Messaging Services and Client Software

Lionel Cons – Massimo Paladin

EGI Technical Forum - Prague, 18th September2012

EMI is partially funded by the European Commission under Grant Agreement RI-261611

Outline

- Recommendations for messaging services
- From use cases to client software
- Recommended libraries and software

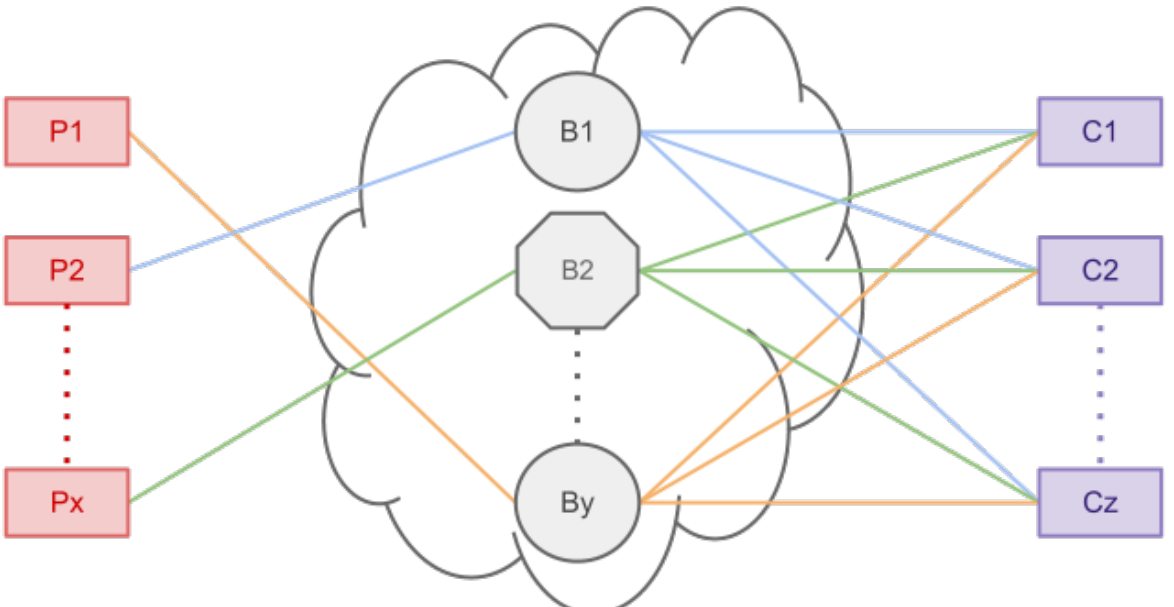
EMI INFSO-RI-261611

Dedicated messaging services of independent brokers

- Dedicated services
  - Application isolation
  - Application specific broker tuning
  - Compatible applications can be grouped
- Independent brokers
  - Horizontal scalability
  - Easier management and operations
  - Heterogeneous products (when needed)
  - Goes well with load-balanced DNS

EMI INFSO-RI-261611

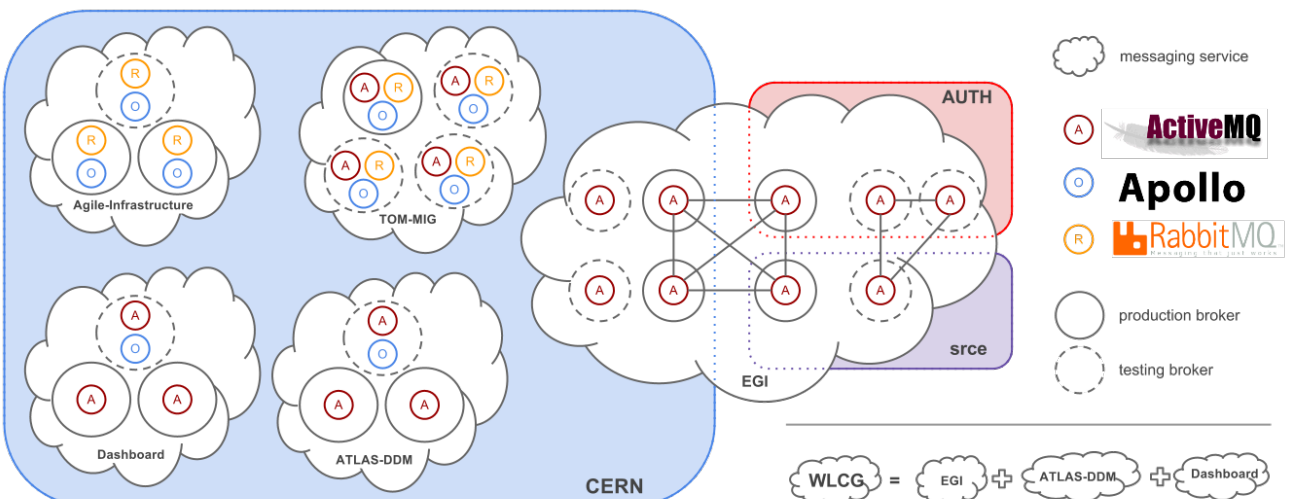
Dedicated messaging services of independent brokers



