Cronograma, AMQP-Céu e FRCP

Carlos Mattoso

29 de Março de 2016

Resumo

- Cronograma
- Plano de Ação para *AMQP-Céu*
- Federated Resource Control Protocol
- *omf_rc*

Cronograma

- **Abril**: Proposta e Implementação do AMQP-Céu
- **Maio**: OMF EC-Céu
- **Junho**: OMF EC-Céu + relatório final
- **Julho**, **Agosto** e **Setembro**: (melhorias pontuais)
- **Outubro**: conclusão de pendências + [OMF_RC-Céu] + relatório
- **Novembro**: relatório + [apresentação de exemplos com OMF_RC-Céu]
- **Dezembro**: relatório e banca

Cronograma

- **Abril**: Proposta e Implementação do AMQP-Céu
  - **11**: envio de proposta p/ avaliação
  - **18**: entrega de proposta final + *connection e channel setup* + progress report
  - **25**: aceitação da proposta + *progress report*
  - **26**: *exchange.declare*, *queue.declare*, *queue.bind*, *message publishing* + checks programáticos usando ferramentas
  - **03 de Maio**: *message consumption* + testes + exemplos oficiais e apresentação

Cronograma

- **Maio**: OMF EC-Céu
  - **10**: estudo da implementação em Ruby
  - **17**: *defProperty* & *property* & *ensureProperty*, *defApplication* + *progress report*
  - **31**: implementação da base de FRCP + apresentação
- **Junho**: OMF EC-Céu + relatório final
  - **14**: *defGroup* + eventos (nativos em Céu) + exemplos + rascunho do relatório final
  - **21**: relatório final + apresentação de exemplos oficiais feitos em Céu

recap: aplicações

```
defApplication('vstream') do |app|
  app.description = “An app definition for a video stream"
  app.binary_path = “/usb/local/bin/vstream”

  # Definição de parâmetros que o app pode/deve receber
  app.defProperty('target', 'Address to stream from', nil, {
    :type => :string,
    :mandatory => true,
    :default => ‘localhost'
  })
  app.defProperty('fps', 'Frames per second', '-fps',
                  {:type => :integer})
  # Continua

  app.defMeasurement('probe_statistic') do |m|
    m.defMetric('dest_addr', :string)
    m.defMetric('ttl', :uint32)
    m.defMetric('rtt', :double)
    m.defMetric('rtt_unit', :string)
  end

  app.defMeasurement('video_stream_statistic') do |m|
    m.defMetric('frame_number', :uint64)
    m.defMetric('drop_rate', :uint32)
    m.defMetric('codec_name', :string)
    m.defMetric('bitrate', :unit32)
  end
end
```

recap: grupos

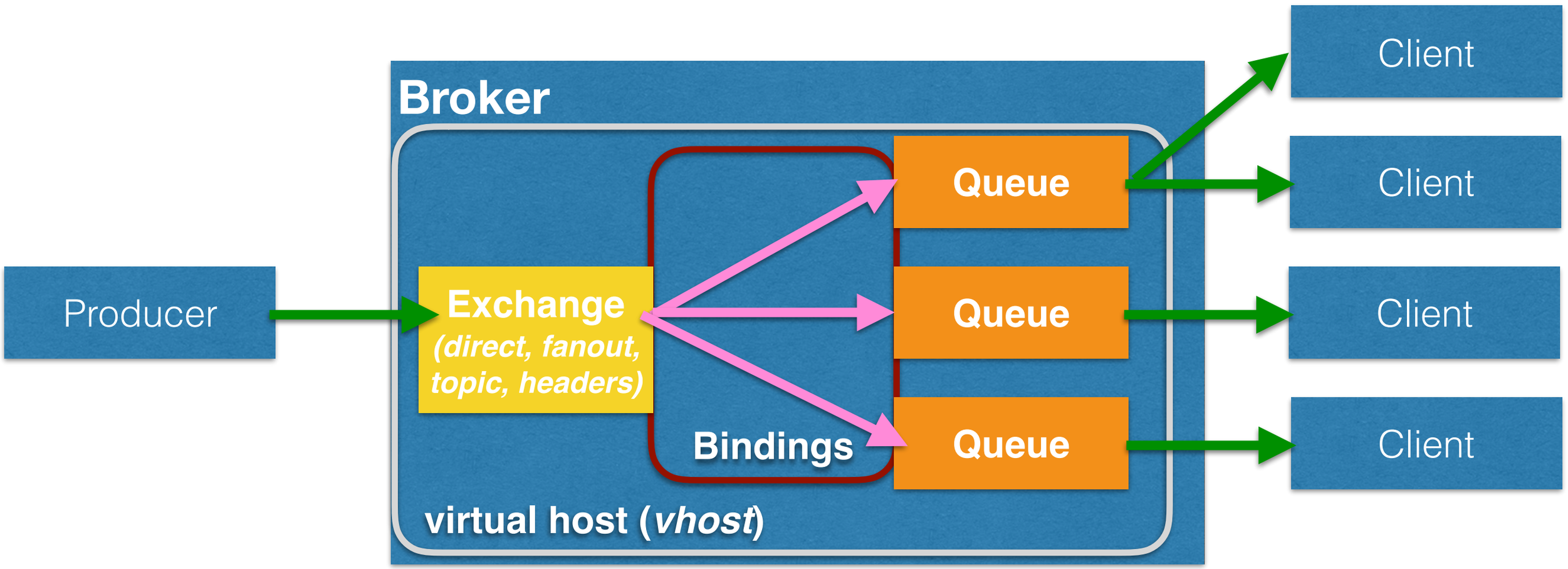
```
defGroup('Sender', "omf.nicta.node2") do |node|
  node.addApplication("test:app:otg2") do |app|
    app.setProperty('udp:local_host', '192.168.0.2')
    app.setProperty('udp:dst_host', '192.168.0.3')
    app.setProperty('udp:dst_port', 3000)

    # Obter métrica em intervalos de 3s
    app.measure('udp_out', :interval => 3)
  end
  node.net.w0.mode = "adhoc"
  node.net.w0.type = 'g'
  node.net.w0.channel = "6"
  node.net.w0.essid = "helloworld"
  node.net.w0.ip = "192.168.0.2"
end
```

