



Departamento de
Informática e Estatística
CTC • UFSC



Mensagens de Interesse e Resposta

INE5424 - Sistemas Operacionais II

Grupo A(Manhã): Vitor Calegari, Matheus Bigolin, Pedro Fountoura, Pedro Tagliajinha

Objetivos da entrega 3

Requisito da entrega:

- Implementar um modelo de interação baseado em mensagens de Interesse e Resposta, similar ao publish-subscribe, mas sem publicações explícitas.
 - Agentes enviam mensagens de Interesse especificando o tipo de dado desejado e um período para recebimento.
 - Agentes capazes de produzir o dado de Interesse respondem periodicamente com mensagens de Resposta.
- Cada agente pertinente deve gerenciar o intervalo de envio de respostas conforme especificado na mensagem de Interesse.

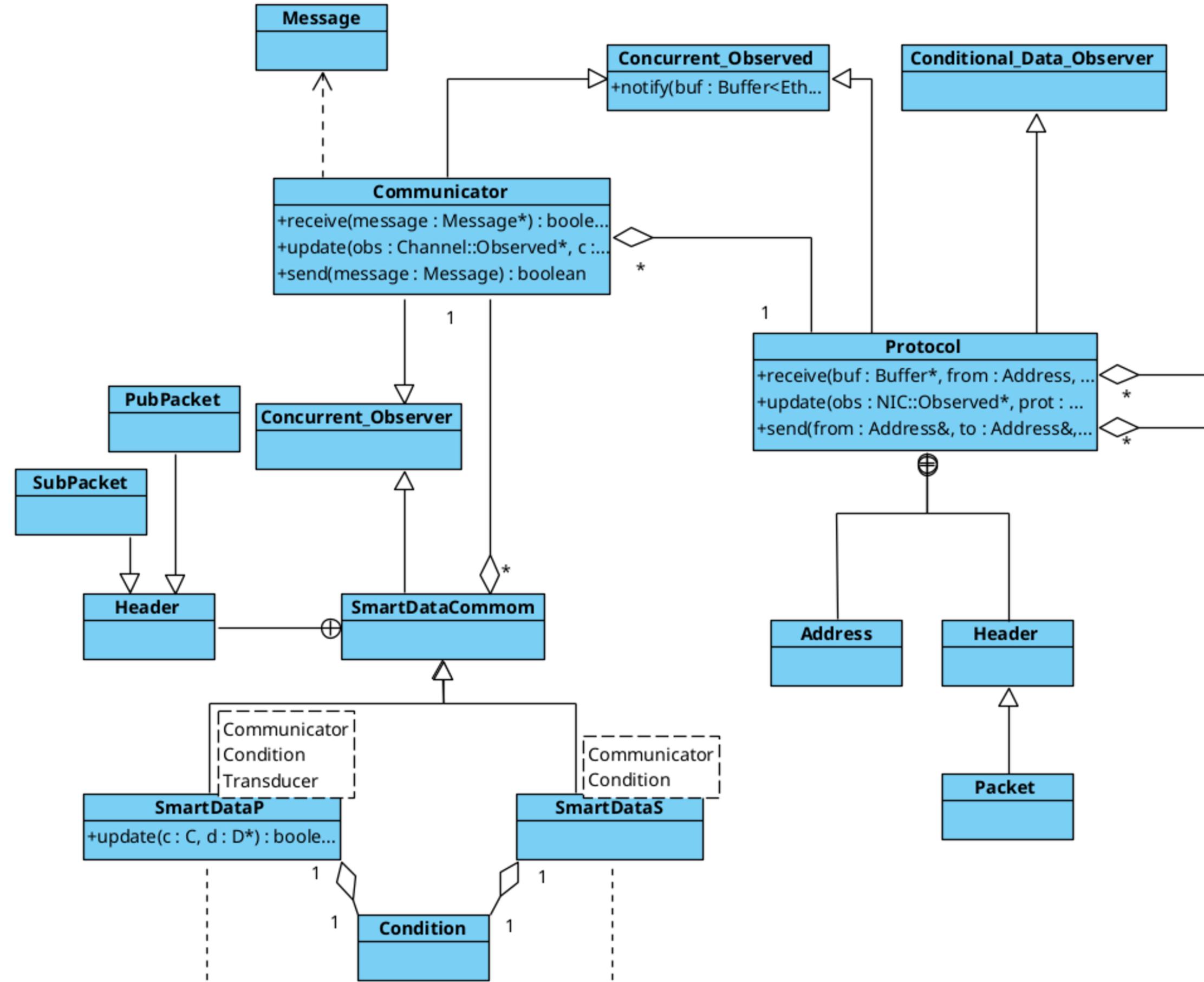
Mudanças entre o T2

Funcionamento sem Publish/Subscriber

- Funcionamento da T1/T2 foi mantido igual: Usuário pode continuar se comunicando sem Pub/Sub utilizando apenas o Communicator.

Adaptação da Message para Tipificar Interações de Interesse/Resposta

- Para diferenciar mensagens comuns das novas mensagens de interesse e resposta foi introduzido o campo `_msg_type` na classe Message:
 - `enum class Type : uint8_t { COMMON, PUBLISH (Resposta), SUBSCRIBE (Interesse) };`
- O `Type::COMMON` permite que a comunicação direta via Communicator continue funcional
- Tipos `SUBSCRIBE` e `PUBLISH` são utilizados por SmartData para enviar um interesse ou resposta.
- Payload da Message:
 - Para `Type::SUBSCRIBE`: Contém a SmartUnit (tipo) e o period.
 - Para `Type::PUBLISH`: Contém a SmartUnit (tipo), period e o value.



No código ambos SmartData tem o nome "SmartData", aqui tivemos que diferenciar seus nomes por limitações da ferramenta utilizada para fazer os diagramas

SmartData e SmartUnit

SmartUnit (Facilitador para o campo Type):

- **Propósito:** Serve como uma classe auxiliar para construir e interpretar de forma padronizada o campo *type* (identificador TEDS - IEEE 1451) das mensagens de Interesse e Resposta.
- Utiliza um campo de 32 bits para uma representação interna das unidades SI, tipos digitais e formatos numéricos. Oferece uma interface para que o SmartData possa facilmente serializar/desserializar esta informação no payload da Message.

SmartData:

- **Propósito:** Organiza a lógica de envio de mensagens de Interesse e o envio periódico de Respostas, abstraindo essa complexidade da aplicação final.
- Constrói mensagens de Interesse e envia via Communicator. O campo *type* é formatado usando SmartUnit.

Operação:

- Observa o Communicator para mensagens de Interesse destinadas às unidades que pode fornecer (via Transducer).
- Gerencia uma lista de inscritos e seus períodos.
- Utiliza um Transducer para obter o value.
- Constrói e envia mensagens de publicação periodicamente.

Lógica de operação do SmartData

Envio de Interesse (subscribe):

- O SmartData configurado para solicitar dados instancia uma Message com Type::SUBSCRIBE.
- O data() da Message é preenchido com a SmartUnit (representando o type) e o period.
- A Message é enviada via _communicator->send(...). Todas as mensagens são broadcast.
- A subscrição acontece periodicamente.

Processamento de Interesse Recebido:

- O SmartData produtor recebe apenas mensagens, via seu Communicator, do tipo SUBSCRIBE que requisitam a unidade produzida por seu transdutor.
- Se SmartData produtor receber a mensagem:
 - Armazena o origin da Message e o period em uma lista interna de solicitantes.
 - Ajusta o período global de envio de respostas

Lógica de operação do SmartData

Envio de Resposta (publish_loop):

- Uma thread interna no SmartData (produtor) é ativada periodicamente.
- Obtém o value do _transd.
- Constrói uma Message com Type::PUBLISH.
- O data() da Message é preenchido com a SmartUnit, periodo de envio atual e o value.
- A Message é enviada via _communicator->send(...) (broadcast).

Processamento de Resposta Recebida:

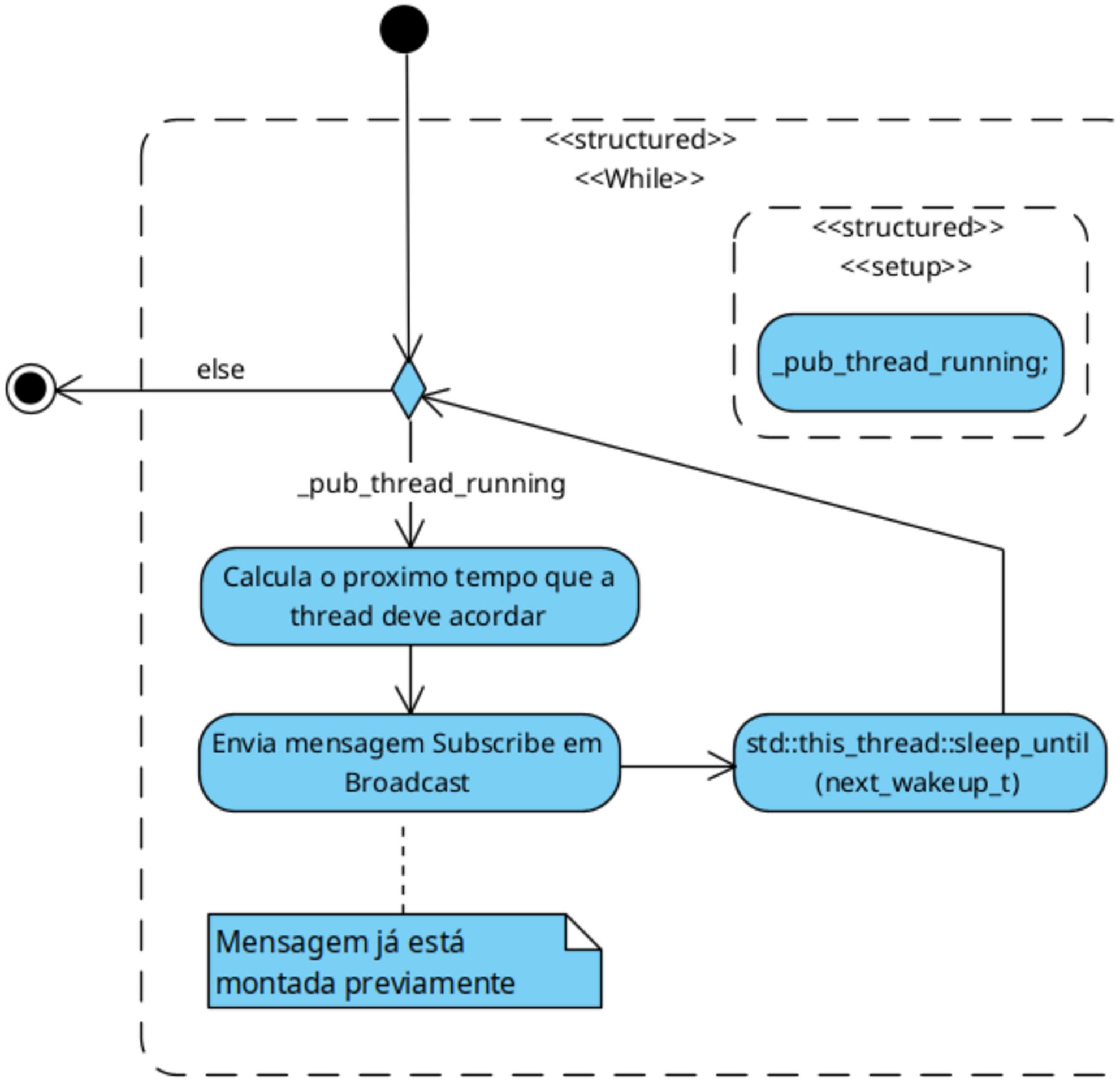
- O SmartData solicitante recebe apenas mensagens, via seu Communicator, do tipo PUBLISH da unidade e periodo requisitados.
- Se a SmartData solicitante receber a mensagem, ela é disponibilizada para ser obtida pela aplicação por SmartData::receive(...).