produce to any *consume from all*

EMI INFSO-RI-261611

WLCG messaging services



* CERN: European Organization for Nuclear Research - AUTH: Aristotele University of Thessaloniki - srce: University of Zagreb Computing Centre

Messaging software

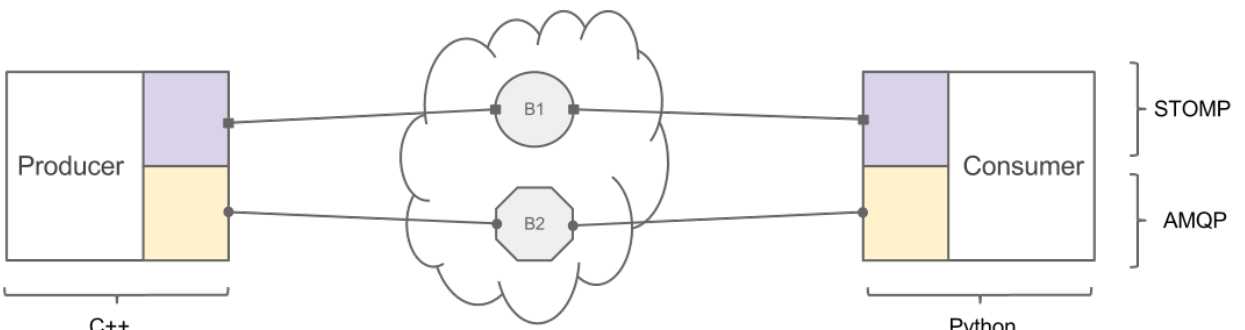
- Many messaging brokers available
  - Recommendations available in our wiki

- Many protocol level client libraries available
  - Different protocols
  - Several alternatives per language

	STOMP	AMQP	OpenWire	…
C / C++	X	X	X	…
Java	X	X	X	…
Perl	X	X		…
Python	X	X		…
Ruby	X	X		…
…	…	…	…	…

Developing messaging clients

- Easy to get something working
- Using different protocols and programming languages leads to code duplication



- Error handling is not trivial at all (messaging is mostly asynchronous)
- Hard to get something working *reliably*

How to solve this?

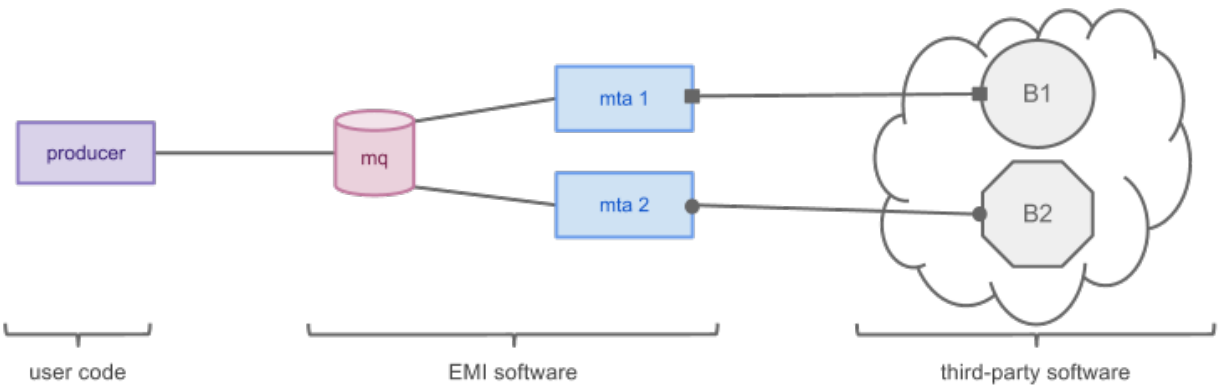
- What about LEGO bricks?
  - Small reusable components
  - Flexible when combined

