Design and Implementation of a MongoDB Driver for Prolog

Sebastian Lundström

April 14, 2011

Abstract

Prolog is good stuff, and MongoDB ...

Contents

1 Diary (not part of the report)

- March 29. Wrote a decoder that decodes {hello: world} and {hello: 32}. Learned some DCG and rewrote the parser with it.
- March 30. Researched how SWI interacts with C. Wrote a bytes8_to_double function in C and managed to integrate it with the rest of the system. Separated bson into separate files, encoder and decoder. Should think about researching PlUnit by now. Thought about and tried PlUnit. Works fine. But how to organize tests? Implemented all bit-hacking in C for the time being. Works. Simple. Fun. Started looking at PlDoc. Generating actual docs seems like a hassle, probably won't do it. But the important thing is unit tests, and some comments inside the source. Started fleshing out the report slightly, with some ideas and stubs.
- March 31. Restructured project folder. Better including of modules now, I think. Starting to write nice unit tests. Continued with the decoder a little, the complex BSON example works. Wrote nice TextMate snippets for test boilerplating. Rewrote Makefile, yet again. Added make test, for instance.
- April 1. When trying to fix proper decoding of strings I ran into issues with Unicode. Finally found SWI-Prolog's memory files, which can be written to and then read back using a different encoding. Five line fix. Implemented a variant that might be more efficient. Might. (And like five times longer, but clean though.) Cleaned up the tests. Implementing the rest of the decoder should be straight-forward now. Next week. It. Will. Be. Awesome.
- April 2. Looked into the Prolog module system again and started adding prefixes ("memory_file:", "builtin:") to SWI predicate calls. "builtin" isn't an actual module, but since it works anyway (it enters the module builtin and calls the predicate from there) I decided to use it. Documented predicates in bson_bits. Fixed a bug with bytes_to_integer/5 because it wasn't using a 32-bit integer. Now it uses int32_t. Started using maplist/2. Found a library(apply_macros) that claims to speed up maplist etc. simply by loading it.
- April 3. All use_module directives now use an empty list in order to ensure that the module is loaded, but nothing is imported. This forces

me to add the appropriate module prefixes to calls. Benchmarked the unicode converter and the old shorter code outperformed the new longer code with more than a factor of three. I guess this is due to more stuff being done by the C library and not Prolog. Anyway, the old code is faster and much shorter – win-win. Also replaced a crappy to-codes-and-then-to-atom conversion with a straight-to-utf8-atom library routine, which made it even faster. A keeper.

- April 4. Refactored decoder slightly. Added relation term_bson/2 to bson, which calls necessary subpredicate. Moved tests into separate files: they were getting too long, and separating them makes it easier to write them simultaneously. As long as tests and implementation are close to each other, I am okay. (Impl in .pl and tests in .plt next to each other.) The decoder should now be feature complete. It does minimal error checking, but it should be able to parse all element types. It might still need some cleaning though. Added version predicates to bson. Made comments more in line with PlDoc.

- April 5. Major decoder cleanup. Took a slow day.

- April 6. Started on the encoder. Unicode issues again. Solved it after an hour or so, using even more library predicates. Factored out Unicode handling to separate module, used by both encoder and decoder. Worked a lot on the encoder, including some C hacking again.

- April 7. Lots more encoding of elements and encoder refactoring. More to come!

- April 8. Clean up a lot of bit-hacking in Prolog. Made entry-predicates into true relations and made the actual conversions into more general list predicates. Changed "builtin:" prefix to "inbuilt:", which looks nicer and is more of a single word. Real cleanup in all the bit-hacking. It sure took must of the day, but things should be pretty solid and clean now, and the strategy is flexible but straight-forward: all integer conversions back-and-forth between signed big/little 32/64-bit and bytes is done in C. If those routines cannot be used, it fails if a negative integer is being encoded, otherwise it is treated as an unbounded unsigned, making for instance ObjectID simple. Everything goes through either the relation FLOAT_BYTES(FLOAT, BYTES) which is just mappings to C, or the more intricate relation INTEGER_BYTES(INTEGER, NUMBYTES, ENDIAN, BYTES) which can be called like: (+,+,+,?) or (?,+,+,+). E.g. (1, 4, little, Bytes) gives Bytes = [1,0,0,0], or (Integer, 4, big, [0,0,0,1]) gives Integer = 1. The first example can be read like "the integer 1 in 4 little(-endian) bytes."

