

Controlador de Experimentos OMF em Céu

Apresentação de Projeto Final

Aluno: Carlos Mattoso

Orientadora: Noemi Rodriguez

15/Dezembro/2016

Motivação

Motivação

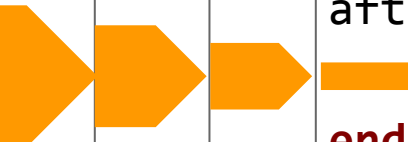
- Facilitar o desenvolvimento de experimentos para *testbeds*.
- Ampliar os casos de uso da linguagem de programação Céu.
- Exposição de API reativas para os artefatos propostos.

Exemplo: experimento em OMF

```
onEvent(:ALL_UP) do |event|  
  after 3 do  
    group("Actor").startApplication("date_LA")  
  
    after 3 do  
      group("Actor").startApplication("ping_google")  
  
      after 3 do  
        Experiment.done  
      end  
    end  
  end  
end  
end
```

Problema: *callback hell*

```
onEvent(:ALL_UP) do |event|  
  after 3 do  
    group("Actor").startApplication("date_LA")  
  
    after 3 do  
      group("Actor").startApplication("ping_google")  
  
      after 3 do  
        Experiment.done  
      end  
    end  
  end  
end
```



The diagram shows four nested orange arrows pointing to the right, representing the sequence of asynchronous operations in the code. The first arrow is the largest and is on the far left. Each subsequent arrow is smaller and positioned further to the right, corresponding to the increasing indentation of the code blocks. The arrows are nested such that the end of one arrow is aligned with the start of the next, visually demonstrating how the code becomes increasingly indented and complex as more asynchronous steps are added, which is the essence of 'callback hell'.

Problema: pensar no tempo global

```
onEvent(:ALL_UP) do |event|
  # 'after' is not blocking. This executes 3 seconds after :ALL_UP fired.
  after 3 do
    info "TEST - do ping app"
    group("Actor").startApplication("date_LA")
  end

  # 'after' is not blocking. This executes 6 seconds after :ALL_UP fired.
  after 6 do
    info "TEST - do date app"
    group("Actor").startApplication("ping_google")
  end

  # 'after' is not blocking. This executes 9 seconds after :ALL_UP fired.
  after 9 do
    Experiment.done
  end
end
```

Problema: pensar no tempo global

```
onEvent(:ALL_UP) do |event|
  # 'after' is not blocking. This executes 3 seconds after :ALL_UP fired.
  after 3 do
    info "TEST - do ping app"
    group("Actor").startApplication("date_LA")
  end

  # 'after' is not blocking. This executes 6 seconds after :ALL_UP fired.
  after 6 do
    info "TEST - do date app"
    group("Actor").startApplication("ping_google")
  end

  # 'after' is not blocking. This executes 9 seconds after :ALL_UP fired.
  after 9 do
    Experiment.done
  end
end
```

Problema: modelo bloqueante

```
onEvent(:ALL_UP) do |event|  
  wait 3  
  info "TEST - do ping app"  
  group("Actor").startApplication("date_LA")
```

```
wait 3  
info "TEST - do da  
group("Actor").sta
```

```
wait 3  
Experiment.done
```

end

⚠️ Calling 'wait' or 'sleep' will block entire EC event loop. Please try 'after' or 'every'

Problema: finalização explícita

```
onEvent(:ALL_UP) do |event|
  # 'after' is not blocking. This executes 3 seconds after :ALL_UP fired.
  after 3 do
    info "TEST - do ping app"
    group("Actor").startApplication("date_LA")
  end

  # 'after' is not blocking. This executes 6 seconds after :ALL_UP fired.
  after 6 do
    info "TEST - do date app"
    group("Actor").startApplication("ping_google")
  end

  # 'after' is not blocking. This executes 9 seconds after :ALL_UP fired.
  after 9 do
    Experiment.done
  end
end
```

Artefatos

Ambiente de Céu

AMQP (*RabbitMQ*)

Biblioteca de Federated Resource Control Protocol

Controlador de Experimentos OMF

Artefatos

Ambiente de Céu

AMQP (*RabbitMQ*)

Biblioteca de Federated Resource Control Protocol

Controlador de Experimentos OMF

Céu

Exemplo: reação a eventos

```
input int KEY;

par/or do
    every 1s do
        _printf("Hello ");
    end
with
    every 1.5s do
        _printf("World!\n");
    end
with
    await KEY;
end
```

Declaração de evento de entrada.

Construção paralela

Exibe "Hello" a cada 1s

com

Exibe "World!" a cada 1.5s

com

Espera pressionamento da chave.

-> Construção morre quando isto ocorre.

fim

Conceito global de tempo

**Premissa de
execução
síncrona**

Programação Reativa & Estruturada

Artefatos

Ambiente de Céu

AMQP (*RabbitMQ*)

Biblioteca de Federated Resource Control Protocol

Controlador de Experimentos OMF

OME

Gerência e controle de experimentos

Arquitetura: Simplificada





Extensão de Ruby para Descrição de
Experimentos

- Aplicações
 - Grupos de Recursos
 - Experimento
-

Exemplo: definição de aplicações

```
defApplication('ping') do |app|
  app.description = 'Simple Definition for the ping application'
  app.binary_path = '/usr/bin/ping'

  app.defProperty('target', 'Address to ping', '')
  app.defProperty('count', 'Number of times to ping', '-c')
end

defApplication('date') do |app|
  app.description = 'Simple Definition for the date application'
  app.binary_path = '/usr/bin/date'

  app.defProperty('date', 'display time described by STRING, not now', '--date')
end
```

