) referring to a highly directional sensitivity pattern (also called polar pattern) of a microphone, as opposed to a rifle.

It is interesting to notice the emergence of some very domain specific 'micro' topics. For example, the topic about the *'Electric Organ'* contains terms related to playing the organ such as *tonewheel* and *drawbar*, an instrument specific term for sliders controlling the timbre of the instrument. Terms such as *hammond*, *leslie*, *rotary*, or *speaker* signifies the common use of the Hammond organ[7] and the Leslie speaker[8] in 1960s and 1970s rock music, which shows that topic modelling can discover interesting musical phenomena in a suitable text corpus.

Next, we map the topics to two sets of ontologies corresponding the Music Ontology and the Studio Ontology frameworks, and derive an *ontology fit* measure, based on methodologies proposed in [Brewster et al., 2004] and [Raimond, 2008].

We first use the ontology fit measure Δ_m of [Raimond, 2008], where each topic is considered a feature, weighted according to its prevalence in the corpus. The ontology fit is then computed by summing the normalised weights of features that can be expressed using the ontology. This measure ranges from 0 to 1, where a result of 1 means that all features are fully expressible by the ontology. Using this measure, we obtain $\Delta_m = 0.3813$, for the Music Ontology, and $\Delta_m = 0.6412$ for the Music Ontology extended with the Studio Ontology.

[7]Electric organs made by the Hammond Organ Company using additive synthesis.
[8]A speaker cabinet with a rotating woofer that simulates the Doppler effect.

This shows that roughly 40% and 60% of important topics of the corpus can be expressed using the two ontology frameworks respectively. Although there is a significant improvement when adding our ontology framework to the Music Ontology, it is in contrast with the value of $\Delta_m = 0.723$ obtained by Raimond [2008] when mapping topics extracted from musical queries (collected from Google Answers and Yahoo Questions, see [Raimond, 2008] for details) to the Music Ontology framework. This difference is at least partly due to the domain specificity of our text corpus.

Since we found that many topics contain a few highly domain specific terms and/or jargon, that may prevent the topic to be easily represented, we argue for a more accurate ontology fit measure, and introduce the following refinement. Given a set of topics Θ, and the set of documents D (the corpus), we derive a weight $w_n = tf_n \times idf_n$ for each term in each topic Θ_k, where tf_n is the term count of the n-th term of topic in the joint corpus, and idf_n is the inverse document frequency given in Equation 6.4. For each topic, we normalise the term weights, so that they sum is equal to 1, therefore the weights represent the general importance of the term in the corpus and also depend on the topic composition. We then manually map each term in each topic to the ontologies, and quantify how well it can be represented using a score $\delta_{o,k,t}$ given by Equation 6.9, where m is a mapping of term t in topic Θ_k to a term T, and V_o is the vocabulary of ontology o.

$$\delta_{o,k,t} = \begin{cases} 0 & \text{if } m : t \in \Theta_k \to T \notin V_o, \text{ or } T \in \{\mathsf{owl\!:\!Thing}, \mathsf{owl\!:\!Nothing}\} \\ 0.5 & \text{if } m : t \in \Theta_k \to T \notin V_o, \exists S \in V_o : T \subseteq S \\ 1 & \text{if } m : t \in \Theta_k \to T \in V_o \end{cases} \tag{6.9}$$

Topic: Studio production		**Ontology Mapping**		**Score** ($\delta_{o,k}$)	
Term	Weight (w_k)	MO	STUDIO (MOS)	MO	STUDIO
record	0.1744	mo:Recording	mo:Recording	1.0	1.0
work	0.0963	-	-	0.0	0.0
studio	0.1076	-	studio:RecordingStudio	0.0	1.0
song	0.1085	mo:MusicalWork	mo:MusicalWork	1.0	1.0
album	0.0831	mo:Album	mo:Album	1.0	1.0
producer	0.0816	foaf:Agent	studio:Producer	0.5	1.0
band	0.0880	mo:MusicGroup	mo:MusicGroup	1.0	1.0
engineer	0.0652	mo:SoundEngineer	studio:SoundEngineer	1.0	1.0
mixed	0.0775	-	studio:Mixing	0.0	1.0
master	0.1177	mo:SignalGroup	studio:MasterSignal	0.5	1.0
Topic Score (Δ_k):				**0.6190**	**0.9037**

Table 6.7: Mapping a topic to different ontologies

We perform the mapping by finding a concept, a property or an individual in the ontology that can be used in a logical statement involving a representation of the term. The score depends on the success of the mapping, such that it is zero if we cannot map a term within the vocabulary of an ontology (or the mapping is only possible using the most general concepts), 0.5 if there is a broader term in the ontology that represents the word from the topic, but the most specific term remains undefined, and one if mapping to a term within the vocabulary of the ontology is successful. Table 6.7 shows an example of mapping a topic to two ontology frameworks, and the resulting scores. For each topic, we may compute a topic score Δ_k by summing up the normalised weights multiplied by the mapping scores (see also Equation 6.11). A topic score of one means that the topic is fully representable in the ontology.

By computing the weights α_k, we take the prevalence of each topic in the corpus into account, when evaluating the ontologies against the full corpus. This is shown in Equation 6.10, where the numerator is the number of documents in D where every term in topic Θ_k co-occurs. The *ontology fit* Δ_o is then computed by Equation 6.11, where $K = |\Theta|$ is the number of topics, $w'_{k,n}$ are the normalised term weights, and $\delta_{o,k,n}$ are the mapping scores for term n in topic k to ontology o.

$$\alpha_k = \frac{|\{d \in D | \forall t \in \Theta_k : t \in d\}|}{|\Theta|} \tag{6.10}$$

$$\Delta_o = \frac{1}{K} \sum_{k=1}^{K} \sum_{n=1}^{10} \alpha_k \times w'_{k,n} \times \delta_{o,k,n} \tag{6.11}$$

This more fine grained measure is independent of the number of topics considered, takes the relative importance of terms and topics into account, as well as the different precision of term representations using a given ontology. If all topics are fully describable by an ontology we get $\Delta_o = 1$, however, failure to represent less frequent topics diminishes the result more gracefully. This is useful since, for instance, the topic *'Electric Organ'* (see Table 6.6) contributes only to a handful of articles, while the *'Studio Production'* topic contributes to over 400. Compared to vector space modelling, our method provides a better indication of how well an ontology may cover a body of domain specific knowledge, since *i)* manual term mappings take the topic specific semantics into account, and *ii)* assigning different scores to different mappings allows to measure the precision of the logical statements resulting from using a specific ontology framework.

We obtain $\Delta_o = 0.49810$, for the Music Ontology framework alone, using the refined ontology fit, and $\Delta_o = 0.7734$ for the Music Ontology extended with the Studio Ontology. All of the measures computed so far show a significant improvement in the representation of music production specific information when using the Studio Ontology in conjunction with the Music Ontology. However all measures indicate the lack of fine grained modelling of certain areas of music production, for instance, the lack of detailed properties of electro-acoustic instruments, tool specific characteristics and application of audio effects and recording devices, computer hardware for audio recording, or recording room characteristics.

6.3.1.8 Options for logical ontology structure evaluation

The methods discussed so far quantify the expressiveness of our ontologies, but not their logical structure or the quality of knowledge representation within the ontologies. In this section, we discuss some methods that can be used to overcome the limitations of previously considered data-driven evaluation techniques.

A promising approach to evaluate various aspects of ontologies against a text corpus is the use of ontology learning from text, or the use of automatic methods for populating a knowledge base given an ontology. Both ontology learning and population are currently active and challenging fields of research. While fully automatic knowledge acquisition techniques are not yet feasible [Cimiano et al., 2009], ontology learning techniques have a high potential to support the full ontology engineering process, discussed in Section 3.1.2, including evaluation. For instance, one may measure the agreement between an automatically generated and a manually created ontology, or attempt to quantify how information extracted from a text corpus can be represented using different ontologies.

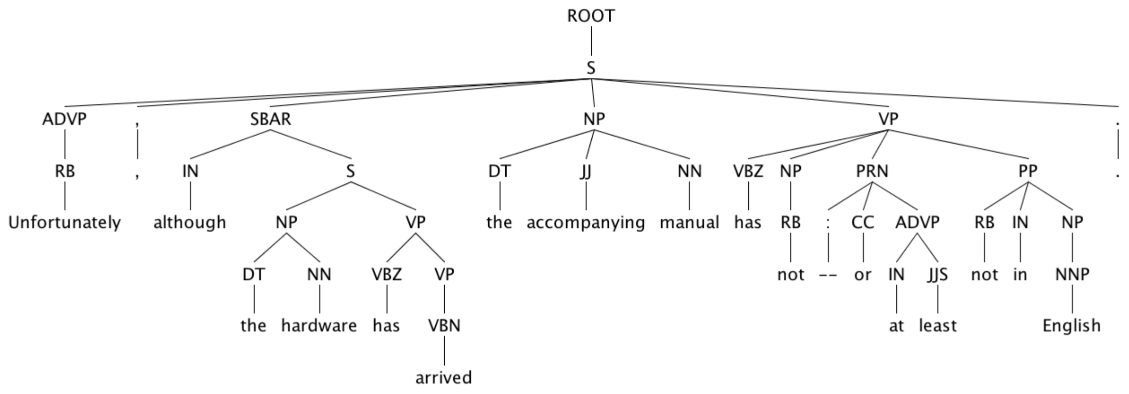


Figure 6.1: Parse tree of a Part-of-Speech tagged sentence from our evaluation corpus.

The first step in both ontology learning and information extraction is the analysis of text, usually at the sentence or clause level. This involves annotating the words with *'Part-of-Speech'* (POS) tags[9], and then parsing sentences into a phrase structure (constituency) tree, or a graph of syntactic dependencies. POS tagging involves the identification of linguistic word categories such as *noun, verb, adjective, adverb*, which depend on both lexical definition and context. Statistical approaches have succeeded in yielding high quality parse-trees, using, for instance, probabilistic context free grammars (PCFG), in which probabilities are assigned to grammar rules learned from large manually created tree banks[10]. Natural lan-

[9]Automatic POS tagging tools include the TreeTagger available from: http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/

[10]A text corpus in which each sentence has been annotated with syntactic structure, for instance, the British English tree bank available from: http://www.ucl.ac.uk/english-usage/projects/ice-gb/

guage parsers using variants of PCFGs are described in [Carroll and Rooth, 1996] and [Klein and Manning, 2003] in detail. Figure 6.1 shows an example of a tagged and parsed sentence from our Sound-on-Sound text corpus, generated using the Stanford statistical parser[11]. Collapsed grammatical dependencies [de Marneffe et al., 2006] provide an alternative, more easily interpreted representation of the syntactic structures, where all sentence relationships are uniformly described as typed dependency relations. The analysis of the sentence *'Bell makes electronic products.'* would yield for instance the relations `subject(makes, Bell)`, `object(makes, products)`. These relations map straightforwardly onto a directed graph, in which words correspond to nodes in the graph, and grammatical relations provide the edge labels[12].

In a simplistic approach, one could use certain types of grammatical dependencies, such as the direct subjects and direct objects of verbs, to verify range and domain types of properties in ontology schemata. More complex techniques for automatic ontology construction are also based on extracting typed dependencies from text corpora. For instance, Cimiano et al [2005] use verb/prepositional phrase, verb/object and verb/subject dependencies, and Formal Concept Analysis (FCA), an algebraic method for abstracting conceptual hierarchies from a set of individuals and the set of their properties [Ganter and Wille, 1999], to learn concept hierarchies automatically. This approach is extended in [Hacene et al., 2008] to learn taxonomic as well as non-taxonomic relationships using Relational Concept Analysis (RCA). These state of the art techniques however do not yet yield good ontologies from arbitrarily complex text, that may be used to verify or evaluate manually created domain ontologies. This is, at least in part, due to the difficulty in natural language parsing. Finding direct relationships between words in complex sentences, and selecting the ontologically relevant relations from a set of dependencies are both non-trivial problems. We found that our music production specific corpus is written in a style that is too informal, and uses sentences that are too complex for this type of analysis. The parse tree shown in Figure 6.1 is a good case in point. The ontologically most relevant relation in this sentence is the one between a piece of *hardware* and the *accompanying manual*, which is spread over different clauses, and do not appear as direct dependencies in a dependency graph. Moreover, the exact information of what *hardware* actually refers to is given in a preceding sentence.

Two approaches could be used to resolve these issues. One would require more sophisticated natural language understanding methodologies, and extend the analyses to the paragraph level, as opposed to parsing individual sentences. Alternatively, we could compile a reduced representation of the text corpus using, for instance, automatic document summarisation. A summary then can be manually pruned, and split into more easily parsed sentences. How then the resulting dependency graphs are best used to evaluate our ontologies is a future research question.

[11]Available from `http://nlp.stanford.edu/software/lex-parser.shtml`

[12]See `http://nlp.stanford.edu/software/stanford-dependencies.shtml` for more complex examples.

Alternatively, the concept taxonomy within an ontology may be evaluated using topic modelling, and various techniques to learn concept graphs directly from text. Building on the foundations of Latent Dirichlet Allocation, Chambers et al [2010] use a probabilistic framework to learn relationships between topics from text, which is then used to learn concept graph structures. We consider future work to assess the concept hierarchies extracted this way from our text corpus, and find a suitable method to evaluate ontologies against these results.

6.3.2 Qualitative evaluation

So far we have discussed quantitative methods to evaluate the Studio Ontology framework, and measure its contribution to the Music Ontology for describing music production in the recording studio. However, we argue that quantitative methods are generally not sufficient for judging the overall quality of an ontology. Therefore we propose evaluating our ontology against a set of use cases appearing both in the previously described music production text corpus, and additional Web resources.

One of the main challenges in qualitative evaluation is choosing the right users, equally knowledgeable in the domain of the ontology, as well as the usage and underlying concepts of Semantic Web ontologies. While it may be feasible for smaller and very specific ontologies to train a large number of domain experts to use and evaluate an ontology, it becomes problematic with ontology libraries such as the Studio Ontology, which cover many subdomains of the field. Therefore, we base this part of our evaluation on task-based self-assessment. We first choose a number of topics, outline typical requirements, attempt to represent the topic using our ontology, and finally examine how well this representation fulfils the requirements.

6.3.2.1 Use Case: Describing recording studios

Considering the Semantic Web as a counterpart of the Web, we need to be able to represent the information, one would normally find on the web site of a recording studio. This information typically includes a list of recording spaces and their characteristics, post-production facilities, lists of equipment available for the engineers, producers and artists, associations between recording or mastering facilities, engineers and the clientele. Providing a machine readable representation of this information is a contribution to the Semantic Web in itself, since, for instance, user agents could find recording studios using complex queries based on the equipment available, technical expertise, or the previous projects of the studio.

Requirements: Based on an informal summary of our music production text corpus, we collected the following information needs in describing recording studios:

- describe recording and other facilities
- describe the characteristics of recording rooms
- describe available equipment
- associate agents with studios and facilities
- describe technical expertise
- describe previous recording projects

Examples and discussion: The Web pages of Abbey Road Studios[13] contain information in all of the above categories. In the following, we use this data to evaluate how it can be represented using the Studio Ontology. First, we describe the studio and its facilities. Please note that all RDF examples in each of the following sections should be seen as part of a single knowledge base, data set or RDF file.

```
@base <http://example.org/resource/>         .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix mo: <http://purl.org/ontology/mo/>  .
@prefix studio: <http://purl.org/ontology/studio/> .

# Description of Abbey Road Studios
:AbbeyRoadStudios a studio:CommertialStudio ;
       studio:label [ a mo:Label ; foaf:name "EMI" ] ;
       studio:facility :Studio1, :Studio2, :Studio3 ;
       studio:facility :MasteringRoom5 ;
       studio:homepage <http://www.abbeyroad.com> .

:Studio1 a studio:RecordingFacility .
:Studio3 a studio:RecordingFacility .
:Studio2 a studio:RecordingFacility ;
       studio:facility :control_room, :recording_room,
       [ a studio:Lounge ] .
```

Listing 6.1: RDF description of Abbey Road Studios and its facilities (partial).

In Listing 6.1, we associate a recording studio with a record label (which may be the owner or main client of the studio), and its facilities. We identify Abbey Road as a commercial studio

[13] Detailed information about Abbey Road Studios is available at: http://www.abbeyroad.com/

(the corresponding concept is a subclass of `mo:CorporateBody` and `studio:RecordingStudio`.) The general purpose property `studio:facility` is used to link studios with self-contained sub-units, such as *Studio Two*, which is useful to express the common organisational structure of larger studios, but also used to subdivide facilities to smaller units such as a control room or a recording room. This conceptualisation supports the easy formulation of the typical query: *List all facilities available in this studio.*

Studio facilities may be associated with simple descriptions regarding the size (floor area, width, depth, height) or volume of the space. We can also link recording rooms with simple textual or numerical descriptions for wall and floor materials, absorption coefficients, standard reverberation times, and impulse responses, which are identified as instances of the `mo:Signal` concepts. Presently, 9 simple data properties are provided, which include, for instance, `studio:floor_area` or `studio:rt60_decay_time`. More complex architectural characteristics are not well aligned with the purpose of the studio ontology, therefore these may only be expressed using a specialised extension or a suitable domain ontology. However, at the time of writing, we are not aware of such an ontology. Studio facilities can be explicitly identified as recording, control, or mastering spaces (among others), and linked with equipment available in these spaces. This is exemplified in Listing 6.2.

```
@prefix device: <http://purl.org/ontology/studio/device/> .
@prefix mx: <http://purl.org/ontology/studio/audiomixer/> .
@prefix mic: <http://purl.org/ontology/studio/microphone/> .

# Organisations
:akg a foaf:Organization ;
       foaf:name "AKG" .

# Recording room equipment
:recording_room a studio:RecordingRoom ;
       rdfs:label "Recording space in Studio2 at Abbey Road"
       studio:equipment :mic1, :mic2, :mic3 .

:control_room a studio:ControlRoom ;
       studio:equipment :neve_console, :monitoring_system .

:neve_console a mx:AnalogConsole ;
       device:vendor [ a foaf:Organization ; foaf:name "Neve" ] ;
       device:model "VRP Legend" ;
       mx:channel_count "60"^^xsd:int .
```

```

:monitoring_system a studio:MonitoringSystem ;
        device:vendor [ a foaf:Organization ;
                foaf:name "Quested" ] .

:mic1 a mic:CondenserMicrophone ;
        device:vendor :akg ;
        device:model "C 12" .

:mic2 a mic:CondenserMicrophone ;
        device:vendor :akg ;
        device:model "C 414 EB" .

:mic3 a mic:CondenserMicrophone ;
        device:vendor :akg ;
        device:model "C 414 EB" .
```

Listing 6.2: Partial description of recording spaces and equipment at Abbey Road Studios.

In the above example, we provide a partial equipment list of the recording room of Studio 2 at Abbey Road. The property `studio:equipment` and its sub properties are used to link facilities to equipment, whose range is simply `device:Device`, which is used to identify technological artefacts. This conceptualisation supports the common use case of querying for a list of equipment, but also allows very simple queries to be written for such specific cases as retrieving all condenser microphone models. These microphones are frequently used for vocal recordings, and it is often the first question a recording artist may ask. In query 6.3, for instance, we retrieve a list of recording spaces, where a model "C 414 EB" condenser microphone is available.

```
SELECT ?room WHERE {
        ?room studio:equipment ?mic .
        ?mic a mic:CondenserMicrophone ;
             device:model "C 414 EB" . }
```

Listing 6.3: Finding the recording room featuring a specific type of microphone.

A limitation of referring to all equipment types using a generic `studio:equipment` property is that we need to know the specific equipment sub-categories available in the ontology to retrieve, for instance, all recording devices, or all types of microphones. This limitation can be resolved by using sub-properties of `studio:equipment`, such as `studio:daw` or `studio:recording_device`, whose range is restricted to specific device categories in the vocabulary of audio engineering tools. This is exemplified in Listing 6.4, where we also show how to associate a studio facility with the engineers working there.

```
:MasteringRoom5 a studio:MasteringRoom ;
        studio:mastering_engineer [ a studio:SoundEngineer ;
                foaf:name "Geoff Pesche" ] ;
        studio:mastering_engineer [ a studio:SoundEngineer ;
                foaf:name "Christian Wright" ] ;
        studio:daw [ a studio:DAW, mx:DigitalConsole ;
                device:vendor [ a foaf:Organization ;
                        foaf:name "SADiE" ] ;
                device:model "Series 5 PCM 8 Digital Audio Workstation" ;
                device:firmware_version "5.6.1" ] ;
        studio:recording_device [ a studio:AnalogTapeMachine ;
                device:vendor [ a foaf:Organization ;
                        foaf:name "Ampex" ] ;
                device:model "Ampex ATR 100" ] ;
        studio:monitor [ a studio:NearFieldMonitor ;
                device:vendor [ a foaf:Organization ;
                        foaf:name "Yamaha" ] ;
                device:model "NS-10M" ] .
```

Listing 6.4: Description of Abbey Road Mastering Room 5

Our ontology provides the necessary flexibility to represent and associate recording facilities, engineers (and other staff), and tools, but it's up to the user of the ontology to choose the desired granularity of knowledge representation for an application. For instance, whether to associate facilities and devises using the generic property `studio:equipment`, or its more specific sub-properties, depends on the typical application of a recording studio database. For instance, using `studio:equipment` supports the more general query *'List all available equipment.'*, even if an inferential component, such as a deductive query engine which takes assertions of the ontology schema into account when evaluating a query is not available. Since specific kinds of devices are used for different purposes in music production, this knowledge can be assumed, therefore we believe that our conceptualisation makes the optimal compromises from an audio engineering point of view. The best practice in general, is the use of the most specific concepts and properties available in the ontology, and make more general assertions if required by the system or the application.