Cronograma

- **Julho**, **Agosto** e **Setembro** (melhorias pontuais)
    - A ser definido, mas: melhorias, bug fixes, etc dentro do possível
    - *19 de setembro*: encontro para retomada
- **Outubro**: conclusão de pendências + *[OMF_RC-Céu]* + relatório
- **Novembro**: **relatório** + *[apresentação de exemplos com OMF_RC-Céu]*
- **Dezembro**: wrap-up, **relatório** e banca

AMQP: Entidades



C API

- Síncrona.
- Utiliza *non-blocking sockets* (nas últimas 2 versões, *mid2015*)
- Duas “classes” de métodos da API
  - *blocking* sem controle do usuário (métodos AMQP)
  - *blocking* controlado pelo usuário (e.g. conexão e recebimento de mensagens)
- Ambos limitados por algum *timeout*
- *select/poll* usados por trás dos panos

C API

- Síncrona.
- Utiliza *non-blocking sockets* (nas últimas 2 versões, *mid2015*)
- Duas “classes” de métodos da API
  - *blocking* sem controle do usuário (métodos AMQP)
  - *blocking* controlado pelo usuário (e.g. conexão e recebimento de mensagens)
- Ambos limitados por algum *timeout*
- *select/poll* usados por trás dos panos

C API: Métodos AMQP

- *amqp_exchange_declare_ok_t * AMQP_CALL* ***amqp_exchange_declare****(…);* <-> **exchange.declare**

- *amqp_queue_declare_ok_t * AMQP_CALL* ***amqp_queue_declare****(…);* <-> **queue.declare**

- *amqp_queue_bind_ok_t * AMQP_CALL* ***amqp_queue_bind****(…);* <-> **queue.bind**

- *amqp_basic_consume_ok_t * AMQP_CALL* ***amqp_basic_consume****(…);* <-> **basic.consume**

- e demais métodos AMQP são *blocking* invisíveis (timeout: *heartbeat)*

C API: Arquitetura RPC

- Métodos dependem da função ***amqp_simple_rpc***
- Consiste em um *send* (*non-blocking*) com *timeout == hearbeat*
  - `status = amqp_send_method(state, channel, request_id, decoded_request_method);`
  - `send_with_timeout`
- Posteriormente um *receive* com *timeout == heartbeat*
  - `status = wait_frame_inner(state, &frame, NULL);`
  - `=> amqp_try_send`

C API: Arquitetura RPC

- Algumas funções permitem ao usuário definir o *timeout*
  - *int AMQP_CALL* ***amqp_socket_open_noblock****(…, struct timeval **timeout**);*
  - *amqp_rpc_reply_t AMQP_CALL* ***amqp_consume_message****(…, struct timeval **timeout**, …);*
    - *=>* ***amqp_simple_wait_frame_noblock***

C API: Arquitetura RPC

- *Caveat:*
  - *amqp_rpc_reply_t AMQP_CALL* ***amqp_consume_message****(…, struct timeval **timeout,** …);*
  - *=> ret = **amqp_read_message**(state, envelope->channel, &envelope->message, 0); => **amqp_simple_wait_frame_on_channel** => res = **wait_frame_inner**(state, decoded_frame,* ***NULL****);*
  - Pode bloquear carregando o corpo da mensagem, apesar do usuário ter definido um *timeout* na chamada a *amqp_consume_message*

“rabbitmq-c does not currently support non-blocking operation. You could use threads though: have one thread responsible for AMQP communictions, which passes recieved (sic) messages to other threads for procesing (sic).”

--

David Wragg
Staff Engineer, RabbitMQ
SpringSource, a division of VMware
[17/01/2011]

http://rabbitmq.1065348.n5.nabble.com/Regarding-the-non-blocking-call-of-amqp-simple-wait-frame-tp5278p5279.html

"AMQP is intended to be run on a low-latency, high-bandwidth LAN, […]"

--

Alon Antonuk
Autor principal da *librabbitmq-c*
[16/02/2012]

http://rabbitmq.1065348.n5.nabble.com/rabbitmq-c-Nonblocking-recv-tp19912p19913.html

"rabbitmq-c does not currently have an event loop. Support for heartbeats is still considered partial: they are correctly serviced doing a blocking read from the socket, and PR#256 will land support for heartbeat servicing when in a blocking write (publish)."