Fluxo de Subscrição (Agente Solicitante Agentes Produtores)

Para se inscrever em determinada unidade, basta instanciar um SmartData<Communicator, Condition>:

- **Condition é formada por Unit, Period**
- **O SmartData (Solicitante) iniciará a inscrição periódica na rede com Unit e Period da Condition.**
- **Criação da Mensagem de Interesse:**
 - Message msg_interesse(sizeof_payload_interesse, Message::Type::SUBSCRIBE);
 - O sourceAddr() da msg_interesse é o endereço do _communicator do Solicitante.
 - O destAddr() é configurado para broadcast (e.g., Channel::BROADCAST_SID, Channel::BROADCAST_PORT) para alcançar todos os agentes.
 - Payload (msg_interesse.data()):
 - Serializa Unit.get_int_unit() (o type TEDS). Unit é transmitido como uint32_t
 - Serializa Period.
- **Envio: _communicator->send(&msg_interesse);**
 - A mensagem é propagada para todos os agentes.

act [SmartData re-Subscribe Thread]



Fluxo de Subscrição (Agente Solicitante Agentes Produtores)

Receptor (Potenciais Agentes Produtores via SmartData::update(...) - Observador do Communicator):

- Cada SmartData (Produtor) recebe a msg_interesse através do update(...) chamado por seu _communicator.
- **Filtragem e Processamento da Mensagem de Interesse:**
 - Verifica message_recebida.getType() == Message::Type::SUBSCRIBE.
 - Dessaializa received_type e received_period da mensagem recebida.
 - Compara received_type com a unidade que este Produtor oferece.
- **Registro do Solicitante (Se type compatível):**
 - Adiciona {origin = message_recebida.sourceAddr(), period = received_period} à lista subscribers.
 - Recalcula período de publicação do Produtor, MDC de todos os received_period dos subscribers.
- **Ativação/Ajuste da Thread de Publicação:**
 - Se esta é a primeira subscrição a thread pub_thread é liberada(por has_first_subscriber_sem.release()) e é estabelecido o período de envio.
 - Se o período de envio deve ser modificado, seu período é ajustado.

Fluxo de Publicação (Agente Produtor Agentes Solicitantes)

Para começar a publicar determinada unidade, basta instanciar um SmartData<Communicator, Condition, Transducer>:

Thread de Publicação do Produtor (SmartData::pub_thread loop):

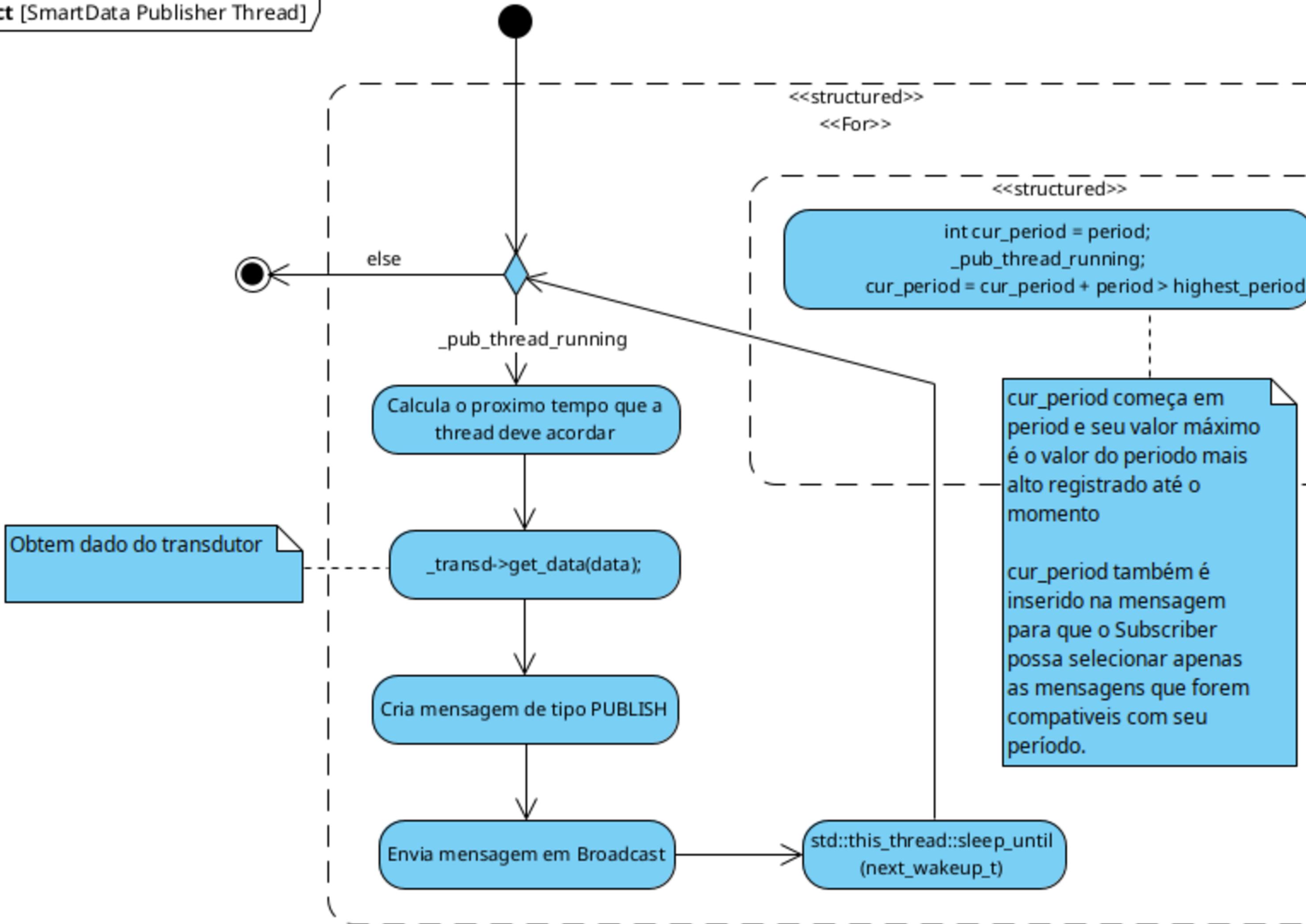
1. Loop executado enquanto _pub_thread_running e existem subscribers.
2. period_sem.acquire(); (para acesso seguro a this->period).
3. Calcula next_wakeup_t baseado em std::chrono::steady_clock::now() + std::chrono::microseconds(this->period).
4. period_sem.release();
5. **Obtenção do Dado:** _transd->get_data(data); data é preenchido por get_data de acordo com o tamanho do tipo declarado na unidade.

6. Criação da Mensagem de Publicação:

- Message msg_resposta(sizeof_payload_resposta, Message::Type::PUBLISH);
- sourceAddr() é o endereço do _communicator do Produtor.
- destAddr() é configurado para broadcast.
- Payload (msg_resposta.data()):
 - Serializa _transd->get_unit().get_int_unit() (o type TEDS do dado).
 - Dados gravados em data.

7. Envio: _communicator->send(&msg_publish); 8.std::this_thread::sleep_until(next_wakeup_t);

act [SmartData Publisher Thread]

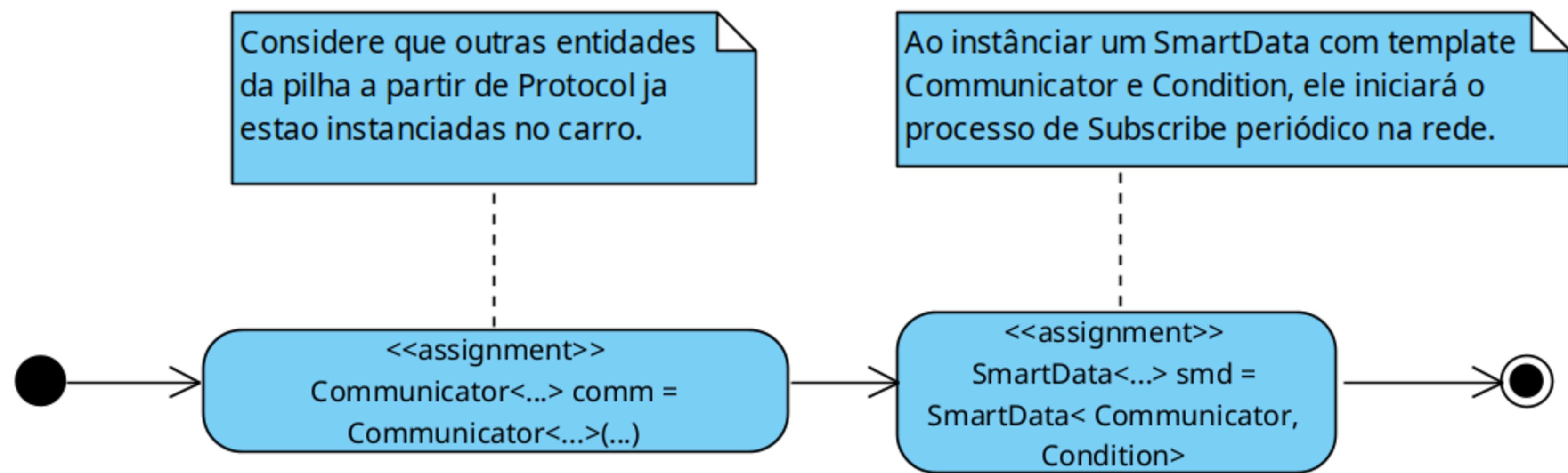


Fluxo de Publicação (Agente Produtor Agentes Solicitantes)

