A Distributed API for Searching Multimedia Databases

Dale E. Parson
Agere Systems
1110 American Parkway NE
Allentown, PA, US, 18109
dparson@agere.com

Lisa Frye
Kutztown University
Kutztown, Pennsylvania 19530
frye@kutztown.edu
http://faculty.kutztown.edu/frye/

Abstract

UPnP™ (Universal Plug'n'Play) specifies standards for interaction between user interface / control devices, called Control Points, and embedded application devices, called UPnP Devices, on an IP network. UPnP includes a six layer generic protocol for IP address establishment, UPnP Device discovery, Device and Service Description retrieval, Device Control in the form of interpreted XML procedure invocation from a Control Point to a Service, Service Eventing that notifies subscribers of changes in a Service's advertised state, and Presentation for vendor-specific extensions. UPnP AV (audio-visual) specifications extend these protocols for multimedia by defining Media Server and Media Renderer Device types, along with the Content Directory Service of metadata that describe and locate available video, audio and image content. This paper examines the standard operations available for metadata creation, maintenance, browse and search, and discusses deficiencies in UPnP's SQL-like search capability. We propose a keyword-based alternative to search whose benefits include ease of use familiar to users of Web search engines, incremental refinement of query results when browsing a large database, structuring of results in a classification hierarchy, and applicability to graphical manipulation and display of results. We have built a reference implementation of this query interface that uses only established database structures such as B-trees. Our results fit readily into the distributed UPnP AV framework.

Keywords: distributed database, Internet, multimedia, plug and play, search engine

1. UPnP Interfaces for Multimedia

1.1 Universal Plug'n'Play Standards

UPnP™ (Universal Plug'n'Play) defines a set of standards for interaction of embedded devices across communications networks in homes, businesses and other environments [1]. The generic UPnP device model applies to systems as simple as basic home appliance control / monitoring systems distributed across local area networks (LANs). At the other end of the networked continuum, this model applies to high definition video players receiving high bandwidth content streams across wide area networks (WANs). The generic UPnP model classifies all participating devices into one of two categories — Control Points and UPnP Devices. A Control Point is a remote control that monitors Device status and controls Devices using UDP/IP and TCP/IP; a UPnP Device is an application-specific device under the control of one or more Control Points. A given Control Point or Device may be an embedded processing device, or it may be application software running on a computer.

UPnP is primarily a set of standards for specifying extensible Control Point and Device protocol layers on top of UDP/IP and TCP/IP. This section outlines the generic, application-neutral layers of UPnP protocol.

Addressing (UPnP Layer 0) establishes a unique IP address for each UPnP Control Point or Device on a network via DHCP (Dynamic Host Configuration Protocol) or Auto-IP [2].

Discovery (UPnP Layer 1) uses SSDP (Simple Service Discovery Protocol [3]) over UDP/IP to allow Control Points to query a network for Devices, and to allow Devices to advertise themselves to Control Points. When a Control Point joins a network, and at periodic intervals thereafter, it sends SSDP messages over multicast UDP/IP to a dedicated UDP port to query for available Devices; each Device replies via unicast UDP/IP to the Control Point. Similarly, when a Device joins a network, and at periodic intervals thereafter, it sends SSDP messages over multicast UDP/IP to a dedicated UDP port to notify all Control Points of its availability. Finally, when a Control Point or Device leaves a network, it uses SSDP/UDP/IP multicast to notify network devices of its departure.

Description (UPnP Layer 2) uses HTTP/TCP/IP to obtain a set of Device and *Service* descriptions, via HTTP GET, in the form of XML documents. From this layer upward all UPnP interactions build on point-to-point TCP/IP. A Device houses one or more UPnP

Services; each is a collection of identifiers, networked functions and events. Device and Service descriptions distinguish application-specific features of a Device such as a Media Server or Player. A Control Point uses these descriptions to determine whether it is in the application class of a discovered Device; a Control Point gives user access only to Devices in the application class of the Control Point.

Control (UPnP Layer 3) consist of invocation of Service *actions*, described in a Service description, by a Control Point. A UPnP action is an application-specific remote procedure within a Service. A Control Point invokes an action in a Service by sending a SOAP (Simple Object Access Protocol) message, where SOAP is implemented as XML messages over HTTP/ TCP/IP. A Service interprets a SOAP message and returns a result as an XML reply. Action invocation is the primary means of controlling and polling a UPnP Device.