--

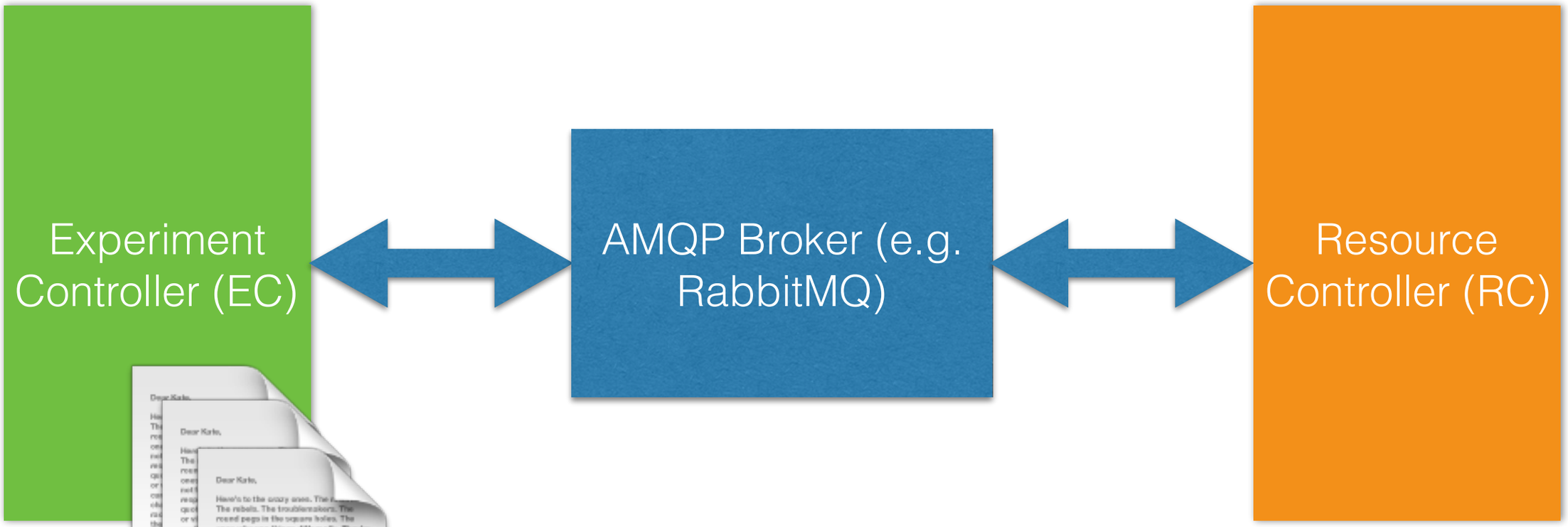
Alon Antonuk
Autor principal da *librabbitmq-c*
[17/04/2015]

https://groups.google.com/forum/#!topic/rabbitmq-users/w_xWfrEOJLA

Plano de Ação

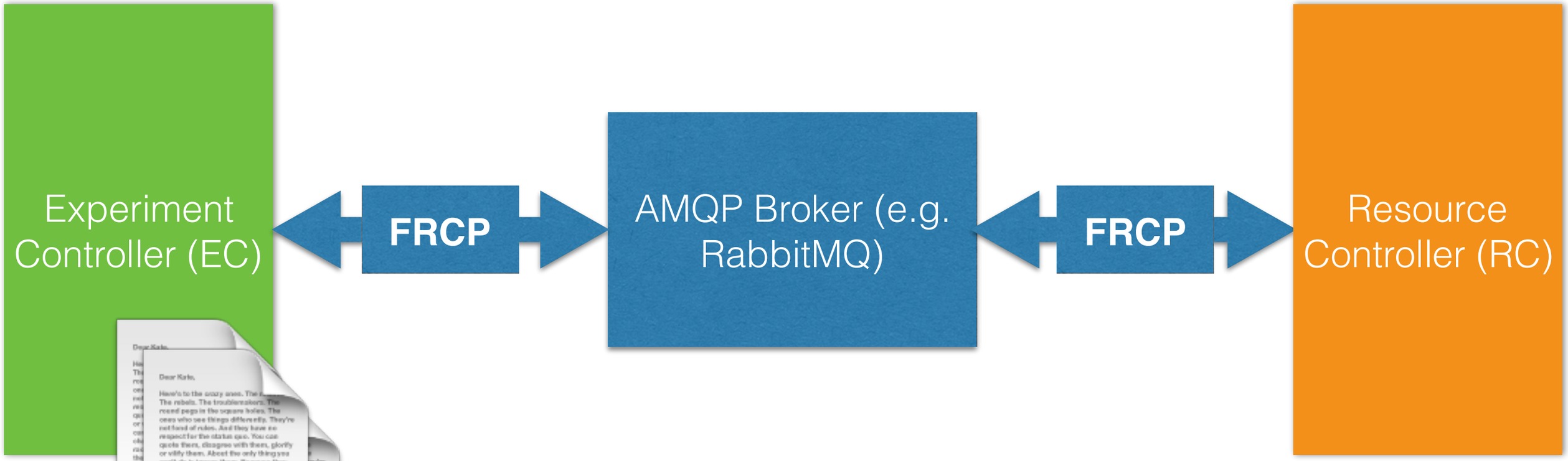
- Delegar a uma *thread* o envio e recebimento de mensagens
- Supera a limitação estrutural do tratamento de *heartbeats* durante *leituras/escritas*

OMF: Mais simples



Experiment Description (em OEDL)

OMF: Mais simples



Experiment Description (em OEDL)

Federated Resource Control Protocol

- Protocolo de troca de mensagens para controle e orquestração de recursos distribuídos.

- Mensagens tem tipos: *inform*, *configure*, *request*, *create*, *release*.

- Troca de mensagens deve seguir regras.