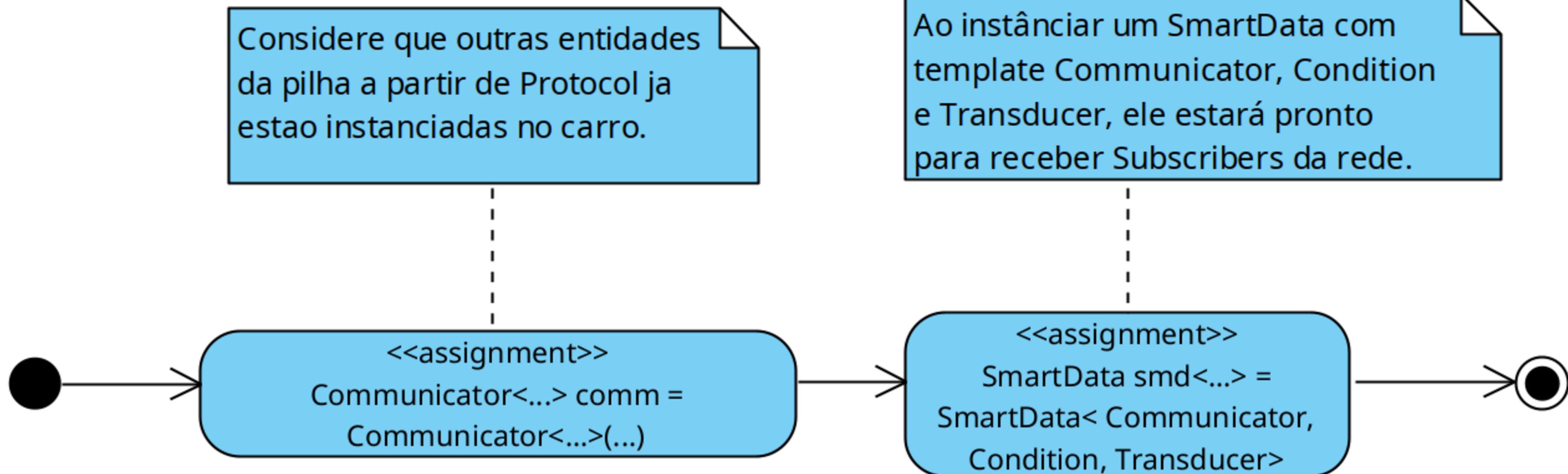
Receptor (Agentes Solicitantes via SmartData::update(...) - Observador do Communicator):

- Cada SmartData (Solicitante) recebe a msg_resposta através do update(...).
- **Filtragem e Processamento da Mensagem de Resposta:**
 - Verifica msg_resposta.getType() == Message::Type::PUBLISH.
 - Dessaializa unit e period do payload da mensagem de resposta.
 - Verifica se unit corresponde a unidade para a qual este Solicitante tem interesse (comparando com sua Unit declarada na Condition).
 - Verifica se o period da mensagem de resposta é compatível com seu período (comparando com seu period declarado na Condition)
 - Caso, não sejam compatível, libera buffer.
- **Disponibilização do Dado (Se type e period compatível):**
 - O Observer::update(c, buf) é chamado, o que libera o semáforo _semaphore da classe Concurrent_Observer e enfileira o Buffer* buf (que contém a Message) para que a aplicação a desenfileire.
 - A aplicação do Solicitante, que chamou smartData.receive(&msg_para_preencher), estava bloqueada em _semaphore.acquire(). Agora ela é desbloqueada, obtém o Buffer da fila, e a **msg_para_preencher** é preenchida com os dados do Buffer recebido.

act [User View Subscribe]



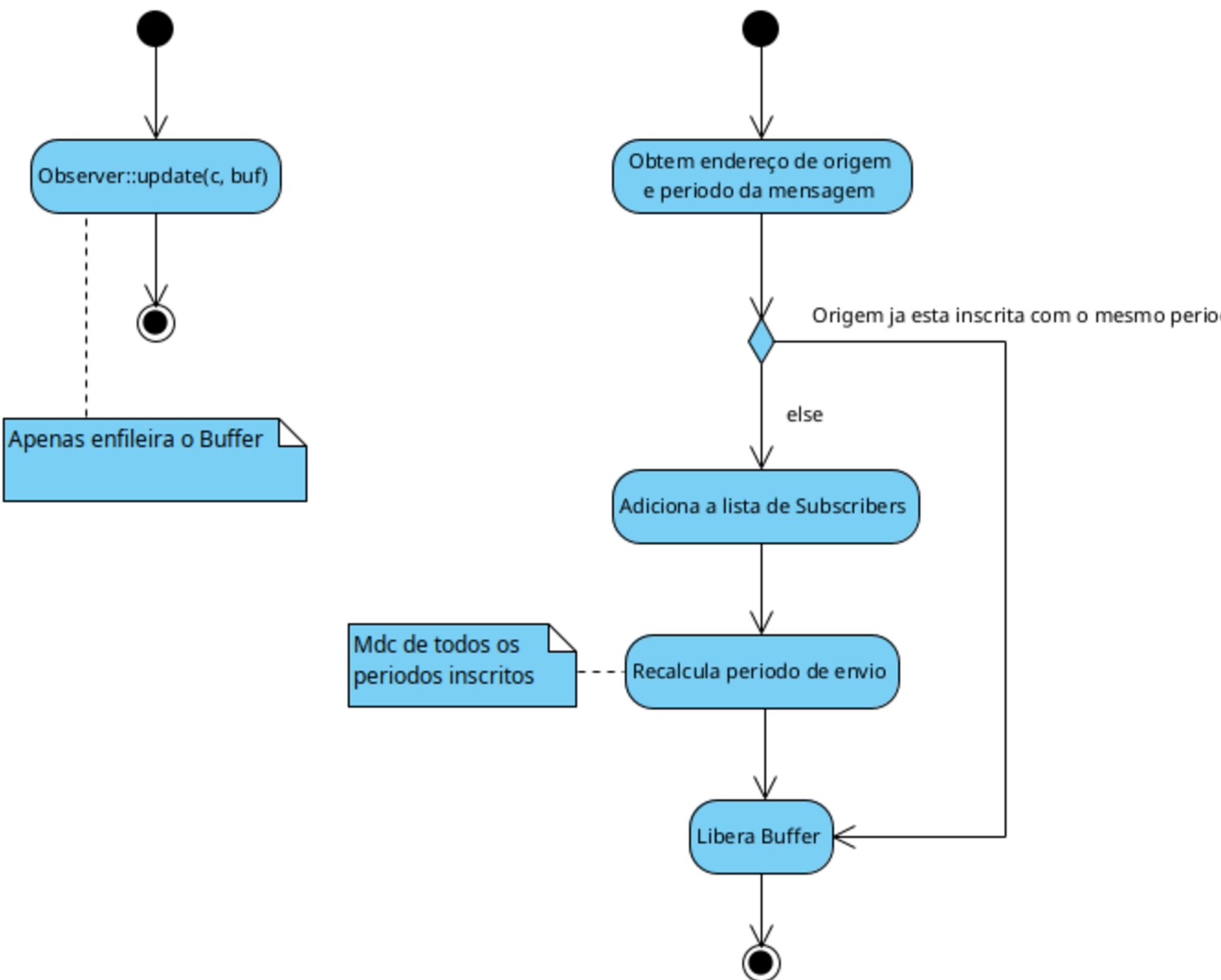
act [User View Publish]



act [SmartData.update(...)]

Subscriber SmartData

Publisher SmartData



Gerenciamento de Período

- **SmartData Produtor:**

- Faz a coleta de períodos recebendo mensagens de Interesse com desiredPeriod de múltiplos Subscritores (SmDS).
- Faz a definição de período de publicação publicando dados em intervalos baseados no Máximo Divisor Comum dos desiredPeriod de todos os seus subscritores ativos. Isso garante a frequência mínima necessária.
- Contador de Ciclo: Cada publicação é marcada por um período cíclico incremental
 - Valor mínimo é mdc de todos os períodos recebidos.
 - Valor máximo é o período mais alto requerido. Após chegar no máximo, retorna ao mínimo.

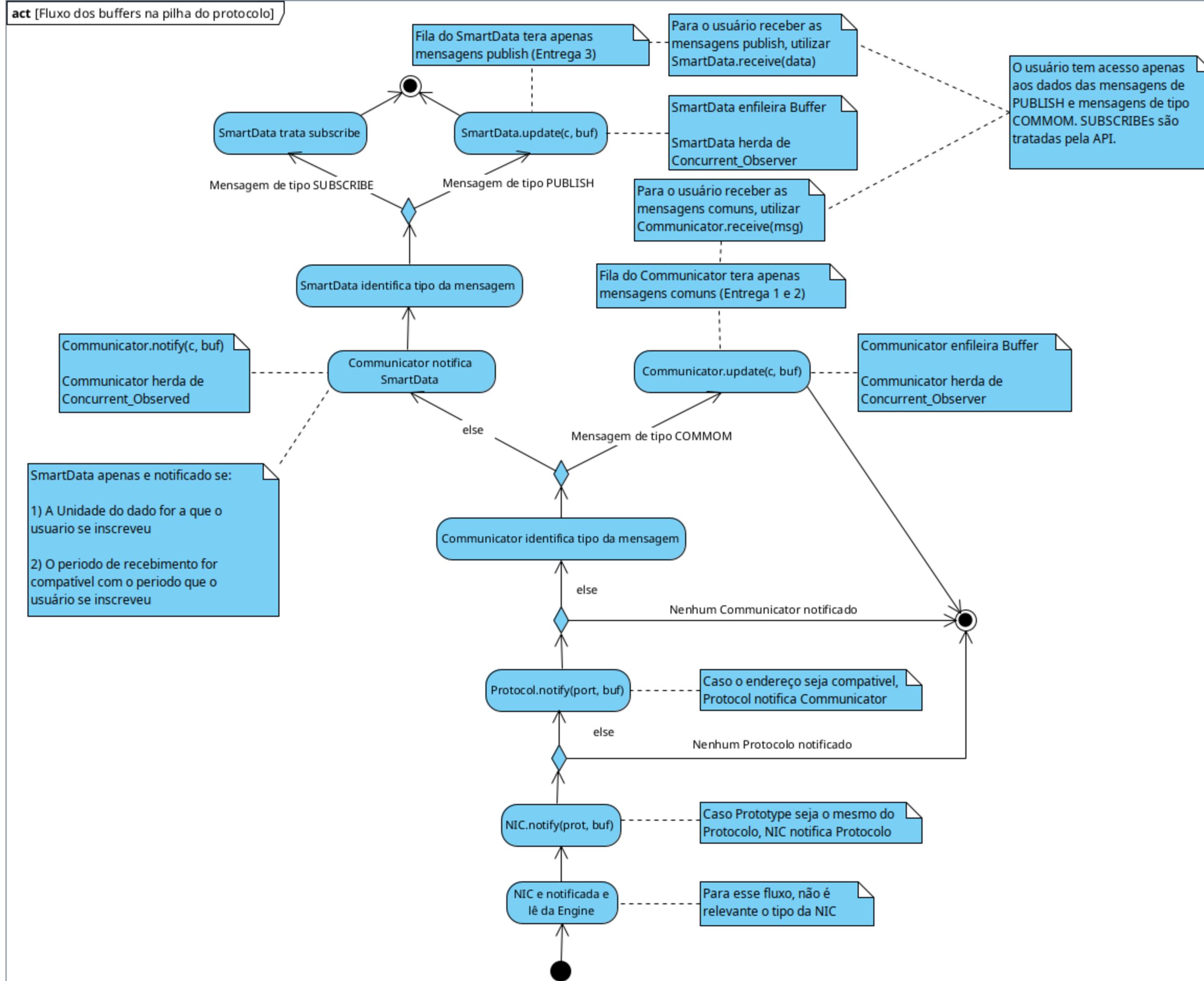
- **SmartData Subscritor:**

- Recebe Publicações: Obtém todas as publicações enviadas por outro SmartData Produtor.
- Verificação Dupla para Aceitar Dado:
 - Tipo Correto: A unidade da publicação corresponde à unidade de interesse do subscritor.
 - Momento Correto (Condição de Múltiplo): O ciclo atual do produtor é um múltiplo do período que este subscritor solicitou (`producer_cycle_count % this->desiredPeriod == 0`).