Eventing (UPnP Layer 4) is the means by which a Device notifies subscribed Control Points and Devices of a change in its advertised state. Control actions include subscription to application-specific events. A subscriber supplies a URL, and a Device sends a GENA (General Event Notification Architecture [4]) event in the form of an XML message over HTTP/TCP/IP to each subscriber whenever advertised Service state changes.

Presentation (UPnP Layer 5) is a vendor-specific URL atop HTTP/TCP/IP where Device vendors can add proprietary extensions to the above UPnP layers.

1.2 Standards for Multimedia Devices

The UPnP protocols described to this point are application neutral. UPnP also specifies a set of audio-visual (AV) standards that pertain to multimedia Devices, Services and Control Points. These standards specify multimedia application-specific bindings for the Device and Service Descriptions, Control actions and Events of UPnP layers 2 through 4.

Figure 1 shows the generic UPnP Device Service model adapted for the Media roles of AV Control Point and Media Server and Media Renderer Devices [5]. Each Device in Figure 3 provides three AV-specific Services.

The primary role of a Media Server is to provide access to *metadata* that describe available multimedia content via the Media Server's Content Directory Service [6,7]. Metadata include property tags for a multimedia object (e.g., a video, audio or image file) that a user can understand, such as title, artist, genre and creation date. Metadata also include tags such as bitrate, number of audio channels or image resolution intended to be interpreted by a media player. Perhaps the most important metadata tag for a multimedia object is its *resource URL*, which is the network path that a media player uses to locate and obtain the multimedia content. Content Directory Service deals entirely with tags for describing and locating a content stream, and not with the content itself.

The Connection Manager Service of Figure 1 describes

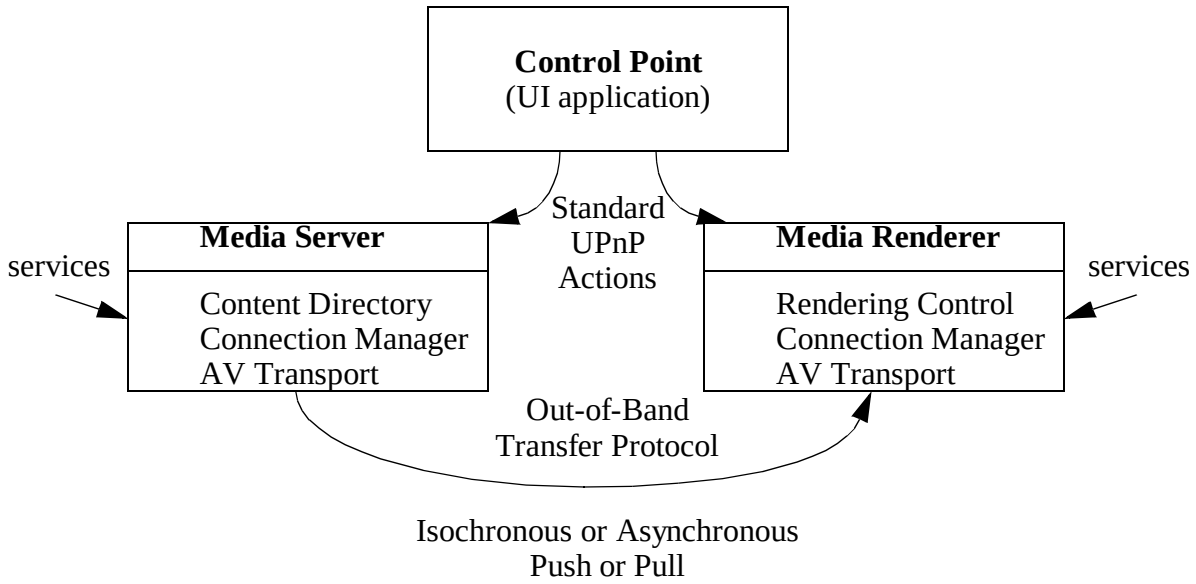

Figure 1: UPnP AV Devices and Services (from [5])

available protocols for streaming actual media content, and AV Transport includes basic mechanisms for streaming this content from a Media Server to a Media Renderer that plays or displays content via its Rendering Control Service [8,9]. Figure 1 illustrates that the actual transfer of media content is outside the domain of UPnP specification. The "out-of band transfer protocol" is some standard, high bandwidth means for streaming multimedia content that is not specified directly by UPnP. Note further that the content described by Content Directory Service metadata need not reside on the same networked machine as that metadata. Although these data often are coresident, a resource URL can point to a media stream stored on a remote machine.