We have shown that the first four requirements, *i)* describing studio facilities, *ii)* the characteristics of recording rooms, *iii)* the available equipment, and *iv)* associate agents with studios and facilities, are easily fulfilled using the Studio Ontology. Describing the technical expertise of staff, and past projects are not currently supported by the ontology, or any of its extensions. The Music Ontology however, supports linking music releases to engineers or producers, therefore parts of these requirements can be, at least indirectly, expressed within our framework, but this does not allow easy query formulation. For instance, the property `mo:engineered` may be used to link an engineer to a performance, recording or recording session event, which in turn can be localised in a recording studio, but this is a rather complex way to express simple information like a list of albums produced in a studio. We consider future work to fulfil these requirements in a more simple way.

6.3.2.2 Use Case: Describing recording scenarios

Another common information need in describing music production concerns the details of recording, in particular, the placement and configuration of microphones, and the description of microphone techniques.

Requirements: The main requirements when describing the details of recording are:

- describe microphones and their application (such as variable parameters)
- describe microphone placement and specific techniques
- describe the recording of individual instruments
- describe the recording of groups and orchestras
- describe spatial and ambient recording techniques
- describe pre and post processing of recorded sound and signals

Examples and discussion: The Microphone Ontology extension of the Studio Ontology enables the description of recordings and recording techniques, as described in Section 4.2.5.1. This allows denoting the main characteristics of microphones used in a particular recording. The `studio:microphone` property — whose domain includes recording events (`mo:Reocrding`) and range is the top-level `studio:Microphone` concept with specific subtypes defined in the Microphone Ontology — can be used to link recording events to microphones.

```
:rec1 a mo:Recording ;
       studio:microphone :mic1 .

:mic1 a mic:CondenserMicrophone ;
       device:vendor [ a foaf:Organisation ; foaf:name "Neumann" ] ;
       device:model "M 49" ;
       rdfs:comment """The M 49 is a classic Neumann studio microphone of
           the 1960s. It is a short-bodied design with a twin large-
          diaphragm capsule and variable polar pattern...""" ;
       mic:diaphgram_type "large" ;
       mic:max_spl "125.0"^^xsd:float ;
       mic:output_impedance "200"^^xsd:nonNegativeInteger ;
       mic:output_sensitivity "6.0"^^xsd:float ;
       mic:high_frequency_rolloff "16000"^^xsd:int ;
       mic:connector_type con:Tuchel_8 ;
       mic:polar_pattern mic:MultiPolar ;
       mic:configuration :cfg1 .

:cfg1 a mic:Configuration ;
       mic:polar_pattern mic:Omnidirectional ;
       mic:distance "2"^^xdd:float ;
       mic:azimuth "0"^^xdd:int ;
       mic:elevation "0"^^xdd:int .
```

Listing 6.5: Application of a Neumann M49 vintage microphone in a recording event.

In Listing 6.5 we provide detailed information about a microphone[14] and show how it is applied in a recording event. The Microphone Ontology allows to describe both static and variable parameters of the device, including physical and electrical characteristics. For instance, the size of the diaphgram can be expressed using the literal values *miniature*, *small* and *large* which are common designations in audio engineering. We may also specify the exact size using `mic:diaphgram_size`. The maximum sound pressure level (SPL) the device can process can be expressed using `mic:max_spl`. Various other electrical and frequency characteristics can be included in these descriptions as shown in our RDF example.

Variable parameters of microphones are expressed using the `mic:Configuration` concept, which is considered an abstraction of changeable states of devices modelled as events. This is discussed in Section 4.2.3.6. The domain of variable microphone properties includes the

[14]Microphone data was obtained from the online catalogue: http://www.microphone-data.com/microphones/m49/

`mic:Configuration` concept to satisfy this requirement. Without additional information regarding the temporal extent of this event, it shall be interpreted as coinciding with the main recording event. We currently do not require this event to be specifically timed, therefore the interpretation of the data relies on a convention, which can be seen as a limitation of the ontology. However, multiple configurations may be described and linked with specific temporal objects.

Similarly, the measurement units of all parameter values are fixed to be the most conventionally used ones, for instance, `mic:max_spl` has to be expressed in deci Bels (dB). The ontology specification includes these designations. User agents may provide different views on these data, and data entry interfaces may allow entering values in different units, but implementations have to take care of necessary conversions. We believe that this is a sensible compromise over alternative solutions, such as reified parameter values, allowing units to be expressed together with values, or the use of different properties for different measurement units.

```
:rec2 a mo:Recording ;
       studio:microphone_technique :arr1 .

:arr1 a mic:SpacedPair ;
       mic:left [ a mic:CondenserMicrophone ;
              device:vendor [ a foaf:Organisation ;
              foaf:name "Schoeps" ] ;
              device:model "CCM 41 VL/U" ;
              mic:polar_pattern mic:Supercardioid
       ] ;
       mic:right [ a mic:CondenserMicrophone ;
              device:vendor [ a foaf:Organisation ;
              foaf:name "Schoeps" ] ;
              device:model "CCM 41 VL/U" ;
              mic:polar_pattern mic:Supercardioid
       ] ;
       mic:element_distance "1"^^xdd:float ;
       mic:distance "2"^^xdd:float ;
       mic:azimuth "10"^^xdd:int ;
       mic:elevation "0"^^xdd:int .
```

Listing 6.6: Describing a microphone technique: ambient recording using a spaced pair.

Microphone placement is described using the properties `mic:distance`, `mic:azimuth` and `mic:elevation`, which express the (approximate) distance between the microphone pickup and the sound source, as well as the vertical and horizontal angles between the main axis of the microphone and the principal radiating direction of the instrument, (by convention, zero means the microphone is placed perfectly on-axis). As we previously noted, this is somewhat problematic, as the interpretation of these data may be instrument specific. The solution is to consider the use cases of recording, and provide the grounding and reference points necessary for the description of more precise microphone placement in a musical instrument ontology corresponding to our ontology library (see Section 4.5).

```
:rec3 a mo:Recording ;
       studio:microphone_technique :arr2 .

:arr2 a mic:DeccaTree ;
       mic:left [ a mic:CondenserMicrophone ;
              device:vendor [ a foaf:Organisation ;
              foaf:name "Neumann" ] ;
              device:model "M 50" ;
              mic:polar_pattern mic:Omnidirectional
       ] ;
       mic:right [ a mic:CondenserMicrophone ;
              device:vendor [ a foaf:Organisation ;
              foaf:name "Neumann" ] ;
              device:model "M 50" ;
              mic:polar_pattern mic:Omnidirectional
       ] ;
       mic:center [ a mic:CondenserMicrophone ;
              device:vendor [ a foaf:Organisation ;
              foaf:name "Neumann" ] ;
              device:model "M 50" ;
              mic:polar_pattern mic:Omnidirectional
       ] .
```

Listing 6.7: Description of a Decca Tree

When describing microphone techniques or arrangements, we have the option of describing each microphone individually, which is currently the only option for very complex arrangements, such as the recording of a grand piano or a drum kit using several microphones. Alternatively, we can use one of the microphone technique concepts available in the ontology. This is exemplified in Listing 6.6, describing a technique commonly used for recording ambient sound. This technique is called *Spaced Pair*. It consists of two identical microphones, placed relatively far from the source. In case of microphone arrangements, the placement predicates relate the centre and principal direction of sensitivity of the microphone array to the sound source. Microphone arrangements are linked to recording events using the `studio:microphone_technique` predicate, while any number of constituent microphones can be linked to the arrangement using `mic:constituent_microphone`. For specific arrangements where relative positioning of the microphones within an array are defined by the applied technique itself, we provide sub-properties such as `mic:left` or `mic:center` to link individual microphone descriptions to the array. For instance, the placement of microphones in a standard Decca Tree — which is a stereo technique used for recording large ensembles — is fixed, therefore it doesn't have to be included in the description shown in Listing 6.7. However, variations of this and similar stereo and spatial recording techniques exists, where the individual spatial co-ordinates of each microphone may be specified relative to the centre of the array.

Describing the recording of electric and electro-acoustic instruments with direct analogue or digital signal output is more straightforward. The ontology provides means for describing recording interfaces, such as sound cards, break-out boxes and direct injection (DI) boxes frequently used to match the signal level and output impedance of the instrument to the recording device in these recording scenarios. Currently, the ontology allows the inclusion of these devices in a signal flow description, but only their basic characteristics can be described.

Although the ontology fulfils most requirements mentioned in this section, there is scope for future improvement. Currently the ontology provides very limited knowledge representation when it comes to instrument specific recording techniques, such as the choice, configuration and placement of microphones given an instrument. For instance, we could restrict certain microphone characteristics, or express the most suitable transducer technology to be used, the desirable frequency response, or polar pattern for recording a certain instrument. Given a recording situation, we could for instance describe whether to use a dynamic, a condenser or another type of microphone. We could also include constraints on the type of microphones used in stereo and spatial recording techniques, end express facts such as the standard Decca Tree has to be constructed of identical omnidirectional microphones. This embedded knowledge could then be used to aid the engineer in common recording situations. Our ontology currently lacks this level of knowledge representation. Lastly, although pre and post processing of recorded sound is can be seen as part of the recording process, we discuss these issues in a wider context in the following sections.

6.3.2.3 Use Case: Describing Recording Sessions

Describing recording sessions primarily concerns the denotation of a workflow in the studio. The Music Ontology provides a fundamental model of music production workflows discussed in Section 4.1.3.2. This model focuses on broader, more general concepts, such as composition, musical works, performances, sounds, recordings, album releases and individual musical items. In studio production, we need to be able to talk about studio sessions corresponding to elements of this broader workflow. For instance, describing composition and song writing sessions in the studio environment. However, we also need to be able to express a narrower, more specific workflow discussed in Section 4.2.4.2, related to the production of individual musical recordings, and the audio engineering workflow.

Requirements: The most important requirements in describing a recording session are:

- the ability to link individual recording events to a single session,
- the identification of the resulting signals,
- the description of a signal flow, for instance the ability to express how signals from a microphone array are mixed together,
- the ability to denote the audio engineering workflow,
- liking parts of the recording session to participants, artists and engineers.

Examples and discussion: In the following, we show how the Music and Studio ontologies can be used to fulfil these requirements. In Listing 6.8, we first describe a performance as part of a recording session.

```
:s1 a studio:RecordingSession ;
       event:place <http://dbpedia.org/resource/Abbey_Road_Studios> ;
       event:sub_event :perf1, :rec1 .

:perf1 a mo:Performance ;
       rdfs:comment "A solo performance on guitar and vocals." ;
       mo:performer [ a mo:MusicArtist ; foaf:name "Les Paul" ] ;
       dc:title "How High the Moon" ;
       mo:produced_sound :snd1, :snd2 .
```

Listing 6.8: Describing a performance during a recording session

We distinguish between different types of sessions in the studio, such as composition, arrangement, rehearsal, recording, as well as post production, including mixing and mastering. These sessions can all be interpreted as part of the wider `mo:RecordingSession` concept, which encompasses all the diverse activities related to producing an album. In a studio specific ontology however, the above mentioned distinctions are also important. The concept `studio:RecordingSession` can be used to designate a session strictly for the case of recording new material. Our simple recording session example consists of a single performance event, producing two distinct sounds.

```
:snd1 a mo:Sound ;
       rdfs:comment "The sound of the voice." ;
       mo:recorded_in :r1 .

:snd2 a mo:Sound ;
       rdfs:comment "The sound of the guitar." ;
       mo:recorded_in :r2 .

:rec1 a mo:Recording ;
       rdfs:comment "Recording event corresponding to the song." ;
       mo:produced_signal :sig_mix ;
       event:sub_event r1, r2 .

:r1 a mo:Recording ;
       rdfs:comment "Recording the vocal part" ;
       mo:produced_signal :sig1 ;
       studio:recording_engineer [ a studio:SoundEngineer;
              rdfs:comment "Our recording engineer" ;
              foaf:name "John" ] ;
       studio:microphone :mic1 .

:r2 a mo:Recording ;
       mo:produced_signal :sig2, :sig3 ;
       rdfs:comment "Recording the guitar using a stereo technique" ;
       studio:recording_engineer [ a studio:SoundEngineer;
              rdfs:comment "Our other recording engineer" ;
              foaf:name "Mike" ] ;
       studio:microphone_arrangement :arr1 .
```

Listing 6.9: Describing the recording of a performance

The sound of the voice and the guitar are individually recorded, as described in Listing 6.9. A complex recording event corresponding to the song is decomposed into two individual recordings, expressing the fact that two different engineers were involved in setting up the different microphones. The configuration of the vocal microphone, and the stereo arrangement for recording the guitar is described in listings 6.10 and 6.11 respectively.

```
:mic1 a mic:DynamicMicrophone ;
       device:vendor [ a foaf:Organisation ;
              foaf:name "Electrovoice" ] ;
       device:model "RE 20" ;
       mic:polar_pattern mic:Cardioid ;
       mic:configuration :cfg1 ;
       mic:output [ a con:AnalogOutput ;
              con:signal :sig1 ] .

:cfg1 a mic:Configuration ;
       mic:distance "0.3"^^xdd:float ;
       mic:azimuth "0"^^xdd:int ;
       mic:elevation "0"^^xdd:int .
```

Listing 6.10: Describing the vocal microphone configuration

In these examples, we also show how the microphone outputs are linked to signals, which is necessary to describe the signal flow. The details of how microphone placement and configurations are described was discussed in the previous section. The following example show an X/Y stereo arrangement. This example highlights a current limitation of the ontology, that is, microphone placement in some situation cannot be fully expressed in a machine readable way. We had to include an extra `rdfs:comment` predicate to express that we place the microphone relative to a particular fret on the neck of the guitar as opposed to the sound hole.

```
:arr1 a mic:XY ;
       mic:left [ a mic:CondenserMicrophone ;
              device:vendor [ a foaf:Organisation ;
              foaf:name "DPA" ] ;
              device:model "4011" ;
              mic:polar_pattern mic:Cardioid ;
              mic:output [ a con:AnalogOutput ;
                     con:signal :sig2 ]
```

```
        ] ;
        mic:right [ a mic:CondenserMicrophone ;
                device:vendor [ a foaf:Organisation ;
                foaf:name "DPA" ] ;
                device:model "4011" ;
                mic:polar_pattern mic:Cardioid ;
                mic:output [ a con:AnalogOutput ;
                        con:signal :sig3 ]
        ] ;
        mic:distance "0.6"^^xdd:float ;
        mic:azimuth "20"^^xdd:int ;
        mic:elevation "30"^^xdd:int ;
        rdfs:comment "Placed relative to the 10th freat." .
```

Listing 6.11: Describing the stereo arrangement

Now that we have described the details of a performance and the recording, in the following examples of Listing 6.12 and 6.13, we denote how the individual recordings are mixed together.

```
:sig1 a mo:AnalogSignal .
:sig2 a mo:AnalogSignal .
:sig3 a mo:AnalogSignal .

:m1 a studio:Mixing ; # a mixing event
        studio:mixing_engineer [ a studio:SoundEngineer;
                foaf:name "Joe" ] .
        studio:consumed_signal :sig1, :sig2, sig3 ;
        studio:produced_signal :sig_mix ;
        studio:console :mixer1 .

:sig_mix a mo:DigitalSignal ;
        rdfs:comment "The mixed signal" ;
        mo:channels "2"^^xsd:int .
```

Listing 6.12: Describing the mixing of analogue signals

An additional event `studio:Mixing`, discussed in Section 4.2.4.2, is introduced in the workflow model. This event is linked to the audio signals produced by parallel recording events, as well as agents and factors, such as the mixing console and the mixing engineer. The mixing event produces a stereo digital signal, however, we are yet to describe the details about the mixing console and its configuration. First we describe the signal connections which allow us to associate input and output signals with the mixer channels and internal mixing buses.

```
:mixer1 a mx:DigitalConsole ;
       device:vendor [ a foaf:Organisation ;
              foaf:name "Yamaha" ] ;
       device:model "MC3000" ;
       mx:mixer_channels :ch1, :ch2, ch3 .

:in1 a con:AnalogueInput, con:BalancedTerminal ;
       con:input_signal :sig1 ;
       con:terminal_channels "1"^^xsd:int ;
       con:connector con:XLR_3F .

:in2 a con:AnalogueInput, con:BalancedTerminal ;
       con:input_signal :sig2 ;
       con:terminal_channels "1"^^xsd:int ;
       con:connector con:XLR_3F .

:in3 a con:AnalogueInput, con:BalancedTerminal ;
       con:input_signal :sig3 ;
       con:terminal_channels "1"^^xsd:int ;
       con:connector con:XLR_3F .

:out1 a con:DigitalOutput, con:StereoTerminal ;
       con:output_signal :sig_mix ;
       con:terminal_channels "2"^^xsd:int ;
       con:connector con:EBU .

# mixing and effect busses
:output_bus1 a mx:MixingBus ;
       mx:output :out1 .

:send_bus1 a mx:SendBus ;
       mx:output :send_output1 .
```

Listing 6.13: Describing the connections of the mixer

Finally, we can describe the channel settings and routing of the mixing console during the mixing and corresponding recording event.

```
:ch1 a mx:Channel ;
       rdfs:comment "The vocal channel" ;
       mx:input :in1 ;
       mx:bus :output_bus1 ;
       mx:send_bus :send_bus1 ;
       mx:pan "0"^^xsd:float ;
       # equvalent property mx:left_right_position
       mx:fader_level "0"^^xsd:float ;
       mx:gain_level "10"^^xsd:float .

:ch2 a mx:Channel ;
       rdfs:comment "The left channel of the stereo pair" ;
       mx:input :in2 ;
       mx:bus :output_bus1 ;
       mx:pan "-70"^^xsd:float ;
       mx:fader_level "-30"^^xsd:float ;
       mx:gain_level "20"^^xsd:float .

:ch3 a mx:Channel ;
       rdfs:comment "The right channel of the stereo pair" ;
       mx:input :in3 ;
       mx:bus :output_bus1 ;
       mx:pan "70"^^xsd:float ;
       mx:fader_level "-30"^^xsd:float ;
       mx:gain_level "20"^^xsd:float .
```

Listing 6.14: Describing the configuration of the mixer

Using the `mx:input` and `mx:output` predicates we associate channels and buses with input and output terminals receiving or emitting signals. The `mx:pan` predicates can be used to describe standard stereo panning, with the units adopted from the AES-31-3-2008 standard, where a value of +100.0 means fully right, -100.0 means fully left and 0 means centre. Similarly, surround spatial positioning can also be expressed using the equivalent property `mx:left_right_position` and the corresponding `mx:front_rear_position` predicate. Finally, in the last example, we describe a tape recording device connected to the digital output of the mixing console in Listing 6.15.

```
:rd1 a studio:DATMachine ;
        device:input [ a con:DigitalInput ;
                con:terminal_channels "2"^^xsd:int ;
                con:connector con:EBU
                con:input_signal :sig_mix ] ;
        rdfs:comment "A recording device connected to the mixer."
```

Listing 6.15: Connecting a recording device

The above examples show that our ontology framework fulfils the requirements for describing recording sessions in details. We have extended the music production workflow model of the Music Ontology for this purpose. Our model provides two levels of granularity in representing audio engineering workflows. Firstly, it has the ability to describe a set of parallel or successive events (event flow) corresponding to performances, recordings and audio engineering tasks. Secondly, it also provides a way to describe the detailed signal flow and connection of recording and audio engineering tools, and their configurations. A current limitation of the ontology is the lack of predicates for a more detailed description of tools other than microphones and consoles. Although many other tools may be included in the recording workflow, using the generic terms of the device and connectivity ontologies, further work is required to provide ontology extensions for other devices, based on the fundamental vocabulary of tools present in the core Studio Ontology.

6.3.2.4 Use Case: Describing post-production

So far we have discussed how well our framework can be used to represent information about recording studios, the specific details of recording, such as microphone placement, and examined how it can be used to describe whole recording sessions. We need to consider one remaining area of audio engineering, the post production of recorded material, including audio editing, audio effects, or how recordings can be managed in digital audio workstations. We note that the application of audio effects is not only a post-production task.

Requirements: Notable requirements for describing post-production tools and tasks are:

- description of multitrack production tools (multitrack editors and workstations),
- the audio editing workflow,
- complex signal processing devices and their application,
- audio transformations (such as mastering).

