Ontology Concepts, Uses and Evaluation

Vinay K. Chaudhri

This note is aimed at the developers engaged in applied research and development tasks or trying to develop end-user applications and attempts to explain how an ontology can be useful to them and how can they evaluate an ontology for their needs. We also consider a concrete scenario of representing and reasoning with music preferences to show how the ontology concepts apply to a specific situation.

Uses of an Ontology

The word ontology is used to refer to a wide range of representations ranging from simply a list of terms to a knowledge base with detailed axiomatizations and problem solving methods. A term is usually a word or a phrase in the spoken language. A practical way to understand an ontology is in relation to what it does or the value it adds for an end-user or an application. Here are some example uses of an ontology:

1. **Support navigation:** An ontology provides a hierarchically organized set of terms that help a user in navigating through a set of documents, web pages, or more generally, through a collection of information resources. Yahoo’s open directory (http://dir.yahoo.com) and the Open DMOZ (http://www.dmoz.org) are examples of such use.

2. **Controlled vocabulary:** Provide a set of terms that can be consistently used by multiple users in performing a task; for example, in annotating research articles, or the use of disease codes by a doctor. There usually needs to be a clear definition of every term, a unique term to define a situation, and a lack of redundancy. The Gene Ontology (see http://www.geneontology.org) is an example of such use and is used widely in the bio-informatics community.

3. **Semantic interoperability:** Provide a set of terms that can be used by software programs for exchanging information. For example, the iCalendar standard (http://en.wikipedia.org/wiki/iCalendar) is a popular exchange format for exchanging calendar information.

4. **Schema information:** Information systems for a variety of applications require a schema for the data they will store. Database schemas are usually used by multiple application programs and in that sense, are reused, and hence can be viewed as an ontology. Every database management system has a schema definition. Java programs use class definition files that serve as the schema.

5. **Domain descriptions:** An ontology can be used to describe a domain for the purpose of arriving at a clear description, and for communicating it across multiple people. Requirement specification documents, for example, UML diagrams (See http://en.wikipedia.org/wiki/Unified_Modeling_Language), are an example of such use.

6. **Problem solving:** An ontology could enable a reasoning system to derive conclusions by applying the rules and the axioms it contains. A common form of problem solving is answering questions. Other examples are planning, recommendation systems, etc. Ontologies such as the component library
are designed to support problem solving.

An alternative way to understand an ontology is in terms of the level of detail or the formality with which it is represented. Here are some alternative representations used by currently prevalent ontologies:

1. **Collection of words and phrases:** This is the least common denominator representation used in every ontology. Every ontology uses a collection of words or phrases to describe the terms it contains. The name space of those words could either be self-contained within that ontology, or the ontology could refer to and use names from ontologies external to it.

2. **Semantic relationships:** The terms in an ontology could be related using a set of relationships that have a specific meaning associated with them; for example, one term may be more general than the other.

3. **Class descriptions:** An ontology can have explicit classes, individuals, and class descriptions. The class descriptions may be defined using a description logic style description language and may contain some form of constraints. Web Ontology Language (OWL) is a common representation language used for class descriptions.

4. **Rules:** There are detailed axioms and rules defining the ontology, which are more expressive than what might fit in any of the common concept description languages. Rule Interchange Format or RIF (See [http://www.springerlink.com/content/e7v2802743688216/fulltext.pdf](http://www.springerlink.com/content/e7v2802743688216/fulltext.pdf)) is a common representation language to express knowledge using rules. RIF is also capable of capturing concept descriptions.

5. **Problem solving methods:** The ontology has associated with it specialized reasoning mechanisms, for example, a procedure to diagnose faults or to make recommendations, etc. Ontologies usually represent an interface to the problem solving method. The implementation of a problem solving method is generally done outside the ontology.

**Should I use an Ontology?**

In general, any application that will store data has a schema, and thus, at some level, has an ontology. An ontology is implicit in a relational data base schema or in an interface description language, or Java class definitions and is sufficient for a large fraction of application needs.