- What are our bricks?

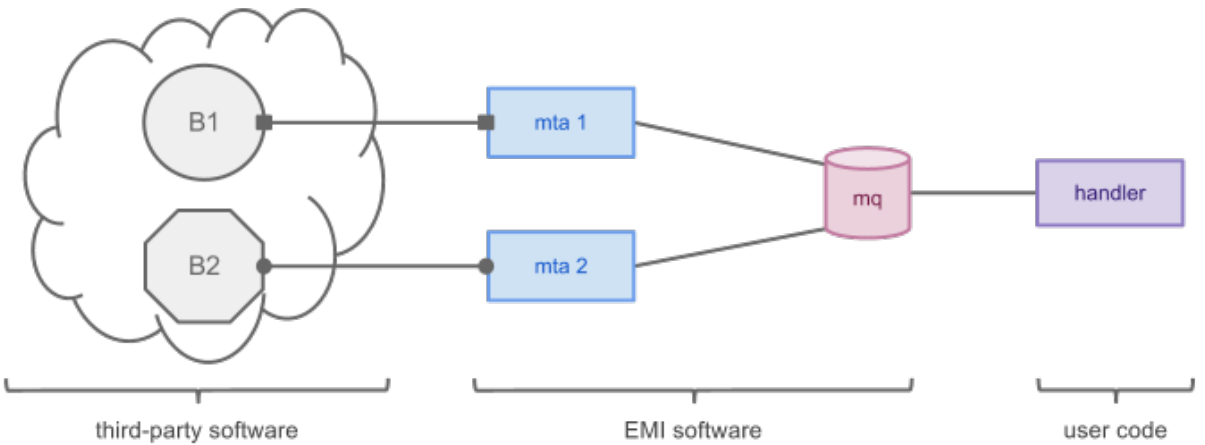
- Message Queue
  - File based message queue
  - Simple and robust API

- Messaging Transfer Agent
  - Transfer messages between a broker and a message queue (all combinations)