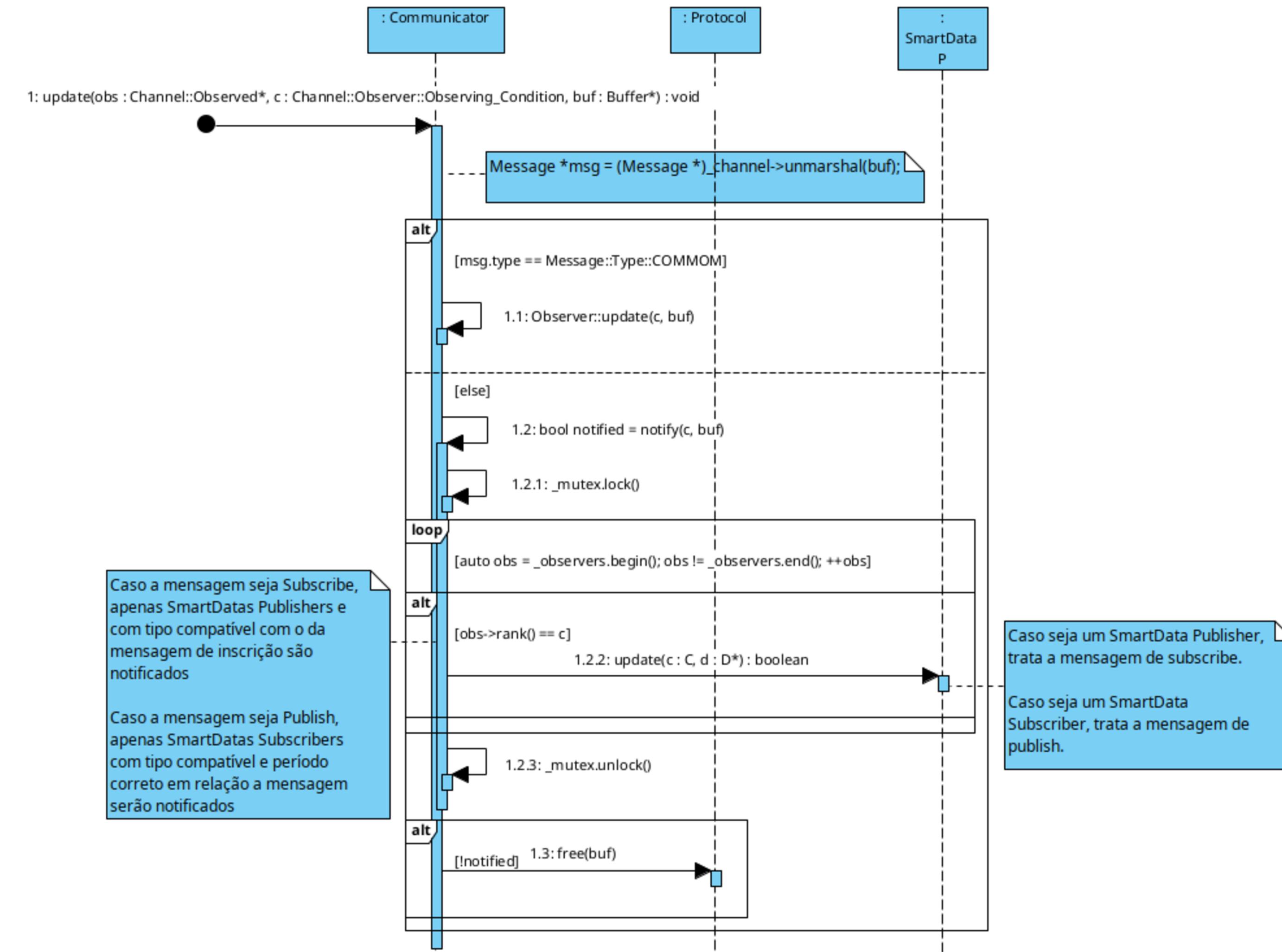
Resumo: **Produtor:** "Grita" atualizações na frequência mais rápida que satisfaz o subscritor mais exigente, marcando cada "grito" com um número de ciclo.

- Subscritor: "Escuta" todos os "gritos", mas só processa o dado se:

- For do tipo que ele pediu.
- E o número do ciclo do "grito" indicar que é a "vez" dele receber, conforme seu período solicitado.

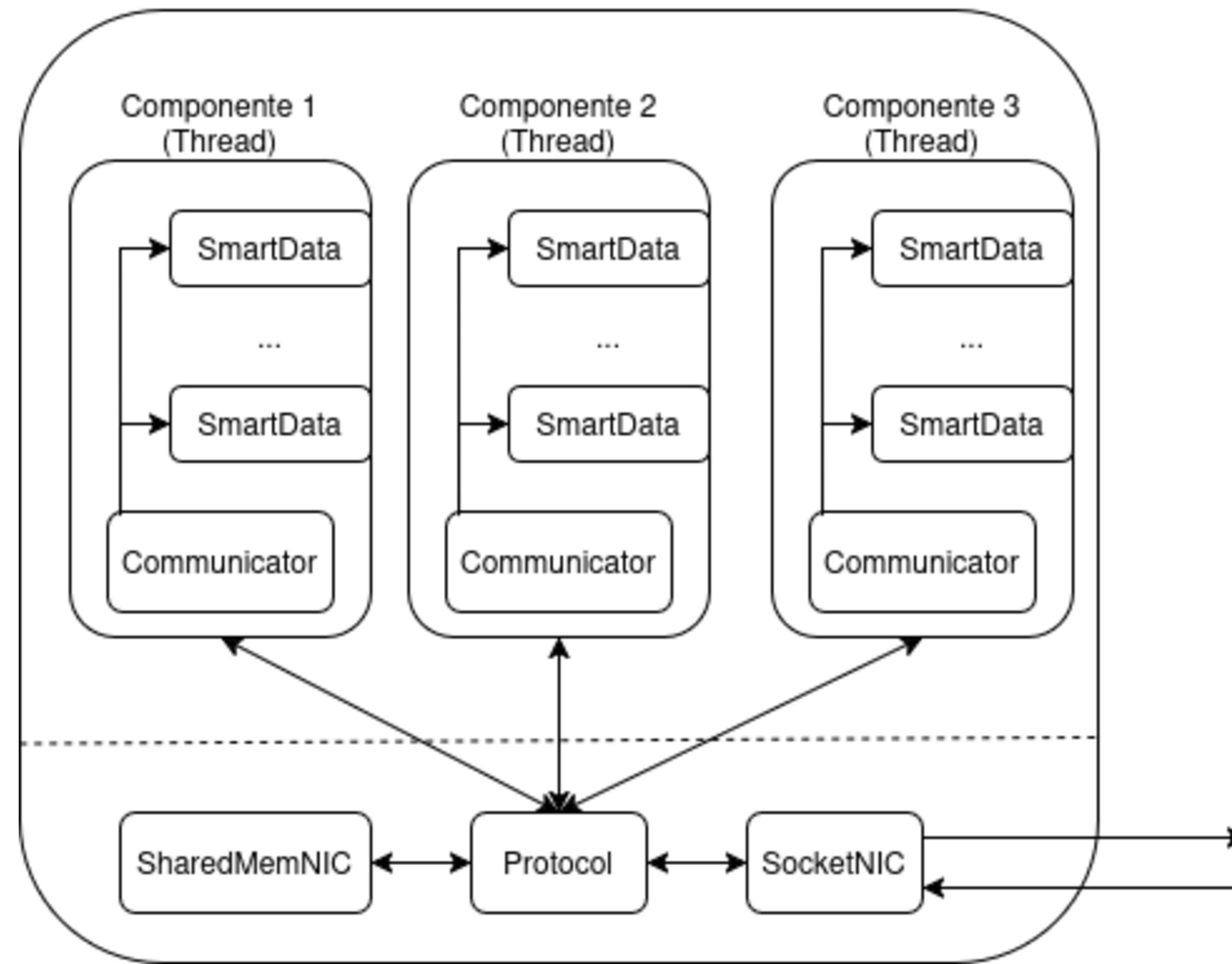


sd [Communicator.update(...)]