The Digital Living Network Alliance (DLNA™) has extended the UPnP AV specifications into a practical set of working standards by specifying some optional Services, actions, and events as mandatory, others as disallowed [10], and by providing forums for interoperability testing and certification. According to DLNA guidelines, the AV Transport Service and transport-oriented actions of Connection Manager are not used, and DLNA 1.0 compliant Media Renderers do not advertise themselves on a UPnP network. Each media player comes with its own embedded Control Point that locates DLNA compliant Media Servers and retrieves their metadata. HTTP GET is the mandatory, baseline transport mechanism for a media streaming, although other protocols are allowed. A DLNA-compliant media player fetches a multimedia object's resource URL from a Media Server's Content Directory Service, and then uses that URL to stream the media content.

1.3 An Interface for Metadata Access

The Content Directory Service of Figure 1 organizes metadata objects into a hierarchical directory structure, where an *object* is either an *object.container* that collects subordinate objects, or an *object.item* that houses metadata tags for a specific audio, video or image media file or stream. Content Directory specifies an extensible type system including object classnames such as the following.

• object.container.album.musicAlbum is a subclass of the album type containing object.item.audioItem.musicTracks.

• object.container.genre.musicGenre is a subclass of the genre type that contains music albums of a given genre.

• object.item.videoItem.movie is a subclass of the class of video stream objects.

There is a large set of predefined classnames. Other mandatory object tags besides classname include a unique database identifier for each object, a title, parent container ID, along with optional tags such as artist and genre. Medium-specific resource related tags include size, duration, bitrate, sampleFrequency, bitsPerSample, number of audio channels, resolution, colorDepth, protocolInfo, and the resource URL that locates the actual media data. With its UPnP Service interface and classification and containment hierarchies, Content Directory is a distributed, object-oriented, hierarchical database.

There are four mandatory and ten optional actions that an AV Control Point can use to interact with a Content Directory. The primary actions for this discussion are the following.

• **Browse** is a mandatory action that allows Control Points to walk a Content Directory tree structure and examine object tags, including the resource URL. It allows a client Control Point to request a subset of objects within a given container being browsed; it is not necessary for Browse to return all objects in a container being browsed in a single networked interaction.

• **Search** is an optional action that allows Control Points to search for objects whose property tags match values specified in a SQL-like query string. Search allows a Control Point to request a subset of objects satisfying a given Search string, caching results for subsequent client calls to Search.

• **CreateObject**, **UpdateObject** and **DestroyObject** are optional actions that support database modification. Most commercial Media Servers do not implement these actions because of potential dangers in making DB modification available to the UPnP network. UPnP provides no means for authentication or protection when these actions are available. Instead, most Media Servers provide non-UPnP means for modifying and maintaining metadata in the Content Directory Service that support secure access.

• **ImportResource**, **ExportResource** and **DeleteResource** are similar, optional actions, not usually implemented, for modifying the storage of actual media files. These operations are normally implemented by secure, non-UPnP means.

Content Directory thus normally acts as a means for Control Points to identify content, based on metadata tags, and to locate that content via its URL.

2. Distributed Multimedia Search

The NAS (network attached storage) family of chips from Agere Systems is designed to work as a UPnP Media Server as well as a more general networked disk

server [11]. This device architecture includes custom hardware for bypassing its microprocessor in accelerating outgoing and incoming streams of multimedia data such as concurrent, high-definition video streams. It also contains embedded support for various reliable RAID disk configurations.

Two of Agere's contributions to the UPnP oriented middleware of this Media Server NAS system have been the creation of a C++ library interface that provides the actions and semantics of the Content Directory Service, and the creation of a keyword-based Query capability that is an alternative to UPnP Search. C++ class *CdsAction* is a C++ interface consisting strictly of pure virtual functions whose names and parameters match those of the Content Directory Service actions such as Browse and CreateObject. Unlike Content Directory Service, where each action takes the form of an XML-over-TCP interpreted function invoked from a Control Point, CdsAction specifies pure virtual functions invoked by media utilities such as *content import*, which reads metadata from media files (e.g., ID3 tags in MP3 music files) and writes them into the Content Directory Service via CdsAction::CreateObject. CdsAction gives a utility programmer a means for interacting with the database that lies beneath an implementation of Content Directory Service without requiring the utility programmer to learn the details of the database system. A concrete class derived from CdsAction implements the virtual functions for a given database system; changing to a different database system requires writing a different class derived from CdsAction, but it does not require changes to client utility code. Easing the conceptual burden on utility programmers and enhancing portability are the results.