Examples and discussion: Using the Multitrack Ontology discussed in Section 4.2.5.4 we can describe how recorded material is represented in multitrack music production tools. In listings 6.16, we exemplify the use of this ontology, describing the outcome of a simple recording session, and its representation in an multitrack audio editor.

```
:our_singer a mo:SoloMusicArtist ;
    foaf:name "Superstar Singer" .

:chorus_take a mo:Performance ;
    mo:performer :our_singer ;
    mo:recorded_as project:chorus .
:intro_take a mo:Performance ;
    mo:performer :our_singer ;
    mo:recorded_as project:intro .
:verse_take a mo:Performance ;
    mo:performer :our_singer ;
    mo:recorded_as project:verse .

:chorus a mo:Signal .
:intro a mo:Signal .
:verse a mo:Signal .

:voclas_001 a mt:AudioClip ;
    rdfs:label "The first take of the intro" ;
    mt:signal :intro .
:voclas_002 a mt:AudioClip ;
    rdfs:label "The first take of the verse" ;
    mt:signal :verse .
:voclas_003 a mt:AudioClip ;
    rdfs:label "The first take of the chorus" ;
    mt:signal :chorus .

:vocals_001 a mt:AudioTrack ;
    mt:clip :voclas_001 , :voclas_002 , :voclas_003 .

:myProject a mt:MultitrackProject ;
    mt:track :bass_001 , :keyboard_001 , :vocals_001 .
```

Listing 6.16: Multitrack Ontology example (simplified)

The above listing shows how the recording of a set of performances may be represented in a multitrack audio editor, that is, how the audio clip and audio track composition of a multitrack project may be encoded, how these entities are related to each other, and how they are linked to audio signals. However, this representation is only sufficient for encoding a static project state, assuming that recordings of different instruments are synchronised. In order to express an editing workflow in a multitrack workstation, first, we need to be able to express the temporal relations of audio clips to signals, tracks, or the whole project. This is exemplified in listings 6.17.

```
:voice_take1 a mo:Signal ;
       mo:time [ a tl:Interval ;
   tl:onTimeLine :signal_timeline1 ] .

:signal_timeline a tl:TimeLine .
:track_timeline a tl:TimeLine .

:voice_clip1 a mt:AudioClip ;
       mt:signal :voice_take1 ;
       mt:signal_time [ a tl:Interval ;
              tl:at "PT01.000000000S"^^xsd:duration ;
              tl:duration "PT30.00000000S"^^xsd:duration ;
              tl:onTimeLine :signal_timeline ] ;
       mt:track_time [ a tl:Interval ;
              tl:at "PT20.500000000S"^^xsd:duration ;
              tl:duration "PT30.000000000S"^^xsd:duration ;
              tl:onTimeLine :track_timeline ] .
```

Listing 6.17: Describing audio clips in a temporal context

Here, we define a timeline for a recorded audio signal, and a timeline for a track in an audio editor. Multitrack audio editors often conceptualise clips as a *'window'* over an underlying signal, that is, a clip may not necessarily contain the whole signal and its boundaries may be variable. The temporal relation of a clip and a signal can be expressed in our ontology using the property `mt:signal_time` which links the clip to an interval defined on the signal timeline. The property `mt:track_time` describes the position and duration of the clip within an audio track. This conceptualisation is flexible enough to represent even the most complex cases, for instance, when both clips and tracks can be moved relative to a project timeline. However, we do not require a full description in all circumstances. If an audio editor always represents a complete signal in a clip, the `mt:signal_time` property does not need to be used, and the duration of the clip is *assumed* to be equivalent to the duration of the signal. These sort of assumptions may be seen as limitations of the ontology. They may be however resolved using various OWL language features, for example, cardinality constraints on the various time related predicates. Requiring all temporal associations to be present in all descriptions may lead to significantly more verbose description when describing complex editing workflows. Finding the best trade off is not trivial, and we regard this future work.

Next, we consider how edit decisions within an audio editing workflow may be described using the Multitrack and the Edit ontologies. In Section 4.2.5.4 we discussed some basic audio editing operations. In listings 6.18 we exemplify how these can be used to describe the manipulation of audio clips.

```
:ed1 a edit:Move ;
       edit:consumed_clip :voice_clip1 ;
       edit:produced_clip :voice_clip2 ;
       edit:media_time [ a tl:Interval ;
              tl:at "PT0S"^^xsd:duration ;
           tl:duration "PT10.000000000S"^^xsd:duration ;
              tl:onTimeLine :track_timeline ] .

:voice_clip2 a mt:Clip ;
       mt:signal :voice_take1 ;
       mt:signal_time [ a tl:Interval ;
              tl:at "PT01.000000000S"^^xsd:duration ;
           tl:duration "PT30.000000000S"^^xsd:duration ;
              tl:onTimeLine :signal_timeline ] ;
       mt:track_time [ a tl:Interval ;
              tl:at "PT30.500000000S"^^xsd:duration ;
           tl:duration "PT30.000000000S"^^xsd:duration ;
              tl:onTimeLine :track_timeline ] .
```

Listing 6.18: Describing a move operation

Recall that all edit decisions are modelled as time based events. These events may be linked to the universal timeline as well as the timeline of the track, clip or signal entities an event operates on. Therefore both the order of execution and the edit decision's relation to the audio material are captured. In our example, we describe moving a clip, defined previously in listings 6.17, relative to the timeline of a track. The time extent of the move is expressed using the time interval linked to the event by the `edit:media_time` predicate. The description above is perhaps the most trivial example, which shows the consequence of modelling all transformations as events that change the identity of the entities they operate on. This conceptualisation, although produces an excessive amount of data, enables us to describe complex sound editing workflows. Similarly to edit decisions, audio transformations can also be described using our ontology. It provides a rather generic term `edit:Transform`, specialised to describe basic operations of an editor for instance, `edit:Mute`, `edit:Gain`. Similarly to the `studio:Transform` concept, which is intended to be used when describing real-time signal processing workflows discussed in Section 4.2.4.3, `edit:Transform` concept may be linked to devices, such as audio processing plugins to describe off-line operations.

Having more than one way of encoding audio transformations seems somewhat problematic, and certainly, the ontology does not satisfy the principles of clarity and minimal ontological commitment in this respect (see Section 3.1.4). It requires further refinements and modelling experiments to create a simpler and more unified model which is able to fulfil the description requirements for real-time signal transformations using hardware or software, as well as transformations in an audio editor using either built in commands or plugins.

6.4 Summary and discussion

In this chapter a formal evaluation of the Studio Ontology framework was presented. We first reviewed prominent techniques available for ontology evaluation and choose to perform quantitative evaluation using data-driven automated testing, as well as qualitative evaluation using self-assessment based on specific case studies.

Our data-driven evaluation measures the lexical coverage (see Section 6.3.1.4) and a more subjective ontology fit (see Section 6.3.1.5) given a large domain specific text corpus obtained from the internet archive of a domain-specific periodical. These measures demonstrated that the Studio Ontology has a significant contribution to the set of ontologies comprising the Music and Studio Ontology frameworks when describing relevant topics in this text corpus. We also outlined a methodology to automatically test the quality of the logical model within the ontologies given a text corpus. However, it remains future work to find specific techniques and the algorithms most suitable to perform these tests. The difficulty lies in improving information extraction techniques using natural language processing (see Section 6.3.1.8).

Our qualitative evaluation relies on objective self-assessment examining how our ontologies may be used to represent information in four music production related use cases; describing recording studios, concrete recording events, complete recording sessions, and post production. Here, we showed that albeit the ontologies generally provide effective knowledge representation for describing these topics, there are also gaps in the knowledge representation available. For instance, we cannot always describe microphone placement unambiguously. We also identified the need for further extensions describing detailed parameters of music production tools such as audio effects.

Chapter 7

Conclusions and Future Work

This thesis outlines techniques, utilities and applications of semantic audio. We were particularly interested in two application areas: *i)* music production, and the development of an intelligent audio editing environment, and *ii)* music informatics, in particular, the development of Web-based tools that use audio analysis.

We developed a set of ontologies for describing music production, and a set of tools for supporting the analysis of audio content on the Web, as well as to facilitate the use of ontologies by MIR researchers. Furthermore, we built software tools for collecting metadata in music production, prototyping audio analysis applications, and for indexing of music timbre features. We believe that these ontologies and tools are significant contributions to several areas, including audio engineering, the Semantic Web, MIR research, and music informatics. In the rest of this chapter, we provide a quick summary of the novelties and achievements of this work, and then outline future work.

7.1 Summary of this thesis

Several use cases were discussed throughout this text which lead to the hypothesis detailed in Section 1.6, namely, that collecting data in the recording studio should lead to improvements in the applications of semantic audio. This hypothesis guided the development of a set of ontologies and applications.

We first reviewed semantic audio analysis techniques, and outlined a system that can be used to improve the state of the art by representing audio content analysis results as structured data, and by using a knowledge base in the analysis process.

One of the core ideas behind this work is the use of Semantic Web technologies in audio analysis and audio processing. Therefore, we provided a general overview of Semantic Web related technologies. The field of application for these technologies is novel in this work. As a consequence, we found that the adaptation of these technologies with original requirements that are different from ours presents a significant challenge. However, this is balanced by the possibility of opening new areas for research and applications, including the potential generalisation and facilitated applicability of audio analysis techniques. We believe, that this is necessary for Web-based audio analysis tools as well as audio engineering applications using semantic audio to be successful in professional settings.

In Chapter 2 we reviewed information management and knowledge representation issues, including the state of the art of metadata management in musical applications, and discussed how the Semantic Web data model can be used to improve the current state of the art.

We reviewed music information management technologies in Chapter 3, and the ontologies we use, namely the Music Ontology and its extensions. Several new ontologies were introduced for collecting detailed information about music production. This includes the *Studio Ontology* (see Section 4.2) framework, a novel conceptualisation of the recording studio domain.

In Chapter 5, several software architectures were discussed including RDF-MOP, a specialised RDF library which can be used to manage content analysis results and user information in an intelligent audio editing application. The *SAWA framework* discussed in Section 5.3 allows for developing Web-based applications involving semantic audio analysis. Several applications of this framework were demonstrated, which show that a variety of applications can be built using this system. We briefly discussed VamPy, which is used as an environment for prototyping music analysis algorithms, however it can also be used in production as demonstrated in Section 5.4.3.

Finally, the Studio Ontology framework was evaluated by measuring its coverage of a domain specific text corpus. We compared the domain coverage of different ontology frameworks, and assessed the quality of knowledge representation in our framework when describing music production specific use cases and recording scenarios.

7.2 Summary of contributions

The two major contributions of this work are as follows:

- A Semantic Web ontology for describing music production in the recording studio: the Studio Ontology framework (see Section 4.2), as well as set of extensions to the Music Ontology described in Chapter 4. The ontologies contribute towards a larger framework of ontologies depicted in Figure 7.1 for describing music related information.

- A software framework for Web-based audio analysis: SAWA described in Chapter 5, and a library that facilitates the use of evolving ontologies in music production tools.

In addition, the thesis has several minor contributions which extend previous works, or represent collaborative research. Notably, the applications built using the SAWA framework include SAWA-recommender, a system that utilises the audio similarity algorithm discussed in [Levy and Sandler, 2006a] and an improved search technique presented in this work, TempEst,

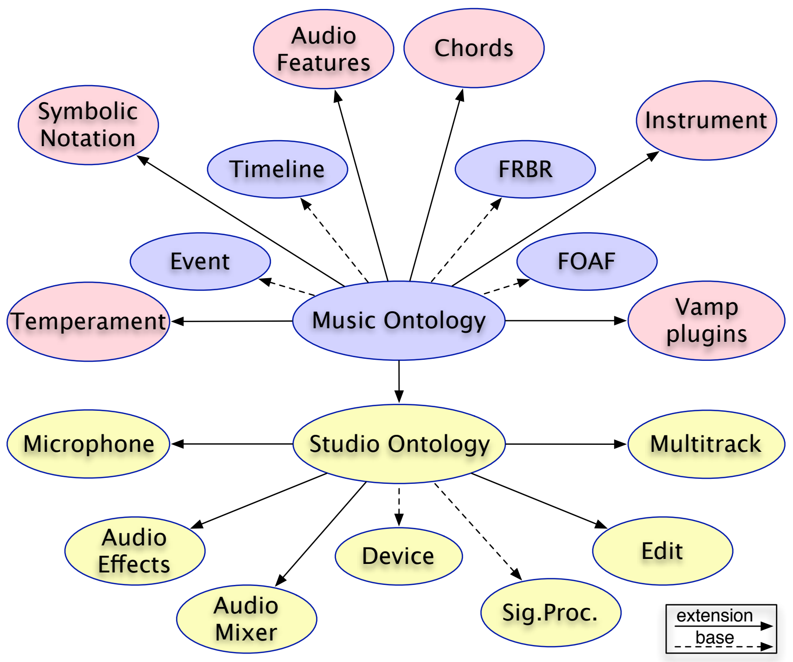


Figure 7.1: Overview of contributions towards a framework for describing music related information. Our contributions include the Studio Ontology, the Temperament Ontology or collaborative work towards an ontology of musical instruments.

a Web-based tool for the estimation of musical temperament of uploaded audio files [Tidhar et al., 2010a], and Hotttabs, an interactive Web application for music learning [Barthet et al., 2011]. VamPy (see Appendix B.2), a Python interface for the Vamp plugin API [Cannam, 2009], provides for writing Vamp plugins using Python, which is becoming a favoured language for rapid development in signal processing and machine learning research. VamPy brings the flexibility of a dynamic scripting language together with the robustness of the C++ API of Vamp plugins, which is supported by many host applications including SAWA. It is a core component in many of our applications.

The Temperament Ontology described in Section 4.6 is designed to describe both automatic temperament classification [Tidhar et al., 2010a] results and standard temperament profiles.

Other contributions include components which can be used to realise the idea of intelligent audio editing using semantic audio and the *Semantic Audio Desktop* introduced in this work (see Section 1.5 and 5.2), and the concept of a *Knowledge Machine* introduced in [Abdallah et al., 2006]. These ideas provide important motivation for our work. They are mentioned here to facilitate the interpretation of the major contributions in the right context. However, detailed discussion of these components deemed outside the scope of this thesis.

A potential application of the software and ontologies frameworks developed here, and the information captured and formalised using these frameworks is in the improvement of music analysis and audio engineering workflows. We hypothesise that intelligent semantic audio applications can take advantage of these frameworks to support engineering decisions. The following section outlines a possible model.

7.3 Knowledge-based audio analysis and processing

When human beings make decisions (whether it is cognition or interpretation of a musical event, or a resulting action) there are several types of different processes at work. These processes rely on a multitude of *percepts*, coming through the senses from the physical world, as well as innate or previously acquired *knowledge*. Although it is not yet known precisely how primary or contextual information coming from separate sources are fused during cognitive processes, we believe that significantly different methodologies are needed to approximate how humans understand and react to sound and music.

We know for instance that the physical layer of auditory perception can be modelled using straightforward signal processing techniques, such as a linear filter that approximates the transfer function of the middle ear, or a bank of overlapping filters to model the behaviour of the Basilar membrane, which has the strongest response for a particular frequency at each different site (for a very recent and thorough overview see e.g.[Meddis et al., 2010]).

Higher-level processes such as the perception of harmony can be modelled within probabilistic frameworks. These are able to incorporate learnt associations between percepts and musical phenomena, which is demonstrated by the success of statistical models and supervised machine learning in areas like chord recognition and music transcription (see Section 1.4.2). Complex probabilistic models can also integrate expert knowledge (such as musicological cues), and information related musical context (e.g. the dependency of chords on musical key), as demonstrated in [Leistikov, 2006], and more recently in [Mauch, 2010].

Every day experience and common sense suggests, however, that some of our decisions are heavily influenced by factual knowledge and strong associated rules. This is exemplified by the revelation, for instance, when false perception is suddenly overridden and corrected by factual knowledge (i.e. when we see or hear something and realise that it can not possibly be true), or when we have justified belief of something (e.g. that we listen to a piece from the western musical tradition, played on the harpsichord) and the search space of our interpretation of the sound is limited accordingly. We therefore believe that any system with the aim of modelling human interpretation of music, or designed to facilitate the process of working with audio, has to incorporate all layers of sensation, perception and cognition, and all associated methodologies that involve signal processing, as well as probabilistic and logical inferences.

The need for using different methodologies and different sources of information is perhaps best explained by using Rasmussen's model of human information processing [Rasmussen, 1983] shown in Figure 7.2. The arch of the diagram represents the information-flow through an individual or a corresponding computational model. The left half of the figure corresponds to the low-level processing of data collected from the environment, (such as sensory stimulus processing) the right half corresponds to resulting actions or some other form of output. Information processing can be divided into three main categories. These categories represent activities at different levels of complexity.

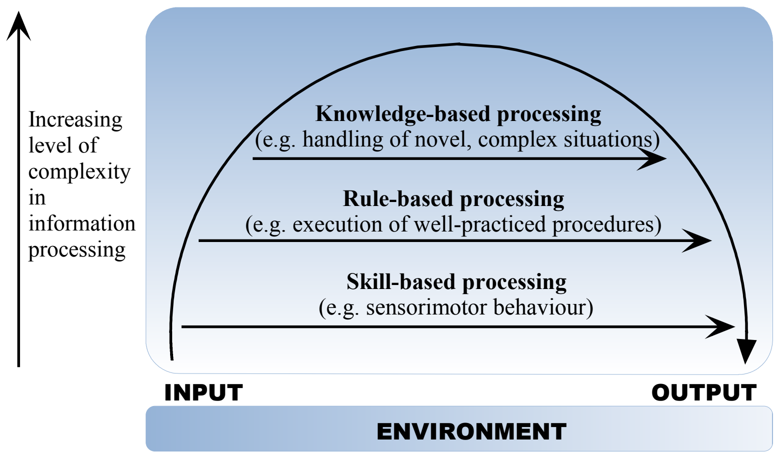

Increasing level of complexity in information processing
Knowledge-based processing
(e.g. handling of novel, complex situations)
Rule-based processing
(e.g. execution of well-practiced procedures)
Skill-based processing
(e.g. sensorimotor behaviour)
INPUT
OUTPUT
ENVIRONMENT

Figure 7.2: Rasmussen's hierarchy of human cognitive processing [Rasmussen, 1983]

The lowest level of the diagram corresponds to *skill-based processing*, such as the perception of simple features of sound. Rule-based audio analysis algorithms and statistical models resulting from supervised learning (inherently encoding association rules in a framework that allows to deal with uncertainty) correspond to the second level of information processing in the diagram: *rule-based processing*. The human analogy is the execution of well-practiced procedural skills, such as the detection of significant events. This works if we accept a well-defined set of assumptions, and expect a system's abilities accordingly. In human cognition, the ability to solve difficult problems and reasoning about unfamiliar events represents the most complex behaviour, *knowledge-based processing*.

A music analysis system which is able to choose the most appropriate algorithm, the correct model or the right parameters given a musical context or some user input, may be seen as a knowledge-based system. In Figure 7.2 we find this at the highest level. Using structural segmentation of music as an example, we may create a set of timbre models independently, using different musical data-sets corresponding to different musical styles. If the association of models and musical styles is encoded in a knowledge-base, the system is able to use contextual information to choose the correct model for a given audio signal. This represents simple factual knowledge which may be obtained by interacting with the user, or as a result of fusing the output of different, possibly probabilistic, inferences working at a lower level.

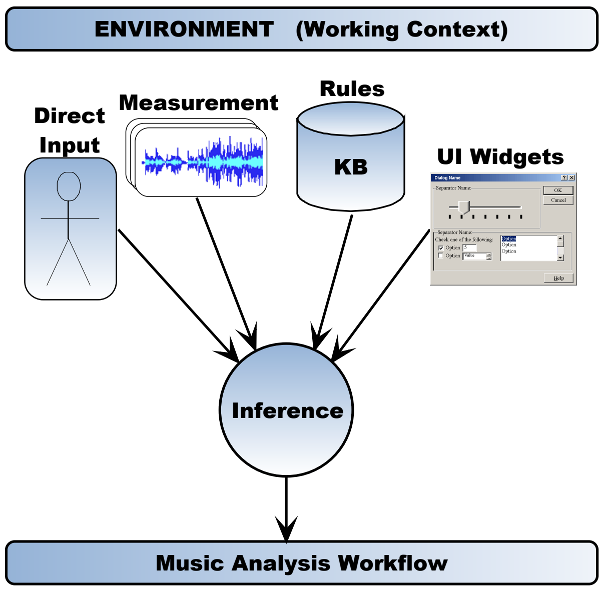


Figure 7.3: Context adaptation in audio analysis (KB: knowledge base, UI: user interface)

Recognising the need for these different levels of processing, and that fact that most high-level audio analysis techniques follow certain typical patterns or workflows, we motivate future work by the *information needs* of a system outlined in Figure 7.3. We postulate that a system that is able to collect information from the recording context is able to infer the most suitable workflow for analysing a piece of music or a simple sound. This, however, requires a complex information management framework, which facilitates the encoding of heterogeneous pieces of information about recording and musical context. The systems described in this work to encode and capture intricate information about recording in the music studio may facilitate the creation of intelligent semantic audio tools using the principles described above.

7.4 Future work

We can divide our future work into different areas, focusing on ontologies, the Semantic Web and Web-based tools, applications in music production and more general goals.

7.4.1 Ontologies

We identified a set of weaknesses and a further requirements in our ontology framework (see Section 6.4). This guides our future work in this area:

- Extensions of the Studio Ontology, to provide detailed knowledge representation of the particularities of a more extensive set of hardware and software audio engineering tools.
- Development of musical instrument ontologies that support the detailed description

of the recording of acoustic instruments, for instance, reference points for microphone placement.