Exemplo: definição de grupos

```
defGroup('Actor', 'omf-rc') do |g|
  g.addApplication("ping") do |app|
    app.name = 'ping_google'
    app.setProperty('target', 'google.com')
    app.setProperty('count', 3)
  end

  g.addApplication("date") do |app|
    app.name = 'date_LA'
    app.setProperty('date', 'TZ="America/Los_Angeles" 09:00 next Fri')
  end
end
```

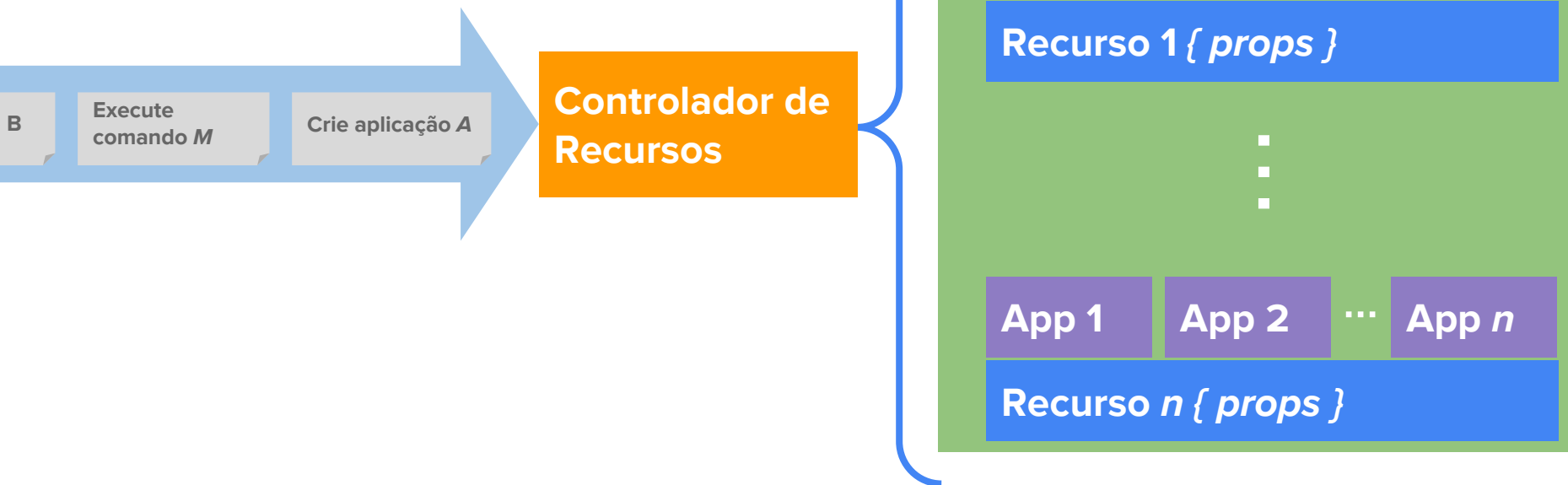
Exemplo: experimento

```
onEvent(:ALL_UP) do |event|
  # 'after' is not blocking. This executes 3 seconds after :ALL_UP fired.
  after 3 do
    info "TEST - do ping app"
    group("Actor").startApplication("date_LA")
  end

  # 'after' is not blocking. This executes 6 seconds after :ALL_UP fired.
  after 6 do
    info "TEST - do date app"
    group("Actor").startApplication("ping_google")
  end

  # 'after' is not blocking. This executes 9 seconds after :ALL_UP fired.
  after 9 do
    Experiment.done
  end
end
```


OEDL: Entidades



omf-ec-céu

Ambiente de OMF em Céu

- Definição de entidades por tabelas Lua.
 - Sem “*nesting*” de reações.
 - Conceito global de tempo provido pelo ambiente.
 - Finalização implícita.
-

Exemplo: definição de aplicações

```
apps = {  
    ping = {  
        description = "Simple Definition for the ping application",  
        binary_path = "/usr/bin/ping",  
        props = {  
            target = {  
                description = "Address to ping",  
                command      = "-a",  
            },  
            count = {  
                description = "Number of times to ping",  
                command      = "-c",  
            },  
        },  
    },  
},  
  
// continua ...
```

Exemplo: definição de aplicações

```
date = {  
    description = "Simple Definition for the date application",  
    binary_path = "/usr/bin/date",  
    props = {  
        date = {  
            description = "Display time described by STRING, not now",  
            command      = "--date",  
        },  
    },  
},  
}
```

Exemplo: definição de aplicações (*OEDL*)

```
defApplication('ping') do |app|  
  app.description = 'Simple Definition for the ping application'  
  app.binary_path = '/usr/bin/ping'  
  
  app.defProperty('target', 'Address to ping', '')  
  app.defProperty('count', 'Number of times to ping', '-c')  
end  
  
defApplication('date') do |app|  
  app.description = 'Simple Definition for the date application'  
  app.binary_path = '/usr/bin/date'  
  
  app.defProperty('date', 'display time described by STRING, not now', '--date')  
end
```

Exemplo: definição de grupos

```
groups["Actor"] = {  
  nodes = { "omf-rc" },  
  apps = {  
    ping_google = {  
      app_id = "ping",  
      target = "www.google.com",  
      count  = 3,  
    },  
    date_LA = {  
      app_id = "date",  
      date   = 'TZ="America/Los_Angeles" 09:00 next Fri',  
    }  
  },  
}
```