It is also very common for applications to exchange information with others and semantic interoperability is an important concern. If one invests enough resources, either manual or semi-automatic, it may be possible to convert data from one ontology to another. But, if the applications commit to using some shared ontology and use it consistently, the cost of semantic interoperability can be drastically reduced. There are significant practical challenges in the consistent use of an ontology, as in spite of best intentions, the applications can differ in subtle but important ways. There are some examples of
successes in the use of an ontology as seen in the standard usage of standard XML data types.

A developer of a new application who is likely to import data from others can also significantly benefit by committing to a standard ontology. For example, if one wishes to use data about people, the friend of a friend ontology (FOAF) is a commonly used format, and there is a significant amount of published data about people that is available that can be more easily imported if one were using the FOAF ontology.

In a problem solving application that needs to model a significant aspect of domain, the use of an ontology is unavoidable. Even though such an application may never exchange information with an outside application, there needs to be a vocabulary that could be used in writing rules or by the problem solvers.

In summary, the question of "should I use an ontology" is better phrased as "what level of ontological description will best meet the needs of the application at hand?" This is because the ontologies always exist implicitly – even if there is no explicit definition. A secondary question is whether the ontology should be developed as a separable piece of the system and exposed and published as an end in itself? The answer to these questions can vary widely depending on the business need one is trying to fulfill.

**How does an ontology relate to the code base?**

The system implementers need to deliver code using programming languages such as Java, C, or database programming languages. Conversely, ontologies have traditionally been developed using knowledge representation languages such as OWL. In cases such as OWL, where the representation language has been standardized, the focus is usually on representation expressiveness concerns so that the knowledge can be well-specified and the semantics are rigorous. Because of these different perspectives, the developers find themselves at a crossroad: If they commit to using a rich representation language for defining an ontology they are not able to ensure simplicity and the performance that is required of a well-engineered systems. What should then be done in such a situation?

For any performance-oriented systems, a recommended rule of thumb is to err on the side of simplicity in that one should choose the simplest possible solution even if it means sacrificing representational sophistication. Many applications can be simplified to the extent that the code base contains nothing that comes anywhere close to explicit ontology encoded in an expressive representation language. There are, however, other possible middle grounds. One promising approach is to develop and maintain the ontology external to the system’s code base but construct a simplified representation for inclusion in the production code. In this scheme of things, one is driven by representation expressiveness, completeness, and functionality while defining the ontology, but driven by performance and simplicity while engineering the system. Such a dual model still requires that there be a business case for maintaining two different representations and that the external ontological model is done in a way that the simpler representation required for the production system can be programmatically generated.
How does an Ontology relate to Tags?

Tags are free-form strings that a casual user uses to annotate objects. Tags are a powerful collaborative and simple way to way to label and organize large collections of data. The goal of such tagging is to enable the future retrieval of objects. The goal of enabling an easier retrieval is not new: the traditional libraries meet the same goal through a centralized catalogue system. The users of a tagging system can quickly label a large numbers of objects as opposed to a library in which a book must have a unique call number based on its information content. The tags, however, are much less informative than a call number in the library: they tell us little more than the free-form string that they represent. In spite of this limitation, the tags enable information browsing in the following ways: listing all objects with the same tag, listing objects with most popular tags, and a list of tags which have a high degree of overlap with a tag that the user is currently investigating.

A natural question is whether a collection of tags can be viewed as an ontology? In the context of the different classes of uses introduced earlier, the tags are intended to support information navigation. In terms of the representation languages, they are simply a collection of words and phrases. In that sense, tags are weakest possible version of an ontology that we considered earlier.

There have been some efforts to introduce a hierarchical structure amongst tags. In one possible approach, (see http://heymann.stanford.edu/taghierarchy.html) tags can be automatically organized into a hierarchy, thus, providing a better structure for navigation than simply a flat list of tags.

Some Useful Ontologies

We consider here a few ontologies that are available in public domain and have a broad applicability.

Friend of a Friend: Data about people is a very commonly used data in applications. The FOAF ontology provides a way to describe people and share that information with others. See http://www.foaf-project.org/ for details.

Geonames: The GeoNames geographical database covers all countries and contains over eight million place names that are available for download free of charge. The GeoNames Ontology makes it possible to add geospatial semantic information to the Word Wide Web. See http://www.geonames.org/ontology/ for more details.