Simplifying the producer

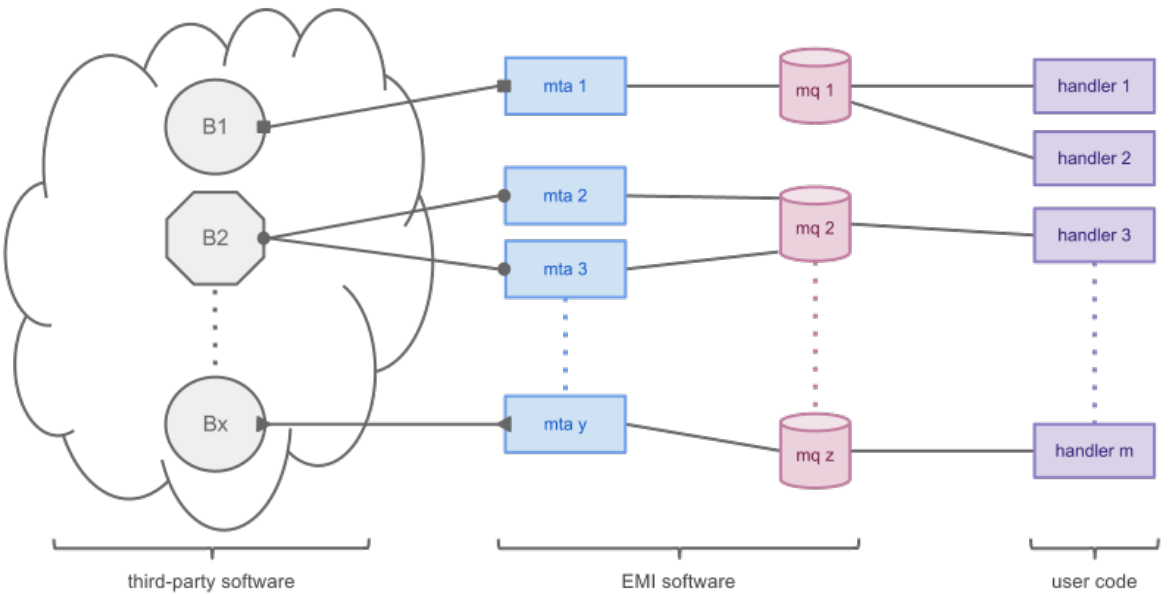


Simplifying the consumer



EMI INFSO-RI-261611

Scaling the consumer side



EMI INFSO-RI-261611

In practice 1/2

Producers and consumers only need to interact with the Message Queue

- Perl: **`perl-Messaging-Message`** + **`perl-Directory-Queue`**
- Python: **`python-messaging`** + **`python-dirq`**
- simple algorithm, easy to port to other languages

```
from messaging.message import Message
from messaging.queue.dqs import DQS

# create a message queue
mq = DQS(path = "/some/where")

# add a message to the queue
msg = Message(body = "hello world")
print("msg added as %s" % mq.add_message(msg))

# browse the queue
for name in mq:
    if mq.lock(name):
        msg = mq.get_message(name)
        # unlock the element
        mq.unlock(name)
        # othwerwise, if you want to remove the element
        # mq.remove(name)
```


EMI INFSO-RI-261611

In practice 2/2

Messaging Transfer Agents

- STOMP protocol: **`stompclt`** (production ready)
- AMQP protocol: **`amqpclt`** (being tested)

`stompclt` sender example:

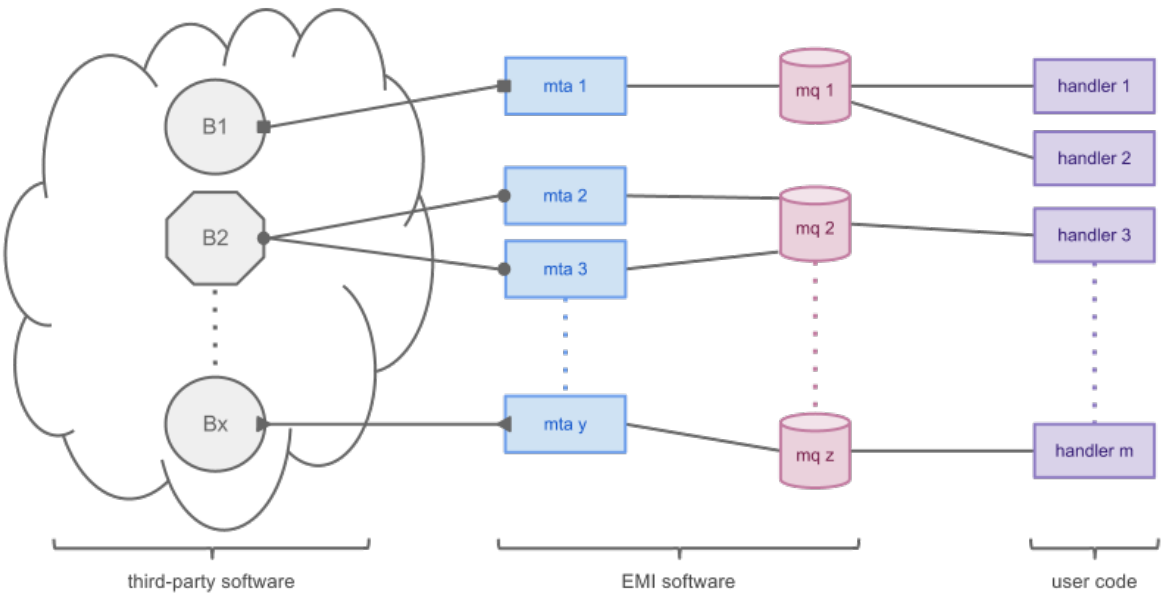
```
<incoming-queue>
 path = /var/spool/sender
</incoming-queue>

callback-code = <<EOF
$hdr{destination} = "/queue/myapp.data";
$hdr{persistent} = "true";
EOF

<outgoing-broker>
 uri = "stomp://broker.acme.com:6163"
</outgoing-broker>

pidfile = /var/run/sender.pid

loop = true
remove = true
```