Exemplo: definição de grupos (*OEDL*)

```
defGroup('Actor', 'omf-rc') do |g|
  g.addApplication("ping") do |app|
    app.name = 'ping_google'
    app.setProperty('target', 'google.com')
    app.setProperty('count', 3)
  end

  g.addApplication("date") do |app|
    app.name = 'date_LA'
    app.setProperty('date', 'TZ="America/Los_Angeles" 09:00 next Fri')
  end
end
```

Exemplo: experimento

```
#include "omf_start.ceu"
```

```
code/await Experiment(var& Communicator c) -> void do
```

```
    await outer.omf_all_up;
```

```
    await 3s;
```

```
    vector[] byte group_id = [] .. "Actor";
```

```
    // Dispara app de ping
```

```
    vector[] byte ping_app_id  = [] .. "ping_google";
```

```
    await Start_Application(&c, &group_id, &ping_app_id);
```

```
    await 3s;
```

```
    // Dispara app de ping
```

```
    vector[] byte date_app_id  = [] .. "date_LA";
```

```
    await Start_Application(&c, &group_id, &date_app_id);
```

```
    // Experimento encerrado automaticamente para o usuario (não precisa chamar finalização)
```

```
    await 3s;
```

```
end
```


Exemplo: experimento (*OEDL*)

```
onEvent(:ALL_UP) do |event|
  # 'after' is not blocking. This executes 3 seconds after :ALL_UP fired.
  after 3 do
    info "TEST - do ping app"
    group("Actor").startApplication("date_LA")
  end

  # 'after' is not blocking. This executes 6 seconds after :ALL_UP fired.
  after 6 do
    info "TEST - do date app"
    group("Actor").startApplication("ping_google")
  end

  # 'after' is not blocking. This executes 9 seconds after :ALL_UP fired.
  after 9 do
    Experiment.done
  end
end
```

Vantagens: espera não bloqueia a *thread*

```
#include "omf_start.ceu"
```

```
code/await Experiment(var& Communicator c) -> void do
    await outer.omf_all_up;
    await 3s;

    vector[] byte group_id = [] .. "Actor";

    // Dispara app de ping
    vector[] byte ping_app_id = [] .. "ping_google";
    await Start_Application(&c, &group_id, &ping_app_id);
    await 3s;

    // Dispara app de ping
    vector[] byte date_app_id = [] .. "date_LA";
    await Start_Application(&c, &group_id, &date_app_id);

    // Experimento encerrado automaticamente para o usuario (não precisa chamar finalização)
    await 3s;
end
```

Vantagens: sem *nesting*

```
#include "omf_start.ceu"
```

```
code/await Experiment(var& Communicator c) -> void do
```

```
    await outer.omf_all_up;
```

```
    await 3s;
```

```
    vector[] byte group_id = [] .. "Actor";
```

```
    // Dispara app de ping
```

```
    vector[] byte ping_app_id = [] .. "ping_google";
```

```
    await Start_Application(&c, &group_id, &ping_app_id);
```

```
    await 3s;
```

```
    // Dispara app de ping
```

```
    vector[] byte date_app_id = [] .. "date_LA";
```

```
    await Start_Application(&c, &group_id, &date_app_id);
```

```
    // Experimento encerrado automaticamente para o usuario (não precisa chamar finalização)
```

```
    await 3s;
```

```
end
```

Vantagens: finalização implícita

```
#include "omf_start.ceu"
```

```
code/await Experiment(var& Communicator c) -> void do
```

```
    await outer.omf_all_up;
```

```
    await 3s;
```

```
    vector[] byte group_id = [] .. "Actor";
```

```
    // Dispara app de ping
```

```
    vector[] byte ping_app_id = [] .. "ping_google";
```

```
    await Start_Application(&c, &group_id, &ping_app_id);
```

```
    await 3s;
```

```
    // Dispara app de ping
```

```
    vector[] byte date_app_id = [] .. "date_LA";
```

```
    await Start_Application(&c, &group_id, &date_app_id);
```

```
    // Experimento encerrado automaticamente para o pesquisador (não precisa chamar finalização)
```

```
    await 3s;
```

```
end
```

Artefatos

Ambiente de Céu

AMQP (*RabbitMQ*)