- Harmonisation of the core Device Ontology model with more specific domain ontologies, such as an ontology of audio effects currently in development.
- Better harmonisation of the Music and Studio Ontology frameworks.
- Empirical research and development of methodologies and tools that allow more rigorous logical evaluation of engineering ontologies, lacking a gold-standard and trained domain experts.
- Deployment of ontology based information management in music production tools.

7.4.2 Web-based tools

Our future work in the area of Web-based audio analysis includes extensions and improvements to the SAWA framework:

- On-line deployment of the experimental SPARQL server, and its extension towards a fully automated system.
- Development of clients, such as extensions to Sonic Visualiser or Audacity in order to access SAWA using its SPARQL-endpoint.
- Completion of the SAWA-Experimenter module.
- Evaluation of SAWA-Recommender and the database indexing method on a large collection of audio files.
- Extensions of SAWA-TempEst, providing an interactive graphical user interface, which displays the properties of the recognised tuning systems.
- Deployment of an online "Knowledge Machine" architecture.

7.4.3 General goals

In a broader prospective, we aim to develop tools that support the following goals:

- Improved communication and information sharing between MIR researchers.
- Improved reproducibility of MIR research.
- Generalisation of MIR techniques to facilitate their broader applications in music production and audio engineering.

We aim to develop ontologies closely tied with digital signal processing and machine learning methods to enable the supervision of these techniques in a logic-based environment. The aim is to support the adaptation of MIR algorithms to contextual information such as musical style, the analysed instrument, or the original recording conditions.

In the area of intelligent multitrack music production tools, we aim to further develop the software architectures described in this work. While the architecture and the application concepts will not change, the most significant future work is the extension of our system such that we shall be able to separate the implementation of specific analysis steps, and the workflow description of music analysis tools needed in an intelligent editor. This requires a rule based approach. Extending our framework with an appropriate rule engine, for instance, binding RDF-MOP to a reasoning engine like FactC++[1], shall be completed before we will proceed with further application development. Perhaps, what remains an important research question is to find the most suitable evaluation methods for various systems outlined in this work. This includes ontology evaluation using more sophisticated natural language processing techniques.

7.5 Closing remarks

Opening new opportunities in semantic audio applications is the prime aspiration of this research. I believe that the use of explicitly formalised knowledge about audio engineering processes, capturing information about recording conditions, and the use of Semantic Web technologies have the potential to create new opportunities for research and applications. Aligned with the philosophical framework outlined in Section 1.6, this may indeed trigger entirely new research programmes, or at least extend the hypothesis set of current ones.

A possible new area is in the intersection of recent developments in automatic mixing [Gonzalez, 2010] and the use of high level semantics to create intelligent tools for creative professionals (e.g. audio engineers and music produces). By making knowledge explicit and shared, as opposed to hard coded in tools, we can facilitate the communication between researchers, developers, and engineers, and create tools that can more easily cope with future challenges, requirements, or changing needs. An ambitious idea from an artificial intelligence perspective is the creation of personalised music production tools that learn from human decisions, and use a harmonised framework of digital signal processing, machine learning, knowledge representation and logical inference for decision support and the enhancement of human computer interaction at large.

[1]http://owl.man.ac.uk/factplusplus/

Appendix A

Publications

This appendix includes a list of the author's publications in peer-reviewed journals, conferences, as well as workshops and other publications relevant in this work. Each publication is presented with an outline followed by a statement about the author's contribution.

Outline of publications

Journal Papers

- G. Fazekas, Y. Raimond, K. Jakobson, and M. Sandler. An overview of Semantic Web activities in the OMRAS2 Project. *Journal of New Music Research special issue on Music Informatics and the OMRAS2 Project*, Vol. 39. Issue 4, pp. 295–311, 2010.

  Outline and author's contribution: This paper provides an overview on the development of Semantic Web Ontologies in the OMRAS2 project, and describes research tools data sets and applications that utilise these ontologies. The author's contribution focuses on introducing Semantic Web concepts, providing examples of using Semantic Web ontologies for describing content-based audio features, and discussing the SAWA system (see Section 5.3). Finally the use of ontologies in creating intelligent tools for creative music production and the idea of the Semantic Audio Desktop is introduced.

- D. Tidhar, G. Fazekas, M. Mauch, and S. Dixon. Tempest - harpsichord temperament estimation in a Semantic Web environment. *Journal of New Music Research special issue on Music Informatics and the OMRAS2 Project*, Vol. 39. Issue 4, pp. 327–336, 2010.

  Outline and author's contribution: This paper describes a Semantic Web application for analysing audio in order to estimate the tuning system (temperament) used in the recording. The author's contributions include building the temperament estimator Vamp plugin (TempEst) using VamPy (see Section B.2) and building the back-end architecture utilising the SAWA system (see Section 5.3), as well as writing the relevant sections of the paper.

Conference Papers

- G. Fazekas, T. Wilmering, and M. Sandler. A knowledge representation framework for context-dependent audio processing. *In proceedings of the 42th International Conference of the Audio Engineering Society on Semantic Audio, Ilmenau, Germany, July 22–24*, 2011.

  Outline and author's contribution: This paper presents a general framework for using appropriately structured information about audio recordings in music processing, and shows how this framework can be utilised in multitrack music production tools. The author's contributions include outlining the logical foundations of information management in music production based on the principles discussed in Section 2.2, the development of ontologies (see Section 4.2) and a back-end architecture for using ontologies in audio software (see Section 5.1).

- T. Wimering, G. Fazekas, and M. Sandler. Towards ontological representation of digital audio effects. *In proceedings of the 14th Int. Conference on Digital Audio Effects (DAFx-11), Paris, France, September 19-23*, 2011.

  Outline and author's contribution: This paper discusses the development of ontological representations of digital audio effects and provides a framework for the description of digital audio effects and audio effect transformations. The author's contributions include the provision of motivating ideas for this work, contribution to the ontology design and providing a general framework for an ontology library describing music studio related concepts (see Section 4.2). Note that the audio effects ontology harmonisation approach discussed in Section 4.2.5.3 is not used in this paper.

- G. Fazekas and M. Sandler. The Studio Ontology Framework. *In proceedings of the 12th International Society for Music Information Retrieval (ISMIR-11) conference, Miami, Florida, USA.*, Oct, 2011.

  Outline and author's contribution: This paper introduces the Studio Ontology Framework for describing and sharing detailed information about music production. The primary aim of this ontology is to capture the nuances of record production by providing an explicit, application and situation independent conceptualisation of the studio environment. The author's contribution is the design of the ontology library discussed in Section 4.2.

- S. Kolozali, G. Fazekas, M. Barthet, and M. Sandler. Knowledge representation issues in musical instrument ontology design. *In proceedings of the 12th International Society* *for Music Information Retrieval (ISMIR-11) conference, Miami, Florida, USA.*, Oct, 2011.

Outline and author's contribution: This paper provides an analysis of existing musical instrument classification systems following traditional taxonomic organisation, and examines how well these systems support complex queries related to musical instruments. The author's contributions include the provision of the main motivating ideas for this work, supervising the process of creating ontologies based on existing taxonomies, writing and correcting SPARQL queries and contributing to the introductory and analysis sections of the paper.

- T. Wilmering, G. Fazekas, and M. Sandler. The effects of reverberation on onset detection tasks. *In proceedings of the 128th Convention of the Audio Engineering Society, London, UK*, 2010.

  Outline and author's contribution: This paper discusses the effects of reverberation on onset detection tasks. The author's contributions include the provision of the main motivating ideas for this work, participating in the selection of songs for the evaluation dataset, contributing to the experiment design, and writing the introductory sections of the paper.

- S. Kolozali, M. Barthet, G. Fazekas, and M. Sandler. Towards the automatic generation of a Semantic Web ontology for musical instruments. *In proceedings of the 5th International Conference on Semantic and Digital Media Technologies (SAMT-10) Saarbrcken, Germany*, 2010.

  Outline and author's contribution: This paper describes a novel hybrid system using a formal method of automatic ontology generation for web-based audio signal processing applications. The author's contributions include the provision of the main motivating ideas for this work, supervising the system and the experiment design and contributing to sections on audio signal processing.

- G. Fazekas and M. Sandler. Novel methods in information management for advance audio workflows. *In proceedings of 12th International Conference on Digital Audio Effects (DAFx-09), Como, Italy*, 2009.

  Outline and author's contribution: This paper discusses architectural aspects of a software library for unified metadata management in audio processing applications. The author's novel contribution is the idea of using Meta Object Protocols for building Ontology-based tools in audio software (see Section 5.1).

- D. Tidhar and G. Fazekas and S. Kolozali and M. Sandler. Publising Music Similarity Features on the Semantic Web. *In proceedings of the 10th International Society for Music Information Retrieval (ISMIR-09) conference, Kobe, Japan, Oct*, 2009.

Outline and author's contribution: This paper describes the process of collecting, organising and publishing a large set of music similarity features produced by the SoundBite playlist generator tool, and a recommender application utilising this dataset. The author's contributions include writing D2R mappings for creating a SPARQL end-point, developing a method for indexing the dataset (see Section B.1), and building the recommender described in Section 5.4.2.

- G. Fazekas, C. Cannam, and M. Sandler. Reusable metadata and software components for automatic audio analysis. *In proceedings of the IEEE/ACM Joint Conference on Digital Libraries (JCDL'09) Workshop on Integrating Digital Library Content with Computational Tools and Services, Austin, Texas, USA*, 2009.

  Outline and author's contribution: This paper argues for the need of modularity through interoperable components and data publishing methods in MIR applications. The author's contributions include fusing several software tools and ontologies developed in the OMRAS2 project to demonstrate the utilities of these systems in a Semantic Web application framework (see Section 5.3).

- G. Fazekas and M. Sandler. Ontology based information management in music production. *In proceedings of the 126th Convention of the Audio Engineering Society, Munich, Germany*, 2009.

  Outline and author's contribution: In this paper, we use ontologies to associate metadata, captured during music production, with explicit semantics. The collected data is used for finding audio clips processed in a particular way, for instance, using engineering procedures or acoustic signal features. The author's contribution is in the novel application of the Resource Description Framework and Semantic Web ontologies for representing data about music production.

- G. Fazekas, Y. Raimond, and M. Sandler. A framework for producing rich musical metadata in creative music production. *In proceedings of the 125th Convention of the Audio Engineering Society, San Francisco, USA*, 2008.

  Outline and author's contribution: In this paper, we propose a framework for producing and managing meta information about a recording session, a single take or a subsection of a take. As basis for the necessary knowledge representation we use the Music Ontology with domain specific extensions. We provide examples on how metadata can be used creatively, and demonstrate the implementation of an extended metadata editor in a multitrack audio editor application. The author's main contribution is the design of a data entry interface in the open source audio editor Audacity (see Section 5.2.2), to capture metadata and structuring it using the Music Ontology (see Section 4.1).

- G. Fazekas and M. Sandler. Structural decomposition of recorded vocal performances and its application to intelligent audio editing. *In proceedings of the 123rd Convention of the Audio Engineering Society, New York, USA*, 2007.

Outline and author's contribution: This paper describes a new approach to extract both low and high level hierarchical structure from vocal tracks of multi-track master recordings. Contrary to most segmentation methods for polyphonic audio, we utilise extra information available when analysing a single audio track. The author's main contribution is the novel approach of analysing individual instrument track in a multitrack recording context, and the use of a reduces size similarity matrix based on comparing quasi-stationary segments of audio content.

- G. Fazekas and M. Sandler. Intelligent editing of studio recordings with the help of automatic music structure extraction. *In proceedings of the 122nd Convention of the Audio Engineering Society, Vienna, Austria*, 2007.

  Outline and author's contribution: This paper describes the development of new tools that allow an audio engineer to navigate multitrack recordings using a hierarchical music segmentation algorithm. Segmentation of musical audio into intelligible sections like chorus and verses is discussed and an overview of segmentation by timbre is provided. The author's main contribution is the implementation of the structural segmentation algorithm in Audacity, and the development of a navigation interface that utilises automatic audio segmentation.

White Papers, Posters and Miscellaneous Publications

- M. Barthet, A. Anglade, G. Fazekas, S. Kolozali, R. Macrae. Music recommendation for music learning: Hotttabs, a multimedia guitar tutor. *In proceedings of the 2nd Workshop on Music Recommendation and Discovery (WOMRAD'11) in conjunction with ACM RecSys, Chicago, USA, Oct. 23*, 2011.

  Outline and author's contribution: This paper presents Hotttabs, an online music recommendation system dedicated to guitar learning. The author's main contribution is developing the ontology based data fusion approach (see Section 5.4.5) that utilises the Music Ontology to combine diverse data sources and building the back-end architecture using the SAWA system (see Section 5.3).

- S. Kolozali and M. Barthet and G. Fazekas and D. Tidhar and M. Sandler The musical instrument ontology. *Presented at the Digital Music Research Network Workshop*, London, UK, Dec. 2010.

  Outline and author's contribution: This poster discusses progress in the design of a musical instrument ontology. The author's contributions include the provision of the main motivating ideas for this work and supervising the process of creating the ontology.

- G. Fazekas and D. Tidhar TempEst - Temperament estimation Web service. *Presented at the Digital Music Research Network Workshop*, London, UK, Dec. 2010.

  Outline and author's contribution: This poster discusses progress in the design of a musical instrument ontology. The author's contributions include the provision of the main motivating ideas for this work and supervising the process of creating the ontology.

- D. Tidhar, G. Fazekas, M. Mauch, and S. Dixon. Temperament Estimation as an MIR task. *Presented at the 11th International Society for Music Information Retrieval Conference (ISMIR-10), Late-breaking session*, Utrecht, The Netherlands, Aug. 9–13, 2010.

  Outline and author's contribution: This demonstration showcases the TempEst Web application discussed in Section 5.4.3. The author contribution is building the Web application and the TempEst VamPy plugin used by this application.

- D. Tidhar and G. Fazekas and M. Sandler The Temperament Ontology. *Presented at the Digital Music Research Network Workshop*, London, UK, Dec. 2009.

  Outline and author's contribution: This poster presents the Temperament Ontology (see Section 4.6) designed collaboratively relying on musicological knowledge and use cases provided by the first author. The author contribution is the implementation and publishing of the ontology and its use in related software systems.

- C. Cannam and G. Fazekas and K. Noland A Demonstration of Sonic Visualiser. *Presented at the Special SIGMUS Symposium*, Tokyo, Japan, Nov. 2009.

  Outline and author's contribution: This poster presents Sonic Visualiser. The author contribution is demonstrating the development of Vamp plugins using VamPy during the demo session.

- G. Fazekas and C. Cannam and M. Sandler A Simple Guide to Automated Music Analysis on the Semantic Web. *Centre for Digital Music white paper*, Apr, 2009.

  Outline and author's contribution: This paper describes the first prototype of SAWA (see Section 5.3), a simple Web-based system for automated audio analysis. The author's contributions include fusing several software tools and ontologies developed in the OMRAS2 project to demonstrate the utilities of these systems in a Semantic Web application framework.

- G. Fazekas and M. Sandler. Uncovering the details of music production using ontologies. *Presented at the Unlocking Audio 2 Conference, March 16-17 London, UK*, 2009.

  Outline and author's contribution: This paper argues for the need for structured data representations in music production applications and outlines the discrepancies in existing metadata management frameworks. The author's contributions include a review of metadata standards and the design of an ontology based systems to harmonise these overlapping frameworks.

- G. Fazekas. Information Interaction in Context (conference review). *Published in The Informer – BCS Information Retrieval Specialist Group (IRSG)*, (30):4 pp. 2–4, 2009.

  Outline and author's contribution: This article provides a review of the IIiX'08 conference and discusses key concepts in related to the role of context in information seeking.

- G. Fazekas and M. Sandler. Ontology based information management in music production. *Presented at the Digital Music Research Network Workshop*, Dec, 2008.

  Outline and author's contribution: In this poster, the use of ontologies is described to associate metadata captured during music production with clear semantics. The author's main contribution is in the novel application of the Resource Description Framework and Semantic Web ontologies for representing data about music production.

Appendix B

Components of SAWA

This appendix describes two components of the SAWA system that are vital in several of its applications and essential in understanding its operation. While SAWA is built mainly on collaborative work (see Section 5.3), the following components are unique contributions by the author that are not discussed elsewhere.

B.1 Similarity assessment in SAWA-Recommender

SAWA-Recommender (Section 5.4.2) uses a widely adopted method of modelling the overall timbre of a recording by first extracting frame-wise Mel-Frequency Cepstral Coefficients, and then modelling the overall timbre of the recording by fitting a single Gaussian to the resulting MFCC vectors [Logan and Salomon, 2001]. This method makes several simplifying assumptions. For one, it ignores musical structure, and also the fact that the distribution of timbre features is not necessarily Gaussian. A solution to these problems may be the use of a mixture of Gaussians (MoG) or a sequence of Gaussians fitted on coherent segments (for instance, a single Gussian representing each bar or each structural segment) of the music for modelling a track. However, approaches to estimate similarity between these models such as MonteCarlo sampling are computationally expensive. A more detailed discussion on timbre models and the effects of the above assumptions can be found, for instance, in [Aucouturier, 2006], [Casey and Slaney, 2006] and [Casey et al., 2008].

Albeit modelling timbre using a single Gaussian is a very simple approach, it was shown in [Mandel et al., 2005] that it can perform comparably to mixture models when computing similarity between tracks. It was also shown to be effective and computationally efficient for finding similar songs in personal music collections in [Levy and Sandler, 2006a]. An important advantage of using this model is that the similarity between two tracks can be computed using closed form expressions, such as the Jensen-Shannon (JS) or Kullback-Leibler (KL) divergences. Here, following [Levy and Sandler, 2006a], we use the symmetrised KL distance given in Equation B.1, where p and q are Gaussian distributions, with μ mean and

Σ covariance, and d is the dimensionality of the feature vectors. Besides modelling tracks using a single Gaussian, a further simplifying assumption is introduced by using Gaussians with diagonal covariance. This has the advantage of reducing the number of multiplications and eliminating the need for matrix inversion for each distance computation.

$$\begin{aligned} KL_s(p\|q) &= 2KL(p\|q) + 2KL(q\|p) \\ &= tr(\Sigma_q^{-1}\Sigma_p + \Sigma_p^{-1}\Sigma_q) \\ &\quad +(\mu_p - \mu_q)^T(\Sigma_q^{-1} + \Sigma_p^{-1})(\mu_p - \mu_q) - 2d \end{aligned} \tag{B.1}$$

Linear search was deemed sufficient for personal collection management, however the size of our current database is over 150,000 tracks and it is expected to grow. Therefore, for efficient query evaluation in a recommender system, we need a method which avoids explicit calculation of similarity between a query song and each entry in our feature database. This requires indexing the database in such a way that a model representing a query can be compared with index entries, as opposed to each element of the database. Creating such an index table for a database of timbre models requires *i)* an unsupervised clustering algorithm, since manual labelling required for supervised learning would not be feasible, and *ii)* a method that can work with Gaussian models and the associated divergences, for instance, the symmetrised KL divergence.

A possible way of creating an indexing method for our database is the use of the Self-Organising Map (SOM) [Kohonen, 1995], commonly used in data visualisation, but also as a method of Vector Quantisation (VQ) [Vignoli and Pauws, 2005]. The original Self-Organising Map can be thought as a two dimensional array of nodes (neural-network cells), where each node is associated with a model vector. These vectors are iteratively trained to recognise (become close to) classes of input signal patterns in a competitive and unsupervised fashion. Only one node is activated at a time corresponding to each input. Typically, the node that is the closest to the input in Euclidean space, is called the *best matching unit* (BMU) or the *winner*.

The locations of the responses in the array tend to become ordered in the learning process as if some meaningful nonlinear coordinate system for the different input features were being created over the network [Kohonen, 1995]. More generally, a SOM can be used to map between data of different dimensionality, moreover the target space or the topology of the map does not necessarily have to be in a Euclidean space [Ritter, 1999]. The basic training algorithm for the SOM can be outlines as follows. Given a ordered set of map nodes i associated with a location, and model vectors m_i, the model vectors are initialised randomly, or using more advanced techniques such as principal component analysis. During training, a randomly selected input vector $x(t)$ is compared with all model vectors at each iteration t. The best matching unit is selected by choosing the node that is closest to the input. Then, the model vectors are updated using the adaptation rules described in the following equation (B.2),

$$m_i(t) = \begin{cases} m_i(t-1) + \alpha(t)[x(t) - m_i(t-1)] & \text{if } i \in N_c(t) \\ m_i(t-1) & \text{otherwise,} \end{cases} \tag{B.2}$$

where $\alpha(t)$ is a weight that depends on the distance from the BMU, and $N_c(t)$ is a neighbourhood function that defines which nodes around the BMU are updated. Typically, the weight $\alpha(t)$ is decreased in every iteration given by a predefined learning rate. The initial weight may be a linear function of distance, or computed using a Gaussian function, and $N_c(t)$ may be abandoned in favour of updating all nodes in each iteration. Training a SOM continues either until some convergence criterion has been reached, or a fixed number of training iterations are used. Finally, if we wish to use the map for indexing, each input vector in a data set must be assigned to the closest node on the trained map.