OMF: *deep dive*

Res1 ... ResN

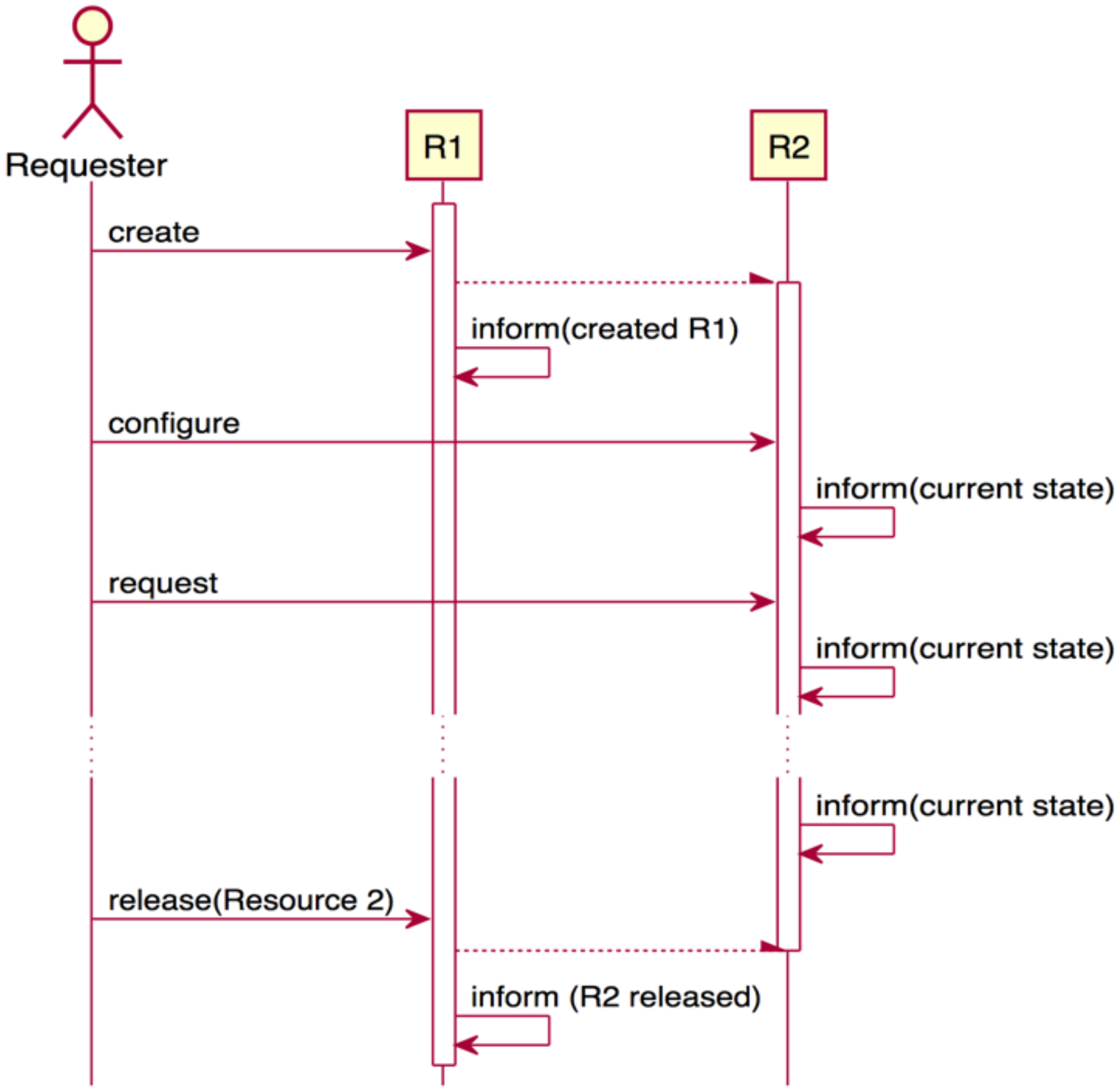
Experiment Controller (EC)

Resource Controller (RC)

FRCP

FRCP

message broker (XMPP or AMQP)


Requester
R1
R2
create
inform(created R1)
configure
inform(current state)
request
inform(current state)
inform(current state)
release(Resource 2)
inform (R2 released)

Formato da Mensagem

```
{
  "op": MTYPE,
  "mid": ID,
  ["cid": ORIG_CONFIG_MSG_ID,]
  "src": RID,
  "ts": TIMESTAMP,
  "rp": TOPIC,
  ["it": INFOTYPE,]
  "props": {
    "@context": “http://foo.com/goo",
    "key1": "some_other_values",
    "key2": {
      "val": “some_value"
    },
    "key3": {
      "val": 1024,
      "unit": “bps"
    }
  },
  ["guard": {
    "@context": "http://foo.com/goo",
    "key1-to-match": "val-to-match",
    "key2-to-match": "val-to-match",
    "key2-to-deep-match": ["val1-to-match", "val2-to-match"]
  }]
}
```

Inform

1. Podem ser publicadas espontaneamente para notificar mudanças no estado do recurso.

2. Publicadas para responder a comandos recebidos através dos demais tipos de mensagem.

3. Devem ser publicadas no tópico principal e, caso *rp* estivesse definido, uma cópia neste tópico.

Configure

1. Deve tentar atualizar as propriedades definidas em *props*.

2. Deve *informar* em seu tópico principal o resultado da atualização (setando *cid* e *props*).

Request

1. Ao receber uma mensagem *request*, deve publicar uma mensagem *inform* em seu tópico principal com os atributos definidos nos *props* de request.

2. Caso o *request* tenha um *rp* uma cópia da mensagem deve ser publicada no tópico definido em *rp*.

Create

1. O recurso pai *P* recebe um pedido de criação de um novo recurso filho *C*.

2. Caso aprovado, *P* define uma chave única global e um tópico para *C*.

3. Recurso é preparado e inicializado.

4. A mensagem pode conter *props* que configurem o recurso.

5. *P* deve publicar uma mensagem *inform* em seu tópico notificando a criação de *C*.

OEDL: ED Simples (Bruno/Miguel)

```
# Parte 2: Criando uma aplicação
if msg['op'] == 'create' and 'props' in msg and
'membership' in msg[‘props’]:

   # . . .
```

```
   # Gerar um novo UUID para o novo recurso. Recurso
filho pedido pelo EC.
   uid = str(uuid.uuid4())
   hrn = msg['props']['hrn']

   # Criar um novo tópico para o novo recurso.
   q = create_topic(uid)
   channel.basic_consume(omfrc_resource_cb, queue=q)
```