Music Ontology: The Music Ontology is an attempt to link all the information about musical artists, albums and tracks together, from MusicBrainz to MySpace. The goal is to express all relations between musical information in order to help people find anything about music and musicians. It is based around the concept of machine readable information provided by any web site or web service on the Web. See http://wiki.musicontology.com/index.php/Main_Page for more information.
**DBpedia Ontology**: The DBpedia Ontology is a shallow, cross-domain ontology, which has been manually created based on the most commonly used infoboxes within Wikipedia. It provides a good coverage of people, places, organizations and buildings.

**Evaluating an Ontology**

The ontology should be evaluated in the context of a specific use. Even though generic multi-use ontologies exist, there are few universal criteria for the goodness of an ontology. A few alternative upper ontologies exist that are meant to be multi-purpose and multi-use, but there is no consensus in the research and user communities over their relative contribution and value to end-user applications. We suggest here a few approaches that are pragmatic and can help an application developer/researcher assess the usefulness of an ontology.

1. **Coverage**: The most common use of an ontology is to provide a collection of terms. An evaluation based on coverage simply measures how many of the terms needed for a specific end-use are represented or covered in the ontology. For example, an ontology of driving actions will be considered to have good coverage if it provides terms for denoting all possible driving actions of interest, such as: commute, race, city drive, highway drive, scenic drive, etc. Suppose, an applications requires \( n \) terms, and \( k \) of them are represented in the ontology, then the coverage can be measured by simply \( k/n \).

2. **Redundancy**: The redundancy of an ontology measures the number of terms in the ontology that are not directly needed by an application. A general-purpose and multi-use ontology is likely going to contain things that are useful across multiple applications, and we do not expect an ontology to exactly match what an application requires. But, on the other hand, if one needs to use only 1% of an ontology for an application, there can be questions about its usefulness especially if using the complete ontology may have repercussions for the overall performance of the system. Suppose an application requires \( n \) terms that are present in an ontology of size \( N \), then the redundancy in ontology is \( 1-n/N \).

3. **Consensus**: The terms in the ontology should be defined in a way that their meanings are clear, different users agree on the meaning, and they can use the terms in an identical manner. The measurement of consensus requires human involvement. A simple measurement of consensus can be done by giving the terms in an ontology to say \( N \) different people, and ask them to express their agreement with a score of 0 or 1. If the sum of the scores is \( n \), then one can claim that there is \( n/N \) consensus on the meaning of these terms. (Of course, more complex scoring and measurement schemes are possible.) To measure the consensus usage, one could ask \( N \) different people to perform a task that requires using a term from the ontology. Suppose, the task involves making \( k \) choices, then for each choice, \( i \) one can measure \( n_i \) as the number of people that use identical terms giving \( 1/k \sum_{i=1}^{k} n_i / N \) as the overall measure of consensus in usage.

4. **Competency questions**: For an ontology that has rules and axioms associated with it, a possible way to evaluate it is to define a suite of test questions that characterize the knowledge of the domain. The ability of the ontology and the
associated reasoners to correctly answer those questions can be viewed as a competency measure of that ontology. The fraction of questions answered correctly in this case would be viewed as the overall competency score of the ontology.

5. **Representation evaluation:** It is possible to evaluate the quality of representation used in an ontology from a purely representation point of view. Some example characteristics of representation quality are:
   a. Cycles in the class hierarchy
   b. Redundant class definitions
   c. Inconsistent definitions or rules
   d. Overly strict or underspecified constraints
   e. Contradictory conclusions from the ontology
   f. Conciseness

6. **Non-functional evaluation:** The ontologies also have non-functional properties that are often relevant in choosing them. Some examples of non-functional properties of an ontology are:
   a. Licensing issues – whether it is open domain or requires license
   b. Existing user base – whether there is a community of users and researchers behind it

**Example Scenario: Music Preference Consensus**

In this section, we consider an example scenario that comes up during a driving situation. Within the context of this scenario, we will consider the application of the ontology concepts that we have considered so far.

Consider a situation in which a group of passengers are driving together in a car. In such a situation, we will like to develop a music recommendation application that identifies tracks that will be liked by a majority of the passengers.

The following technical problems need to be solved in developing a group music recommendation application: representing the content of the music, representation and acquisition of preferences, and computing a recommendation that works for the group. These steps are inter-related in the sense that the representation of the content of the music influences how the preferences are acquired which in turn influences how the group recommendation is computed.