Biblioteca de Federated Resource Control Protocol

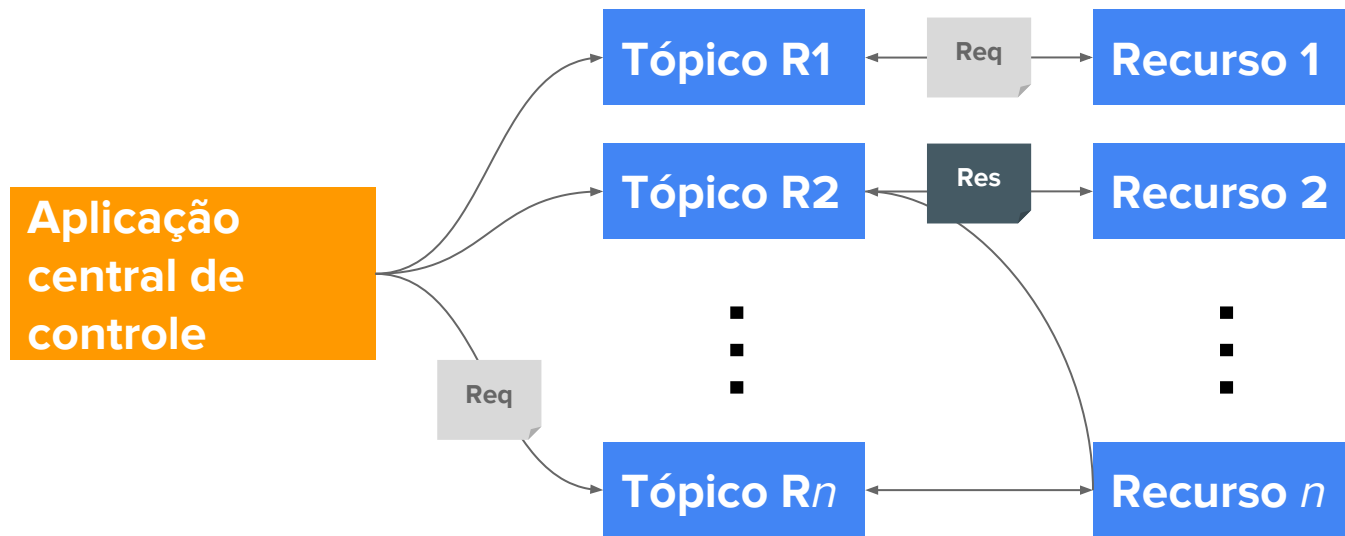
Controlador de Experimentos OMF

ERCP

Federated Resource Control Protocol

Controle e Orquestração de Recursos Distribuídos

Arquitetura

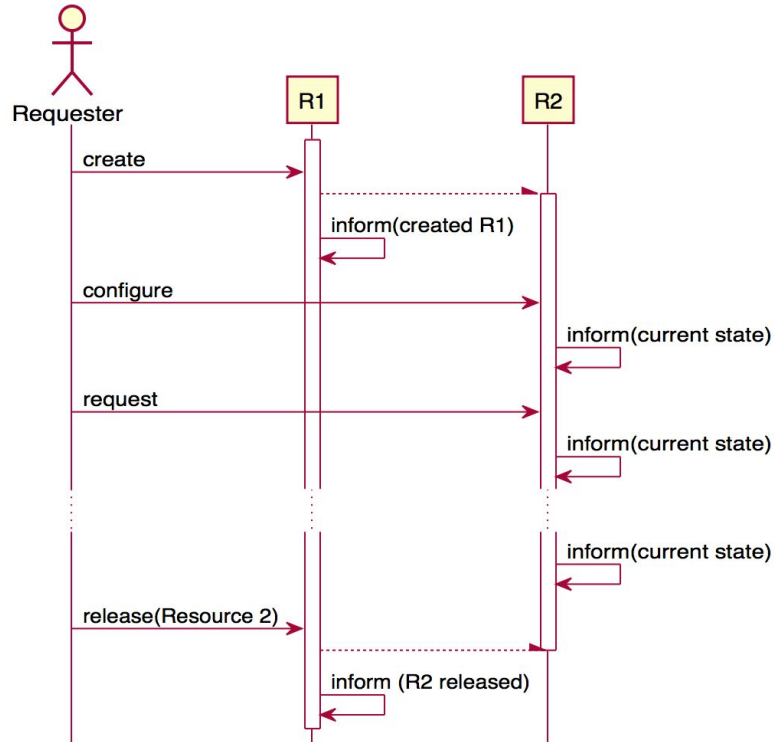


Mensagens

Definem operações dos recursos

- Inform
 - Create
 - Configure
 - Request
 - Release
-

Diagrama de troca de mensagens



Envelope

```
{  
  // Cabeçalho  
  "op" : /* Tipo da mensagem */,  
  "mid": /* ID da mensagem */,  
  "src": /* ID do recurso */,  
  "ts" : /* Unix timestamp */,  
  [ "rp": /* Tópico que recebe cópia da mensagem */ ]  
  // Corpo  
  "props": {  
    // ...  
  }  
}
```

Envelope

```
{  
  // Cabeçalho  
  "op" : /* Tipo da mensagem */,  
  "mid": /* ID da mensagem */,  
  "src": /* ID do recurso */,  
  "ts" : /* Unix timestamp */,  
  [ "rp": /* Tópico que recebe cópia da mensagem */ ]  
  // Corpo  
  "props": {  
    // Dados de interesse da aplicação no recurso.  
  }  
}
```



Controle automático da orquestração

- Expõe ao recurso apenas os dados pertinentes a aplicação.
- Lida com a configuração do ambiente.
- Gerencia respostas e tratamento de erros.