Release

1. O recurso pai *P* deve informar ao filho *C* que este será liberado.

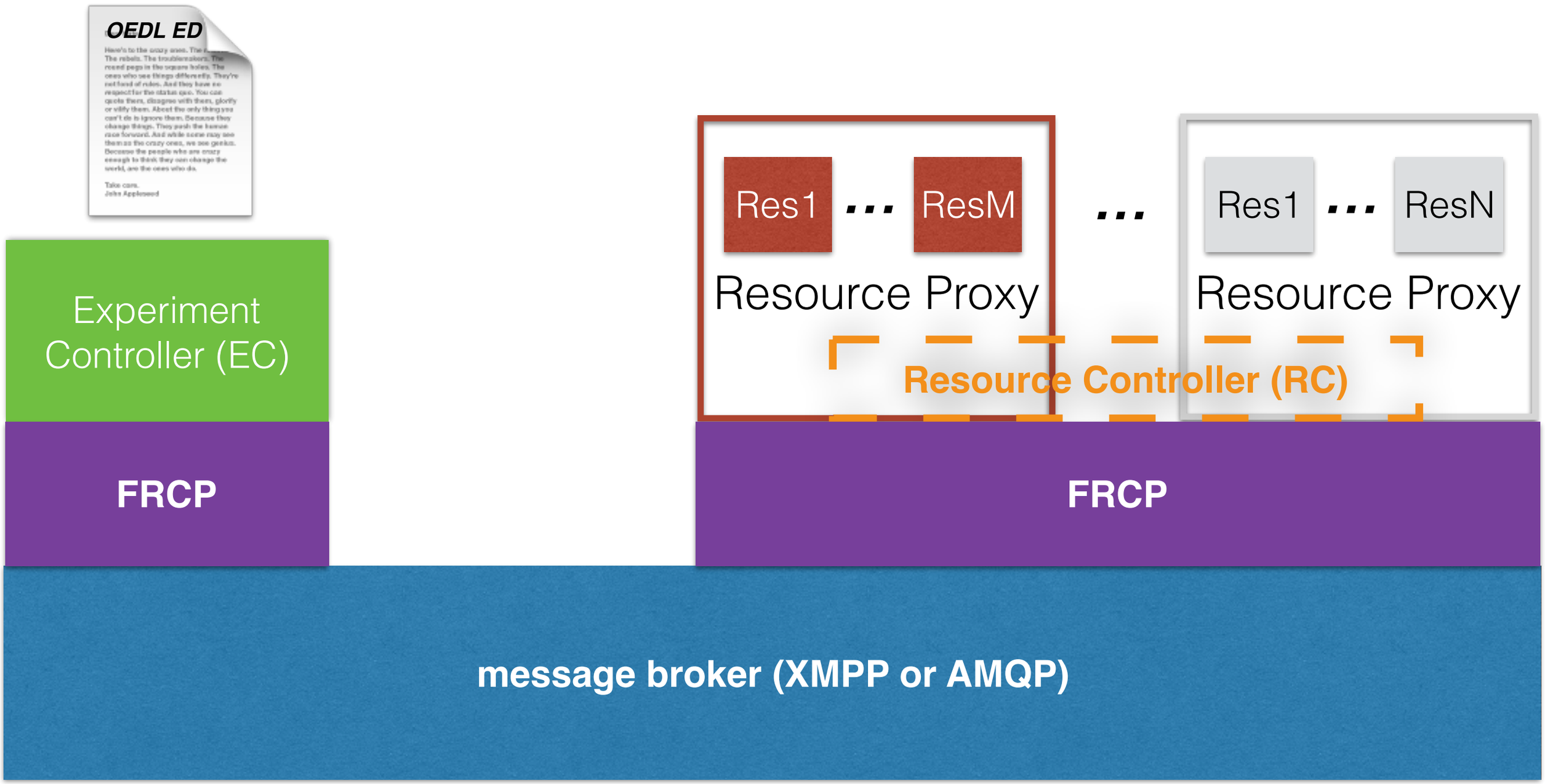
2. Quando *C* terminar, *P* deve remover o topico de *C* do sistema de mensagens.