**Representing the Content of the Music**

We will limit our attention here to only those approaches that explicitly represent the content of music in a symbolic form.

A systematic approach to describe the content of music is to start from a contemporary encyclopedia of music that systematically enumerates different kinds of music, and their properties. Such an enumeration provides a principled framework for categorizing different tracks.
A music genre is a categorical construct that classifies musical sounds into different categories. There are thousands of musical genres, and a systematic enumeration of them is available in public domain from Wikipedia. See http://en.wikipedia.org/wiki/List_of_music_genres for a detailed listing of genres.

There are also several commercial efforts to represent the content of the music. Some of the leading commercial efforts are listed below.

- **Music Genome Project:** The music genome project has defined four hundred attributes that capture the essence of music. A close enumeration of these attributes is available at: http://en.wikipedia.org/wiki/List_of_Music_Genome_Project_attributes. Given such enumeration, each song is also represented by an n-dimensional database vector in which each element corresponding to one of n musical characteristics of the song. An n-dimensional source song vector that corresponds to the musical characteristics of a source song is determined. A Distance between the source song vector and each of database song vector is calculated, each distance being a function of the differences between the n musical characteristics of the source song vector and one of source database song vector. The calculation of the distances may include the application of a weighted factor to the musical characteristics of resulting vector. A match song is selected based on the magnitude of the distance between the source song and each database songs after applying any weighted factors. This technology is protected under a US patent. (See http://www.google.com/patents?id=LJ54AAAIAEBAJ&dq=7,003,515)

- **All Music Guide (AMG):** The AMG covers a vast range of artists, albums, musical styles, and provides following details about them:
  - **Meta Data:** Facts about an album or artist, including title, tracks, genre, label, credits, release date, and cover and artist images.
  - **Descriptive Content:** Deeper details that really illuminate an artist or album including styles, moods, years active, instruments, birth/death date/place, and country of origin.
  - **Relational Content:** Information that helps you make meaningful connections between artists and the music – such as major influencers and followers, similar artists, top artists, and top albums.
  - **Editorial Content:** Original and insightful writing by AMG’s staff and network of professional freelance music contributors. This content includes biographies, album and song reviews, style descriptions, composition descriptions, and AMG ratings and picks.

The content in the AMG is curated by its editorial staff along with hundreds of expert contributors. Its goal from the inception has been to allow the users to effectively navigate music. See the following URL for more information: http://www.allmusic.com/cg/amg.dll?p=amg&sql=32:amg/info_pages/a_about.html.

- **MusicBrainz:** The goal of the MusicBrainz project is to create an open database of music which can be used for identifying audio CDs and digital audio tracks.
The basic metadata includes a list of artists and artist aliases (e.g. alter-ego names, alternate band names and common abbreviations) and for each artist a list of albums and the tracks for each album. MusicBrainz assigns each artist, album and track a unique identifier, which can be used to refer to a particular artist/album/track without having to deal with the semantics of correct spelling and conflicting names in the database.

These identifiers provide the Internet community with a means to establish a meaningful computer-based dialog about music. This unambiguous dialog is enabled by an RDF based web service interface and presents the first baby steps towards the "Semantic Web", where computers can carry on a meaningful discussion without involving human beings. The RDF used in the web service uses portions of the Dublin Core and is documented on the MusicBrainz site. MusicBrainz encourages others to use the RDF in other future music applications to enable a host of new applications and features that are not possible today.

For instance, it is not possible today to exchange a playlist with a friend, since your friend may not have the same files that you do; even if your friend does, the files may be located in a different location on the hard drive. Using MusicBrainz, a user can create a playlist that consists solely of MusicBrainz track identifiers, and then send that playlist to their friend. Their friend will be able to feed the playlist to their MusicBrainz-enabled audio player and then have the player match up the available tracks. If some of the tracks are not available in the collection, the player could go out to music sites such as EMusic.com, MusicNet or Pressplay to download the missing tracks. The MusicBrainz identifiers allow future audio applications to carry on unambiguous conversations about music and to enable a whole new set of features for music enjoyment and music discovery.


