RBNB INTERNAL DATA STRUCTURES

The RBNB system is designed to store and move data between networked applications. Each data source can have multiple "channels" of data. Channel data are referenced by name and time. Thus, the RBNB data structures are inherently a triad of *data*, *name*, and *time*.

1 DATA

Each RBNB *server* has one or more data *sources*. Each data source has one or more time-stamped *frames* of data. Each data frame has one or more data *channels*. Each channel contains a block of *data* (per frame). Thus, the organization from RBNB server to data is as follows:

Server -> Source -> Frame -> Channel -> Data

RBNB data are simply blocks of bytes. No "meaning" is assigned to the data by the RBNB; this is a matter between the data source and consumer (sink). A *Source* also maintains static metadata for each channel.

2 NAME

Data is addressed via a three-part naming system:

Server / *Source* / *Channel*

The S*erver* name uniquely identifies the RBNB server. It is typically a network host:port address. The S*ource* name uniquely identifies the application that sends the data. The *Channel* name uniquely identifies each data channel for the source. It may be generic (e.g. "C1", "C1") or descriptive (e.g. "audio", "stage 3 pressure").

3 TIME

As in the natural world, RBNB imposes a *time* dimension. All data is time-stamped, and time must monotonically increase from one source frame to the next. This time-stamp can be a simple index that counts frames (1,2,3…), an actual time-of-day, or an application-specific value. It is this time dimension that distinguishes RBNB from other middleware "message" servers. The RBNB server keeps track of time for incoming data, and ring-buffers data for time-selected access to historical information.

The restriction that time-stamps monotonically increase keeps data in sorted-order such that they may be very efficiently searched and accessed. This restriction may at times seem onerous, but it matches the natural time-sequencing of events in the real world and therefore applies to many practical situations.

4 HIERARCHICAL DATA STRUCTURES (RMAPS)

RBNB *data*, *name*, *time* paradigms are integrated to become the "RBNB Channel Map" or RMap object. RMaps store and manipulate all RBNB data. An RMap is a simple object comprised of sub-objects *Name*, *TimeRange*, *DataBlock*, and optionally other RMaps, as shown in Figure 1:

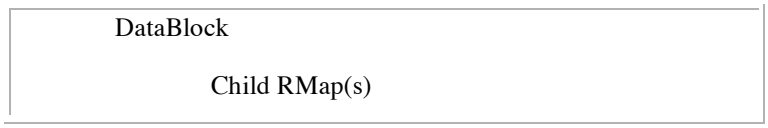

Figure 1: RMap Object Structure

Through the power of recursion (e.g. fractals), the simple RMap structure of Figure 1 can encompass sophisticated, hierarchical representations of interdependent information.

4.1 DATAMAPS

DataMaps are a collection of RMaps (and thus an RMap themselves), and are the "workhorses" of the RBNB server. Sources send data to *DataMaps*, and sinks request data from *DataMaps*. There are different structures to *DataMaps*, all constructed from the fundamental RMap objects. The different *DataMap* structures address the various needs and operating modes of practical RBNB applications.

The collection of information sent in one bundle by a data source is called a *DataFrame* . The simplest form of *DataFrame* is a single RMap with a name, time, and data. A collection of *DataFrames* is known as a *DataMap* (also an RMap). *DataFrames* in a *DataMap* can be organized in various hierarchical schemes. Figure 2 illustrates a *DataMap* comprised of two simple *DataFrames*.

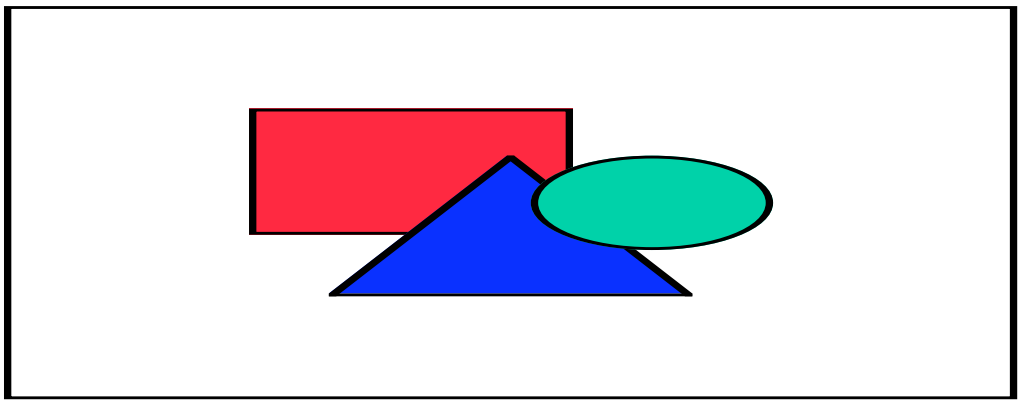

Figure 2: DataMap with Two Fully-Specified DataFrames

In Figure 2, each *DataFrame* is is a fully-specified RMap, containing the three RMap objects: *Name*, *TimeRange*, and *DataBlock*. Every RMap has each of these three objects either explicitly as in Figure 2, or implicitly through inheritance.