3. Processo recursivo! Toda a árvore de *C* deve ser eliminada.

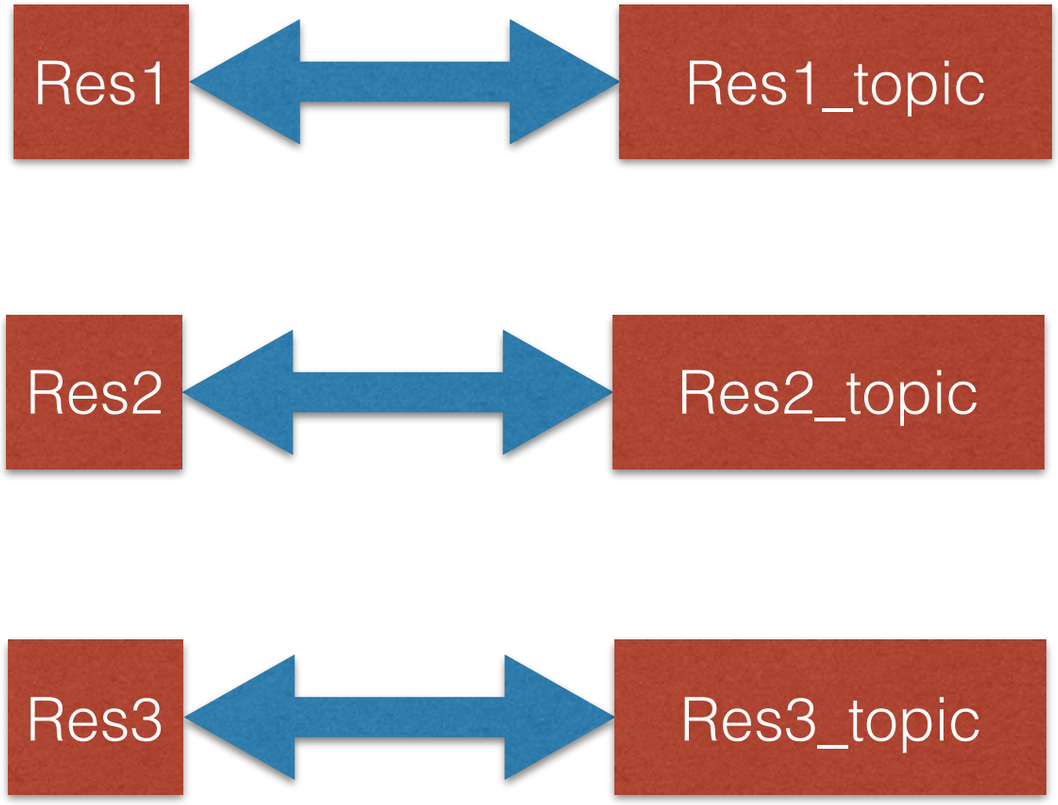
omf_rc

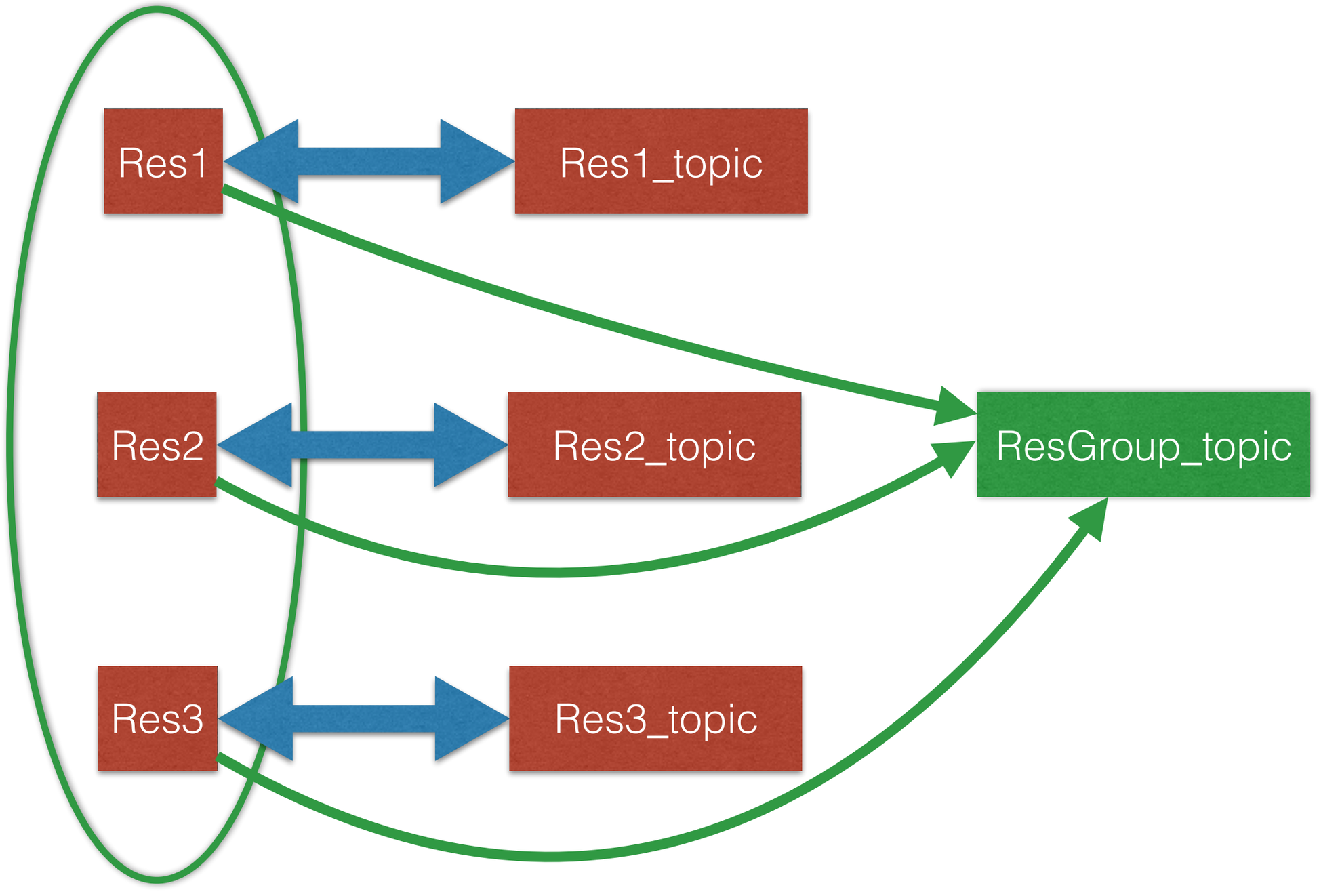
- Biblioteca para definição de *resource proxies*.
- Abstrai a inscrição em tópicos e envio de mensagens dos recursos.
- Possibilita a definição de regras sobre recursos.