`stompclt` receiver example:

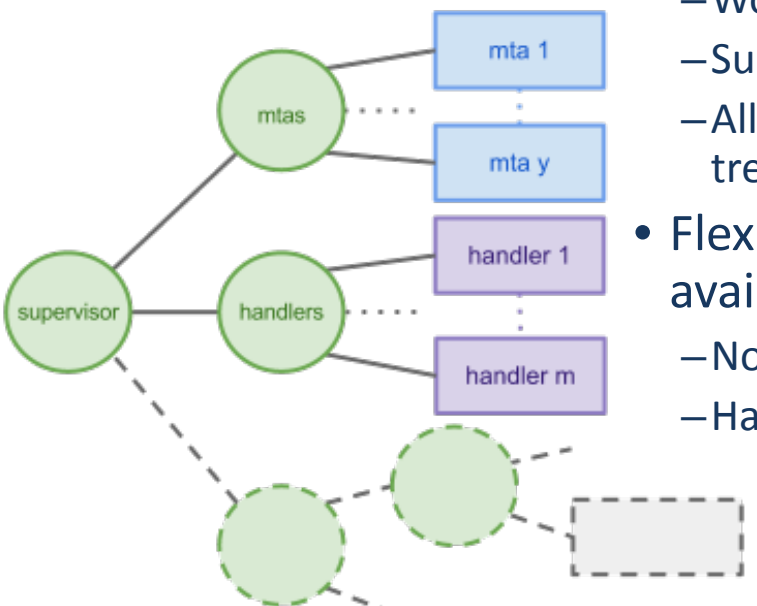
```
<incoming-broker>
 uri = "stomp://broker.acme.com:6163"
 <auth>
   scheme = plain
   name = receiver
   pass = secret
 </auth>
</incoming-broker>

<subscribe>
 destination = /queue/myapp.data
</subscribe>

<outgoing-queue>
 path = /var/spool/receiver
</outgoing-queue>

pidfile = /var/run/receiver.pid
```


EMI INFSO-RI-261611

How can we handle an elastic service?



Reliable services through supervision



- Proven concept (Erlang/OTP)
  - Workers do their work
  - Supervisors monitor workers
  - All are defined in a supervision tree
- Flexible implementation available (**`simplevisor`**)
  - Non intrusive
  - Handle service evolution

EMI INFSO-RI-261611

More about software

- The EMI Messaging Product Team
  - Identified the reusable components
  - Improved the existing ones
  - Developed the missing ones
- All the components are available
  - Most are production ready
  - The others are being finalized
  - All are available in EPEL

Main messaging brokers

Broker	*Qpid*	*MRG*	*HornetQ*	*ActiveMQ*	*Apollo*	*RabbitMQ*
Language	Java	C++	Java	Java	Scala	Erlang
Main Protocols	AMQP	AMQP	proprietary STOMP	OpenWire STOMP	OpenWire STOMP AMQP	AMQP STOMP
Owner (*)	Red Hat	Red Hat	Red Hat	Fuse Source (Progress)		VMware

7 September 2012 : Red Hat completed its acquisition of FuseSource

"Two Gorillas in the Open Source Market Join Forces"

http://fusesource.com/redhat/

EMI INFSO-RI-261611

For more information

- CHEP 2012 paper & poster
  http://cern.ch/messaging-chep2012

- Our wiki
  https://twiki.cern.ch/twiki/bin/view/EMI/EMIMessaging
  (short: http://goo.gl/JZ8o5)

- If you are interested in using messaging or want to provide feedback, contact us

  emi-jra1-messaging@eu-emi.eu

EMI INFSO-RI-261611

Thank you!

EMI is partially funded by the European Commission under Grant Agreement RI-261611