The *Name* object is the label by which RMaps are referenced and retrieved. The *TimeRange* object specifies the time (start and duration) of the corresponding *DataBlock* object, which holds data "payload" of each *DataFrame*.

4.2 RMAP INHERITANCE

In many situations, certain parts of a *DataMap* are repeated over and over again, as new *DataFrames* are added. For example, the channel names may remain constant as new frames of time-stamped data are sequentially added by a data source.

A concept called *inheritance* can greatly reduce redundancy and improve the efficiency of building and accessing a *DataMap*. Using inheritence, a more complex (e.g. multi-channel) *DataFrame* can be comprised of a collection of partially-specified RMaps. An RMap that lacks one or more sub-objects (*Name*, *TimeRange*, *DataBlock*) inherits the missing pieces from its parent RMap.

Figure 3 illustrates how a "time ordered" *DataMap* is a natural structure that evolves as a data source sequentially sends *DataFrames* comprised of a common group of channels. TimeRange-only RMaps are parents to groups of named-data RMaps. The data-containing RMaps inherit *TimeRanges* from their parent. Thus, the *TimeRanges* common to a group of channels need not be repeated; saving storage. With each update from the data source, another *TimeRange*-based branch of the *DataMap* is simply and efficiently added.

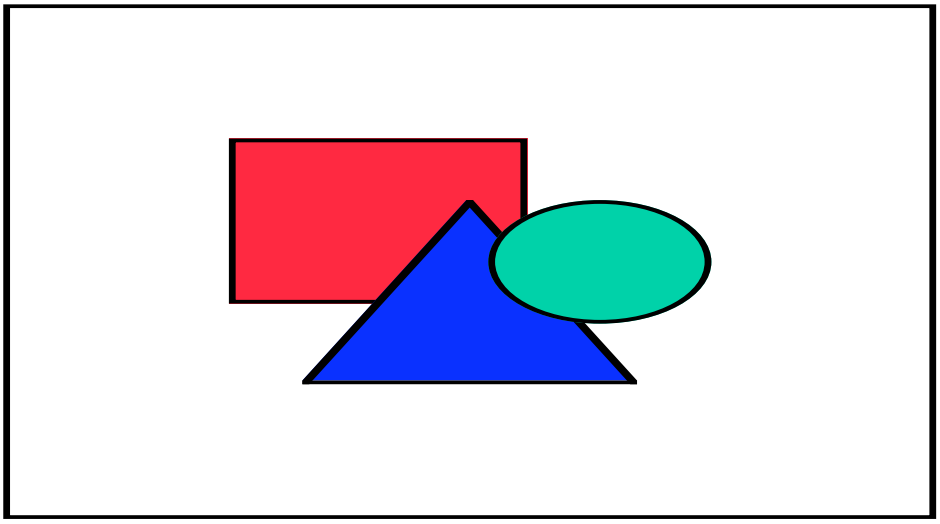

Figure 3: Time-Ordered DataMap

4.3 OTHER RMAP STRUCTURES

A variety of sophisticated and efficient *DataMap* structures are possible through the use of other RMap structures. Whereas time-ordered *DataMaps* are most efficient for data sources, "channel-ordered" *DataMaps* are usually preferred by data sinks. That is, a sink can most conveniently deal with data sorted such that it has contiguous blocks of data for each channel. Figure 4 illustrates a channel-ordered *DataMap*.

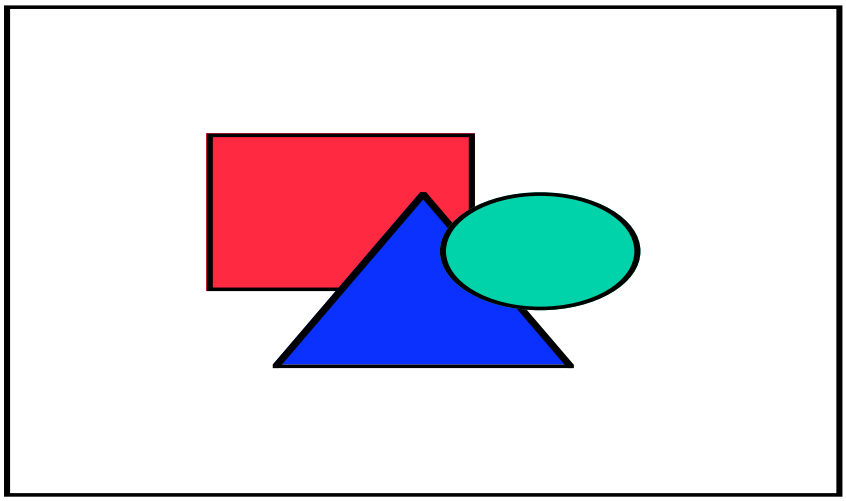

Figure 4: Channel-Ordered DataMap

The RBNB automatically translates *DataMaps* from time-ordered to channel-ordered in response to sink requests. The RBNB can also transform DataMap structures on-the-fly to using inheritance to efficiently store only the parts of the structure that are changing.