OMF: *deeper dive*



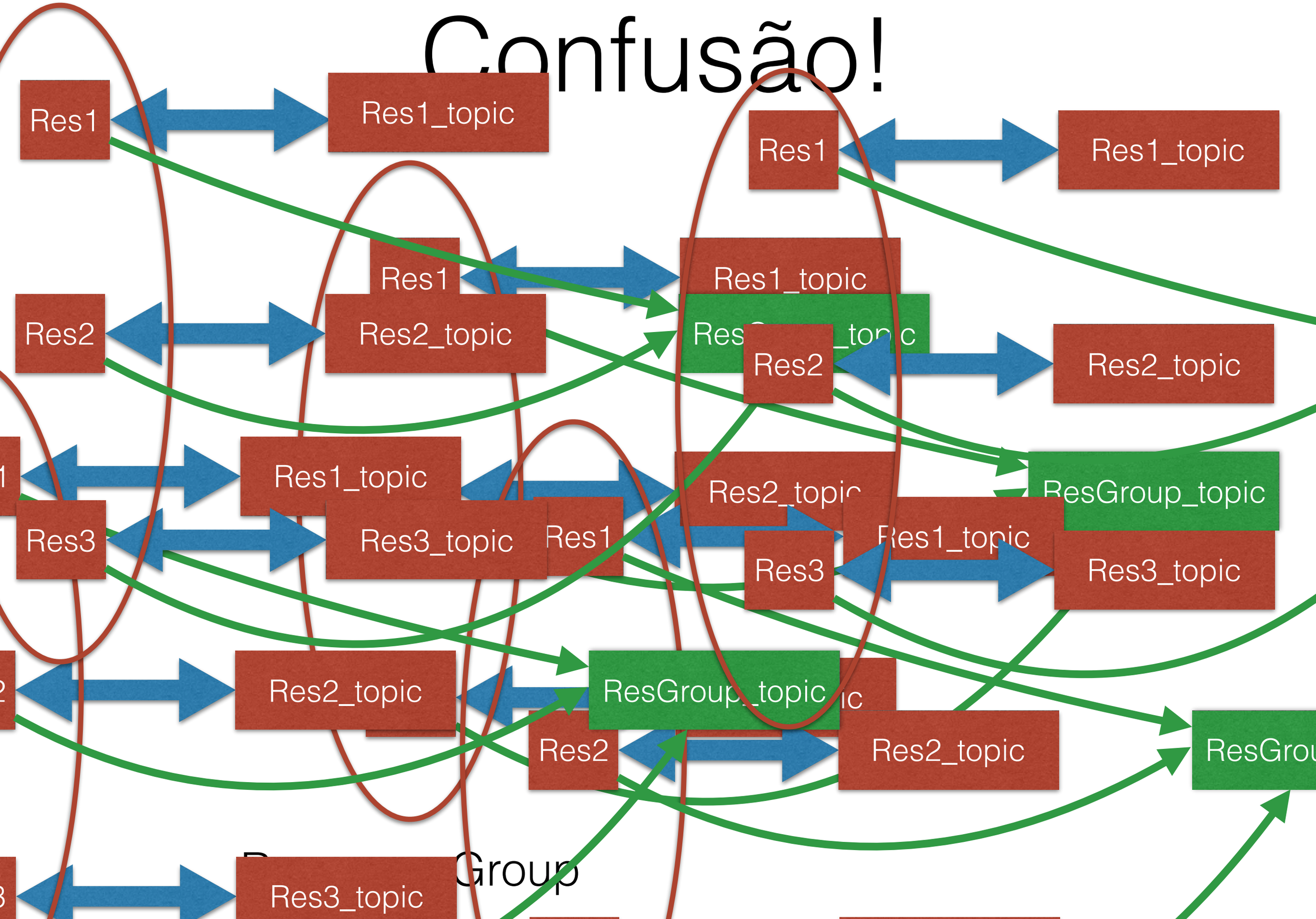
Por que *proxies?*



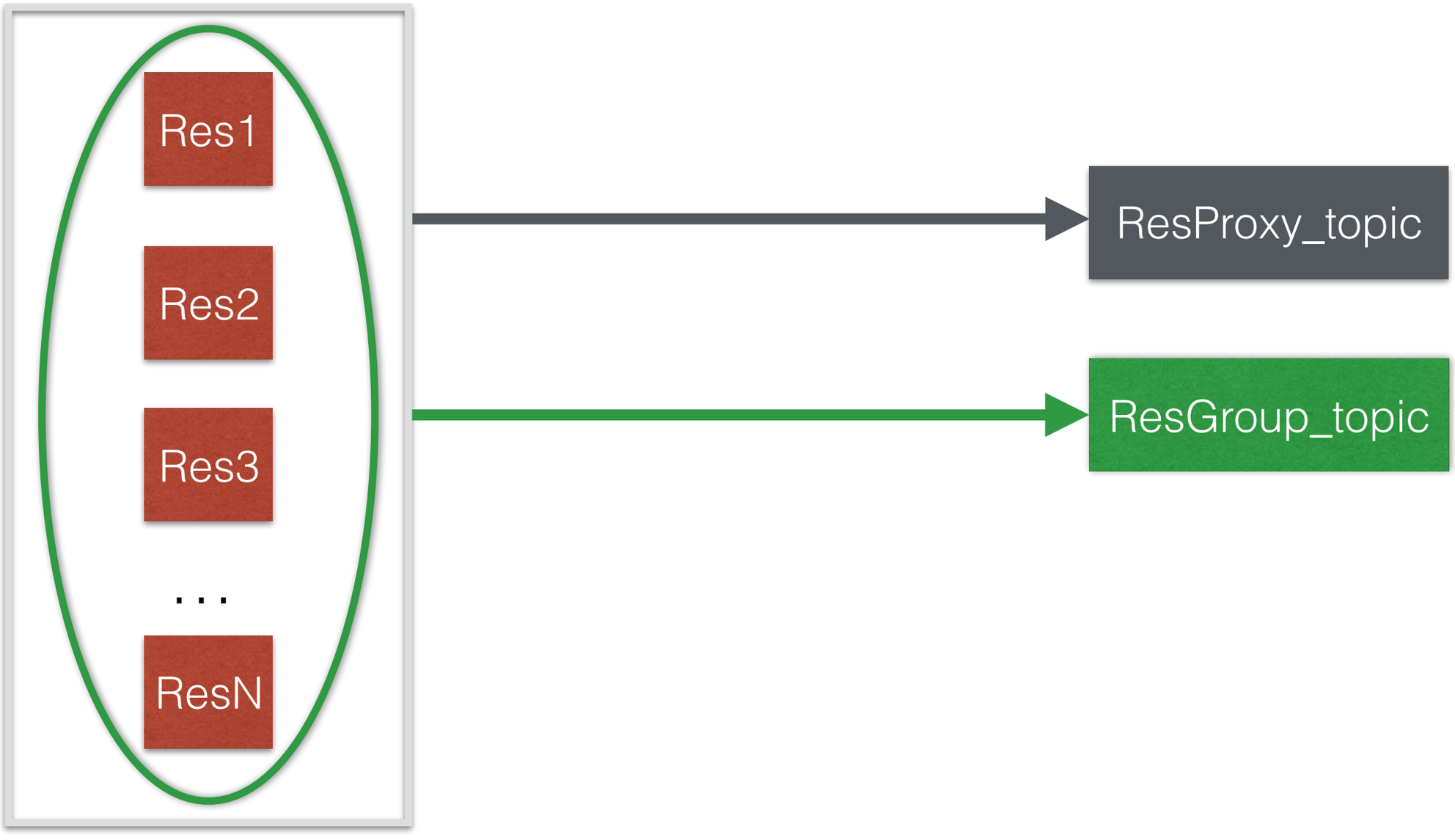


Resource Group

Confusão!