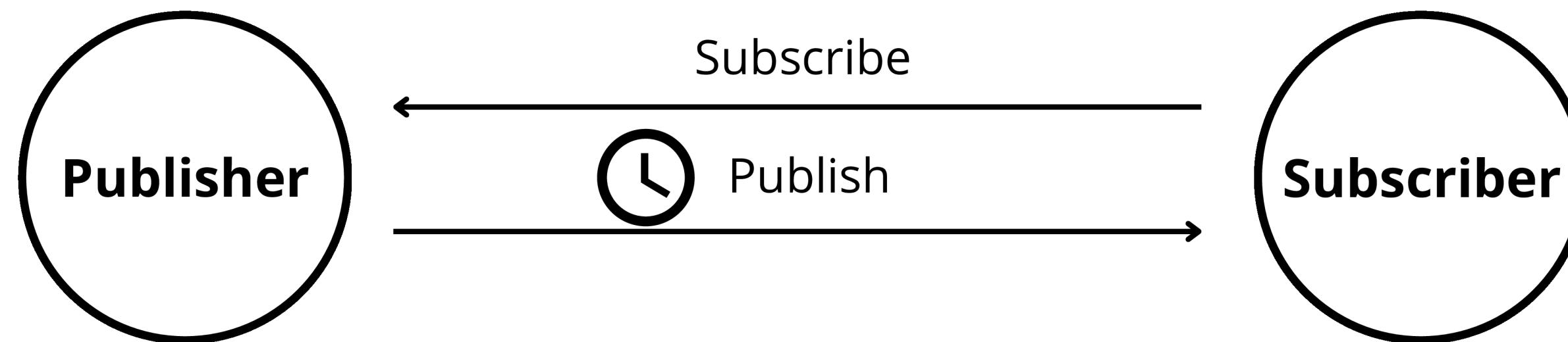


Estrutura do Veículo

Veículo (Processo)

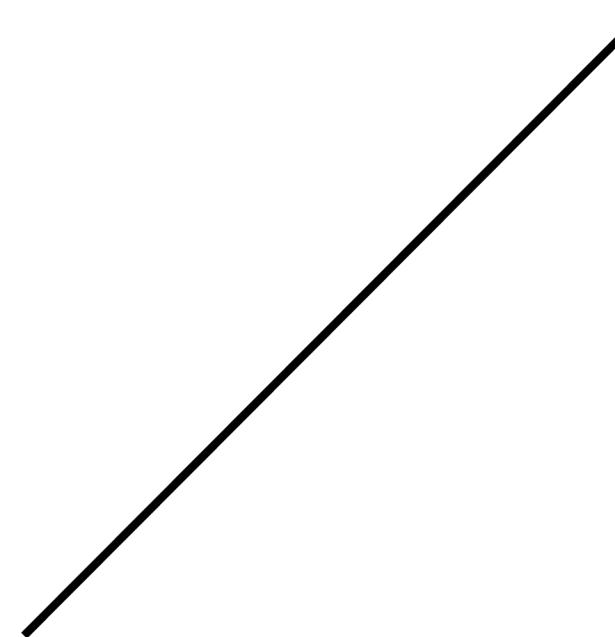


Teste Básico



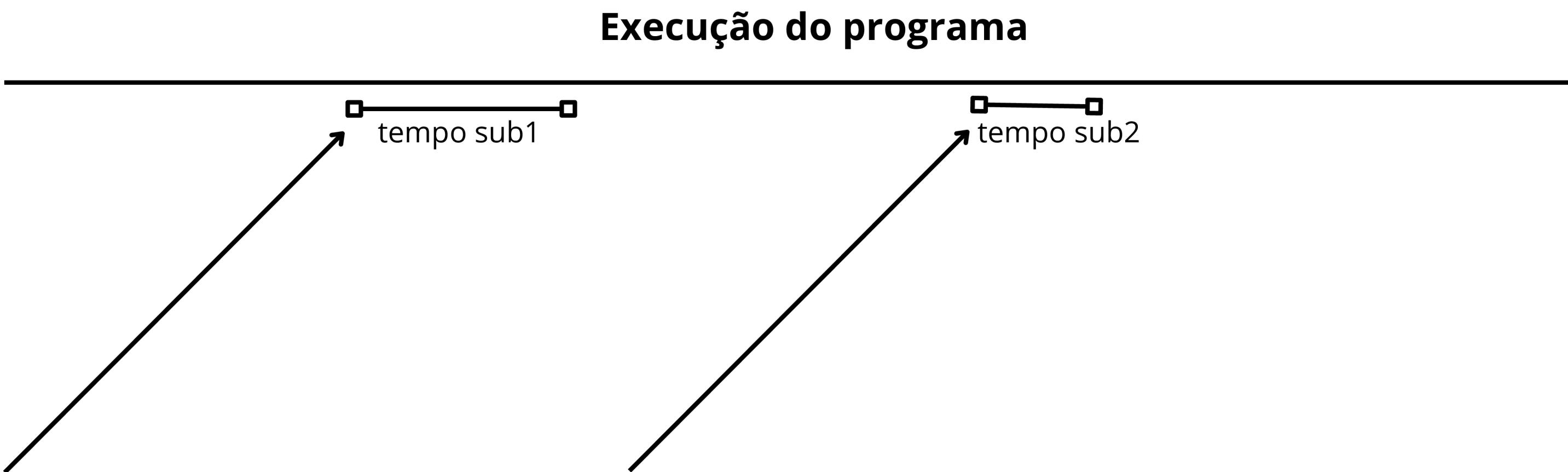
Teste de Resubscribe

Execução do programa



**Executa um resubscribe no
meio da execução do
programa**

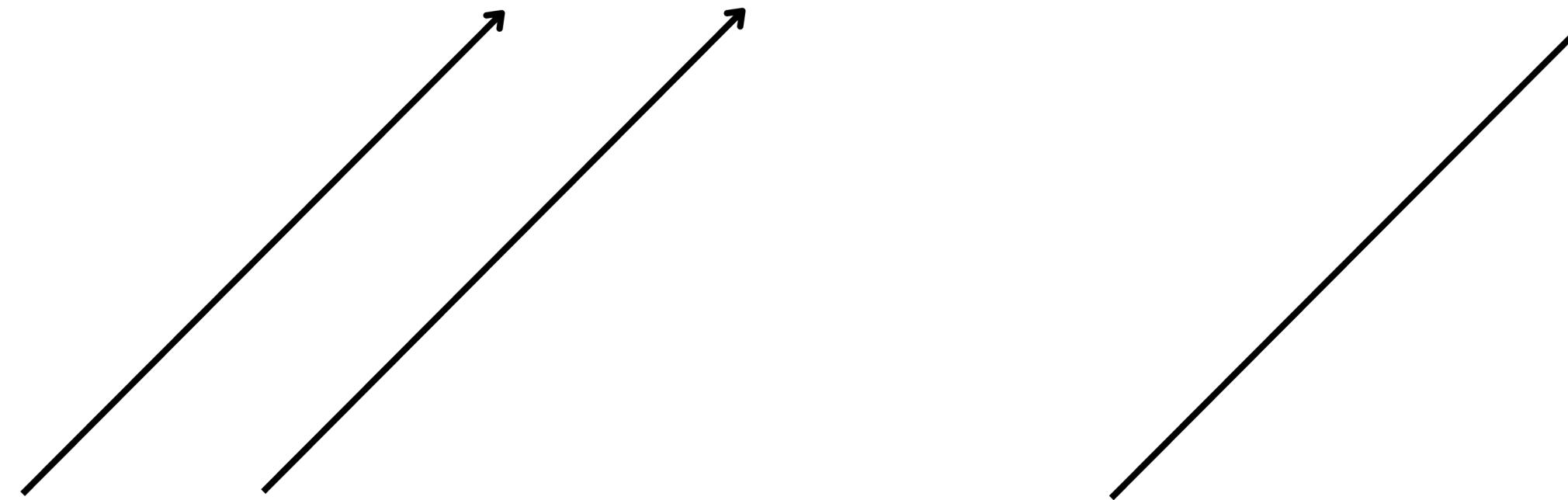
Teste de Disparidade de Tempo



Verifica a diferença entre o tempo em que as mensagens foram recebidas e o período esperado. Imprime a média das diferenças

