# Design and Implementation of a MongoDB Driver for Prolog

Sebastian Lundström

June 8, 2011

**Abstract**

[XXX Prolog is good stuff, and MongoDB ...]

# Contents

# 1 Introduction

[XXX Is the following more suitable as Abstract material? If so, what should the introduction contain?]

MongoDB is a document-oriented database system with a strong focus on flexibility, scalability and performance. Document-orientation involves leaving the row-centric concept of the relational database model, and introducing the much more flexible notion of a document. Document-orientation avoids rigid database schemas and also promotes a certain degree of denormalization which allows embedding documents into each other, leading to potentially much better performance by avoiding the need for expensive join operations.

Prolog, being an untyped language, agrees with the document-oriented approach of relaxing manifests in order to create more dynamic and flexible systems. Embedding terms in other terms is natural in Prolog, and embedding documents in other documents is natural in MongoDB.

In order to use MongoDB from a programming language, some kind of interface, or "driver", must sit in-between the database system and the language to facilitate the communication.

# 2 Purpose

The purpose of this thesis is to discuss the design and implementation of a MongoDB driver for Prolog.

# 3 Scope

The aim of this thesis is not to create a feature-complete production-ready driver, but to cover basic CRUD (Create, Read, Update, Delete) functionality and connection handling, laying the foundation for further development. More advanced MongoDB features such as replication and file storage are not covered.

Due to the lack of proper standardization within the Prolog community, maintaining portability between different Prolog implementations is difficult and not actively pursued.

# 4 Background

This chapter briefly discusses MongoDB and Prolog.

## 4.1 MongoDB

MongoDB is a document-oriented database management system that emphasizes speed, scalability and flexibility. This is accomplished by avoiding fixed database schemas and, by sacrificing joins, requiring a certain degree of denormalization by embedding documents into each other. [XXX ref: MongoDB: The Definitive Guide, p 4]

### 4.1.1 Document-Orientation

[XXX lots of references missing here right now]

MongoDB employs a document-oriented view of data. Instead of focusing on the "row" as the primary data entity, document-orientation introduces the notion of a "document". [xxx ref mongodb p 1]

A document in MongoDB resides in a "collection", and no predefined schema is enforced on the collection. Therefore, documents in a collection need not share the same structure. This allows for great flexibility, especially with data migrations, as individual documents can be modified freely without updating a schema and affecting all documents in a collection.

A document consists of key/value pairs, where keys are strings and values can be of many different types, including other documents (embedded documents). Allowing recursive structures in the database is a step away from the more traditional concepts used in relational databases, where a high degree of normalization is often desired. Relaxing the need for normalization makes it possible to create highly efficient systems by avoiding expensive join operations.

### 4.1.2 Main Features

**Document-Oriented**  MongoDB is designed to be easy to scale, a goal which is easier to achieve when the relational model is not used. Document-oriented databases are more flexible and impose less restrictions on the layout of data, allowing for complex data structures to be stored within a single document. [2][XXX ref? Mongo book p 1]

**Scalability**  The MongoDB approach to scaling involves "scaling out" – the process of adding more commodity servers to a cluster instead of replacing existing servers with better (and more expensive) ones. [xxx ref mongodb p 2] Adding more machines when the need arises is straight-forward and the database system itself figures out how to best configure the machines. [xxx ref mongodb p 3]

**Low Maintenance**  Because there are no database schemas, and databases and collections can be created on-the-fly without explicit commands, MongoDB requires very little manual maintenance. The database system is designed to be very automatic. [xxx ref mongodb]

4

### 4.1.3 BSON

At the core of MongoDB lies the BSON (Binary JSON) data format which is used to communicate data as well as store data on disk. When a driver communicates with a MongoDB instance, documents are transmitted over the network as a series of BSON encoded bytes.

BSON is similar to and heavily influenced by JSON, the main difference being that BSON is a binary format whereas JSON is plain-text. The reason for using a binary format is efficiency. With JSON, numbers need to be converted to and from text in order to be used. This kind of conversion is generally slow, and BSON therefore extends the JSON model with a set of data types that can be parsed directly. For instance, a 32-bit integer is encoded in BSON as four consecutive little-endian bytes. This wastes some space for small integers, but is much faster to parse. [xxx ref bsonspec.org faq?]