- April 9. Factored out the unbounded unsigned handling into a separate predicate UNSIGNED_BYTES(UNSIGNED, NUMBYTES, ENDIAN, BYTES). Now things are starting to look really good. float_bytes/2 handles doubles, integer_bytes/4 handles all signed 32/64-bit integers and unsigned_bytes/4 handles all unbounded unsigneds. Simple. Also documented the predicates.

- April 10. Made all Unicode handling atom-only. The code became much less confusing, and using code lists for text would be bad anyway because it can be ambiguous (is [97,98,99] a list of numbers or 'abc'?). Yet another implementation of utf_to_bytes/2. Uses fewer weird predicates (no open_chars_stream any more) and seems a tiny bit faster. The code is a bit longer though (but arguably more straight-forward).

- April 11. Encoder is now feature complete. Tomorrow I will start to think about the report and maybe, just slightly, investigate sockets.

- April 12. Started hacking a little on the report structure. Once again renamed "inbuilt:" to "core:". Nicer and shorter. Started looking at sockets. Created mongo module stubs. Managed to setup a basic socket and got a successful response from MongoDB when issuing a query. Can also insert arbitrary docs using the BSON encoder! Things are progressing nicely, and well ahead of schedule.

- April 13. Changed key:value to key=value due to the fact that key:-5.05 is parsed like a rule. This is more in line with the existing JSON parser also. Cleaned up some tests (gave them proper tag numbers). Started working on the API slightly, experimenting away. Produced a MONGO:INSERT(MONGO,DOCUMENT,'DB.COLLECTION') predicate which actually works. Also implemented a MONGO:COMMAND(MONGO,COMMANDDOC,'DB') that can send arbitrary special commands to a database. This is awesome. Wrote an experimental key/value term pretty-printer. Don't really know why, but it works. Might be useful when analyzing response documents returned by the server.

- April 14. Changed key=value to key-value, hoping that this will be final. Using a dash is more idiomatic Prolog and goes hand-in-hand with association lists, should I choose to use them, which seems more and more likely (logarithmic worst-case when fetching value by key etc.). They seem to be well supported by SWI and SICStus, at least, and it seems very easy to convert between key-value lists and assoc lists. Renamed bson:term_bson to bson:pairs_bson, and added bson:assoc_bson, enabling you to choose if a simple list of pairs suffices or if you need to do heavy operations on the document and would prefer predictable performance. The assoc_bson predicate currently just converts to-and-from lists, but the encoder/decoder must of course be changed to deal with association lists from scratch. Rewrote module comments to be acceptable to PlUnit (documentation compiling is great, although it is a bit strict in what it accepts). Removed old n_bytes_to_unsigned_integer from decoder – we have bson_bits for that kind of heavy lifting.

2 Introduction

From spec:

[MongoDB is a young document-oriented database system that has started to gain much attention recently. Document-orientation involves removing rigid database schemas and advanced transactions, in favor of flexibility. Document-orientation 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.

Many drivers exist, both official and unofficial, that enable the use of MongoDB from various programming languages. At the time of writing, no such driver for Prolog seems to exist.]

3 Scope

From spec:

[Creating a driver that is usable with, or at least easily portable to, other Prolog environments is desirable, but development and testing will focus on SWI-Prolog.

More complex features of MongoDB, such as advanced connection management and GridFS, will not be implemented.]

4 Method (necessary?)

Research other drivers, docs.

Test-driven development.

5 Requirements

Various requirements on the implementation. See for example

http://www.mongodb.org/display/DOCS/Writing+Drivers+and+Tools

6 Design

- BSON encoder/decoder
  - Some parts written in C. Why? (Basically didn't know how to easily handle bytes-to-float in Prolog. And perhaps some efficiency.)
  - Discuss data structures, term [key:value] maps to bytelist [4,1,7,9,3,...] etc.
  - Design choices: text as atoms (why not list of codes, [97,98,99]?) Inspired slightly by JSON parser: http://www.swi-prolog.org/pldoc/doc_for?object=section%283,
- Network communication
  - How does the communication work? How simplistic is the communication administration? Sockets, connections, etc.
- MongoDB API
  - Thoroughly discuss design of wrapper API, how lists and structures are represented etc. What algorithms are used etc.

7 Implementation

How do I solve it?

8 Evaluation

Did it work? Is it usable? What should have been done differently?

9 Related Work

Compare to existing drivers? Erlang? And something completely different?

10 Conclusion/Future Work

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? Critical parts (BSON) in pure C? Even write a C extension with more/most functionality? Don't know how portable that would be though, but SWI (and probably SICStus also?) has a mature interface to C. What is missing for it to be a "real" driver?

11 References

References for Prolog DCG? References for various stuff used in SWI? At all relevant? How much web references? Most MongoDB driver stuff is web.

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