- Last.fm: Last.fm is a website that allows users to discover new music they like based on the music they already listen to. For each registered user, the website provides a profile, that records all the music they listen to. Based on the history of music listened, the site will recommend new music, upcoming events, as well as users with similar taste. This project is actually a commercial website that utilizes commonalities among user preferences to suggest music.

Last.fm supports user-end tagging or labeling of artists, albums, and tracks to create a site-wide folksonomy of music. Users can browse via tags, but the most important benefit is tag radio, permitting users to play music that has been tagged a certain way. This tagging can be by genre ("garage rock"), mood ("chill"), artist characteristic ("baritone"), or any other form of user-defined classification ("seen live"). However, since tagging is not moderated, it is prone to manipulation by the site's users, most often resulting in genre disagreements among users or pushing certain artists higher up on certain tag charts.

Subscribers are also able to create personal tag radio stations containing only tracks and artists that they have tagged themselves. All tag radio stations
(including subscriber's personal tag stations) can be played by anybody, including non-subscribers.

Last.fm provides an API that allows anyone to build their own programs using the data on its site. The documentation of the API is available at http://www.last.fm/api.

From an ontological point of view the four resources considered here represent very different points in the design space. The Music Genome project can be characterized as list of features that can be used by a machine learning program, and serves as a controllable vocabulary. The AllMusic data is manually curated and designed to support navigation. The MusicBrainz data is also manually curated, tied to the RDF representation language, and primarily aimed at semantic interoperability. The Last.fm data consists of tags and aimed at allowing collaborative creation of labels in the context of a social network.

Representing and Acquiring Music Preferences

An approach to preference representation and acquisition has to take into account the overall system usability, the level of user interaction possible, the type of reasoning possible with them, and the quality of the ultimate decisions. Popular classes of systems that take preferences into account to make specific suggestions to a user are known as recommender systems. We will first review some recommender systems, and then discuss different approaches for representing preferences for music.

Some systems generate recommendations with as little effort as possible from the user. An example of such a technique is Amazon.com’s “People who bought this item also bought...” These systems provide recommendations to the user based on demographics, content (for example, viewed items), or similar users’ data. The system uses this information to provide the recommendations to the passive user; thus preferences are implicitly and indirectly elicited.

In contrast, conversational recommender systems involve the user elaborating his or her requirements and desiderata over the course of an extended dialogue. They are often used in situations where the user is actively searching for a particular item but he or she has only partial knowledge of the domain, a task sometimes called a preference-based search.

In the recommender systems based on collaborative filtering, the idea is that similar people like similar things. An implicit model of a user’s preferences is inferred from items that he or she has rated. Reasoning is based on statistical predictions to estimate the missing ratings. Collaborative filtering has obtained attention because it can uncover serendipitous associations. It is applied by websites such as Amazon.com (for example, books, videos) and Last.fm and Mystrands (music), and by services such as Netflix. The interaction is asynchronous—the user can provide new feedback and obtain new recommendations at any time—and the recommendation engine works inside the framework of an online user community.
Let us now turn our attention to representation of preferences for music. This issue needs to be addressed at two levels: preferences of an individual, and preferences of a group. We will consider each of them in turn.

There are two broad approaches for representing preferences for music: cardinal and ordinal. A cardinal number is a number used in counting, for example, 1, 2, 3, etc. (See http://mathworld.wolfram.com/CardinalNumber.html) for more detail. An ordinal number, on the other hand, is an adjective that describes the numerical position of an object, e.g., first, second, third, etc. (http://mathworld.wolfram.com/OrdinalNumber.html)

When applied to representing preferences about music, in a cardinal ranking scheme, the user can specify different degrees of preference between candidates (i.e., the distance between candidates in the ranking is a meaningful value). In an ordinal ranking scheme, the user simply provides an order over the list of candidates. Cardinal rankings contain more information, which can be useful; however, ordinal rankings are much easier for most users to specify.

In Pandora, the preference scheme is a degenerate case of an ordinal scheme, as the user has an option to give thumbs up or thumbs down to specific songs. Another common example of an ordinal scheme is to ask the user to give a star value (between 1 and 5) to songs.