2.1 Drawbacks of UPnP Multimedia Search

The Agere NAS middleware team pursued creation of keyword-based Query capability that is an alternative to UPnP Search because of difficulties in using Search in a media customer environment. UPnP Search uses SQL-like query syntax requiring a user to supply tag names (e.g., "artist") matched to values, using relational operators from the set '=', '!=', '<', '<=', '>', '>=', 'exists', 'contains', 'doesNotContain', 'derivedfrom' for dynamic class verification, and the 'and' and 'or' logical connectives [7, sections 2.5.5 and 2.7.5]. Search returns results in the form of a container housing the objects that satisfy the query constraints.

A formal query mechanism such as the Search action is unnecessarily cumbersome to use. Many Content Directory Service providers do not implement Search because of the difficulties of user interaction and the costs of implementing query parsing and results caching. The keyword matching approach used in web surfing is much easier to use and better known to typical customers of media delivery systems.

One partial solution taken by most Content Directory suppliers has been to present search indices to users as virtual containers open to the hierarchical Browse operation. Suppose a search index that maps the 'artist' tags for all Content Directory objects to their respective IDs is available. An implementation of Browse can present the set of all artists as an object.container under the Browse tree root, and present a specific artist such as "Miles Davis" as a container of type object.container.person.musicArtist, tagged with a title of "Miles Davis," that contains references to the music albums of Miles Davis. This approach can associate search indices on genre tags to object.container.genre virtual containers, and associate the media implied by classname tags (e.g., video, audio, image, text, etc.) to media-specific virtual containers. When a user examines such a hierarchy via Browse, the user is really traversing search indices to concrete containers such as music albums, photo albums and playlists.

The Browse-virtual containers alternative to UPnP Search is very helpful in simplifying user interaction requirements, but it suffers from the drawback that it does not eliminate data from view the way that Search can. Search, while difficult to use, effectively reduces the size of the metadata database that a user must Browse. It allows a user to converge more rapidly on media objects of interest than Browsing the full set of objects in a very large database.

2.2 Extending the UPnP Media Server with a Keyword Query API

Our solution within Agere has been to create an extension to the C++ interface CdsAction, which is based on the Content Directory Service actions, with derived interface class *CdsQueryAction* as shown in Figure 2. CdsQueryAction is another class of pure virtual functions that specifies the behavior of the Query, AndQuery, OrQuery and DiffQuery operations.

Each of these operations takes a simple keyword, or a keyword ending in '*' signifying a trailing match wildcard, and builds a **result set** of all concrete object.containers (e.g., music albums or user playlists) and contained object.items in the database tagged with that keyword. The keyword is not an SQL-like relational expression; it is a simple piece of text such as "Miles Davis." An implementation class for interface CdsQueryAction maintains search indices for commonly used Content Directory tags such as title,

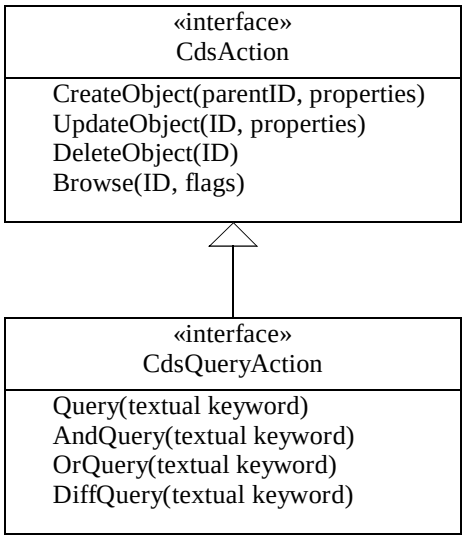

Figure 2: Extending Content Directory with Keyword Query

artist, genre, contributor, director, and publisher. If the keyword matches any of the indexed tags, the corresponding Content Directory ID enters the result set.