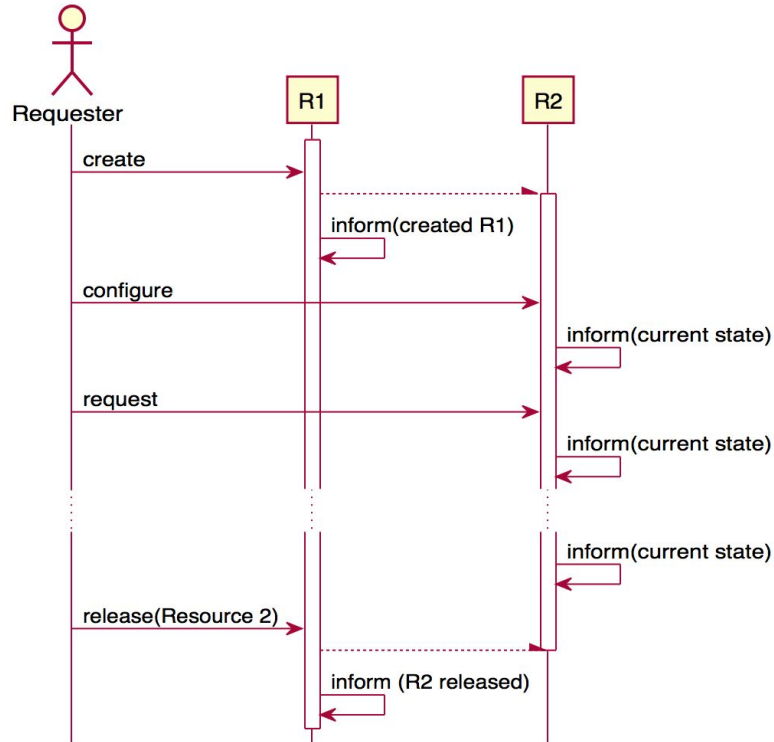
Destaque: Tratador geral de mensagens

```
if /* INFORM */ then
    handler_err = await Handle_Inform( /* ... */ );
else
    if /* CREATE */ then
        spawn Handle_Create(&in_payload.props) -> ( /* ... */ );
    else/if /* CONFIGURE */ then
        if /* membership */ then
            // Inscrição automática do recurso em novo tópico
        else
            spawn Handle_Configure(&in_payload.props) -> ( /* ... */ ); /* ... */
        end
    else/if /* REQUEST */ then
        spawn Handle_Request(&in_payload.props) -> ( /* ... */ ); /* ... */
    else/if /* RELEASE */ then
        spawn Handle_Release(&in_payload.props) -> ( /* ... */ ); /* ... */
    else /* ... */ end
    // Automaticamente define insere mensagem de erro
    if handler_err == FRCP_RET_ERROR /* ... */ end
    // Publica mensagem preenchida em algum trecho acima
end
```

Recurso: Recebe apenas dados

```
if /* INFORM */ then
    handler_err = await Handle_Inform( /* ... */ );
else
    if /* CREATE */ then
        spawn Handle_Create(&in_payload.props) -> ( /* ... */ );
    else/if /* CONFIGURE */ then
        if /* membership */ then
            // Inscrição automática do recurso em novo tópico
        else
            spawn Handle_Configure(&in_payload.props) -> ( /* ... */ ); /* ... */
        end
    else/if /* REQUEST */ then
        spawn Handle_Request(&in_payload.props) -> ( /* ... */ ); /* ... */
    else/if /* RELEASE */ then
        spawn Handle_Release(&in_payload.props) -> ( /* ... */ ); /* ... */
    else /* ... */ end
    // Automaticamente define insere mensagem de erro
    if handler_err == FRCP_RET_ERROR /* ... */ end
    // Publica mensagem preenchida em algum trecho acima
end
```

Diagrama de troca de mensagens



Controlador: Recebe *payload* e tipo da resposta

```
if /* INFORM */ then
    handler_err = await Handle_Inform(/* ... */, &in_payload, &request_type);
else
    if /* CREATE */ then
        spawn Handle_Create(&in_payload.props) -> ( /* ... */ );
    else/if /* CONFIGURE */ then
        if /* membership */ then
            // Inscrição automática do recurso em novo tópico
        else
            spawn Handle_Configure(&in_payload.props) -> ( /* ... */ ); /* ... */
        end
    else/if /* REQUEST */ then
        spawn Handle_Request(&in_payload.props) -> ( /* ... */ ); /* ... */
    else/if /* RELEASE */ then
        spawn Handle_Release(&in_payload.props) -> ( /* ... */ ); /* ... */
    else /* ... */ end
    // Automaticamente define insere mensagem de erro
    if handler_err == FRCP_RET_ERROR /* ... */ end
    // Publica mensagem preenchida em algum trecho acima
end
```

Artefatos

Ambiente de Céu

AMQP (*RabbitMQ*)

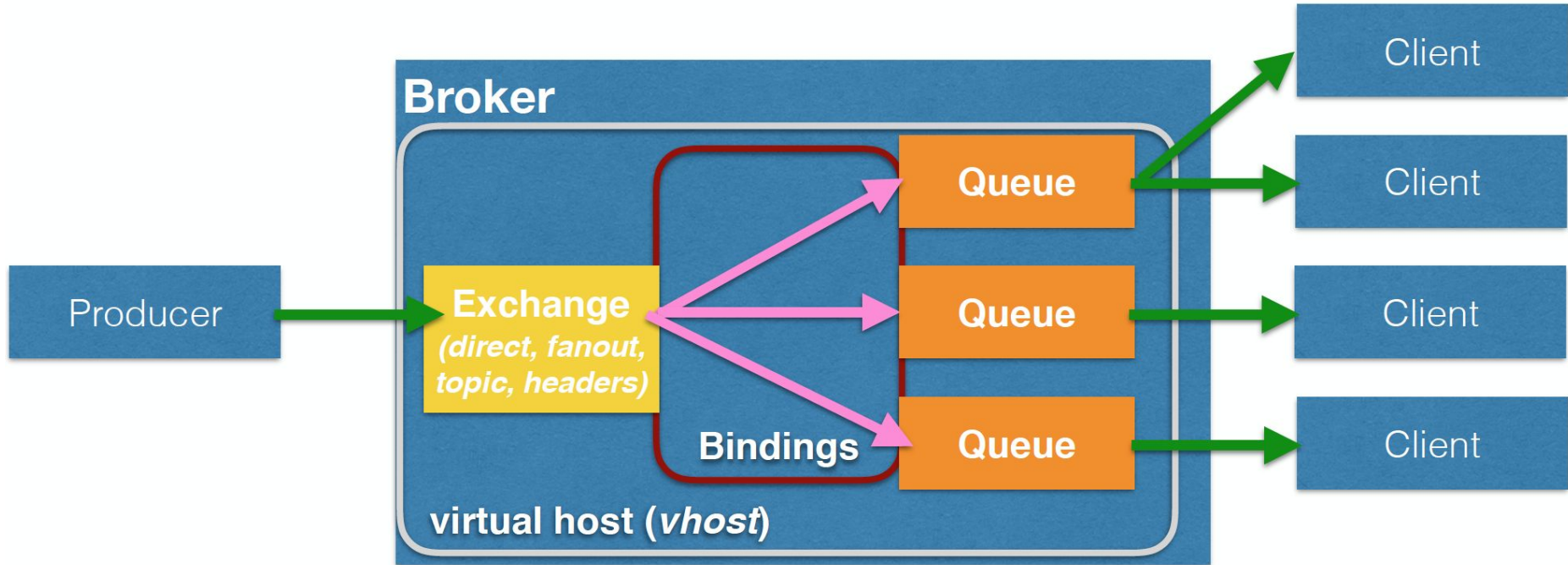
Biblioteca de Federated Resource Control Protocol