Res1
Res1_topic
Res2
Res2_topic
Res3
Res3_topic
ResGroup_topic

Organizado



Resource Proxy

Hierarquias

Experiment Controller (EC)

FRCP

Res1 ... ResK

Res1 ... ResM

Resource Proxy

...

Res1 ... ResN

Resource Proxy

FRCP

message broker (XMPP or AMQP)

omf_rc: definição

```
require 'omf_rc'

module OmfRc::ResourceProxy::Garage
  include OmfRc::ResourceProxyDSL

  register_proxy :garage
end

module OmfRc::ResourceProxy::Engine
  include OmfRc::ResourceProxyDSL

  register_proxy :engine, :create_by => :garage

  property :manufacturer, :default => "Cosworth"
  property :max_rpm, :default => 12500
  property :rpm, :default => 1000
  property :throttle, :default => 0

  hook :before_ready do |engine|
    OmfCommon.eventloop.every(2) do
      engine.property.rpm += (engine.property.throttle * 5000 - 500).to_i
      engine.property.rpm = 1000 if engine.property.rpm < 1000
    end
  end

  configure :throttle do |engine, value|
    engine.property.throttle = value.to_f / 100.0
  end
end
```

omf_rc: definição

```
OmfCommon.init(:development, communication: { url: 'xmpp://localhost' }) do
  OmfCommon.comm.on_connected do |comm|
    info "Garage controoler >> Connected to XMPP server"
    garage = OmfRc::ResourceFactory.create(:garage, uid: 'garage')
    comm.on_interrupted { garage.disconnect }
  end
end
```

omf_rc: *script* de teste

```
require 'omf_common'

def create_engine(garage)
  garage.create(:engine, hrn: 'my_engine') do |reply_msg|
    if reply_msg.success?
      engine = reply_msg.resource

      engine.on_subscribed do
        info ">>> Connected to newly created engine #{reply_msg[:hrn]}(id:
#{reply_msg[:res_id]})"
        on_engine_created(engine)
      end

      OmfCommon.eventloop.after(10) do
        release_engine(garage, engine)
      end
    else
      error ">>> Resource creation failed - #{reply_msg[:reason]}"
    end
  end
end
```

omf_rc: *script* de teste

```
def on_engine_created(engine)
  info "> Now we will apply 50% throttle to the engine"
  engine.configure(throttle: 50)

  OmfCommon.eventloop.every(2) do
    engine.request([:rpm]) do |reply_msg|
      info "RPM >> #{reply_msg[:rpm]}"
    end
  end

  OmfCommon.eventloop.after(5) do
    info "> We want to reduce the throttle to 0"
    engine.configure(throttle: 0)
  end
end

def release_engine(garage, engine)
  info ">>> Release engine"
  garage.release(engine) do |reply_msg|
    info "Engine #{reply_msg[:res_id]} released"
    OmfCommon.comm.disconnect
  end
end
```

omf_rc: *script* de teste

```
OmfCommon.init(:development, communication: { url: 'xmpp://localhost' })
do
  OmfCommon.comm.on_connected do |comm|
    info "Engine test script >> Connected to XMPP"

    comm.subscribe('garage') do |garage|
      unless garage.error?
        create_engine(garage)
      else
        error garage.inspect
      end
    end

    OmfCommon.eventloop.after(20) { comm.disconnect }
    comm.on_interrupted { comm.disconnect }
  end
end
```

omf_rc: *OEDL* de teste

```
def_property('garage', 'garage', 'Name of garage')

defEvent :engine_created do |state|
  # state is an array holds list of resources,
  # and automatically updated once OMF inform
  # messages received.
  state.find_all do |v|
    v[:type] == 'engine' && !v[:membership].empty?
  end.size > 0
end

defEvent :rpm_reached_4000 do |state|
  state.find_all do |v|
    v[:type] == 'engine' && v[:rpm] && v[:rpm] >= 4000
  end.size >= 1
end

# Define a group and add garages to it.
defGroup('garages', prop.garage)
```

omf_rc: *OEDL* de teste

```
# :ALL_UP is a pre-defined event,
# triggered when all resources set to be part of groups are available
# and configured as members of the associated groups.
onEvent :ALL_UP do
  group('garages') do |g|
    g.create_resource('my_engine', type: 'engine')

    onEvent :engine_created do
      info ">>> Accelerating all engines"
      g.resources[type: 'engine'].throttle = 50

      # We periodically check engine RPM
      every 2.second do
        g.resources[type: 'engine'].rpm
      end
    end

    onEvent :rpm_reached_4000 do
      info ">>> Engine RPM reached 4000"
      info ">>> Reduce engine throttle"
      g.resources[type: 'engine'].throttle = 0
    end

    after 20.seconds do
      info ">>> Release engines"
      g.resources[type: 'engine'].release
      done!
    end
  end
end
```

Cronograma

- **Abril**: Proposta e Implementação do AMQP-Céu
- **Maio**: OMF EC-Céu
- **Junho**: OMF EC-Céu + relatório final
- **Julho**, **Agosto** e **Setembro**: (melhorias pontuais)
- **Outubro**: conclusão de pendências + [OMF_RC-Céu] + relatório
- **Novembro**: relatório + [apresentação de exemplos com OMF_RC-Céu]
- **Dezembro**: relatório e banca