Various applications of the SOM have been explored in natural language processing, information retrieval as well as music information retrieval and collection management. For instance, Honkela [1997] uses a SOM to create word category maps, the use of the map has also been shown to scale up to millions of documents in the organisation of large text archives in [Kohonen, 1998]. Organisation and visualisation of music collections using a SOM is introduced in [Rauber et al., 2003], while Pampalk et al [2004b] use a hierarchical SOM for clustering and visualisation of sounds or samples, finally Vembu and Baumann [2005] applies a SOM in music recommendation. These approaches however were training the SOM directly on the features, as opposed to Gaussian models of features. It can easily be noticed, that the SOM training algorithm is not limited to using feature vectors in a Euclidean space. The model vectors associated with the nodes can be substituted with arbitrary models, as long as the adaptation and distance calculation between these models can be substituted in Equation B.2. Given our simple model of MFCCs represented by multivariate Gaussian mean and variance vectors, the SOM can easily be extended by having each node linked with the two parameter vectors separately. Finding the best matching unit is then performed by computing the KL distance instead of Euclidean distance, while the model vectors are separately updated during training using the update rules discussed previously. For the more general case, where Gaussians with full covariance are used to model each track, the centroid of the symmetrised KL distance [Veldhuis, 2002] has to be computed in order to update the model. However, we need to deal with the issue that the KL divergence is asymmetric, therefore the symmetrised centroid need to be found with respect to a left and right centroid of the KL divergence. This can be well approximated by the arithmetic or normalised geometric mean of the left and right centroid as empirically shown in [Veldhuis and Klabbers, 2003], or more precisely solved as an optimisation problem. An efficient algorithm for finding the symmetrised centroid is presented in [Nielsen and Nock, 2009]. A similar model for data visualisation using centroid approximation was later presented in [Schnitzer et al., 2010], while the information geometry framework of [Cont et al., 2011] provides an even more general extension of this idea using generic families of Bregman divergences (such as the KL divergence) and extends the approach to many machine learning problems related to audio similarity analysis.

An important property of the SOM is that it can easily accommodate the need for adding new elements to the modelled data set, i.e. the set of songs in our database. The idea of growing and hierarchically organised (multilayer) SOM is discussed in [Dittenbach et al., 2001] and [Pampalk et al., 2004b]. There is no reason why this cannot be applied in our case. The SOM training algorithm itself can be used in an online fashion. However, it is a more common case that a large database is used as a starting point, to which new elements are added later in batch. Therefore we use batch training, which avoids the need for reassigning the song models to map units after each training iteration. We set the number of nodes to a fixed initial value (100 in case of our current database). For inserting new elements, we adopt the criterion described in [Pampalk et al., 2004b], where the mean quantisation error across the map is used to decide whether new nodes need to be added to the map. The error in our case is computed by Equation B.3 for each node, while the mean of these values with regard to the whole map describes how well all units together represent the data.

$$ME_i = \frac{1}{|U_i|} \times \sum_{k \in U_i} KL_s(p_k \| q_i) \tag{B.3}$$

where $U_i = \{k | c_k = i\}$ is the set of song models (p) assigned to node i, and q represents the parameter vectors assigned to the node.

Since the one dimensional topology is sufficient in a database indexing use case, we further simplify the map by using a set of nodes arranged in one dimensions, while keeping an the important property of the map which ensures that neighbouring nodes are associated with features that are also close in the original feature space. To create a continuous space, avoiding the problem of nodes on the edges having less neighbours then other nodes, we use a circular topology. New nodes then can be inserted any point, where the error above exceeds a suitable threshold. This model, although it is yet to be tested on a database of larger size, can potentially scale up to millions of songs, and with the introduction of multiple layers, a balanced performance can be maintained.

Given a database of Gaussian timbre models, we partition the data space by similarity to form self-similar groups of songs. These groups or clusters can then be used to index the database using the model described above. Query processing and database search is performed in three steps. First, we extract features from the audio files representing a query, and compute the Gaussian model parameters for these features. Next, for optimised search, the models are matched against the above self-organising model trained on the whole database. We can limit the search space by first choosing the best matching unit based on its proximity to the query song, and then calculate the distance between the query and all entries assigned to the selected node. Hence, the number of direct similarity calculations is greatly reduced. To avoid the problem of query songs only marginally closer to one node than another, we extend the search space to nodes in the direct neighbourhood of the best matching unit. Finally, the results a ranked based on proximity to the query, and these results are returned.

B.2 VamPy: A rapid prototyping tool for SAWA

VamPy provides Python bindings for the Vamp plugin API [Cannam, 2009] and plays a crucial role in the audio analysis functionalities of the SAWA framework. For example, it provides for the implementation of the analysis engine in the SAWA-TempEst system (see Section 5.4.3) and enables the use of Python's numerical as well as Semantic Web libraries that are vital in this system. It also provides for prototyping algorithms in a high-level dynamic programming language, which is important for instance in SAWA-Experimenter (see Section 5.4.4).

Vamp plugins and VamPy The Vamp plugin system consists of separate host and plugin APIs. This facilitates the development of audio analysis plugins and host applications independently. The Vamp plugin API is provided in C/C++ requiring the developer to write plugins in a statically typed and compiled language, which is not always the best choice for experimental and research applications or complex semantic audio applications. VamPy bridges the Vamp plugin API with a dynamically typed and interpreted language: Python, which was chosen for its support for numerical analysis and scientific computation, the availability of numerous Semantic Web libraries, and its growing use in the audio research community. In order to run Python scripts in a Vamp host execution context, VamPy embeds the Python interpreter. It also extends the interpreter with data structures defined by the Vamp C++ API within a single shared library. The software architecture of VamPy and its use in semantic audio tools is illustrated in Figure B.1.

Function call translation using meta-programming Vamp plugins consists of a set of callbacks, for instance, a `process()` method called by a host iteratively passing small blocks of audio. This method may return the results of analysis available at the end that processing block, however a plugin may accumulate samples to execute non-causal algorithms and return the results when the host calls its `getRemainingFeatures()` method. Other callbacks include methods for querying plugin metadata and initialisation.

VamPy uses function call translation with meta-programming to redirect calls from a C++ host application to an appropriate Python class realising a VamPy plugin. This is implemented generically using a set of overloaded C++ templates wrapping Python's embedding C API. These methods first check if a VamPy plugin instance realises the requested

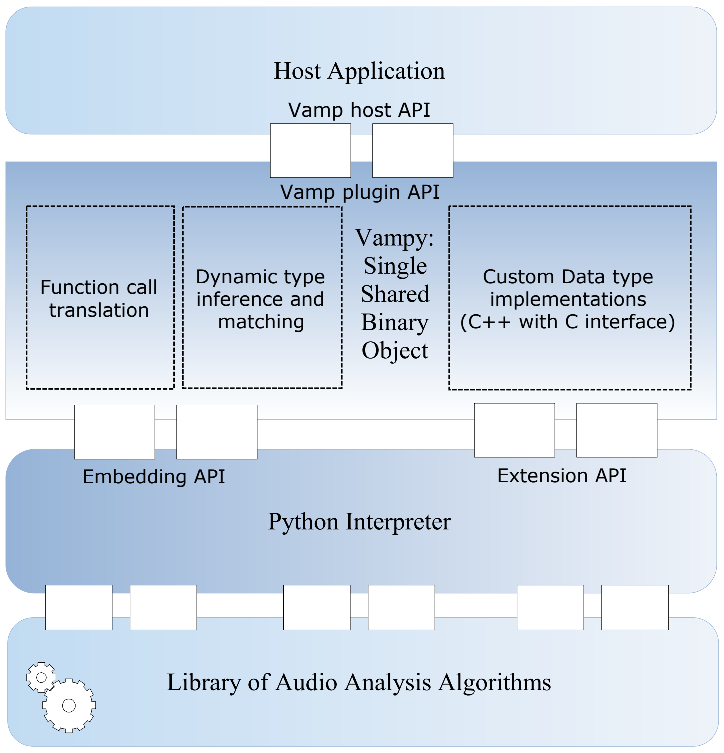

Host Application
Vamp host API
Vamp plugin API
Function call translation
Dynamic type inference and matching
Vampy: Single Shared Binary Object
Custom Data type implementations (C++ with C interface)
Embedding API
Extension API
Python Interpreter
Library of Audio Analysis Algorithms

Figure B.1: Audio Analysis Software Architecture Using VamPy

Vamp plugin method, prepare the arguments to be interpreted by the VamPy plugin, and convert any returned Python data structures to C++ data structures.

Dynamic type inference Python is a dynamically typed language, that is, the programmer is not forced to declare variable types strictly and specifically. The Vamp API however is defined in C++ using static data types, which forces the Python programmer to have detailed knowledge of the C++ API. VamPy relaxes this requirement by using a runtime type inference mechanism. VamPy can convert any suitable Python data object to the appropriate C++ data type expected by a Vamp plugin host, including arrays used by the NumPy scientific library. This type conversion is dynamic, and it is decided based on the plugin context and the expected data type defined by the Vamp plugin API in that context. The mechanism also takes advantage of the high-level Python number sequence and mapping protocols and the taxonomical relations of numerical types defined by the interpreter. For instance, any returned value will be converted to a vector of the appropriate element type when the expected return type is a sequence of values. This allows the programmer to omit type conversions in Python code, even when for instance a single element list (vector) would be returned, and just return the value instead of a single element list. VamPy also extends Python with the data types defined by the Vamp plugin API. This extension module is included in the VamPy shared library which is loaded by the embedded Python interpreter, therefore it doesn't need to be installed separately. Type conversion can be controlled specifically for each plugin using a flag. VamPy supports the use case of prototyping C++ Vamp plugins in Python by using a strict type conversion mechanism. In this mode, an error message is issued if the Python object does not correspond to a C++ type according to a strict one-to-one mapping. This mapping can be briefly outlined as follows:

- Numerical types require direct correspondence between Python and C++ types where available (e.g. a C++ float is mapped to a Python float),
- Data structures defined in the Vamp Plugin API require a type exported by the VamPy extension module, for instance:

  `Vamp::FeatureSet() -> vampy.FeatureSet()`

  `Vamp::RealTime() -> vampy.RealTime()`

Vamp uses `RealTime` time stamps to indicate the position of a processing block passed to the plugin or the position of any returned features relative to the start of the audio. The `RealTime` data type consists of two integers to represent time to nanosecond precision. VamPy provides a Python compatible representation of this type which can be imported and used in any VamPy plugin.

The use of these mechanism as well as the use of generic functions allow for writing a thin generic wrapper around the Vamp API itself such that the specific translation code required to make use of the Vamp API in Python is automatically generated by the C++ compiler. The Vamp plugin initialiser function for instance is implemented in VamPy as shown in Figure B.1.

```
bool PyPlugin::initialise(size_t channels, size_t stepSize, size_t blockSize)
{
 return genericMethodCallArgs<bool>("initialise",channels,stepSize,blockSize);
}
```

Listing B.1: Example of a Vamp function call implementation

Interfaces and application VamPy supports three interfaces for passing audio data to the `process()` function. This includes a *legacy interface* which passes a list of list of values (the native Python representation of a matrix) to the plugin corresponding to the time or frequency domain samples for each channel. Using the *buffer interface*, both time and frequency domain plugins are passed a list of objects describing shared memory buffers where each buffer corresponds to an audio channel. Finally when using the *array interface*, VamPy passes a list of Numpy arrays to the process corresponding to each audio channel. In this case, time Domain plugins are passed an array of `numpy.float32` values where the array size is equal to the block size, while frequency domain plugins are passed an array of $blockSize/2+1$ `numpy.complex64` values. The execution of VamPy plugins, for instance the selection of interfaces and the behaviour of the type inference mechanism is controlled using a set of flags described in the VamPy documentation. The following example shows the declaration of a VamPy plugin for computing Mel-Frequency Cepstral Coefficients using the Numpy array process interface[1].

```
	class PyMFCC(melScaling):
		def __init__(self,inputSampleRate):
			self.vampy_flags = vf_DEBUG | vf_ARRAY | vf_REALTIME
```

Listing B.2: Using run-time flags to control VamPy plugins

The described architectures supports the use of audio analysis algorithms implemented as monolithic processing blocks such as a plugin or a complex function with well defined inputs, outputs and ontological relations. However, in order to implement adaptive components in intelligent semantic audio tools, we need finer granularity and more specific control over how feature extraction may be performed. For instance, by decomposing plugins into smaller DSP components which could be selected for each particular application. This constitutes future work. An example VamPy plugin for note onset detection using high-frequency content can be found in Section C.1.

[1]The full plugin implementation is available online
`http://vamp.svn.sourceforge.net/viewvc/vamp/vamp-vampy/trunk/ExampleVamPyplugins/PyMFCC.py`

Appendix C

Code listings

C.1 A simple onset detector plugin using VamPy

```python
from numpy import *
from vampy import *

class VamPyOnsetExample:

   def __init__(self,inputSampleRate):
      self.vampy_flags = vf_DEBUG | vf_ARRAY | vf_REALTIME
      self.m_inputSampleRate = inputSampleRate
      self.m_stepSize = 0
      self.m_blockSize = 0
      self.m_channels = 0
      self.threshold = 0.05
      return None

   def initialise(self,channels,stepSize,blockSize):
      self.m_channels = channels
      self.m_stepSize = stepSize
      self.m_blockSize = blockSize
      self.wasGreater = False
      self.prevTime = 0
      self.prev=0.0
      return True

   def reset(self):
      return None

   def getMaker(self):
      return 'George Fazekas'

   def getName(self):
      return 'Onset plugin'

   def getIdentifier(self):
      return 'vampy-onset'

```

```
    def getDescription(self):
        return 'An onset detector using high frequency content'

    def getMaxChannelCount(self):
        return 1

    def getInputDomain(self):
        # TimeDomain or FrequencyDomain
        return FrequencyDomain


    def getOutputDescriptors(self):
        # describe what the plugin's output will be like
        Generic = OutputDescriptor()
        Generic.identifier = 'vampy-onset-output'
        Generic.name = 'vampy onset'
        Generic.description ='Onset locations'
        Generic.hasFixedBinCount=True
        Generic.binCount=0
        Generic.hasKnownExtents=False
        Generic.isQuantized=False
        Generic.sampleType = VariableSampleRate
        Generic.unit = ''
        return OutputList(Generic)

    def getParameterDescriptors(self):
        # describe the parameters of the algorithm
        threshold = ParameterDescriptor()
        threshold.identifier='threshold'
        threshold.name='Energy threshold'
        threshold.description='Energy threshold'
        threshold.unit='v'
        threshold.minValue=0
        threshold.maxValue=1
        threshold.defaultValue=0.05
        threshold.isQuantized=False
        return ParameterList(threshold)

    def setParameter(self,paramid,newval):
        if paramid == 'threshold' :
            self.threshold = newval
        return

    def getParameter(self,paramid):
        if paramid == 'threshold' :
            return self.threshold
        else:
            return 0.0

    def process(self,inputbuffers,timestamp):
        # features are computed in this function
        length = self.m_blockSize * 0.5 + 1
        sampleRate = self.m_inputSampleRate
        w = array(xrange(length)) / length

        complexSpectrum = inputbuffers[0]
        magnitudeSpectrum = abs(complexSpectrum) / length
        weightedSpectrum = w * magnitudeSpectrum

        tpower = sum(weightedSpectrum)
        peak = False
        greater = False

        if tpower > self.prev :
            greater = True

        if tpower > self.threshold :
            if self.wasGreater and not greater :
                peak = True

        # return features in a FeatureSet()
        output_featureSet = FeatureSet()
        if peak :
            output_featureSet[0] = Feature()
            output_featureSet[0].timestamp = self.prevTime
            output_featureSet[0].hasTimestamp = True

        self.wasGreater = greater
        self.prevTime = timestamp
        self.prev=tpower

        return output_featureSet
```

Listing C.1: A simple onset detector using VamPy

C.2 A representation of typed RDF literals using a Meta-Object Protocol

```
#ifndef __TYPEDLITERAL__
#define __TYPEDLITERAL__

#include "HashTypes.h"
#include "ontObject.h"
#include "typedLiteralBase.h"
#include "ObjectHandlers.h"

/* generic typed literal class that knows its type */
template<typename W,eMappedTypes D>
class typedLiteral : public typedLiteralBase
{
public:

    /* construct from pointer to value W */
    typedLiteral(W* newValue = NULL, bool proxy = false,
        objectHandler* h = ontObject::defaultHandler) :
    typedLiteralBase(h), mValueProxy(proxy)
```

```
{
    reset();
    if (!newValue) pValue = new W();
    else
    {
        if (!mValueProxy) pValue = new W(*newValue);
        else pValue = newValue;
    }
    flagValueChange();
}

/* construct from reference of value W */
typedLiteral(W& newValue,bool proxy = false, objectHandler* h = ontObject::defaultHandler) :
typedLiteralBase(h), mValueProxy(proxy)
{
    reset();
    if (!restrictValue(&newValue))
    {
        if (!mValueProxy) pValue = new W(newValue);
        else pValue = &newValue;
    }
    else { pValue = new W(); }
    flagValueChange();
}

/* construct from string literal through the interface */
typedLiteral(literalInterface& lit,objectHandler* h = ontObject::defaultHandler) :
typedLiteralBase(h,lit), mValueProxy(false)
{
    reset();
    mXML = (lit.XML == 1) ? true:false;
    mInterface.set(NULL,NULL,mXML,D);
    flagLiteralChange();
}

/* construct from wxString */
typedLiteral(const wxString& lit, bool validate = true, objectHandler* h = ontObject::
     defaultHandler) :
typedLiteralBase(h,lit), mValueProxy(false)
{
    reset();
    if (!validate) flagLiteralChange();
    else
    {
        W* tempValue = new W();
        convertString(tempValue,pLiteralString);

        if (!restrictValue(tempValue))
        {
            pValue = tempValue;
            flagUpdated();
        }
        else
        {
            pValue = new W();
```

```
                delete tempValue;
            }
        }
    }

    /* destructor: delete owned value object */
    ~typedLiteral()
    { if (pValue and !mValueProxy) delete pValue; }

/* interface: */

    /* return the mapped type ID */
    const eMappedTypes getMappedTypeID() { return D; }

    /* return a constant reference to the cached value */
    const W& getValue()
    { updateByGetFunctor(); updateValue(); return *pValue; }

    /* return an reference that may be used for changing the value */
    /* use flagValueChange whenever the value is changed externally*/
    W& getValue(bool edit)
    {
        updateByGetFunctor();
        updateValue();
        if (edit) flagValueChange(); // the value may be changed externally
        return *pValue;
    }

    /* return a reference to the owned literal string */
    const wxString& getLiteral()
    { updateByGetFunctor(); updateLiteral(); return *pLiteralString; }

    /* set the cached value as proxy, the object becomes a wrapper */
    void proxyValue(W& newValue) { setValue(newValue,true); }

    /* modify the cached value */
    void setValue(W& newValue, bool proxy = false)
    {

        if (!restrictValue(&newValue))
        {
            if (proxy)
            {
                if (pValue and !mValueProxy) delete pValue;
                pValue = &newValue;
                mValueProxy = true;
            }
            else
            {
                if (pValue and !mValueProxy) delete pValue;
                pValue = new W(newValue); //*pValue = newValue;
                mValueProxy = false;
            }

            flagValueChange();
            updateBySetFunctor();

        }
        else
        {
            throw typeException("met restriction on value");
        }
    }

    /* modify the literal string owned by the object */
    void setLiteral( const wxString& newLit, bool validate = true)
    {
        if (validate)
        {
            W* tempValue = new W();
            convertString(tempValue,&newLit);

            if (!restrictValue(tempValue))
            {
                *pLiteralString = newLit;
                if (pValue) delete pValue;
                pValue = tempValue;
                flagUpdated();
                updateBySetFunctor();
            }
        }
        else
        {
            *pLiteralString = newLit;
            flagLiteralChange();
            updateBySetFunctor();
        }
    }

    /* update interface : this is called when we access the interface through the object handler */
    void updateInterface()
    {
        updateByGetFunctor();
        updateLiteral();
        if (mUpdateInterface) mInterface.set(pLiteralString,NULL,mXML,D);
        mUpdateInterface = false;
    }

    /* update and return const pointer to the RDF interface */
    const literalInterface* getInterface()
    {
        updateInterface();
        return &mInterface;
    }

    /* foreign method bindings */
    void bindGet(FunctorBase<W>* functor) { mGetFunctor = functor; }
    void bindSet(FunctorBase<void,W>* functor) { mSetFunctor = functor; }
    void unbind() { mGetFunctor = 0; mSetFunctor = 0; }

    /* state modifiers */
    void flagLiteralChange() {
         mUpdateLiteral = false; mUpdateValue = true; mUpdateInterface = true; }
    void flagValueChange() {
         mUpdateLiteral = true; mUpdateValue = false; mUpdateInterface = true; }
    void flagUpdated() {
         mUpdateLiteral = false; mUpdateValue = false; mUpdateInterface = true; }

private:

    /* state variables */
    bool mValueProxy;
    bool mUpdateLiteral;
    bool mUpdateValue;
    bool mUpdateInterface;
    bool mXML;

    FunctorBase<W>* mGetFunctor;
    FunctorBase<void,W>* mSetFunctor;

    W* pValue;

    /* implement type specific conversions using template spec. */
    void convertString(W* val, const wxString* lit);
    void convertValue(wxString* lit, W* val);

    /* update the cached value with new literal: called when accessing the value */
    void updateValue()
    {
        if (!pValue) pValue = new W();
        if (mUpdateValue) {
            convertString(pValue,pLiteralString);
            mUpdateValue = false;
        }
    }

    /* update the literal string with the cached value */
    void updateLiteral()
    {
        if (!pValue) return;
        if (mUpdateLiteral) {
            convertValue(pLiteralString,pValue);
            flagUpdated();
        }
    }

    /* call a bounded get function and caches the returned value */
    void updateByGetFunctor()
    {
        if (!mGetFunctor) return;
        W temp = (*mGetFunctor) ();
        // the bounded object shouldn't accept incorrect values
        if (*pValue != temp)
        {
            *pValue = temp;
            // invoke string conversion on demand
            flagValueChange();
        }
    }

    /* call a bounded set function using the cached value as argument */
    void updateBySetFunctor()
    {
        if (!mSetFunctor) return;
        updateValue();
        (*mSetFunctor) (*pValue);
    }

    /* return true if a new value violates an OWL restriction */
    /* implementation is type specific, default is below */
    bool restrictValue(W* Value) { return false; }

    /* reset */
    void reset()
    {
        mGetFunctor = NULL;
        mSetFunctor = NULL;
        mUpdateLiteral = false;
        mUpdateValue = false;
        mUpdateInterface = true;
        mXML = false;
    }
};

/* Declare Some Template Specializations */
template<> void typedLiteral<int,intID>::convertValue(wxString*,int*);
template<> void typedLiteral<int,intID>::convertString(int*,const wxString*);
template<> bool typedLiteral<int,dayID>::restrictValue(W& Value);

#endif
```

Listing C.2: Representing typed RDF literals in RDF-MOP

Appendix D

Namespaces

Namespace prefix	Ontology, Schema or Vocabulary
`af`	Audio Features Ontology
`con`	Connectivity Ontology
`dc`	Dublin Core Metadata Terms
`device`	Device Ontology
`edit`	Edit Ontology
`event`	Event Ontology
`foaf`	Friend of a Friend Vocabulary
`frbr`	Functional Requirements of Bibliographic Records
`fx`	Audio Effects Ontology
`mic`	Microphone (and microphone techniques) Ontology
`mo`	Music Ontology
`mt`	Multitrack Ontology
`mx`	Audio Mixer Ontology
`owl`	Ontology Web Language
`rdf`	Resource Description Framework syntax
`rdfs`	RDF Schema Language
`studio`	Studio Ontology
`spd`	Signal Processing Device Ontology
`tl`	Timeline Ontology
`xsd`	XML schema data types

Table D.1: Namespace prefixes used in RDF listings and First Order Logic sentences

Bibliography

Aamodta, A. and Nygardb, M. (1995). Different roles and mutual dependencies of data, information, and knowledge an AI perspective on their integration. *Data Knowledge Engineering*, 16:191–222.