Controlador de Experimentos OMF

AMQP

Advanced Message Queuing Protocol

Arquitetura



céu-rabbitmq

Implementação de AMQP focada em
RabbitMQ

- Métodos de AMQP que não fazem *syscalls* bloqueantes.
 - Padrão próprio para tratador de mensagens recebidas.
 - Finalização de entidades e *acknowledgement* de mensagens feitos automaticamente.
-

Desafio: Métodos bloqueantes

- Baseada em biblioteca C de AMQP.
- Muitas chamadas desta API são **bloqueantes** (i.e. fazem chamadas a syscalls bloqueantes), o que impede seu uso direto em Céu.
- **Solução:** Encapsula as chamadas **bloqueantes** em *threads* invisíveis ao programador.

Desafio: Métodos bloqueantes

```
code/await Res( /* ... */ ) -> (event& void ok) -> FOREVER do
  event void ok_;
  ok = &ok_;

  await async/thread( /* ... */ ) do
    _aqmp_blocking_method( /*...*/ );
  end
  emit ok_;

  // codigo de finalizacao...
  await FOREVER;
end

event& void ok;
spawn Res( /* ... */ ) -> (&ok)
await ok;
```

Desafio: Invocação de métodos durante consumo

- A biblioteca C não permite a chamada de métodos enquanto o consumo de mensagens estiver ativo.
- Se não tratado, o programador Céu não poderia realizar interações complexas de criação dinâmica de entidades conforme mensagens fossem recebidas.
- **Solução:** Usar eventos para ativação e suspensão do consumo.

Desafio: Invocação de métodos durante consumo

```
code/await AMQP_Call (var& Channel ch, /* ... */) -> void do
  // ...
  emit ch.pause_consuming;
  await async/thread( /* ... */ ) do
    _amqp_release_call( /* ... */ );
  end
  emit ch.resume_consuming;
  // ...
end
```

Desafio: Tratamento de mensagens

- Céu não tem *callbacks*. Como tratar mensagens recebidas de uma fila?
- **Solução:**
 - No ato de inscrição em uma fila define-se identificador do respectivo tratador.
 - Ambiente de AMQP invoca abstração que deve ser implementada pelo programador.
 - Na invocação, transmite a mensagem e identificador do tratador.

Desafio: Tratamento de mensagens

```
code/await Handler (var& Envelope env, /* ... */) -> void do
    // Implementado pelo usuario...
end
```

```
code/await LowHandler (var _amqp_message_t msg, /* ... */) -> void do
    var int handler_id = /* obtem ID do tratador para `msg` */ ;
    await Handler (Envelope(handler_id, msg));
end
```

```
code/await Consume(var& Channel ch, /* ... */) -> (event& void ok) -> FOREVER do
    // ...
    loop do
        // recebe novas mensagens...
        spawn LowHandler(msg, /* ... */) in handler_pool;
    end
    // ...
end
```

Céu: abstrações de código

```
input Msg RECV;
```

Declaração de evento de entrada.

```
data Msg with  
    // atributos...  
end
```

Estrutura de dados para envelope de mensagem.

```
code/await Msg_Handler (var Msg msg) ->  
void do  
    // trabalho nao bloqueante...  
end
```

Abstração de código (similar a uma função)
-> Pode esperar por eventos: cede controle da exec. a escopo invocante.