Teste de Subscribers em Períodos Diferentes

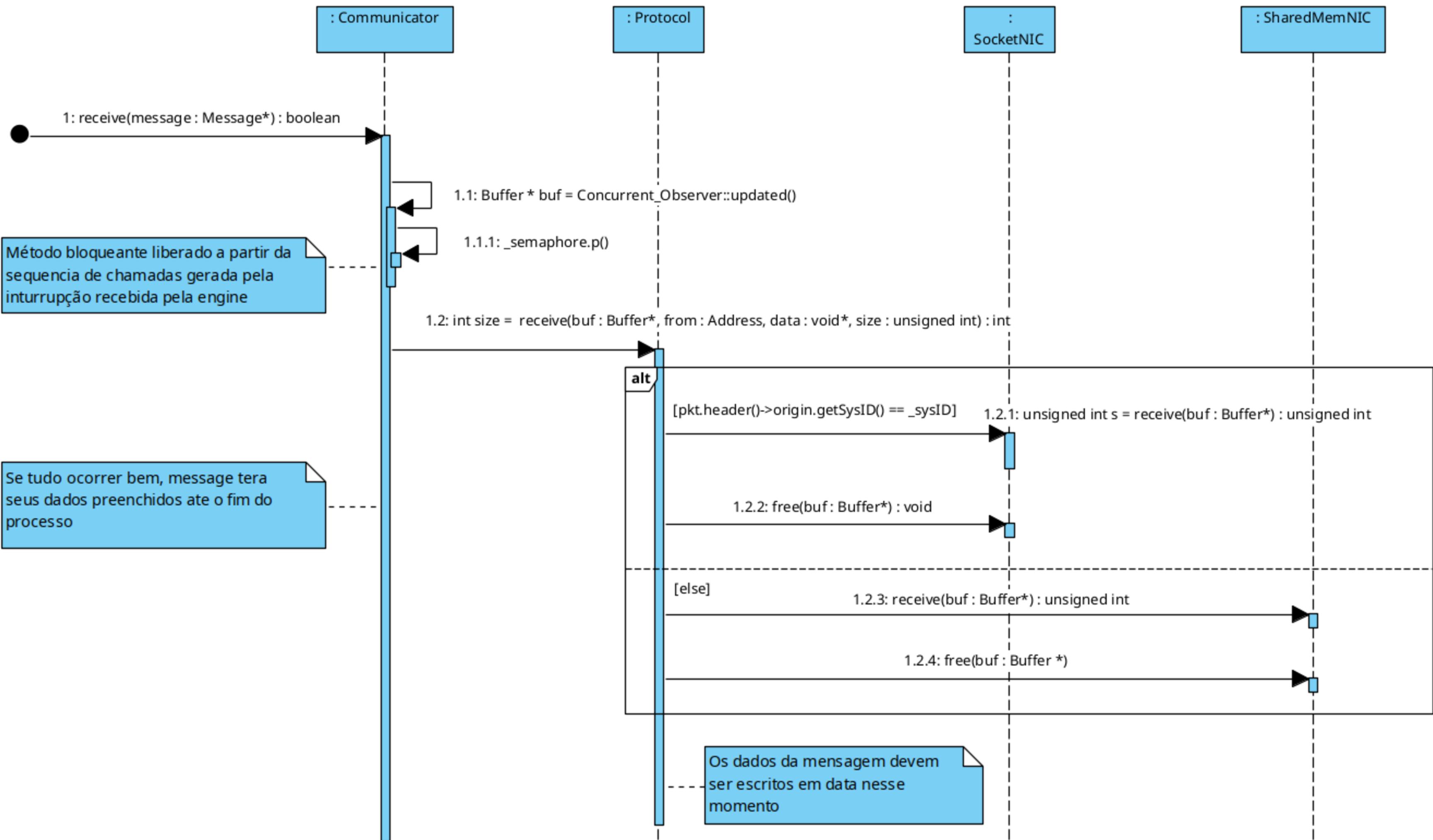
Execução do programa



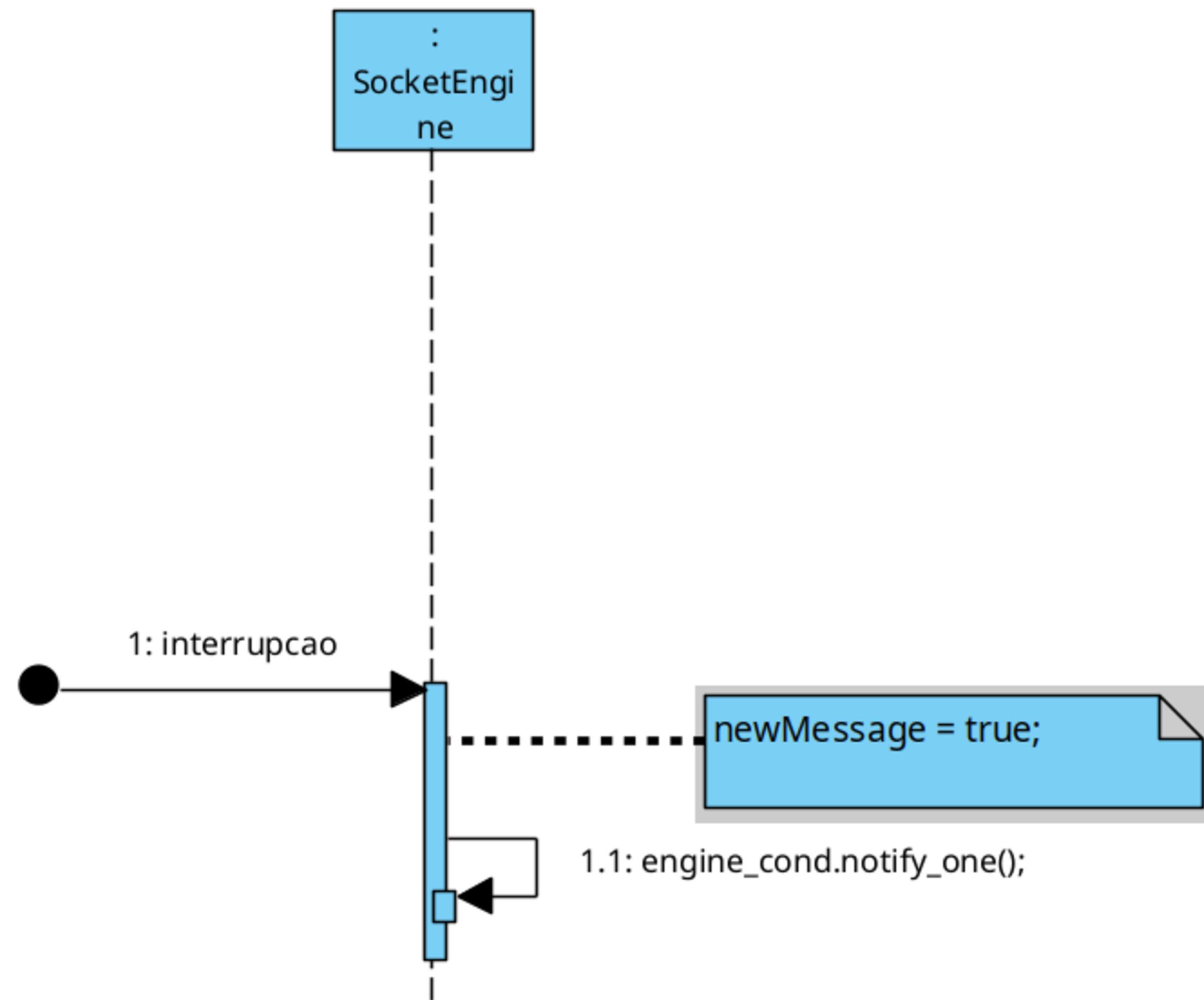
Verifica se todos os
subscribers, apesar de seus
períodos diferentes, recebem
as mensagens corretamente

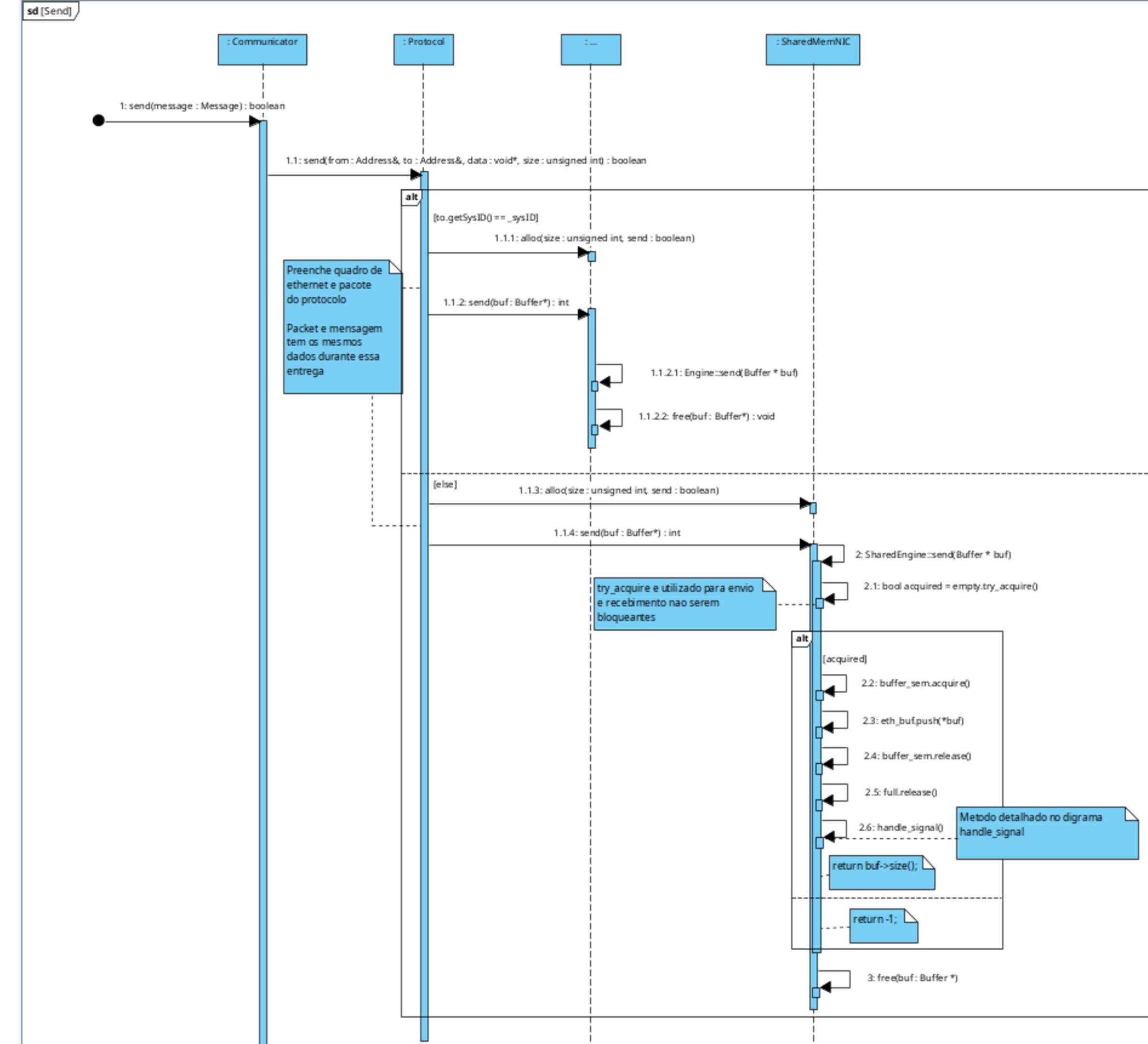
Outros diagramas

Diagramas de Sequência

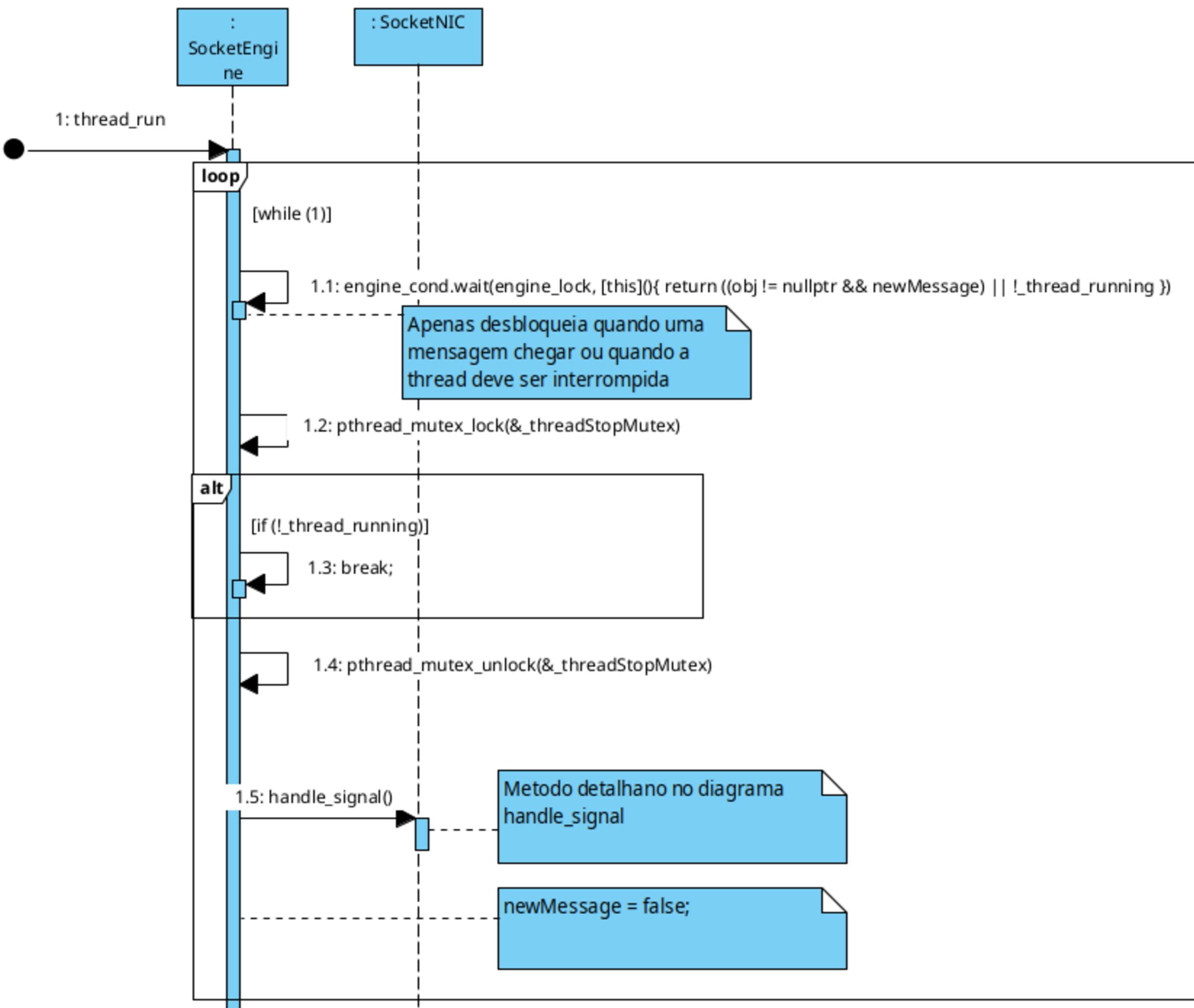
sd [Receive]