Abdallah, S. A., Raimond, Y., and Sandler, M. (2006). An ontology-based approach to information management for music analysis systems. *in proc. AES 120th Convention, 2006. May, Paris, France.*

Ackoff, R. L. (1989). From data to wisdom. *Journal of Applied System Analysis*, 16:3–9.

Allen, J. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26:832–843.

Amatriain, X. (2007). CLAM: A framework for audio and music application development. *IEEE Software*, 24(1):82–85.

Amatriain, X., Bonada, J., Loscos, À., Arcos, J. L., and Verfaille, V. (2003). Content-based transformations. *Journal of New Music Research*, 32(1):95–114.

Arndt, R., Troncy, R., Staab, S., Hardman, L., and Vacura, M. (2007). COMM: Designing a well-founded multimedia ontology for the web. In *Proceedings of the 6th International Semantic Web Conference (ISWC'2007), Busan, Korea*, pages 11–15.

Asuncion, A., Welling, M., Smyth, P., and Teh., Y. (2009). On smoothing and inference for topic models. *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, Montreal, Canada.*

Aubert, O., Champin, P.-A., and Prie, Y. (2006). Integration of semantic web technology in an annotation-based hypervideo system. *In Workshop on Semantic Web Annotations for Multimedia (SWAMM'06).*

Aucouturier, J. J. (2006). *Ten experiments on the modelling of polyphonic timbre.* PhD Thesis, University of Paris 6.

Aucouturier, J. J. and Pachet, F. (2002). Music similarity measures: What's the use? *In Proceedings of the 3rd International Conference on Music Information Retrieval (ISMIR'02), October 13-17, IRCAM, Centre Pompidou,* Paris, France.

Aucouturier, J. J., Pachet, F., and Sandler, M. (2005). The way it sounds: Timbre models for analysis and retrieval of polyphonic music signals. *IEEE Transactions of Multimedia*, 7:1028–1035.

Auer, S., Bizer, C., Lehmann, J., Kobilarov, G., Cyganiak, R., and Ives, Z. (2007). DBpedia: A nucleus for a web of open data. In *Proceedings of the International Semantic Web Conference*, Busan, Korea.

Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., editors (2003). *The Description Logic Handbook: Theory, Implementation and Application.* Cambridge University Press, Cambridge, UK.

Balaban, M. (1999). The music structures approach to knowledge representation for music processing. *Computer Music Journal*, 20(2):96–111.

Balaban, M. and Elhadad, M. (1999). On the need for visual formalisms in music processing. *Leonardo*, 32(2):127–134.

Bandara, A., Payne, T. R., De Roure, D., and Clemo, G. (2004). An ontological framework for semantic description of devices. *in Proc. International Semantic Web Conference (ISWC), Hiroshima, Japan.*

Barga, R. S., Fay, D., Guo, D., Newhouse, S., Simmhan, Y., and Szalay, A. (2008). Efficient scheduling of scientific workflows in a high performance computing cluster. In *Proceedings of the 6th international workshop on Challenges of large applications in distributed environments*, CLADE '08, pages 63–68, New York, NY, USA. ACM.

Barthet, M., Anglade, A., Fazekas, G., Kolozali, S., and Macrae, R. (2011). Music recommendation for music learning: Hotttabs, a multimedia guitar tutor. *2nd Workshop on Music Recommendation and Discovery (WOMRAD'11) in conjunction with ACM RecSys, Chcago USA, Oct. 23.*

Bartsch, M. and Wakefield, G. (2005). Audio thumbnailing of popular music using chroma-based representations. *IEEE Transactions on Multimedia*, 7(1):96–104.

Beckett, D. (2008). The Redland RDF libraries. Availble online: `http://librdf.org/`.

Beenham, D., Schmidt, P., and Sylvester-Bradley, G. (2000). XML based dictionaries for MXF/AAF applications. Technical report, Sony Broadcast and Professional Research Laboratories, UK.

Bello, J., Daudet, L., Abdallah, S., Duxbury, C., Davies, M., and Sandler, M. (2005). A tutorial on onset detection in music signals. *IEEE Transactions on Speech and Audio Processing*, 13(5):1035–1047.

Berners-Lee, T. (2006). Linked data. Availble online: http://www.w3.org/DesignIssues/LinkedData.html, Retrieved: Jan. 2010.

Berners-Lee, T., Connolly, D., Kagal, L., Scharf, Y., and Hendler, J. (2008). N3logic : A logical framework for the world wide web. *Theory and Practice of Logic Programming*, 8:249–269.

Berners-Lee, T., Handler, J., and Lassila, O. (2001). The semantic web. *Scientific American*, pages 34–43.

Berners-Lee, T., Kolovski, V., Connolly, D., Hendler, J., Kolovski, V., and Scharf, Y. (2006). A resoner for the web. *Under consideration for publication in Theory and Practice of Logic Programming (TPLP) special issue on Logic Programming and the Web*, Available online: http://www.w3.org/2000/10/swap/doc/paper/, Retrieved: Jan. 2010.

Bizer, C., Heath, T., Ayers, D., and Raimond, Y. (2007). Interlinking open data on the web. In *Demonstrations Track, 4th European Semantic Web Conference, Innsbruck, Austria.*

Blei, D., Ng, A., and Jordan, M. (2003). Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022.

Blei, D. M. (2011). Introduction to probabilistic topic models. *Communications of the ACM (in press).*

Bod, R. (2001). A memory-based model for music analysis: Challenging the Gestalt principles. *Journal of New Music Research (JNMR)*, 30(3).

Boersma, P. (1993). Accurate short-term analysis of the fundamental frequency and the harmonics-to-noise ratio of a sampled sound. *in proc. Institute of Phonetic Sciences 1993*, 17(97-110).

Bonner, A. and Kifer, M. (1996). Concurrency and communication in transaction logic. *in proc. Joint International Conference and Symposium on Logic Programming*, pages 142–156.

Boolos, G. S., Burges, J. P., and Jeffrey, R. C. (2007). *Computability and Logic.* Cambridge University Press, 5th edition.

Borgida, A. (1995). Description logics in data management. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 7(5).

Borst, W. (1997). *Construction of Engineering Ontologies*. PhD thesis, Institute for Telematica and Information Technology, University of Twente, Enschede, The Netherlands.

Bray, S. and Tzanetakis, G. (2005). Distributed audio feature extraction for music. *Proceedings of the International Conference on Music Information Retrieval*, pages 434–437.

Bregman, A. S. (1990). *Auditory Scene Analysis: The Perceptual Organization of Sound*. The MIT Press.

Brewster, C., Alani, H., Dasmahapatra, S., and Wilks, Y. (2004). Data driven ontology evaluation. *Proceedings of Language Resources and Evaluation*, pages 164–168.

Brickley, D. and Guha, R. (2004). Rdf vocabulary description language. W3C Recommendation, Available online: http://www.w3.org/TR/rdf-schema/, Retrieved: March 2011.

Brickley, D., Hunter, J., and C., L. (1999). A logical model for metadata interoperability. Harmony Project Working Document. Available online: http://www.ilrt.bris.ac.uk/discovery/harmony/docs/abc/abc_draft.html, Retrieved: March 2011.

Brown, J. C. (1991). Calculation of a constant Q spectral transform. *Journal of the Acoustical Society of America*, 89(1):425–434.

Bullock, J. and Frisk, H. (2007). libintegra: A system for software-independent multimedia module description and storage. *in Proceedings of the International Computer Music Conference, Copenhagen, Denmark*.

Bulterman et al. (2008). Synchronized multimedia integration language (SMIL 3.0). Available online: http://www.w3.org/TR/SMIL/, Retrieved: March 2011.

Buneman, P. (1997). Semistructured data. *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 12-14, 1997, Tucson, Arizona, USA.*

Camacho, A. (2007). *SWIPE: A Sawtooth Waveform Inspired Pitch Estimator for Speech and Music*. PhD thesis, Graduate School of the University of Florida.

Cannam, C. (2009). The vamp audio analysis plugin api: A programmer's guide. Availble online: http://vamp-plugins.org/guide.pdf, Retrieved: Dec. 2010.

Cannam, C., Jewell, M. O., and Rhodes, C. (2010a). Linked data and you: Bringing music research software into the semantic web. *Journal of New Music Research special issue on Music Informatics and the OMRAS2 Project*.

Cannam, C., Landone, C., and Sandler, M. (2010b). An open source application for viewing, analysing, and annotating music audio files. *in Proceedings of the ACM Multimedia 2010 International Conference*.

Carroll, G. and Rooth, M. (1996). Valence induction with a head-lexicalized pcfg. In *Proceedings of the 3rd Conference on Empirical Methods in Natural Language Processing (EMNLP 3)*, pages 36–45.

Carrolla, J. J., Bizerb, C., Hayesc, P., and Sticklerd, P. (2005). Named Graphs. *Journal of Web Semantics*, 3(4):247–267.

Casey, M. (2002). Musical applications of mpeg-7 audio. *Organised Sound, Campbridge University Press.*, 6(2).

Casey, M. and Slaney, M. (2006). The importance of sequences in musical similarity. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, (ICASSP 2006).*

Casey, M., Veltcamp, R., Goto, M., Leman, M., Rhodes, C., and Slaney, M. (2008). Content-based music information retrieval: Current directions and future challenges. *Proceedings of the IEEE*, 96(4):668–696.

Chalmers, A. F. (1999). *What is this thing called Science?* Open University Press, Buckingham, United Kingdom.

Chambers, A., Smyth, P., and Steyvers, M. (2010). Learning concept graphs from text with stick-breaking priors. In Lafferty, J., Williams, C. K. I., Shawe-Taylor, J., Zemel, R., and Culotta, A., editors, *Advances in Neural Information Processing Systems 23*, pages 334–342.

Chen, H., Wu, Z., and Mao, Y. (2005). RDF-Based ontology view for relational schema mediation in semantic web. *Lecture Notes in Computer Science: Knowledge-Based Intelligent Information and Engineering Systems*, Vol. 3682(169):873–879.

Cheveigné, A. d. and Kawahara, H. (2002). Yin, a fundamental frequency estimator for speech and music. *The Journal of the Acoustical Society of America*, Vol. 111:1927.

Chiba, S. (1995). Metaobject protocol for C++. *in Proc. Object-Oriented Programming, Systems, Languages Applications (OOPSLA'95)*, pages 285–299.

Church, A. (1936). An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:345–363.

Cimiano, P., Hotho, A., and Staab, S. (2005). Learning concept hierarchies from textusing formal concept analysis. *Journal of Artificial Intelligence Research*, 24(1):305–339.

Cimiano, P., Mädche, A., Staab, S., and Völker, J. (2009). Ontology learning. *In S. Staab and R. Studer (eds.), Handbook on Ontologies, International Handbooks on Information Systems, 2nd Edition, Springer-Verlag Berlin Heidelberg.*

Clarke, E. F. (1999). Rhythm and timing in music. *in Diana Deutsch (ed.), The Psychology of Music, 2nd edition,* pp. 473-500.

Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, 13(6).

Collins, N. (2005). Using a pitch detector for onset detection. *in proc. 6th International Conference on Music Information Retrieval 11-15. Sept, 2005. London, UK in proc. 6th International Conference on Music Information Retrieval 11-15. Sept, 2005. London, UK.*

Collins, N. (2010). Computational analysis of musical influence: A musicological case study using mir tools. In *Proceedings of the 11th International Society of Music Information Retrieval Conference (ISMIR'10), August 9-13, 2010, Utrecht, Netherlands.*

Cont, A., Dubnov, S., and Assayag, G. (2011). On the information geometry of audio streams with applications to similarity computing. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(4):837–846.

Cope, D. (1997). *Techniques of the Contemporary Composer.* Schirmer Books, New York, USA.

Corcho, O., Gómez-Pérez, A., González-Cabero, R., and Suárez-Figueroa, M. (2004). ODEval: a Tool for Evaluating RDF(S), DAML+OIL, and OWL Concept Taxonomies. *First IFIP Conference on Artificial Intelligence Applications and Innovations (AIAI'04) Toulouse, France*, pages 369–382.

Crofts, N., Doerr, M., Gill, T., Stead, S., and Stiff, M., editors (2010). *Definition of the CIDOC Conceptual Reference Model (Version 5.0.2).* First published by the ICOM/CIDOC Documentation Standards Group in 2003, continued by the CIDOC CRM Special Interest Group.

Cross, I. (1998). Music analysis and music perception. *Music Analysis*, 1(17).

da Silva, P. P., McGuinness, D. L., and Fikes, R. (2006). A proof markup language for semantic web services. *Information Systems Journal*, 31(4-5):381–395.

da Silva, P. P., Salayandia, L., and Gates, A. (2007). Wdo-it! a tool for building scientific workflows from ontologies. *Technical Report. University of Texas at El Paso.*

Dahlhaus, C. D. C. and Gjerdingen, R. O. (1990). *Studies in the Origin of Harmonic Tonality.* Princeton University Press.

Dannenberg, R. (1993). Music representation: Issues, techniques and systems. *Computer Music Journal*, 17(3):20–30.

Davies, M. E. P. and Plumbley, M. D. (2007). Context-dependent beat tracking of musical audio. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(3):1009–1020.

de Lavieter L. (Ed.) (1995). Multilingual environmental thesaurus. *Nederlandse Bureau voor Onderzoek Informatie / EEA-TF - European Environment Agency - Task Force, Amsterdam.*

de Marneffe, M.-C., MacCartney, B., and Manning, C. D. (2006). Generating typed dependency parses from phrase structure parses. *Language Resources and Evaluation Conference (LREC'06).*

De Roure, D., Goble, C., and Stevens, R. (2009). The design and realisation of the virtual research environment for social sharing of workflows. *Future Generation Computer Systems*, 25(5):561 – 567.

Decker, S. and Frank, M. (2004). The social semantic desktop. *DERI Technology Report*, Available online: http://www.deri.ie/fileadmin/documents/DERI-TR-2004-05-02.pdf, Retrieved: May, 2011.

Degara-Quintela, N., Pena, A., and Torres-Guijarro, S. (2009). A comparison of score-level fusion rules for onsed detection in music signals. *in Proc. 10th International Society for Music Information Retrieval Conference (ISMIR'09), Kobe, Japan.*

Dijkstra, E. W. (1982). On the role of scientific thought. *Selected Writings on Computing: A Personal Perspective*, pages 60–66.

Ding, L., Bao, J., Michaelis, J., Zhao, J., and McGuinness, D. L. (2010). Reflections on provenance ontology encodings. *Proceedings of the 3rd International Provenance and Annotation Workshop (IPAW), Troy, New York, USA.*

Dittenbach, M., Merkl, D., and Rauber, A. (2001). The growing hierarchical self-organizing map. *Proceedings of the International Joint Conference on Neural Networks, IJCNN'01, Como, Italy*, 6:15–19.

Dixon, S. (2001). Automatic extraction of tempo and beat from expressive performances. *Journal of New Music Research (JNMR)*, 30(1):39–58.

Dixon, S. (2006). Onset detection revisited. *in Proc, Int Conference on Digital Audio Effects (DAFX-06) Sept. 18-20. 2006. Montreal, Canada.*

Dixon, S., Goebl, W., and Widmer, G. (2002). The performance worm: Real time visualization of expression based on langnerís tempo-loudness animation. *in Proc. International Computer Music Conference, ICMC2002.*

Dixon, S., Sandler, M., d'Invoerno, M., and Rhodes, C. (2010). Towards a distributed research environment for music informatics and computational musicology. *Journal of New Music Research special issue on Music Informatics and the OMRAS2 Project*, 39(4):291–294.

Dixon, S. and Widmer, G. (2005). Match: A music alignment tool chest. *in proc. 6th International Conference on Music Information Retrieval (ISMIR), London, UK.*

Dolog, P. (2005). *Model-Driven Navigation Design For Semantic Web Applications with the UML-Guide.* in Maristella Matera and Sara Comai (Eds.) Engineering Advanced Web Applications, Rinton Press.

Downie, S. J., Ehmann, A., and Tcheng, D. (2005). Music-to-knowledge (m2k): a prototyping and evaluation environment for music information retrieval research. *in Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, page 676.

Duhem, P. (1954). *The Aim and Structure of Physical Theory.* Princeton University Press, Oxford, United Kingdom.

Duignan, M. (2008). *Computer mediated music production: A study of abstraction and activity.* PhD thesis, Department of Computer Science, Victoria University of Wellington, New Zeland. Available Online: http://researcharchive.vuw.ac.nz/bitstream/handle/10063/590/thesis.pdf, Retrieved: March 2011.

Dumbill, E. (2002). Support online communities with FOAF. *XML Watch, IBM DeveloperWorks.*

Duxbury, C., Sandler, M., and Davis, M. (2002). A hybrid approach to musical note onset detection. *in proc. 5th International Conference on Digita Audio Effects.*

Elschek, O. (1969). System of graphical and symbolic signs for the typology of aerophones. *Bratislava:VydatelstvS lovenskej Academi Vied.*

Evans, E. (2003). *Domain-Driven Design: Tackling Complexity in the Heart of Software.* Addison-Wesley Professional.

Fazekas, G. (2009). Information interaction in context (conference review). *Informer (BCS, IRSG Information Retrieval Specialist Group)*, (30):2–4.

Fazekas, G., Cannam, C., and Sandler, M. (2009). Reusable metadata and software components for automatic audio analysis. *in Proc. IEEE/ACM Joint Conference on Digital Libraries (JCDL'09) Workshop on Integrating Digital Library Content with Computational Tools and Services, Austin, Texas, USA, 2009.*

Fazekas, G., Raimond, Y., Jakobson, K., and Sandler, M. (2010). An overview of Semantic Web activities in the OMRAS2 Project. *Journal of New Music Research special issue on Music Informatics and the OMRAS2 Project*, 39(4):295–311.

Fazekas, G. and Sandler, M. (2007a). Intelligent editing of studio recordings with the help of automatic music structure extraction. *Presented at the 122nd Convention of the Audio Engineering Society, Vienna, Austria.*

Fazekas, G. and Sandler, M. (2007b). Structural decomposition of recorded vocal performances and its application to intelligent audio editing. *Presented at the 123rd Convention of the Audio Engineering Society, New York, USA.*

Fazekas, G. and Sandler, M. (2009). Ontology based information management in music production. *Presented at the 126th Convention of the Audio Engineering Society, Munich, Germany, 2009.*

Fazekas, G., Wilmering, T., and Sandler, M. (2011). A knowledge representation framework for context-dependent audio processing. *In Proc. of the 42th International Conference of the Audio Engineering Society on Semantic Audio, Ilmenau, Germany, July 22–24.*

Fensel, D. (2001). Ontologies: Dynamic networks of formally represented meaning. *In Proceedings of the 1st Semantic web working symposium, Stanford, CA, USA.*

Fielding, R. T. and Taylor, R. N. (2002). Principled design of the modern Web architecture. *ACM Transactions on Internet Technology*, 2(2):115–150.