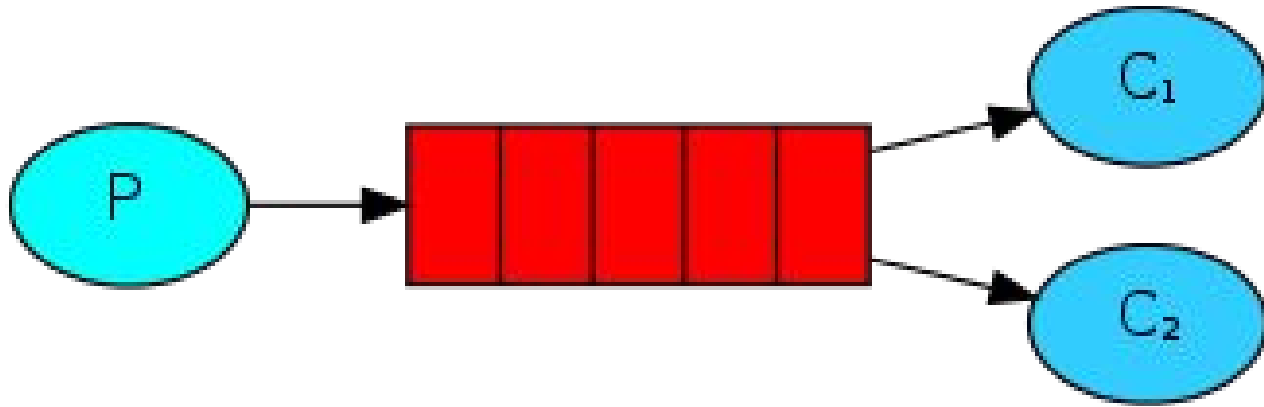
```
pool[] Msg_Handler handlers;  
var Msg msg;
```

Estrutura para armazenamento de invocações de *code/awaits*

```
every msg in RECV do  
    spawn Msg_Handler(msg) in handlers;  
end
```

Despacho de tratador para cada mensagem recebida!

Exemplo: Filas de Trabalho



Exemplo: Produtor de Tarefa

```
conn = Bunny.new(:automatically_recover =>
false)
conn.start

ch  = conn.create_channel
q   = ch.queue("task_queue", :durable =>
true)

msg = ARGV.empty? ? "Hello World!" :
ARGV.join(" ")

q.publish(msg, :persistent => true)
puts " [x] Sent #{msg}"

conn.close
```

```
var& Connection conn;
event& void conn_ok;
watching New_Connection(_) -> (&conn, &conn_ok) do
  await conn_ok;

  var& Channel channel;
  event& void ch_ok;
  spawn New_Channel(&conn) -> (&channel, &ch_ok);
  await ch_ok;

  // Generate task msg
  var      int  task_id = /* gen task id */;
  vector[50] byte task_msg = [] .. "";
  _sprintf((&task_msg[0] as _char&&), "%d", task_id);

  spawn Publish(&channel, &amq_default, PublishContext(/* ... */));
  _printf("Posted a new task: %s\n", (&task_msg[0] as _char&&));
end
```

Exemplo: Worker

```
# setup
ch.prefetch(1)
puts " [*] Waiting for messages. To exit
press CTRL+C"

begin
  q.subscribe(:manual_ack => true, :block =>
true) do |delivery_info, properties, body|
    puts " [x] Received '#{body}'"
    # simula trabalho...
    sleep body.count(".").to_i
    puts " [x] Done"
    ch.ack(delivery_info.delivery_tag)
  end
rescue Interrupt => _
  conn.close
end
```

```
watching New_Connection(_) -> (&conn, &conn_ok) do
  // Setup...

  // Definação de prefetch
  event& void qos_ok;
  spawn Qos(&channel, QosContext(1,_)) -> (&qos_ok);
  await qos_ok;

  // Ativa consumo
  spawn Channel_Consume(&channel, &default_handlers);
  _printf("Consuming messages from queue `task_queue`...\n\n");
  await FOREVER;
End

code/await Handler (var& Channel channel, var Envelope env) -> void do
  var int proc_time = /* obtem tempo de processamento */;

  _printf("\n[%lu] Processing message for %d seconds.\n",
    env.contents.delivery_tag, proc_time);
  await (proc_time)s;
  _printf("[%lu] Processed message for %d seconds!\n\n",
    env.contents.delivery_tag, proc_time);
end
```