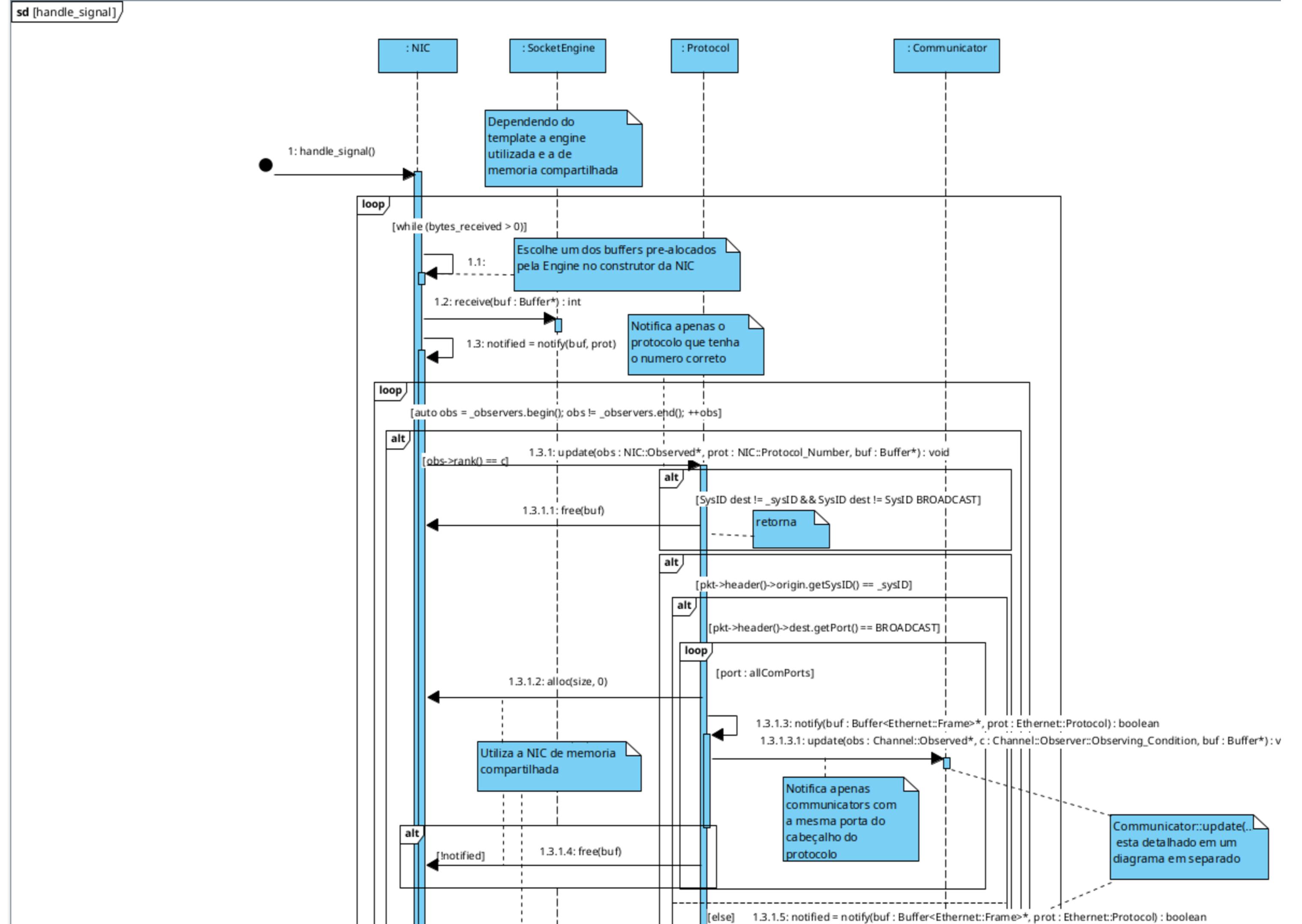
sd [SIGIO handler]

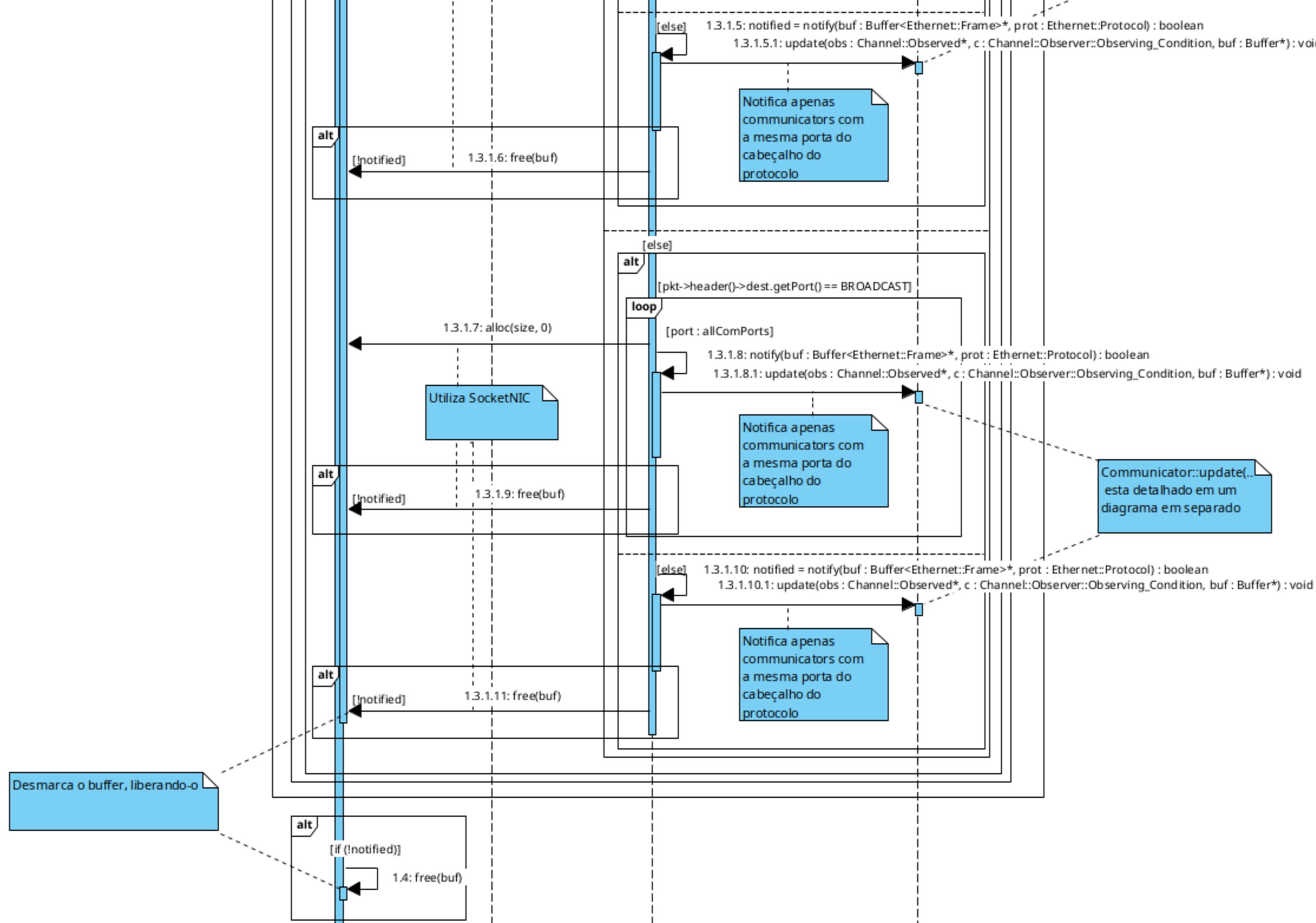




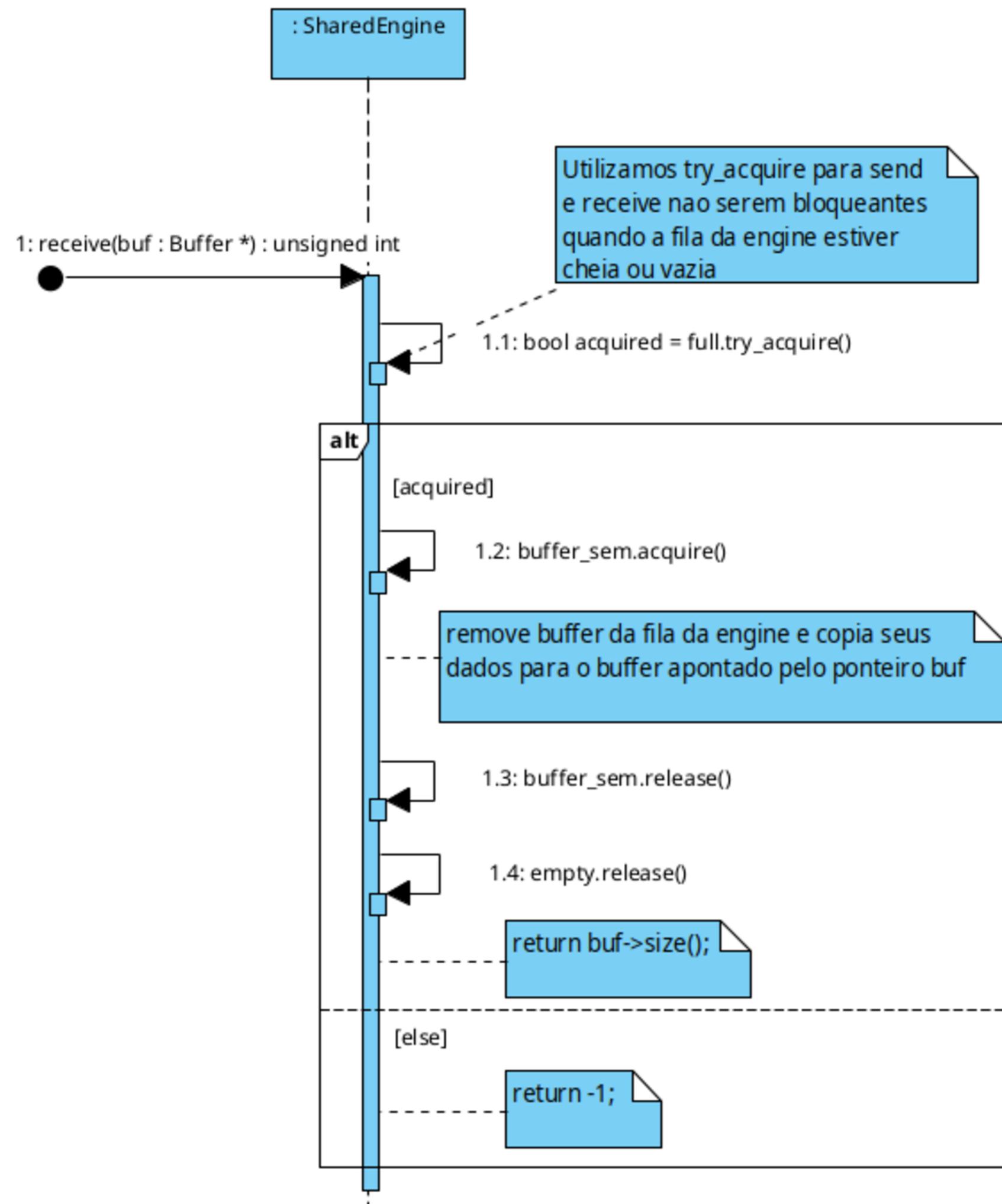
sd [Recv Thread]