FIPA (2002). Fipa device ontology specification. Available online: http://www.fipa.org/specs/fipa00091/PC00091A.html, Retrieved: May 2010.

Floridi, L. (2004). *The Blackwell Guide to the Philosophy of Computing and Information.* Wiley-Blackwell.

Foote, J. (2000). Automatic audio segmentation using a measure of audio novelty. *In Proceedings of IEEE International Conference on Multimedia and Expo*, 1:452–455.

Franconi, E. (2002). *Description Logics.* Lecture Notes, Department of Computer Science, University of Manchester.

Fujishima, T. (1999). Realtime chord recognition of musical sound: A system using common lisp music. *Proceedings of the International Computer Music Conference, Beijing, International Computer Music Association.*

Furini, M. (2007). Mcdl: a reduced but extensible multimedia contents description language. *Int. J. Comput. Appl.*, 29:204–210.

Gangemi, A., Catenacci, C., Ciaramita, M., and Lehmann, J. (2006). Modelling ontology evaluation and validation. *In proc. of the 2006 European Semantic Web Conference (ESWC'06).*

Gangemi, A., Guarino, N., Masolo, C., Oltramari, A., and Schneider, L. (2002). Sweetening ontologies with dolce. *Knowledge Engineering and Knowledge Management, A. Gómez-Pérez, V.R. Benjamins (eds.), Ontologies and the Semantic Web, 13th International Conference, EKAW 2002, Siguenza, Spain, October 1-4, 2002, Springer Verlag,*, pages 166–181.

Gangemi, A. and Presutti, V. (2009). Ontology design patterns. *in S. Staab and R. Studer (eds.), Handbook on Ontologies, International Handbooks on Information Systems,* Springer-Verlag Berlin Heidelberg.

Ganter, B. and Wille, R. (1999). *Formal Concept Analysis Mathematical Foundations.* Springer Verlag.

García, R. and Celma, O. (2005). Semantic integration and retrieval of multimedia metadata. *Proceedings of the 5th International Workshop on Knowledge Markup and Semantic Annotation.*

Gaver, W. W. (1993). What in the world do we hear? explorations in ecological acoustics. *Ecological Psychology*, Vol. 5.(No. 1.):pp. 1–29.

Genesereth, M. R. and Nilsson, N. J. (1987). *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann Publishers, San Mateo, CA, USA.

Gil, Y., Deelman, E., Ellisman, M., Fahringer, T., Fox, G., Gannon, D., Goble, C., Livny, M., Moreau, L., and Myers, J. (2007). Examining the challenges of scientific workflows. *IEEE Computer*, 40(12):24–32.

Goble, C., Wroe, C., and Stevens, R. (2003). The mygrid project: Services, architecture and demonstrator. *Proceedings of UK e-Science All Hands Meeting*, pages 595–603.

Gold, B. and Morgan, N. (2000). *Speech and Audio Signal Processing.* John Wiley and Sons.

Goldberger, J., Gordon, S., and Greenspan, H. (2003). An efficient image similarity measure based on approximations of KL divergence between two Gaussian mixtures. *ICCV'03, Nice, France*, pages 487–493.

Goldfarb, C. F. (1991). HyTime: A standard for structured hypermedia interchange. *IEEE Computer Magazine*, 24:81–84.

Golub, G. H. and Kahan, W. (1965). Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics*, B(2):205–224.

Gomez, E., Peterschmitt, G., Amatriain, X., and Herrera, P. (2003). Content-based melodic transformations of audio material for a music processing application. *in Proc. of the 6th Int. Conference on Digital Audio Effects (DAFx-03), London, UK, September 8-11, 2003.*

Gómez-Pérez, A. (2004). Ontology evaluation. *in S. Staab and R. Studer (eds.), Handbook on Ontologies, International Handbooks on Information Systems, First Edition, Springer-Verlag Berlin Heidelberg.*

Gonzalez, E. P. (2010). *Advanced Automatic Mixing Tools for Music.* PhD Thesis, School of Electronic Engineering and Computer Science, Queen Mary University, London, UK.

Goto, M. (2003). A chorus section detection method for music audio signals. *in Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'03)*, pages 437–440.

Gouyon, F. and Dixon, S. (2005). A review of automatic rhythm description systems. *Computer Music Journal*, 29(1):34–54.

Griffiths, T. L. and Steyvers, M. (2004). Finding scientific topics. *Proceedings of the National Academy of Sciences of the United States of America*, 101(1):5228–5235.

Gruber, T. R. (1993a). Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, 43:907–928.

Gruber, T. R. (1993b). A translation approach to portable ontology specification. *Knowledge Acquisition*, 5(2).

Guarino, N. and Welty, C. (2002). Evaluating ontological decisions with ontoclean. *Communications of the ACM*, 45(2):61–65.

Guha, R., McCool, R., and Fikes, R. (2004). Contexts for the semantic web. *in Proceedings of the 3rd International Semantic Web Conference (ISWC'04), Hiroshima, Japan.*

Haase, P., Haase, P., and Stojanovic, L. (2005a). Consistent evolution of owl ontologies. pages 182–197. Springer.

Haase, P., van Harmelen, F., Huang, Z., Stuckenschmidt, H., and Sure, Y. (2005b). A framework for handling inconsistency in changing ontologies. *in Proc. 4th International Semantic Web Conference (ISWC'05), Galway, Ireland.*

Hacene, M. R., Napoli, A., Valtchev, P., Toussaint, Y., and Bendaoud, R. (2008). Ontology learning from text using relational concept analysis. In *Proceedings of the 2008 International MCETECH Conference on e-Technologies*, pages 154–163, Washington, DC, USA. IEEE Computer Society.

Hainsworth, S. (2004). *Techniques for the Automated Analysis of Musical Audio.* Ph.D. thesis, Deptartment Engineering, Cambridge University, Cambridge, UK.

Hargreaves, S. (2010). *Structural Segmentation of Multitrack Audio.* PhD progress report, Queen Mary University of London (unpublished).

Harte, C., Sandler, M., Abdallah, S., and Gomez, E. (2005). Symbolic representation of musical chords: A proposed syntax for text annotations. In *in Proc. of the 6th International Conference on Music Information Retrieval.*

Harte, C. A. and Sandler, M. (2005). Automatic chord identification using a quantised chromagram. *in Proc. of the 118th Convention of the Audio Engineering Society.*

Hartig, O. and Zhao, J. (2009). Using web data provenance for quality assessment. *Proceedings of the First International Workshop on the role of Semantic Web in Provenance Management (SWPM 2009), collocated with the 8th International Semantic Web Conference (ISWC09), Washington DC, USA, October 25.*

Hattaingadi, J. (1987). *How is Language Possible?* Open Court Publishing Company La Salle, Illinois, USA.

Hayes, P. (2002). RDF model therory. *W3C Working Draft*, Available Online: http://www.w3.org/TR/2002/WD-rdf-mt-20020429/, Retrieved: March, 2011.

Hayes, P. (2004). RDF semantics. *W3C Working Draft*, Available Online: http://www.w3.org/TR/rdf-mt/, Retrieved: March, 2011.

Hayes, P. J. (1996). A Catalog of Temporal Theories. *Artificial Intelligence Technical Report UIUC-BI-AI-96-01, University of Illinois at Urbana-Champaign.*

Hepp, M. and de Bruijn, J. (2007). Gentax: A generic methodology for deriving OWL and RDF-S ontologies from hierarchical classifications, thesauri, and inconsistent taxonomies. *The Semantic Web: Research and Applications, Lecture Notes in Computer Science*, 4519:129–144.

Hoadley, C. (2003). Design-based research: An emerging paradigm for educational inquiry by the design-based research collective. *Educational Researcher*, 32(1):5–8.

Hobbs, J. R. and Pan, F. (2006). Time Ontology in OWL. *W3C Working Draft.* Available online: http://www.w3.org/TR/owl-time/, Retrieved: March 2011.

Hodges, W. (1993). *Model Theory.* Cambridge University Press.

Hoffman, M., Blei, D., and Bach., F. (2010). On-line learning for latent Dirichlet allocation. *Advances in Neural Information Processing Systems (NIPS)*, 23:856–864.

Hofmann, T. (1999). Probabilistic latent semantic analysis. *In Proceedings of 22nd International ACM SIGIR Conference on Research and Development in Information Retrieval,*, pages 50–57.

Hofmann-Engl, L. J. (2003). *Melodic Similarity and Transformations: A theoretical and empirical approach.* PhD thesis, Dep. Of Psychology, Keele University, UK.

Honkela, T. (1997). Self-organizing maps in natural language processing. *PhD Thesis, Helsinki University of Technology, Neural Networks Research Centre, Espoo Finland.*

Horrocks, I. (2008). Ontologies and the Semantic Web. *Communications of the ACM*, 51(12):58–67.

Horrocks, I., Patel-Schneider, P. F., and van Harmelen, F. (2003). From SHIQ and RDF to OWL: The making of a Web Ontology Language. *Journal of Web Semantics*, 1(1):7–26.

Howe, B., Tanna, K., Turner, P., and Maier, D. (2004). Emergent semantics: Towards self-organising scientific metadata. *Lecture Notes in Computer Science*, 3226:177–198.

Huber, D. M. and Runstein, R. E. (2005). *Modern Recording Techniques.* Focal Press.

Hunter, J. (2001). Adding multimedia to the Semantic Web - Building an MPEG-7 Ontology. *The first Semantic Web Working Symposium, Stanford University, July 30 - August 1, 2001, California, USA.*

Hunter, J. (2003). Enhancing the semantic interoperability of multimedia through a core ontology. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(1):49–58.

Hunter, J. and James, D. (2000). Application of an event-aware metadata model to an online oral history archive. *presented at ECDL2000, Lisbon.*

Jacobson, K. (2011). *Connections in Music.* PhD Thesis, School of Electronic Engineering and Computer Science, Queen Mary University, London, UK.

Jacobson, K., Raimond, Y., and Sandler, M. (2009). An ecosystem for transparent music similarity in an open world. *in Proc. 10th International Society for Music Information Retrieval Conference (ISMIR'09), Kobe, Japan.*

Jewell, M. O., Lawrence, F., and Tuffield, M. M. (2005). Ontomedia: An ontology for the representation of heterogeneous media. *ACM SIGIR Multimedia Information Retrieval Workshop (MMIR 2005).*

Kabal, P. and Ramachandran, R. (1986). The computation of line spectral frequencies using chebyshev polynomials. *IEEE Transactions on Acoustics, Speech and Signal Processing (ASSP)*, 34(6).

Kania, A. (2008). The methodology of musical ontology: Descriptivism and its implications. *British Journal of Aesthetics*, 48(4):426–444.

Kanzaki, M. (2007). Music Vocabulary in OWL DL. Available online: http://www.kanzaki.com/ns/music, Retrieved: May 2011.

Kapanci, E. and Pfeffer, A. (2006). A hierarchical approach to onset detection. In *International Computer Music Conference (ICMC'06)*, pages 438–441.

Kaplan, A. M. and Haenlein, M. (2009). Users of the world, unite! the challenges and opportunities of social media. *Business Horizons*, Volume 53(Issue 1).

Kiczales, G., Rivieres, J. D., and Bobrow, D. G. (1991). *The Art of the Metaobject Protocol.* The MIT Press.

Kim, H.-G., Moreau, N., and Sikora, T. (2005). *MPEG-7 Audio and Beyond: Audio Content Indexing and Retrieval.* John Wiley & Sons.

Klapuri, A. P. (2004). Automatic music transcription as we know it today. *Journal of New Music Research (JNMR)*, 33(3):269–282.

Klein, D. and Manning, C. D. (2003). Accurate unlexicalized parsing. *Proceedings of the 41st Meeting of the Association for Computational Linguistics*, pages 423–430.

Klein, M., Fensel, D., van Harmelen, F., and Horrocks, I. (2001). The relation between ontologies and XML schemas. *Linköping Electronic Articles in Computer and Information Science.*

Ko, R. K., Lee, S. S., and Lee, E. W. (2009). Business Process Management (BPM) standards: A survey. *Business Process Management Journal*, 15(5).

Kobilarov, G., Scott, T., Raimond, Y., Oliver, S., Sizemore, C., Smethurst, M., Bizer, C., and Lee, R. (2009). Media meets semantic web - how the BBC uses DBpedia and linked data to make connections. In *Proceedings of the European Semantic Web Conference In-Use track.*

Koestler, A. (1964). *The Act of Creation.* (pp. 513-544), The Macmillan Company, New York.

Kohonen (1998). Self-organzation of very large document collections: the state of the art. *Proceedings of the International Conference on Artificial Neural Networks*, pages 65–74.

Kohonen, T. (1995). *Self Organising Maps.* Springer Verlag, Berlin.

Kolozali, S., Barthet, M., Fazekas, G., and Sandler., M. (2010). Towards the automatic generation of a Semantic Web ontology for musical instruments. *in proc. 5th International Conference on Semantic and Digital Media Technologies (SAMT).*

Kolozali, S., Fazekas, G., Barthet, M., and Sandler., M. (2011). Knowledge representation issues in musical instrument ontology design. *in Proc. International Society for Music Information Retrieval (ISMIR'11), Miami, Florida, USA.*

Kruskal, J. and Wish, M. (1986). *Multidimensional Scaling*. Sage.

Kuhn, T. S. (1962). *The Structure of Scientific Revolutions*. University of Chicago Press, Chicago, USA.

Lagoze, C. and Hunter, J. (2002). The ABC Ontology and Model. *Journal of Digital Information - Special Issue - selected papers from Dublin Core 2001 Conference*, 2(2).

Lakatos, I. (1970). Falsification and the methodology of scientific research programmes. *Criticism and the Growth of Knowledge: Proceedings of the International Colloquium in the Philosophy of Science*, 4:95–196.

Lassila, O. and Swick, R. (1998). Resource Description Framework (RDF) model and syntax specification. Available online: http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/, Retrieved: May 2009.

Lee, J.-S., Kim, T.-H., Yoon, G. S., Hong, J.-E., Cha, S. D., and Bae, D.-H. (1999). Developing distributed software systems by incorporation meta-object protocol with unified modelling language.

Lee, K. and Slaney, M. (2008). Acoustic chord transcription and key extraction from audio using key-dependent hmms trained on synthesized audio. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(2):291–301.

Leistikov, R. J. (2006). *Bayesian Modeling of Musical Expectations via Maximum Entropy Stochastic Grammars*. PhD thesis, Department of Music, Stanford University, USA.

Lerdahl, F. and Jackendoff, R. (1983). *A Generative Theory of Tonal Music*. The MIT Press.

Levine, N. (2003). The fundamentals of clos. *International Lisp Conference ILC-03, New York, 2003.*

Levy, M. and Sandler, M. (2006a). Lightweight measures for timbral similarity of musical audio. *In Proceedings of the 1st ACM Workshop on Audio and Music Computing Multimedia (Santa Barbara, California, USA, October 27, 2006). AMCMM '06. ACM, New York, NY, 27-36.*

Levy, M. and Sandler, M. (2006b). New methods in structural segmentation of musical audio. *in Proc. 14th European Signal Processing Conference, September 4 - 8, 2006, Florence, Italy.*

Levy, M. and Sandler, M. (2008). Structural segmentation of musical audio by constrained clustering. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(2):318–326.

Llorà, X., Ács, B., Auvil, L. S., Capitanu, B., Welge, M. E., and Goldberg, D. E. (2008). Meandre: Semantic-Driven Data-Intensive flows in the clouds. IlliGAL Report 2008013, Illinois Genetic Algorithms Laboratory University of Illinois at Urbana-Champaign, IL, USA.

Logan, B. and Salomon, A. (2001). A music similarity function based on signal analysis. *In proc. Multimedia and Expo ICME*, pages 745–748.

London, J. (2012). *Musical Meter, Musical Expression, and Social Cognition: An Inquiry in Cognitive Aesthetics.* In Studies in Honor of Eugene Narmour, A. Rozin L. Bernstein, eds. Hillsdale: Pendragon Press.

Lozano-Tello, A. and Gómez-Pérez, A. (2004). ONTOMETRIC: A Method to Choose the Appropriate Ontology. *Journal of Database Management. Special Issue on Ontological analysis, Evaluation, and Engineering of Business Systems Analysis Methods*, 15(2).

Lysloff, R. T. A. and Matson, J. (1985). A new approach to the classification of sound-producing instruments. *Ethnomusicology*, 29:213–236.

Macrae, R. and Dixon, S. (2011). Guitar tab mining, analysis and ranking. In *Proceedings of the International Symposium on Music Information Retrieval*, Miami, Florida.

Mandel, M., Poliner, G., and Ellis, D. (2005). Support vector machine active learning for music retrieval. *in Proc. of the 6th International Conference on Music Information Retrieval (ISMIR'05), London, UK.*

Martínez, J. M. (2004). Mpeg-7 overview. International Standard, ISO/IEC JTC1/SC29/WG11, Available online: http://mpeg.chiariglione.org/standards/mpeg-7/mpeg-7.htm, Retrieved: March 2011.

Masolo, C., Borgo, S., Gangemi, A., Guarino, N., and Oltramari, A. (2003a). Wonderweb deliverable d18: Ontology library. WonderWeb, Ontology Infrastructure for the Semantic Web, Available online: http://wonderweb.semanticweb.org/deliverables/D18.shtml.

Masolo, C., Borgo, S., Gangemi, A., Guarino, N., and Oltramari, A. (2003b). Wonderweb deliverable d18: Ontology library. Technical report, Laboratory For Applied Ontology - ISTC-CNR.

Mauch, M. (2010). *Automatic Chord Transcription from Audio Using Computational Models of Musical Context.* PhD Thesis, School of Electronic Engineering and Computer Science, Queen Mary University, London, UK.

Mauch, M. and Dixon, S. (2010a). Approximate note transcription for the improved identification of difficult chords. In *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR'10), Utrecht, Netherlands.*

Mauch, M. and Dixon, S. (2010b). Simultaneous estimation of chords and musical context from audio. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(6):1280–1289.

Mauch, M., Noland, K., and Dixon, S. (2009). Using musical structure to enhance automatic chord transcription. *in Proc. 10th International Conference on Music Information Retrieval (ISMIR'09), Kobe, Japan.*

Mayo, D. G. (1996). *The growth of experimental knowledge.* University of Chicago Press, Chicago, London.

McEnnis, D., McKay, C., and Fujinaga, I. (2006). Overview of OMEN. *Proceedings of the 7th International Conference on Music Information Retrieval. Victoria, BC, Canada.*

McEnnis, D., McKay, C., Fujinaga, I., and Depalle, P. (2005). jaudio: A feature extraction library. *in Proc. of the International Conference on Music Information Retrieval, London, UK.*

McKay, C., Burgoyne, J. A., and Fujinaga, I. (2009a). jMIR and ACE XML: Tools for performing and sharing research in automatic music classification. *Presented at the ACM/IEEE Joint Conference on Digital Libraries Workshop on Integrating Digital Library Content with Computational Tools and Services, University of Texas, Austin, USA.*

McKay, C., Burgoyne, J. A., Thompson, J., and Fujinaga, I. (2009b). Using ACE XML 2.0 to store and share feature, instance and class data for musical classification. *Proceedings of the International Society for Music Information Retrieval Conference*, (303-8).

McKay, C., Fiebrink, R., McEnnis, D., Li, B., and Fujinaga, I. (2005). Ace: A framework for optimizing music classification. *in proc. 6th International Conference on Music Information Retrieval (ISMIR'05), London, UK.*

Meddis, R., Lopez-Poveda, E., Fay, R., and Popper, A., editors (2010). *Computational Models of the Auditory System.* (1st Edition), Neuroscience series, Springer Verlag, 2010, XII.

Miller, L. (2000). Statement/statings: A summary of the threads *a triple is not unique* and *statements/reified statements* from the rdf interest discussion list. Available Online: http://www.ilrt.bristol.ac.uk/discovery/2000/11/statements/, Retrieved: March 2011.

Missier, P. and Goble, C. A. (2011). Workflows to open provenance graphs, round-trip. *Future Generation Comp. Syst.*, 27(6):812–819.

Mitrović, D., Zeppelzauer, M., and Breitender, C. (2010). Features for content-based audio retrieval. *Advances in Computers*, 78:71–150.

Monti, G. and Sandler, M. (2000). Monophonic transcription with autocorrelation. *in Proc, DAFX-00 Dec. 7-9. 2000. Verona, Italy.*

Moore, B. C. J. (2003). *An Introduction to the Psychology of Hearing (General Principles of Perceptual Organization).* Academic Press.

Moorer, J. A. (1979). About this reverberation business. *Computer Music Journal*, 3(2):13–28.