Our Search also allows a keyword to be preceded by '<', '<=', '>' or '>=', signifying a date range comparison. For example, "<2000" gets all entries created earlier than the year 2000; ">=06/01/1955" gets all entries created starting at June 1, 1955.

After performing each Query, an implementation class derived from class CdsQueryAction presents the user with a Browse hierarchy of virtual containers like that shown in Figure 3. This hierarchy is exactly like the Browse virtual containers of the last section, except that it has eliminated from view all Content Directory objects that have not satisfied the cumulative Search criteria. Our CdsQueryAction approach adds three main benefits over the mandatory Browse and optional Search actions of UPnP.

• It is easier to use because of its keyword approach in contrast to the SQL-like syntax of UPnP Search.

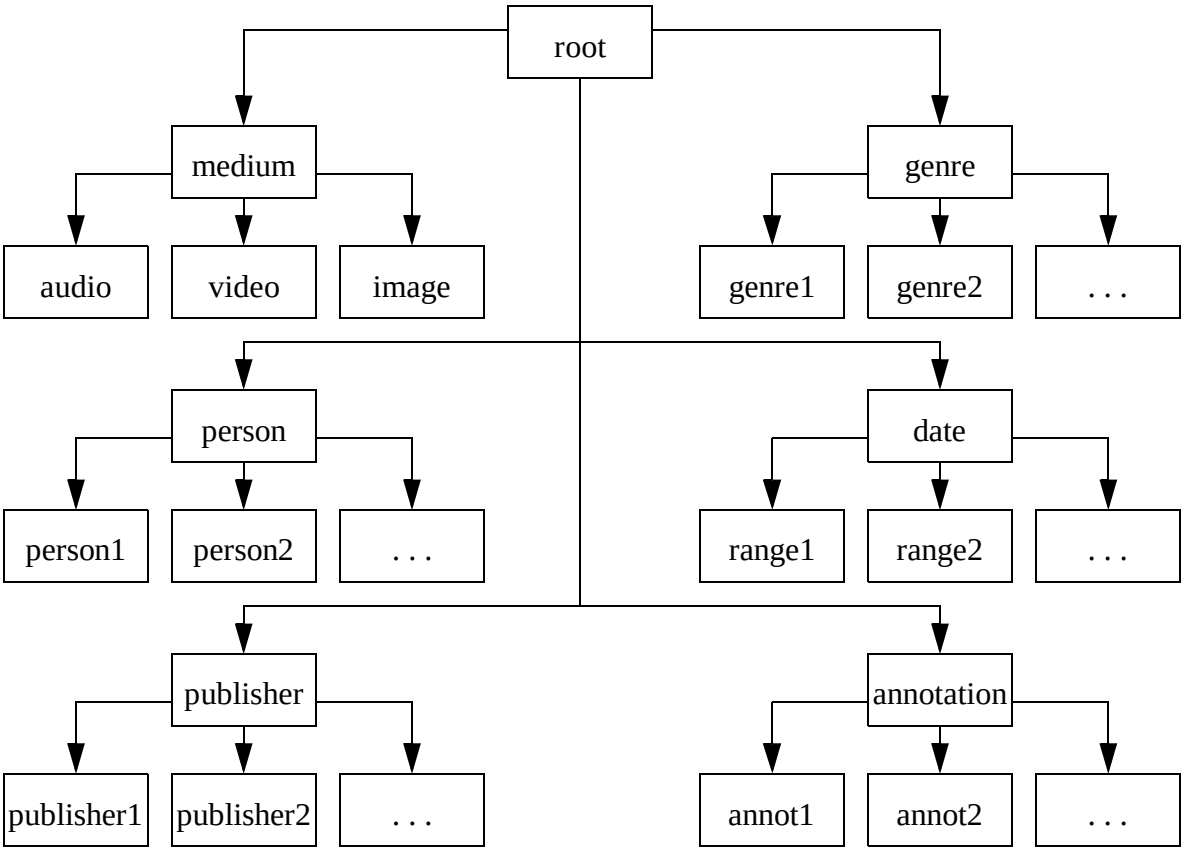

Figure 3: Virtual container hierarchy for CdsQueryAction::Browse

• Our Query supports interactive cumulative queries by providing AndQuery, OrQuery and DiffQuery that refine results.