## 4.2 Prolog

[XXX How much Prolog do I need to describe? Just enough to understand the code examples later on?]

Prolog is an interpreted dynamic language with its roots in logic. It is also a declarative language, implying that the programmer specifies *what* the program is supposed to solve, not *how*.

This section gives a small introduction to Prolog in order to more easily appreciate the code examples in subsequent sections. [xxx should i say something like "for a more thorough explanation see ... "? or is that out of scope?]

### 4.2.1 Type System

The type system used in Prolog is dynamic, and the fundamental data type is the "term", which can be any value. A variable can be bound to any term, but once bound, it is immutable and cannot be rebound. Variables are recognized by an uppercase initial.

### 4.2.2 Program Structure

The fundamental unit of code in Prolog is the predicate. A predicate consists of one or more clauses, together forming a logical disjunction. The value of any predicate is either true or false, indicating whether the predicate succeeded. Any other output must be specified as output-arguments that become bound

by the predicate before it returns. The convention used for argument order is input-arguments first, intermediate values in the middle, and output-arguments last [xxx ref prolog guidelines].

A solution to calculate the number of elements in a list is given as follows, illustrating that the order of arguments is input, intermediate values, and finally output. This predicate uses an accumulator to construct the final result. Anonymous variables, or "don't care" variables, are denoted with an initial underscore. [xxx ref]

```
length([], N, N).
length([_|T], N0, N) :-
    N1 is N0 + 1,
    length(T, N1, N).
```

In an interactive Prolog environment, the above predicate may be used like this, providing a starting value for the accumulator.

```
?- length([a,b,c], 0, Length).
Length = 3
```

A predicate is usually referred to by its name and arity (number of arguments), and, if possible, its module: MODULE:PREDICATE/ARITY. If the LENGTH predicate above would be defined in the module LISTS, it would be referred to as LISTS:LENGTH/3.

### 4.2.3 Data Structures

[XXX lots of missing refs]

The main primitive data types in Prolog are integers, floats and atoms. In SWI-Prolog [xxx ref SWI] integers are unbounded, and floats are represented as 64-bit IEEE 754 doubles. Atoms represent UTF-8 encoded text.

A list is an ordered collection of zero or more terms which need not be the same type. A non-empty list is a recursive structure which consists of the "head" which is the first element, and a "tail" which is itself a list of all elements except the first.

A structure is ... [xxx explain]

### 4.2.4 Control Flow

Branching is performed by means of "backtracking" into an appropriate clause, and repetition is performed by recursion.

The first step in determining which clause to use for a certain set of input is accomplished through "unification", sometimes called "pattern matching". Prolog tries to unify the input with the formal parameters of the first clause of the predicate. If the unification fails, the process is repeated with the second clause, and so on. Invoking the predicate LENGTH shown in 4.2.2 with a non-empty list causes unification to fail in the first clause (an empty list cannot be unified with a non-empty list), but succeed in the second (the first element is selected but ignored, and the tail is bound to the variable T).

### 4.2.5  Documentation Convention

When documenting Prolog predicates, each argument is usually prefixed with a symbol that indicates how the argument is supposed to behave. The following documentation convention is employed by subsequent chapters.

+  Input argument. The argument must be already instantiated.

-  Output argument. The argument must not be already instantiated, but instead instantiated by the predicate.

?  The argument may or may not be instantiated. This is commonly used when the predicate can be used to either verify the argument or instantiate it.

## 5    Requirements

This chapter discusses the functionality required by the driver.

[XXX main ref: http://www.mongodb.org/display/DOCS/Mongo+Driver+Requirements]

### 5.1    BSON Conversion

[xxx ref: http://bsonspec.org/]

The data representation format used by MongoDB is a binary encoded variation of JSON, called BSON (Binary JSON). In MongoDB, BSON is the format used to store data on disk as well as transmit data over the network.

The driver must be able to convert back and forth between some suitable Prolog structure and BSON bytes.

## 5.2 Connection Management

The driver must be able to obtain a connection to a database server instance using a TCP (Transmission Control Protocol) socket. All communication with the database is performed through this socket.

## 5.3 Document Handling

The driver must expose functionality to find, insert, update and delete documents in the database.