From the acquisition point of view, a straightforward way to acquire a cardinal preference on music is to simply track how frequently does a user listen to a song. This creates a cardinal ranking; however, it is not clear if such a ranking actually captures user preferences in meaningful way. A second simple approach is to get an ordinal ranking by simply asking the user to rank songs on some rating scale. This approach likely gets closer to expressing the user's actual preference function, but it may be difficult or annoying for the user to specify. Thus, the level of user interaction needed is a big tradeoff between the two approaches.

The user must be able to specify the preferences in a flexible manner. For example, they could specify the preference for artists, specific songs, specific genres of music, or specific period, or the originating country, etc. One may also want to provide an option to relate the music preference to the time of the day.

Some of the representation and acquisition issues that apply to representing the preferences of a single user carry over to representing the preferences for a group. A group situation, however, may be more conducive to acquisition by voting, because when people are together they will interact about their likes and dislikes.

**Computing Group Recommendation**

The problem of computing group recommendation is known as the "consensus ranking problem" in the decision sciences and has been under study for a long time. Consensus ranking approaches exist for both cardinal and ordinal representation of preferences.

In generally, one can solve cardinal rankings by formulating the preferences as continuous utility functions and then optimizing them. Such a formulation reduces optimization over a convex region, and there are several fast algorithms for it. See
It is difficult to formulate an ordinal consensus ranking algorithm that meets certain social welfare criteria (see http://en.wikipedia.org/wiki/Arrow%27s_impossibility_theorem for a brief introduction). Nevertheless, ordinal ranking has the advantage that it is easier for the user to specify since one does not ask her for degrees of preference. Tavana et al. describe a good, understandable algorithm for preference consensus ranking in "A Hybrid Distance-Based Ideal-Seeking Consensus Ranking Model," and they include a nice numerical example. If a complete ranking of alternatives is not available, Cook et al. has a more complicated algorithm to handle that case, again with a nice numerical example (see "Creating a consensus ranking of proposals from reviewers’ partial ordinal rankings" by Cook et. al for more information).

Instead of completely automating the computation of group recommendation, it may be better for the system to offer suggestions and let the users vote in the real time on what they will like to listen to. Considerable value can be added by consolidating individual user preferences, making them available and offering suggestions to vote on.

**Role of Ontology**

Let us now briefly comment on what role an ontology plays in this scenario.

First observation is that ontology is only a part of the solution. Even after one has a good ontology for this scenario, considerably more work needs to be done to represent preferences, and to reason with them. For this scenario, however, automated reasoning for group recommendation is likely to be less important.

The simplest use of the ontology for this scenario is in supporting semantic interoperability. If every user is represented using the FOAF ontology, and the user’s play list was represented using the MusicBrainz ontology, it will be straightforward to combine them, relate them, and come up with an aggregated set of preferences.

The next possible use of ontology is to allow the user to state preferences in a flexible manner. For example, one user may provide his preference for sixties bands, while another may provide a preference for the Beatles. With an ontology, it is straightforward to infer that there is an overlap in these two preferences.

While using the preferences that are in terms of tags supplied by the user, there are two approaches to introduce better structure. One approach is to organize the tags into a hierarchy as mentioned in a previous section. Another approach is to train a classifier to associate each tag with some class in a background ontology of music. It is an open research question whether such an approach will out perform the approach of organizing the tags into hierarchy by themselves.
In a similar vein, in some cases, there may not be a direct overlap between the preferences of two users. In such cases, using an ontology, one can determine the nearest common ancestor in the ontology and use that as a recommendation of music to be played.

**Open Questions and Possible Next Steps**

Our goal in the current write up has been to articulate some of the basic principles of using ontologies and to show how they apply in the context of a specific scenario. There are several aspects of the scenarios that could be developed in lot more detail. Some of these aspects are as follows:

- Choosing a specific set of publicly created ontologies about music, and assess their effectiveness in enabling interoperability of data
- Investigating how best to exploit the tags – whether by organizing them into a hierarchy or by mapping them to an ontology
- Devising schemes that use the ontological information to improve the recommendation
- Conducting user studies based on different representations of preferences to assess which representation leads to more intuitive recommendation
- Conducting some user surveys to assess what approach is likely to work better to reach group consensus and what level of automation will be acceptable

**Acknowledgment**

This work was supported by Toyota Information Technology Center.