• Our Query returns the hierarchical structure of Figure 3 for subsequent Browse inspections, as contrasted with the flat results list of UPnP Search.

After an initial Query creates a result set, operations AndQuery, OrQuery and DiffQuery modify that result by forming a temporary result set based on a new keyword, and then finding the logical intersection, union, and set difference respectively of the two sets, storing the result in the persistent result set. Following a Query for "Miles Davis" with an AndQuery for "John Coltrane" creates a result set of only the works on which both artists worked together; following this with an AndQuery of ">1955" would yield their combined works only after 1955.

Our cumulative Query allows users to temporarily remove unwanted contents of a Content Directory Service database from view, similar to Search. The result is a hierarchical view into a subset of a hierarchical database. Moreover, mapping the hierarchy of Figure 3 into a graphical user interface (GUI) on a video-capable device allows a user to select keyword text for AndQuery / OrQuery / DiffQuery refinements, as well as to examine object properties and build user playlists, by pointing and clicking. Each incremental Query refinement returns a new virtual Browse tree. Exploring a large metadata DB converges much more rapidly on items of interest to users than Browsing the entire DB or using the difficult and usually unavailable UPnP Search operation.

2.3 Implementing the Keyword Query API

We implemented the CdsQueryAction interface in a concrete class that uses balanced B-trees to map binary key-to-value relations [12]. Our B-trees support log(N) search for a key, where N is the number of entries in the tree, followed by sequential traversal of adjacent elements in ascending order on the key. The secondary, sequential links allow us to traverse multiple mappings with identical keys — many of these mappings are many-to-many — and to satisfy the requirement of UPnP Browse of returning only some of the result set to a client with each call. Typically Browse iterates from the first element (index 0) in a result set, browsing some number of results in each call. After searching out a result via a B-tree search structure, each B-tree maintains an iterator that supports sequential traversal of adjacent elements; this iterator avoids the need for additional caching of results outside a B-tree in support of UPnP Browse.

We build and maintain two primary relations, **string-->id** that maps Content Directory tags of interest (e.g., an artist tag such as "Miles Davis") to each Content Directory ID tagged with that value, along with the inverse **id-->string** mapping used to assist in maintaining the former relation when the underlying Content Directory changes. Note that, in contrast to UPnP Search indices, there is not a separate search index for each tag such as artist or genre. Our approach places all search keys into **string-->id**, avoiding the need for a user to specify tag names in a query.

In addition to these two primary relations, we build secondary relations **id-->medium**, **id-->genre**, **id-->person**, **id-->date**, **id-->publisher** and **id-->annotation** at database indexing time, where *person* represents any person tag such as artist, author or collaborator on a media object, and *annotation* represents a user annotation tag. We build these relations even though these mappings are available via Browse to the id-associated objects in the underlying Content Directory Service, in order to speed construction of Query-created virtual containers discussed next.

When a user provides a Query argument such as "Miles Davis," the Query operator descends the B-tree within **string-->id** and places all IDs associated with Miles Davis into a *result set*, which is implemented as a B-tree keyed on the ID. This B-tree is a strict set; each ID appears once. Each B-tree is actually housed in a C++ template class called *MultiSet* that supports one-to-one, one-to-many and many-to-many mappings, supports template key and value types in the mapping, supports in-core or disk storage, and supports logical AND, OR and DIFF operators for two argument sets. Given the sequential, ascending links between adjacent entries in the B-tree, MultiSet supports these three operators by merging the two ascending sequences in the parameters supplied to AND, OR and DIFF. AND retains in the first MultiSet only the key mappings it shares with the second; OR merges elements from the second set into the first; DIFF eliminates from the first set elements that it shares with the second. Each operation's time is linear on the size of its result. Thus, implementing AndQuery, OrQuery and DiffQuery of Figure 2 is a matter of building a second result set with a new keyword, where the keyword searches into a MultiSet's B-tree, and then combining this result set with the previous Query's result set via the MultiSet AND, OR or DIFF operator.

Query and its variants return the number of matched elements in the result set. CdsQueryAction::Browse supports browsing the IDs in the result set by building a

collection of additional MultiSet B-trees in support of the virtual Browse hierarchy of Figure 3. For each virtual container directly under root in Figure 3, for example *genre*, there is a MultiSet with unique key values such as **genre set** that collects all of the unique genre values tagged onto the IDs in the result set. CdsQueryAction builds sets such as **genre set**, **person set**, etc. at Query time by mapping result set IDs to genre values via **id-->genre**, IDs to person values (artist / director / producer / contributor / etc.) via **id-->person**, and so on. When a user Browses the genre container of Figure 3, the user is really browsing the set of values in **genre set** constructed at Query time.