# 6 Design

This chapter discusses the overall design of the driver.

[xxx this will probably be the biggest chapter, I wager]

## 6.1 Driver Structure

[xxx discuss modules, mongo, bson, etc. What module to include etc. ?]

## 6.2 BSON Handling

At the core of MongoDB lies the BSON binary data format which is used to communicate data as well as store data on disk. When communicating with a MongoDB instance, documents are transmitted over the network as bytes of BSON. The driver converts Prolog structures into series of BSON bytes before sending them to the database, and database responses are converted back to Prolog structures again.

A BSON document is represented in Prolog as a list of key/value pairs. A pair is a structure named '-' (dash) with two arguments, a key and a value. Using a symbol as the structure's name makes it possible to write the name as an infix operator, avoiding the need for parentheses. Keys are UTF-8 encoded atoms, and values can be of several different types.

### 6.2.1 Document-to-BSON Conversion

[XXX 6.2.1 and 6.2.2 will probably be rewritten and merged into one section. The conversion predicate works ways and could be described as such.]

A document with two fields may look like this:

```
[name-'MongoDB', type-db]
```

### 6.2.2  BSON-to-Document Conversion

When converting a document to BSON, a list of bytes is obtained.

When converting between structured documents and BSON encoded bytes, the relation BSON:DOC_BYTES/2 can be used. This is a two-way relation that accepts a document and returns bytes, or accepts bytes and returns a document. xxxxxxxxxxxxxx

Given a list of BSON encoded bytes, the relation BSON:DOC_BYTES/2 can be used to convert it into its structured equivalent. The empty document is represented in BSON as five bytes, the first four being a 32-bit little-endian integer representing the length of the entire document, and the last byte signaling the end of the document with a zero.

```
?- bson:doc_bytes(Doc, [12,0,0,0,16,97,0,42,0,0,0,0]).
Doc = [a-42]
```

If a series of BSON bytes represents a concatenation of documents, the relation BSON:DOCS_BYTES/2 can be used. The following yields an empty and a non-empty document wrapped in a list.

```
?- bson:docs_bytes(
       Docs,
       [5,0,0,0,0,12,0,0,0,16,97,0,42,0,0,0,0]).
Docs = [[],[a-42]]
```

An exception is thrown if the conversion fails [xxx detail the exception].

### 6.2.3  Document Manipulation

[XXX This section got very manual-ish. Is this relevant? Should I keep it, rewrite or omit?]

To make working with documents easier, a set of simple helper predicates are provided.

**bson:doc_is_valid(+Doc)**  True if Doc is a valid document and can be converted into BSON encoded bytes.

9

**bson:doc_empty(?Doc)**    True if Doc is an empty BSON document. Can be used to both check for emptiness as well as obtain an empty document.

**bson:doc_get(+Doc, +Key, ?Value)**    True if Value is the value associated with Key in Doc or +null if Key cannot be found.

**bson:doc_get_strict(+Doc, +Key, ?Value)**    True if Value is the value associated with Key in Doc. Fails if Key is not found or does not match Value.

**bson:doc_put(+Doc, +Key, +Value, ?NewDoc)**    True if NewDoc is Doc with the addition or update of the association Key-Value.

**bson:doc_delete(+Doc, +Key, ?NewDoc)**    True if NewDoc is Doc with the association removed that has Key as key. At most one association is removed. No change if Key is not found.

**bson:doc_keys(+Doc, ?Keys)**    True if Keys is the keys for the associations in Doc.

**bson:doc_values(+Doc, ?Values)**    True if Values is the values for the associations in Doc.

**bson:doc_keys_values(+Doc, ?Keys, ?Values)**    True if Doc is the list of successive associations of Keys and Values.

## 6.3    Network Communication

Communication with a MongoDB instance is conducted using a TCP socket. [xxx ref mongo wire protocol]

### 6.3.1    Obtaining a Connection

A connection can be established by the predicate MONGO:NEW_CONNECTION/1,3. Two versions of the predicate are provided, one which assumes the default MongoDB host and port, and one which allows this to be specified. If an error occurs during the setup, an exception is thrown.

```
?- mongo:new_connection(Connection).
Connection = [opaque connection handle]
```

The following will connect to a MongoDB instance running on localhost attached to the port 1234.

```
?- mongo:new_connection(localhost, 1234, Connection).
Connection = [opaque connection handle]
```

### 6.3.2   Releasing a Connection

When a connection is no longer needed it must be properly released back to the system. Assuming *Connection* is a connection obtained by the predicate MONGO:NEW_CONNECTION/1,3, the following will release any resources associated with it, rendering it unusable.

```
?- mongo:free_connection(Connection).
[Connection may no longer be used]
```

## 6.4   Accessing the Database

[XXX This is where I will discuss the interesting stuff that you would actually use.]