Moreau, L., Clifford, B., Freire, J., Futrelle, J., Gil, Y., Groth, P., Kwasnikowska, N., Miles, S., Missier, P., Myers, J., Plale, B., Simmhan, Y., Stephan, E., and den Bussche, J. V. (2010). The open provenance model core specification (v1.1). *Future Generation Computer Systems*, Preprint.

Motik, B., Horrocks, I., and Sattler, U. (2007). Bridging the gap between OWL and relational databases. *In Proc. of the Sixteenth International World Wide Web Conference.*

Murphy, K. P. (2002). *Dynamic Bayesian Networks: Representation, Inference and Learning.* PhD thesis, University of California at Berkeley,.

Nack, F., van Ossenbruggen, J., and Hardman, L. (2005). That obscure object of desire: multimedia metadata on the web, part 2. *IEEE Multimedia*, 12(1):54–63.

Naik, R. (2006). Introduction to logic programming in C++. Available online: `http://www.mpprogramming.com/Cpp/`, Retrieved: May 2010.

Newton-Smith, W. (1984). *The Russellian construction of instants.* Routledge, London and New York.

Nielsen, F. and Nock, R. (2009). Sided and symmetrized Bregman centroids. *IEEE Transactions on Information Theory*, 55(6):2882–2904.

Noland, K. and Sandler, M. (2009). Influences of signal processing, tone profiles, and chord progressions on a model for estimating the musical key from audio. *Computer Music Journal*, 33(1).

Obrst, L., Ashpole, B., Ceusters, W., Mani, I., Ray, S., and Smith, B. (2007). The evaluation of ontologies. *In Christopher J.O. Baker and Kei-Hoi Cheung (eds.), Revolutionizing Knowledge Discovery in the Life Sciences, Springer, Berlin, 2007*, pages 139–158.

Oinn, T., Addis, M., Ferris, J., Marvin, D., Greenwood, M., Carver, T., Wipat, A., and Li, P. (2004). Taverna, lessons in creating a workflow environment for the life sciences. In *Proceedings of GGF10, Berlin, Germany.*

Olson, H. F. (1952). *Musical Engineering.* McGraw-Hill Book Co. Inc., 242-265 edition.

Ong, B. S. and Herrera, P. (2005). Semantic segmentation of music audio contents. *in Proc. International Computer Music Conference 2005, Barcelona, Spain.*

Ossenbruggen, J. V., Nack, F., and Hardman, L. (2004). That obscure object of desire: Multimedia metadata on the web (part i). *IEEE Multimedia*, 12:54–63.

Pachet, F. (2005). Knowledge management and musical metadata. *Swartz D. (Ed.) Encyclopedia of Knowledge Management, Idea Group.*

Pampalk, E., Hlavac, P., and Herrera, P. (2004a). Hierarchical organization and visualization of drum sample libraries. *in Proc. of the 7th Int. Conference on Digital Audio Effects (DAFx-04), Naples, Italy, October 5-8, 2004.*

Pampalk, E., Widmer, G., and Chan, A. (2004b). A new approach to hierarchical clustering and structuring of data with self-organizing maps. *Intell. Data Anal.*, 8:131–149.

Pap, J. (2002). *Hang - ember - hang Rendhagyó hangantropológia (in Hungarian).* Vince Kiadó Kft. pp. 136-143.

Papadopoulos, H. and Peeters, G. (2007). Large-scale study of chord estimation algorithms based on chroma representation and hmm. *International Workshop on Content-Based Multimedia Indexing, 2007. CBMI '07.*

Paslaru, E., Simperl, B., Tempich, C., and Sure, Y. (2006). Ontocom: A cost estimation model for ontology engineering. *Proceedings of fifth International Semantic Web Conference (ISWC 2006), Athens, GA, USA.*

Passant, A. (2010). dbrec - music recommendations using dbpedia. *9th International Semantic Web Conference (ISWC2010), Nov. 7-11. Shanghai, China.*

Patel, A. D. and Peretz, I. (1997). Is music autonomous from language? A neuropsychological appraisal. *in Perception and Cognition of Music, Irene Deliege, John Sloboda (eds.) pp. 191-215.*

Patel-Schneider, P. F., Hayes, P., and (Eds.), I. H. (2004). OWL web ontology language semantics and abstract syntax. *W3C Recommendation, 10 February 2004*, Available online: http://www.w3.org/TR/2004/REC-owl-semantics-20040210/, Retrieved: Jan. 2010.

Paulus, J. and Klapuri, A. (2009). Music structure analysis using a probabilistic fitness measure and a greedy search algorithm. *IEEE Trans. Audio, Speech, and Language Processing*, 17(6):1159–1170.

Peeters, G. (2004). A large set of audio features for sound description. Technical report, IRCAM, Paris, France.

Peeters, G. (2006). Chroma-based estimation of musical key from audio-signal analysis. *in Proceedings of the 7th International Conference on Music Information Retrieval (ISMIR'06),* Victoria, Canada.

Peeters, G., Burthe, A. L., and Rodet, X. (2002). Toward automatic music audio summary generation from signal analysis. *In Proceedings of the 3rd International Conference on Music Information Retrieval (ISMIR'02),* Paris, France.

Philip, R. (2004). *Performing Music in the Age of Recording*. Yale University Press.

Pike, A. (1971). The perceptual aspects of motivic structure in music. *The Journal of Aesthetics and Art Criticism*, Vol. 30.(No. 1.):pp. 56–58.

Plassard, M.-F., editor (1998). *Functional Requirements for Bibliographic Records, Final Report*. International Federation of Library Associations and Institutions Study Group on the Functional Requirements for Bibliographic Records. Approved by the Standing Committee of the IFLA Section on Cataloguing K . G. Saur München.

Pope, S. T. (1996). Object-orientated music representation. *Organized Sound*, 1(1):56–58.

Popper, K. (1959). *The Logic of Scientific Discovery*. English translation of the original work *Logic der Forschung* published in 1934. Hutchinson & Co., London, England.

Prud'hommeaux, E. and Seaborne, A. (2008). SPARQL query language for RDF. W3C Recommendation, Available online: http://www.w3.org/TR/rdf-sparql-query/.

Puckette, M. S. (1997). Pure data: another integrated computer music environment. *in Proceedings, Second Intercollege Computer Music Concerts, Tachikawa*, pages 37–41.

Purwins, H. (2005). *Profiles of Pitch Classes Circularity of Relative Pitch and Key Experiments, Models, Computational Music Analysis, and Perspectives*. PhD thesis, Elektrotechnik und Informatik der Technischen Universiat, Berlin.

Quine, W. V. O. (1951). Two dogmas of Empiricism. *The Philosophical Review*, 60:20–43.

Quine, W. V. O. (1995). *From Stimulus to Science*. Harward University Press, Cambridge, Massachusetts, London, England.

Rabiner, L. and Juang, B. (1993). *Fundamentals of speech recognition.* Prentice-Hall, New Jersey.

Raimond, Y. (2007). Audio features ontology specification. Available online: http://motools.sourceforge.net/doc/audio_features.html, Retrieved: March 2011.

Raimond, Y. (2008). *A Distributed Music Information System.* PhD Thesis, School of Electronic Engineering and Computer Science, Queen Mary University, London, UK.

Raimond, Y. and Abdallah, S. (2006). The event ontology. Available online: http://motools.sourceforge.net/event/event.html, Retrieved: March 2011.

Raimond, Y. and Abdallah, S. (2007). The timeline ontology. Available online: http://motools.sourceforge.net/timeline/timeline.html, Retrieved: March 2011.

Raimond, Y., Abdallah, S., Sandler, M., and Frederick, G. (2007). The Music Ontology. *in Proc. 7th International Conference on Music Information Retrieval (ISMIR 2007), Vienna, Austria.*

Raimond, Y. and Sutton, C. (2007). A distributed data space for music-related information. *Workshop on multimedia information retrieval on The many faces of multimedia semantics, Emerging applications, Augsburg, Bavaria, Germany.*

Rasmussen, J. (1983). Skills, rules and knowledge: Signs and symbolism and other distinctions in human performance models. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 12.:pp. 257–266.

Rauber, A., Pampalk, E., and Merkl, D. (2003). The som-enhanced jukebox: Organization and visualization of music collections based on perceptual models. *Journal of New Music Research*, 32(2):193 — 210.

Rezinkoff, I. (2004). On primitive elements of musical meaning. *Journal of Music And Meaning (JMM)*, Volume 3. Fall 2004 / Winter 2005.

Ritter, H. (1999). Self-organizing maps on non-euclidean spaces. In *Kohonen Maps*, pages 97–108. Elsevier.

Robertson, S. (2008). The study of information retrieval – a long view. *IIiX'08, Information Interaction in Context, 2008, London, UK.*

Russell, B. (1948). *Human Knowledge Its Scope and Limits*, volume Chapter 5, pp. 51. George Allen and Unwin Ltd.

Sahoo, S. S. and Sheth, A. (2009). Provenir ontology: Towards a framework for escience provenance management. *Microsoft eScience Workshop, Pittsburgh, PA Oct 15-17.*

Salton, G., Wong, A., and Yang, C. S. (1975). A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620.

Sandler, M. and Levy, M. (2007). Signal-based music searching and browsing. *International Conference on Consumer Electronics, ICCE 2007, Jan. 10-17.*

Sauermann, L., Bernardi, A., and Dengel, A. (2005). Overview and outlook on the semantic desktop. *In Proceedings of the 1st Workshop on The Semantic Desktop at the ISWC 2005 Conference.*

Schleusing, O., Zhang, B., and Wang, Y. (2008). Onset detection in pitched non-percussive music using warping compenstaed correlation. *in Proc. 33rd International Conference on Acoustics, Speech, and Signal Processing (ICASSP'08) Las Vegas, USA.*

Schnitzer, D., Flexer, A., Widmer, G., and Gasser, M. (2010). Island of Gaussians: The self organizing map and Gaussian music similarity features. *in Proc. of the 11th International Conference on Music Information Retrieval (ISMIR'10), Utrecht, Netherlands.*

Schomaker, L., Nijtmans, J., Camurri, A., Lavagetto, F., Morasso, P., Benoît, C., Guiard-Marigny, T., Goff, B. L., Robert-Ribes, J., Adjoudani, A., Defée, I., Münch, S., Hartung, K., and Blauert, J. (1995). A taxonomy of multimodal interaction in the human information processing system. *A Report of the ESPRIT PROJECT* Available online: http://www.ai.rug.nl/~lambert/projects/miami/taxonomy/taxonomy.html, Retrieved: March 2012.

Schroeder, M. R. (1961). Improved quasi-stereophony and colorless artificial reverberation. *Journal of the Acoustical Society of America*, 33(8):1061–1064.

Schroeder, M. R. (1962). Natural sounding artificial reverberation. *Journal of the Audio Engineering Society*, 10(3):219–223.

Sheh, A. and Ellis, D. (2003). Chord segmentation and recognition using EM-trained hidden Markov models. *in Proc. of the 4th International Conference on Music Information Retrieval, (ISMIR'03), Baltimore, USA.*

Smith, B. (1995). Formal ontology, commonsense and cognitive science. *International Journal of HumanComputer Studies*, 43.

Smith, J. R. and Schirling, P. (2006). Metadata standards roundup. *IEEE Multimedia, April-June 2006*, 13(2):84–88.

Smits, J. O. (2009). Physical audio signal processing for virtual musical instruments and audio effects. Available online: https://ccrma.stanford.edu/~jos/waveguide/.

Sowa, J. F. (2000). *Knowledge Representation: Logical, Philosophical, and Computational Foundations.* Brooks Cole Publishing Co., Pacific Grove, CA, USA.

Sowa, J. F. and Borgida, A. (1991). *Principles of Semantic Networks: Explorations in the Representation of Knowledge.* Morgan Kaufmann Series in Representation and Reasoning. Morgan Kaufmann.

Steup, M. (2010). Epistemology. In Zalta, E. N., editor, *The Stanford Encyclopedia of Philosophy.* Spring 2010 edition.

Strawson, P. F. (1959). *Individuals. An Essay in Descriptive Metaphysics.* Routledge, London and New York.

Stroud, R. J. and Wu, Z. (1995). Using metaobject protocols to implement atomic datatypes. *Lecture Notes in Computer Science*, 952.

Sure, Y., Staab, S., and Studer, R. (2009). Ontology engineering methodology. *in S. Staab and R. Studer (eds.), Handbook on Ontologies, International Handbooks on Information Systems, Springer-Verlag Berlin Heidelberg.*

Taiani, F., Fabre, J.-C., and Killijian, M.-O. (2005). A multi-level meta-object protocol for fault-tolerance in complex architectures. *International Conferece on Dependable Systems and Networks (DSN'2005), June 28 - July 1, Yokohama, Japan.*

Tartir, S., Arpinar, I. B., Moore, M., Sheth, A. P., and Aleman-meza, B. (2005). OntoQA: Metric-based ontology quality analysis. In *IEEE Workshop on Knowledge Acquisition from Distributed, Autonomous, Semantically Heterogeneous Data and Knowledge Sources.*

Taylor, I., Deelman, E., Gannon, D., and Shields, M., editors (2006). *Workflows for e-Science: Scientific Workflows for Grids.* Springer Verlag.

Temperley, D. (2001). *The Cognition of Basic Musical Structures.* The MIT Press, ISBN 0-262-20134-8.

Tenney, J. and Polansky, L. (1980). Temporal gestalt perception in music. *Journal of Music Theory*, 24(2):205–241.

Tidhar, D., Fazekas, G., Kolozali, S., and Sandler, M. (2009). Publishing music similarity features on the semantic web. *Proceedings of the 10th International Conference on Music Information Retrieval (ISMIR'09), Kobe, Japan.*

Tidhar, D., Fazekas, G., Mauch, M., and Dixon, S. (2010a). Tempest - harpsichord temperament estimation in a semantic-web environment. *Submitted to the JNMR special issue on OMRAS2.*

Tidhar, D., Mauch, M., and Dixon, S. (2010b). High precision frequency estimation for harpsichord tuning classification. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP2010).*

Tillett, B. (2004). FRBR: A conceptual model for the bibliographic universe. *Library of Congress Cataloging Distribution Service.*

Tolonen, T. and Karjalainen, M. (2000). A computationally efficient multipitch analysis model. *IEEE Trans. on Speech and Audio Processing*, 8(1):708–716.

Troncy, R., Bailer, W., Höffernig, M., and Hausenblas, M. (2010). Vamp: a service for validating mpeg-7 descriptions w.r.t. to formal profile definitions. *Multimedia Tools Appl.*, 46:307–329.

Troncy, R., Carrive, J., Lalande, S., and Poli, J.-P. (2004). A motivating scenario for designing an extensible audio-visual description language. *International Workshop on Multidisciplinary Image, Video, and Audio Retrieval and Mining (CoRIMedia), October 25-26, 2004, Universite de Sherbrooke, Quebec, Canada.*

Troncy, R., Celma, Ò., Little, S., García, R., and Tsinaraki, C. (2007). Mpeg-7 based multimedia ontologies: Interoperability support or interoperability issue? *1st Workshop on Multimedia Annotation and Retrieval enabled by Shared Ontologies, Genova, Italy. December 5, 2007.*

Truemper, K. (2004). *Design of Logic-based Intelligent Systems.* Wiley-Interscience John Wiley and Sons.

Tsinaraki, C., Polydoros, P., and Christodoulakis, S. (2004). Integration of OWL ontologies in MPEG-7 and TVAnytime compliant Semantic Indexing. In *Proceedings of the 16th International Conference on Advanced Information Systems Engineering*, pages 398–413.

Turhan, A. Y., Springer, T., and Berger, M. (2006). Pushing doors for modeling contexts with OWL-DL – A Case Study. *Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications Workshops.*

Turney, P. (1996). The identification of context-sensitive features: A formal definition of context for concept learning. *M. Kubat and G. Widmer, (Eds.), Proceedings of the Workshop on Learning in Context-Sensitive Domains.*

Tversky, B. (1989). Parts, partonomies, and taxonomies. *Developmental Psychology*, 25:983–995.

Tzanetakis, G. (2002). *Manipulation, analysis and retrieval systems for audio signals.* PhD Thesis, Computer Science Department, Princeton University, USA.

Tzanetakis, G. and Cook, P. (2000). Marsyas: a framework for audio analysis. In *Organised Sound*, volume 4, pages 169–175.

Varzi, A. C. (2003). *Mereology.* The Stanford Encyclopedia of Philosophy (Spring 2003 Edition), Edward N. Zalta (ed.), Available Online: http://plato.stanford.edu/entries/mereology/, Retrieved: March 2011.

Varzi, A. C. (2006). A note on the transitivity of parthood. *Applied Ontology*, 1:141–146.

Veldhuis, R. (2002). The centroid of the symmetrical Kullback-Leibler distance. *IEEE Signal Procesing Letters*, 9(3):96–99.

Veldhuis, R. and Klabbers, E. (2003). On the computation of the Kullback-Leibler measure for spectral distances. *IEEE Trans. on Speech and Audio Processing*, 11(1):100–103.

Vembu, S. and Baumann, S. (2005). A self-organizing map based knowledge discovery for music recommendation systems. *Lecture Notes in Computer Science*, 3310.

Vemuri, B. and Jian, B. (2005). A robust algorithm for point set registration using mixture of Gaussians. *in Proc. of the 10th IEEE International Conference on Computer Vision (ICCV'05).*

Verfaille, V., Guastavino, C., and Traube, C. (2006a). An interdisciplinary approach to audio effect classification. *Proceedings of the 9th International Conference on Digital Audio Effects (DAFx-06), Montreal, Canada.*

Verfaille, V., Zölzer, U., and Arfib, D. (2006b). Adaptive digital audio effects (a-dafx): A new class of sound transformations. *IEEE transactions on audio, speech, and language processing*, 14(5):1817–1831.

Vignoli, F. and Pauws, S. (2005). A music retrieval system based on user-driven similarity and its evaluation. *in Proc. of the 6th International Conference on Music Information Retrieval, (ISMIR'05), London, UK.*

Viitaniemi, Klapuri, E. (2003). A probabilistic model for the transcription of single-voice melodies. *Finnish Signal Processing Symposium, FINSIG, Tampere University of Technology, May 2003.*

Vila, L. and Reichgelt, H. (1996). The token reification approach to temporal reasoning. *Artificial Intelligence*, 83(1):59–74.

von Hornbostel, E. M. and Sachs, C. (1914). Systematik der Musikinstrumente: EinVersuch. Zeitschriftffir Ethnologie, Translated by A. Baines and K. Wachsmann as A Classification of Musical Instruments. *Galpin Society Journal.*

Vrandečić, D. (2009). Ontology evaluation. *in S. Staab and R. Studer (eds.), Handbook on Ontologies, International Handbooks on Information Systems,* Springer-Verlag Berlin Heidelberg.

Vrandečić, D. and Gangemi, A. (2006). Unit tests for ontologies. *In Mustafa Jarrar, Claude Ostyn, Werner Ceusters, and Andreas Persidis (editors), Proceedings of the 1st International Workshop on Ontology Content and Evaluation in Enterprise* Montpellier, France.

Wainwright, M. and Jordan, M. (2008). Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305.

Wang, J., Chen, X., Hu, Y., and Feng, T. (2010). Predicting high-level music semantics using social tags via ontology-based reasoning. In *In Proc. of the 11th International Society of Music Information Retrieval Conference, August 9-13, 2010, Utrecht, Netherlands.*

Weinstein, E. and Pedro, M. (2007). Music identification with weighted finite-state transducers. *in Proc. ICASSP 2007.*

Welch, I. and Stroud, R. J. (2001). Kava - using byte code rewriting to add behavioural reflection to java. *Proceedings of the in Proc. 6th USENIX Conference on Object-Oriented Technologies and Systems (COOTS '01), Jan. 29 - Feb. 2, San Antonio, Texas, USA.*

Wen, X. and Sandler, M. (2008). Analysis and synthesis of audio vibrato using harmonic sinusoids. *Presented at the 124nd Convention of the Audio Engineering Society, Amsterdam, The Netherlands.*

West, K., Kumar, A., Shirk, A., Zhu, G., Downie, J. S., Ehmann, A. F., and Bay, M. (2010). The Networked Environment for Music Analysis (NEMA). In *6th World Congress on Services, Miami, Florida, USA, July 5-10*, pages 314–317.

Widmer, G. and Goebl, W. (2004). Computational models of expressive music performance: The state of the art. *Journal of New Music Research (JNMR)*, 33(3):203–216.

Wilmering, T., Fazekas, G., and Sandler, M. (2010). The effects of reverberation on onset detection tastks. *in Proc. of the 128th Convention of the AES, London, UK.*

Wilms, T., Harris, S., and Robillard, D. (2007). Lv2 audio plugin standard. Available online: http://lv2plug.in/spec/, Retrieved: Nov. 2011.

Wimering, T., Fazekas, G., and Sandler, M. (2011). Towards ontological representaiton of digital audio effects. *In Proc. of the 14th Int. Conference on Digital Audio Effects (DAFx-11), Paris, France, September 19-23.*

Wright, M., Chaudhary, A., Freed, A., Khoury, S., and Wessel, D. (1999). Audio applications of the sound description interchange format standard. *in Proceedings of the International Computer Music Conference (ICMC), Ann Arbor, Michigan, USA.*

Zentz, D. M. (1992). Music learning: Greater the sum of its parts. *Music Educators Journal*, 78(8):33–36.