Code generation for enterprise integration

Max Lapshin
max@maxidoors.ru

FIX protocol

- Automated stock exchange trading
- Binary protocol for receiving quotes and sending orders
- No opensource erlang implementation

FIX protocol

- Packet is a Key,Value list.
- Keys are ASCII-coded integers
- Values may vary
- Separated with 0x01 byte

FIX protocol

- Binary protocol with three levels: transport (TCP), packet and business
- All business logic is described in protocol spec
- Different message types with many fields and groups of fields
- Very large protocol specification (about 100 different messages)
- 10 us limit for all parsing

FIX protocol

- Received data is decoded to proplists
- But we use records in code
- How to translate? And do it fast!

FIX protocol

- Large XML description how to compose objects from proplists
- XML describes syntax and semantic levels

Problem

- Every FIX user needs only small subset of whole spec
- Impossible to work with full FIX records (more than 100 fields)
- My code works with my records and business logic
- How to fight impedance between our logic and FIX business logic?

Ways to go

- Enterprise
- Ad-hoc
- Controlled code generation

Enterprise way

- Object with 100 fields and 300 methods is ok
- It is Java: abstract singleton factory will make you happy
- Copy-pasted glue code will translate full spec to required subset
- Impossible to use because large size of records
- Not my way

Ad-hoc

- Write custom code for each message type
- Workaround for broker's bugs
- Ok for some situations
- Not my way: I'm too lazy to copy-paste code by hands

Right way

- Autogenerate proplists-to-record translator
- Make code reuseable for others
- Design system, that can fit into other businesses
- Don’t want to write glue code
- I chose configurable code generation

Code generation

- Parse XML
- Reduce messages according to config file
- Generate headers
- Generate parsers
- Profit!

Headers

- Translate only required FIX messages to records
- Include only required fields in these records
- Always include “fields” field to each record for remaining data
- Dialyzer-compatible: types are known at generate time

Example

NewOrderSingle message with 60 fields is translated to

#new_order_single{} with 6 fields

Syntax parser

- Translate binary fields to erlang types and back
- Don’t forget sugar: translate 0,1,2 to atoms buy, sell, make_gift
- Doesn’t depend on config
- Generate C extension (blazing fast)
- Produces readable proplist (not hash!): ordered key-value list

Semantic parser

- Translates proplists to records
- Need to produce records, directly used in business logic
- Can loose some not important information (checksum)
- Need to handle groups of fields (most cryptic part)
- It is generated with config
- Yet have some limited ad-hoc code to keep flexibility

Structure of semantic parser

1. Determine type of message and use compile time record info
2. Process the rest of proplist
3. If key is a known record field, write it to record
4. Else append this {Key,Value} to “fields” field
5. Use “setelement” and generated “field_index(MsgType,KeyName)”

Results

- Simple and useable code
- FIX datatypes are used in business logic directly
- Configurable autogeneration of parser is really helpful
- Working code

Questions?

Max Lapshin

max@maxidoors.ru