Exemplo: Worker

```
# setup
ch.prefetch(1)
puts " [*] Waiting for messages. To exit
press CTRL+C"

begin
  q.subscribe(:manual_ack => true, :block =>
true) do |delivery_info, properties, body|
    puts " [x] Received '#{body}'"
    # simula trabalho...
    sleep body.count(".").to_i
    puts " [x] Done"
    ch.ack(delivery_info.delivery_tag)
  end
rescue Interrupt => _
  conn.close
end
```

```
watching New_Connection(_) -> (&conn, &conn_ok) do
  // Setup...

  // Definação de prefetch
  event& void qos_ok;
  spawn Qos(&channel, QosContext(1,_)) -> (&qos_ok);
  await qos_ok;

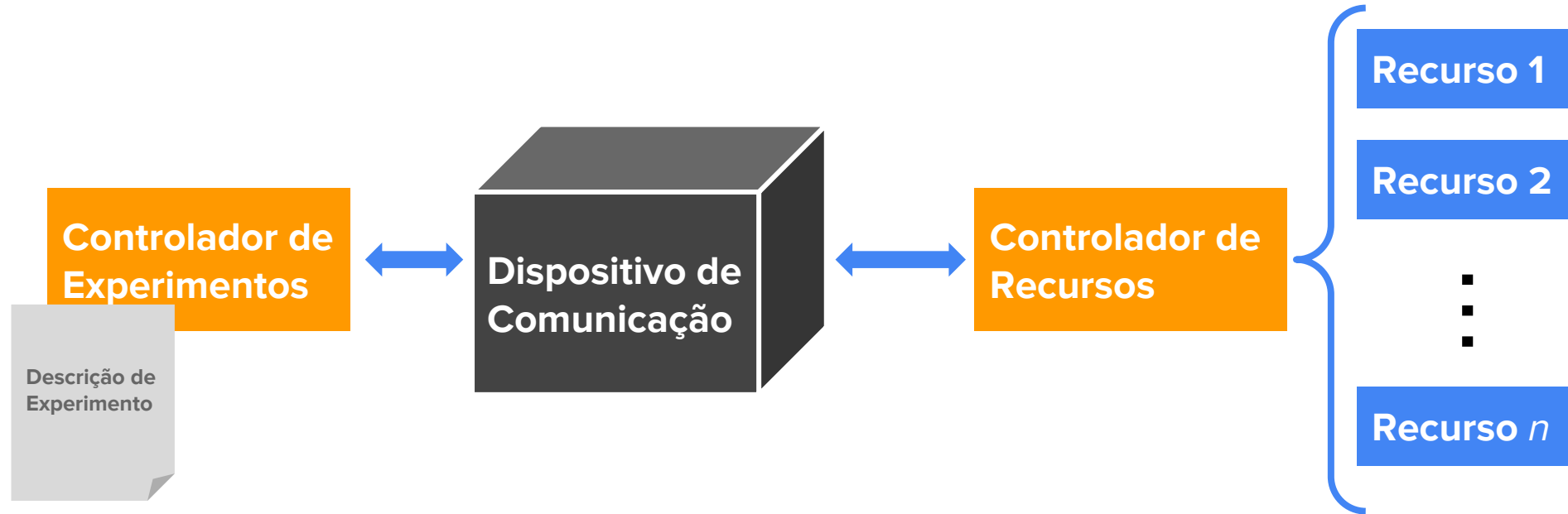
  // Ativa consumo
  spawn Channel_Consume(&channel, &default_handlers);
  _printf("Consuming messages from queue `task_queue`...\n\n");
  await FOREVER;
End

code/await Handler (var& Channel channel, var Envelope env) -> void do
  var int proc_time = /* obtem tempo de processamento */;

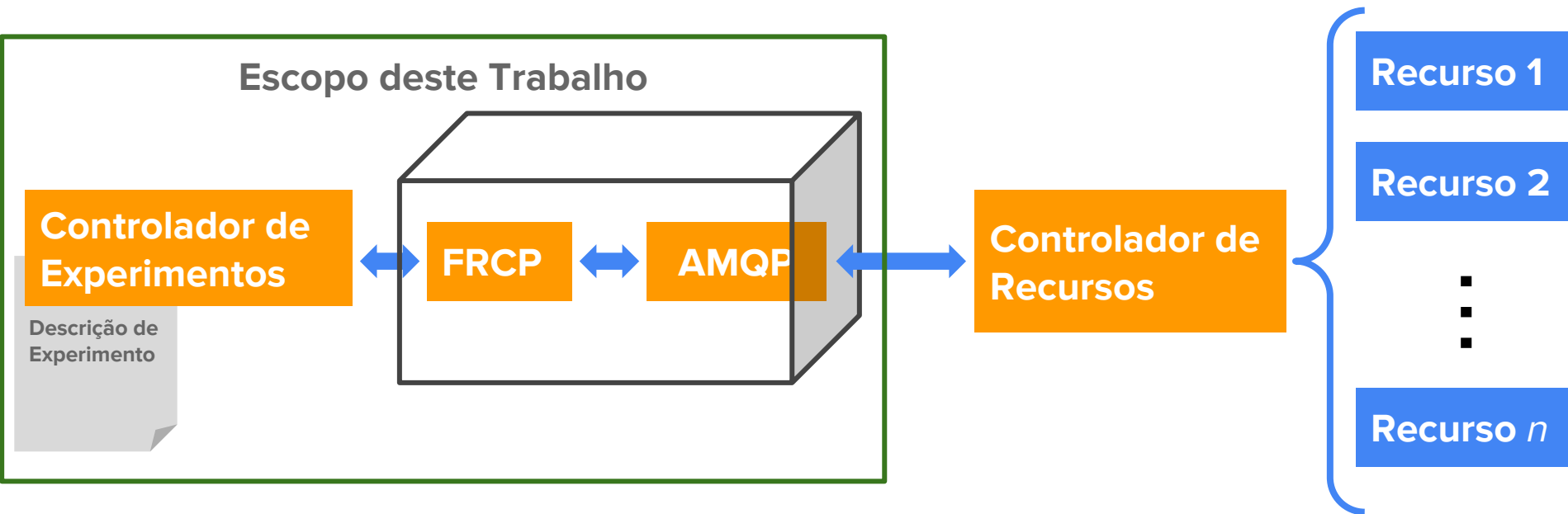
  _printf("\n[%lu] Processing message for %d seconds.\n",
    env.contents.delivery_tag, proc_time);
  await (proc_time)s;
  _printf("[%lu] Processed message for %d seconds!\n\n",
    env.contents.delivery_tag, proc_time);
end
```


Visão geral

Arquitetura: Simplificada



Arquitetura: Simplificada



Trabalhos futuros

Trabalhos futuros

- Testes e verificação de erros mais robustos.
- Expandir o controlador de experimentos para suportar métricas e interfaces de rede em recursos.
- Integração com o *testbed* CéuNaTerra.

Conclusão

Perguntas?

Interessados?

- API Céu de AMQP: <https://github.com/calmatoso/ceu-rabbitmq>
- Controlador de Experimentos OMF: <https://github.com/calmatoso/ceu-omf-ec>

Obrigado!