### 6.4.1   Find Documents

[XXX describe the various interfaces, how cursors work etc]
   find_one/3, find_one/4, find_all/4, find/7, find/8, kill/1, kill_batch/1, get_more/4, has_more/1, exhaust/2

### 6.4.2   Insert Documents

insert/2, insert_batch/3

### 6.4.3   Update Documents

upsert/3, update/3, update_all/3

### 6.4.4   Delete Documents

delete/2, delete/3

### 6.4.5   Database Commands

command/3

## 6.5 Error Handling

Errors issued by a failed query is detected by the driver and an exception is thrown that contains the error document returned by the database. The error document contains a key called '$err' that maps to an atom describing the nature of the error.

# 7 Implementation

[XXX What is this chapter supposed to contain, really? Discuss code organization, modules? Discuss the test suite? Discuss portability? Discuss efficiency (relevant?)?]

## 7.1 C Extension

A small part of the BSON handling is written in C instead of Prolog. This part involves conversion of integers and floats to and from bytes. The two reasons for this are efficiency and ease of programming.

Being able to subvert the type system in C makes it trivial to populate a fixed-width number with individual bytes and then interpret them as a whole. This is especially convenient when converting bytes into floating-point.

BSON conversion is something the driver does very often, and pushing common operations down to a lower-level language is likely more efficient.

## 7.2 Environment

[XXX Is this relevant?]

**Prolog** The driver is developed and tested on SWI-Prolog 5.10.2.

**C Compiler** An ANSI C compiler is required to build parts of the low-level BSON handling.

**MongoDB** The driver is tested on MongoDB versions 1.8.0 and 1.8.1.

# 8 Evaluation

*[xxx Did it work? Is it usable? What should have been done differently?]*

# 9 Related Work

*[xxx Compare to existing drivers? Erlang? And something completely different? Will probably skip this chapter. Doesn't feel that interesting.]*

# 10 Conclusion/Future Work [merge or split?]

*[xxx Not sure how much this section relates to Evaluation above.*

*Portability (Tested on Mac, SWI, GCC/Clang, etc.)? Efficiency? How to improve in the future? Write a C extension with more/most functionality? Incorporate BSON C implementation? Don't know how portable that would be though, but SWI (and probably SICStus?) has a mature interface to C. What is missing for it to be a "real" driver? Discuss choice of list as BSON document. Maybe use some map-like structure instead.]*

# References

[1] [xxx how should these be ordered? alphabetical last names? publish date? in order of first referenced in text?]

[2] Chodorow, K. & Dirolf, M. (2010) *MongoDB: The Definitive Guide.* Sebastopol, United States of America: O'Reilly Media, Inc.

[3] Clocksin, W. F. & Mellish, C. S. (1994) *Programming in Prolog.* 4th ed. New York, United States of America: Springer-Verlag.

[4] [xxx unreferenced] Bratko, I. (2001) *PROLOG Programming for Artificial Intelligence.* 3rd ed. Essex, England: Pearson Education Limited.

[5] [xxx unreferenced] O'Keefe, R. A. (1990) *The Craft of Prolog.* Cambridge, United States of America: The MIT Press.

[6] Some Coding Guidelines for Prolog
Covington, M. A. et al. Retrieved 2011-06-07. [xxx how to format web refs?]
http://www.ai.uga.edu/mc/plcoding.pdf

[7] [xxx unreferenced] MongoDB Driver Documentation
http://www.mongodb.org/display/DOCS/Drivers

[8] [xxx unreferenced] BSON Specification, Retrieved 2011-06-07.
    http://bsonspec.org/

[9] [xxx unreferenced] SWI-Prolog Documentation
    http://www.swi-prolog.org/pldoc/