When Browsing one of these top level virtual containers such as genre or person uncovers a second level virtual container such as "jazz" or "Mile Davis," the user can inspect the contents of that container via Browse as well. CdsQueryAction::Browse supports this level of virtual browsing by inspecting many-to-many MultiSets such as **genre-->id**, **person-->id**, etc., that map specific genre names such as "jazz" or person names such as "Miles Davis," stored in ascending order in a B-tree, to associated Content Directory IDs. Query constructs a mapping such as **genre-->id** at the same time it constructs a set such as **genre set**, by traversing the result set of IDs and mapping each to its respective genre, etc., by examining mapping **id-->genre**, etc., constructed at database indexing time. Finally, at the bottom of the Browse hierarchy of Figure 3, browsing an actual **id** obtained from a mapping such as **genre-->id** goes to the underlying Content Directory Service Browse to obtain UPnP object tags.

2.4 Using the Keyword Query API

We have constructed a prototype graphical interface (GUI) in the form of a conventional folder / file browser, using an existing user interface library, and find it very effective in browsing a virtual media container / item hierarchy like the one outlined in Figure 3. As with other applications of B-tree, most computational cost and delay goes into constructing and maintaining B-tree linkages while modifying the Content Directory database. The benefit of the B-tree comes with searching the Content Directory database for movies, music and photos. As with other database applications of B-trees, search is rapid; as with other keyword-based approaches, search is easy, and is usually graphical. We are planning to expand the Search capabilities of Figure 2 by storing individual words from UPnP object tags as separate search strings (e.g., "Miles" and "Davis" as separate search keys) in support of partial matching of Query strings, at the cost of additional B-tree storage.

The primary outcome of this work is a proposal to extend the available actions of the Content Directory Service with the optional Query actions of interface CdsActionQuery in Figure 2. The benefits discussed include ease of use, incremental refinement of queries, the hierarchical browse structure of query results, and applicability to graphical query and display of results. The UPnP framework and the underlying database technology such as using B-trees as search indices is well established. Our goal is to incorporate these technologies into a productive new combination.

3. References

1. UPnP Device Architecture 1.0, December, 2003, http://www.upnp.org/resources/documents/CleanUPnPDA101-20031202s.pdf.
2. Droms, R., "Dynamic Host Configuration Protocol," RFC 2131, Internet Engineering Task Force, http://www.ietf.org/rfc/rfc2131.txt, March, 1997.
3. Simple Service Discovery Protocol/1.0, Operating without an Arbiter, October, 1999, http://www.upnp.org/download/draft_cai_ssdp_v1_03.txt.
4. General Event Notification Architecture Base: Client to Arbiter, September, 2000, http://www.upnp.org/download/draft-cohen-gena-client-01.txt.
5. UPnP AV Architecture:0.83, June 12, 2002, http://www.upnp.org/download/UPnPAvArchitecture%200.83.prtad.pdf.
6. MediaServer:1, Device Template Version 1.01, June 25, 2002, http://www.upnp.org/download/MediaServer%201.0.pdf.
7. ContentDirectory:1, Service Template Version 1.01, June 25, 2002, http://www.upnp.org/download/ContentDirectory%201.0.prtad.pdf.
8. ConnectionManager:1, Service Template Version 1.01, June 25, 2002, http://www.upnp.org/download/ConnectionManager%201.0.pdf.
9. AVTransport:1, Service Template Version 1.01, June 25, 2002, http://www.upnp.org/download/AVTransport%201.0.prtad.pdf.
10. DLNA Home Networked Device Interoperability Guidelines v1.0, http://www.dlna.org/members/DLNA_Home_Networked_Device_Interoperability_Guidelines_v1.0.pdf.
11. Agere Systems Announces World's Fastest Multimedia, Storage Chip Portfolio for Homes and Businesses, http://www.agere.com/NEWS/PRESS2005/052505b.html, May, 2005.
12. Aho, Hopcroft and Ullman, *Data Structures and Algorithms*, Reading, MA: Addison-Wesley, 1983.