Robotics Service Bus - Enhancement #997
Replace ProtoBuf Serialization of RSB Notifications through a Rosetta-based Serialization

06/06/2012 06:23 PM - S. Wrede

Status:	In Progress	**Start date:**	06/06/2012
Priority:	Normal	**Due date:**	
Assignee:	J. Moringen	**% Done:**	20%
Category:	Protocol	**Estimated time:**	40.00 hours
Target version:	rsb-1.0		

Description

We should consider to replace the current protobuf-based serialization of RSB notfications (cf. RSB-Protocol project) by a Rosetta-based Type / Converter solution and add the pre-compiled data holders to the RSB code base (as they shoudn't change so frequently this should be fair). Anticipated benefits:

- fewer dependencies (no pbuf, no rosetta)
- even RSB Protocol will no longer be visible for non-core developers
- leaner base library
- python 3 compatibily may be in reach with this

A precondition for this is of course that we can generate with Rosetta the corresponding serialization code also for Python and Java.

Any opinions (are there severe drawbacks, @Jan: how much work are the Java & Python code generators)?

History

#1 - 06/06/2012 06:39 PM - J. Moringen

> *A precondition for this is of course that we can generate with Rosetta the corresponding serialization code also for Python and Java.*

Actually, we have to choose/invent a serialization mechanism first. In this particular case, I would argue against Protocol Buffer serialization since we do not exploit the advanced features anyway.
Another issue is the runtime/compile time support library (e.g. project:rosetta-lib-cpp).

- Do we want to manage these as separate libraries (on which project:rsb subprojects) would then depend?
- Should project:rosetta-tools-cl just generate them on-the-fly?

> *Any opinions (are there severe drawbacks, @Jan: how much work are the Java & Python code generators)?*

Depends on

- Chosen serialization mechanism
- Required performance

As a final note, I would advise against scheduling this for version:0.7.

#2 - 06/06/2012 06:45 PM - J. Wienke

I would propose to first have a running benchmark solution before changing this. I think we should not introduce a performance degradation.

#3 - 06/11/2012 11:47 AM - J. Moringen

- Status changed from Feedback to In Progress

- % Done changed from 0 to 10

As a preliminary experiment, I'm trying to generate data holders and ROS MSG serialization code for rsb.protocol.Notification for C++, Python and Java.

#4 - 06/11/2012 04:24 PM - J. Moringen

- % Done changed from 10 to 20

Initial C++ benchmark results for a simplified scenario.
Caveats:

- Minimal notification content (see attachment:benchmark.cc)
- Made optional fields in rsb.protocol.Notification required
- Both compiled with -O2
- Notification objects are re-allocated in every iteration (as in the current transport implementation)
- The project:rosetta implementation is handicapped since it has to copy the whole buffer twice in each iteration (for ProtoBuf API compatibility reasons)
- ProtoBuf implementations uses ProtoBuf serialzation format; rosetta uses (less complicated) ROS MSG format

Result of serializing and deserializing 1,000,000 times:

```
[jmoringe@potash protocol-test]$ time ./benchmark-rosetta
real    0m1.394s
user    0m1.390s
sys    0m0.000s
[jmoringe@potash protocol-test]$ time ./benchmark-protoc
real    0m1.813s
user    0m1.800s
sys    0m0.000s
```

#5 - 06/11/2012 04:25 PM - J. Moringen

- File benchmark.cc added

#6 - 06/11/2012 04:36 PM - J. Moringen

Turns out the benchmark-rosetta programs spends ~ 30 % and ~ 20 % of its runtime in deserialization and serialization code respectively. The rest is constructing and destructing the message objects.

#7 - 06/11/2012 04:39 PM - J. Moringen

Same benchmark, but reusing the same Notification object in all iterations:

```
[jmoringe@potash protocol-test]$ time ./benchmark-rosetta
real    0m0.434s
user    0m0.420s
sys    0m0.010s
[jmoringe@potash protocol-test]$ time ./benchmark-protoc
```

```
real    0m0.539s
user    0m0.530s
sys     0m0.000s
```

#8 - 06/11/2012 04:53 PM - J. Moringen

When using memcpy instead of loops in the project:rosetta-lib-cpp support code:

```
[jmoringe@potash protocol-test]$ time ./benchmark-rosetta
real    0m0.334s
user    0m0.330s
sys     0m0.000s
[jmoringe@potash protocol-test]$ time ./benchmark-protoc
real    0m0.608s
user    0m0.600s
sys     0m0.000s
```

#9 - 06/11/2012 11:49 PM - J. Wienke

Is this protobuf encoding? Otherwise what is the difference in message size?

#10 - 06/11/2012 11:57 PM - J. Moringen

The encodings are as follows:

- protoc-based program uses Protocol Buffer serialization with variable-length integers, start codes etc.
- rosetta-based program uses ROS MSG serialization with fixed-length integers, length-delimited coding for arrays

Serialized sizes later. Protocol Buffer serialization is probably a bit smaller for highly structured data with small leafs (e.g. many nested structures but short strings, blobs, etc.). Otherwise, the difference is probably negligible.

#11 - 06/14/2012 03:03 PM - J. Moringen

- Target version changed from rsb-0.7 to rsb-0.9

I think we should not change this for the 0.7 version. The change could be very inconvenient for people who currently use the trunk version and plan to siwtch to 0.7 once it is released.

#12 - 06/14/2012 03:03 PM - J. Moringen

- Category changed from C++ to Protocol

#13 - 06/14/2012 03:08 PM - J. Wienke

I completely agree with the version change.

#14 - 06/14/2012 03:23 PM - J. Moringen

Serialized sizes and running times (real time) for different "size factors" (lengths of strings like scope, wire-schema etc. are multiplied by this factor)
Condition

- 1,000,000 repetitions
- No reuse of Notification objects
- Protocol Buffer data-holder client API
- Native serialization API (avoids rosetta handicap mentioned earlier)

Size factor	Protocol buffers	ROS MSG (via rosetta)
4	55 / 1.953s	104 / 1.154s
8	83 / 2.005s	132 / 1.129s
16	139 / 2.047s	188 / 1.216s
32	251 / 2.476s	300 / 1.211s
64	477 / 2.401s	524 / 1.281s
128	933 / 2.519s	972 / 1.497s

#15 - 06/15/2012 03:26 PM - S. Wrede

Even if the numbers look promising, I agree with targeting at 0.8.

#16 - 02/01/2013 09:09 AM - J. Wienke

- Target version changed from rsb-0.9 to rsb-0.10

#17 - 12/10/2013 11:47 PM - J. Moringen

- Target version changed from rsb-0.10 to rsb-0.11

#18 - 02/06/2014 05:30 PM - J. Moringen

- Target version changed from rsb-0.11 to rsb-0.12

#19 - 02/06/2014 05:32 PM - J. Moringen

- Target version changed from rsb-0.12 to rsb-1.0

Files

benchmark.cc	1.48 KB	06/11/2012	J. Moringen