sd [Receive SharedMemEngine]



Diagramas de Atividade

act [User View Send]

Cada Communicator é identificado unicamente pelo seu Address

Mensagem tem o seguinte formato:

```
Address ::= SEQUENCE {  
    mac OCTET STRING (SIZE(6)),  
    sysID OCTET STRING (SIZE(4)),  
    port OCTET STRING (SIZE(2))  
}
```

```
Message ::= SEQUENCE {  
    destAddress Address,  
    srcAddress Address,  
    type OCTET STRING(SIZE(1)),  
    data_length OCTET STRING (SIZE(4)),  
    data OCTET STRING (SIZE(0..1471))  
}
```

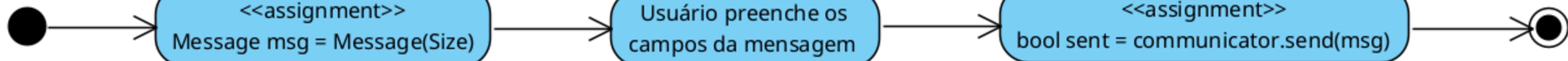
Endereços especiais:

0 = Broadcast_SysID

Caso uma mensagem seja enviada com o SysID 0, todos os veículos (Processos) alcançáveis dentro da rede recebem a mensagem.

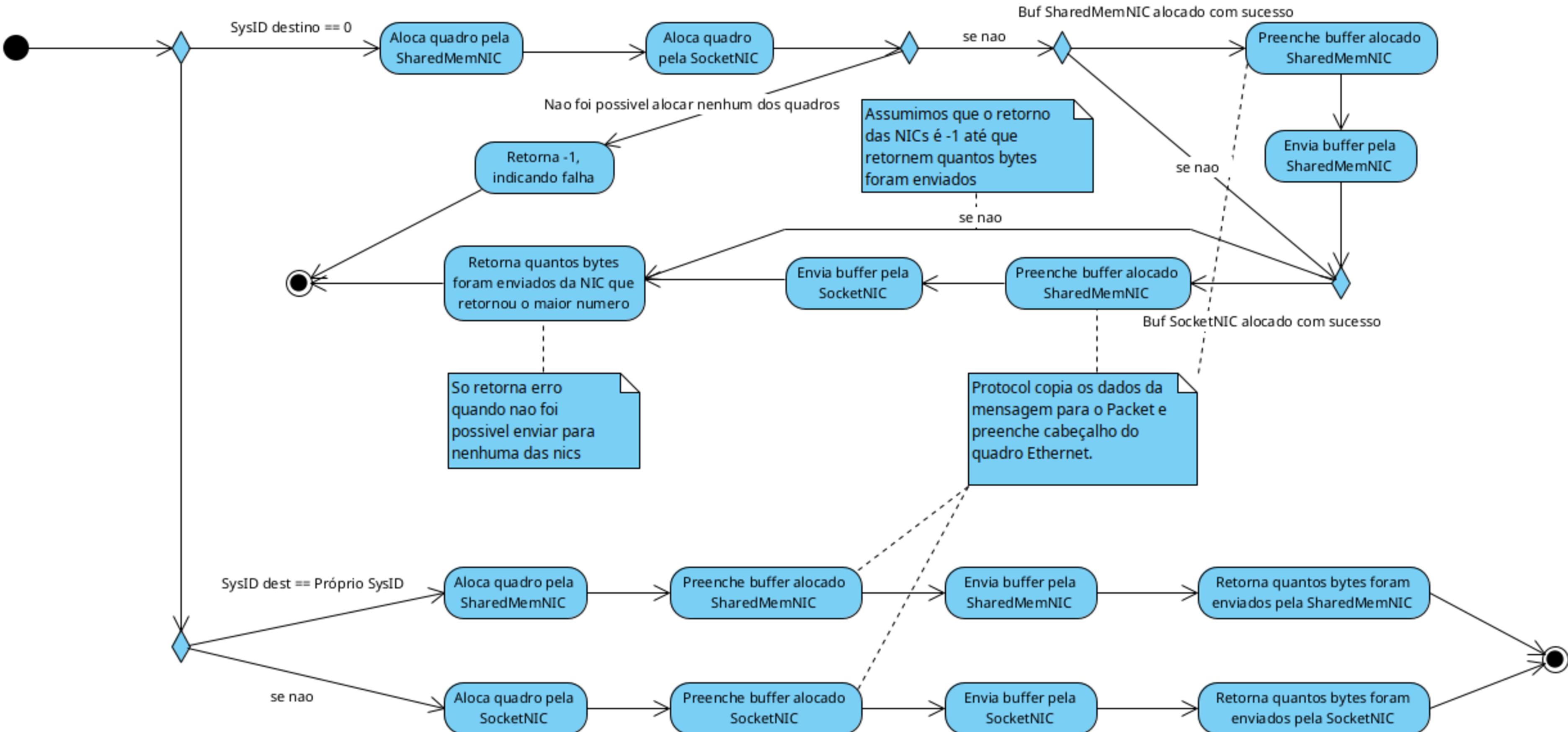
0xFFFF = Broadcast_Port

Caso uma mensagem seja enviada com Port 0xFFFF, todos os componentes(Thread) de um veículo (Processo) recebem a mensagem.



Usuario pode obter seu proprio endereço
(Endereço do comunicador) pelo metodo
Communicator::addr()

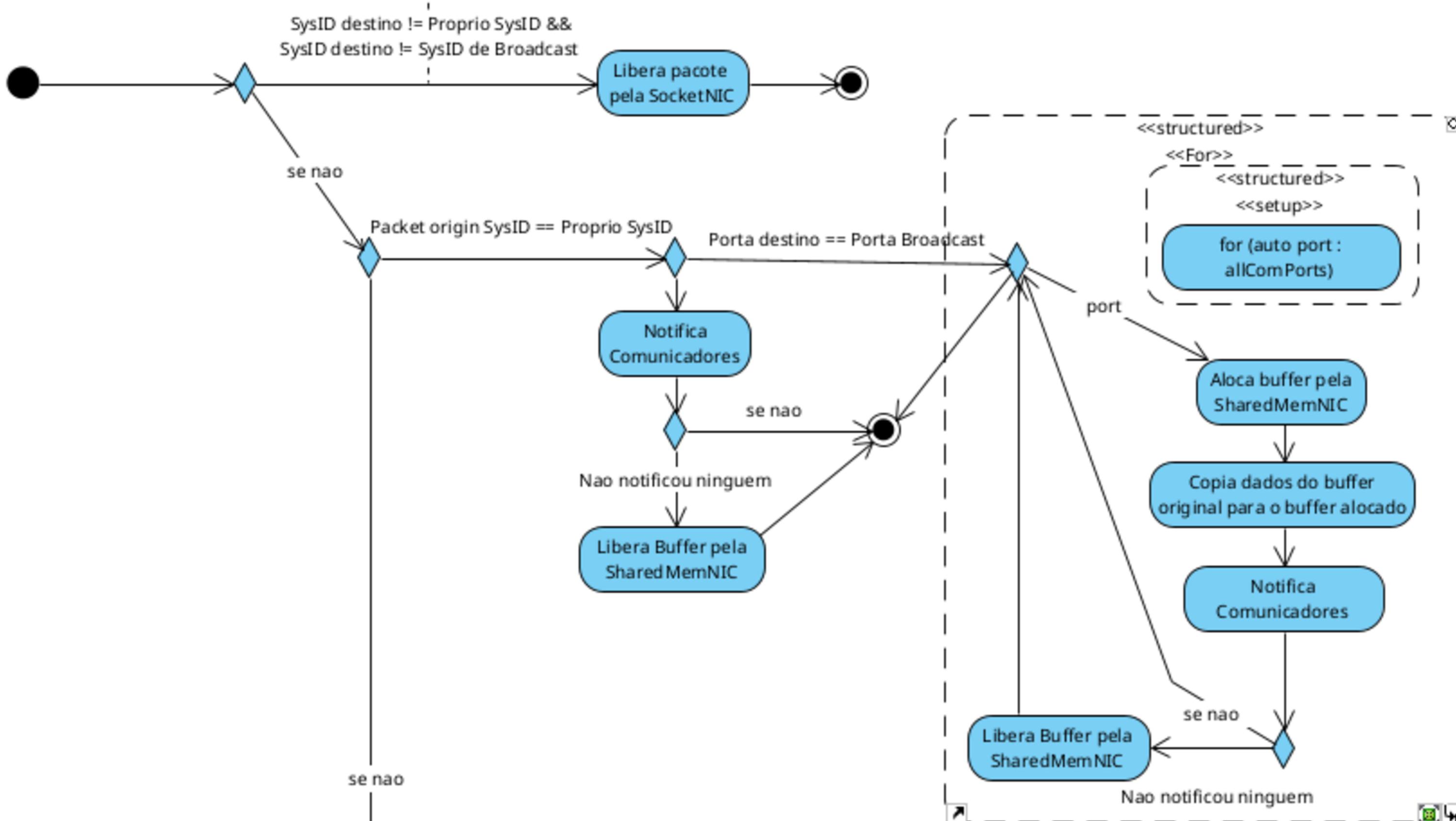
act [protocol.send(...)]

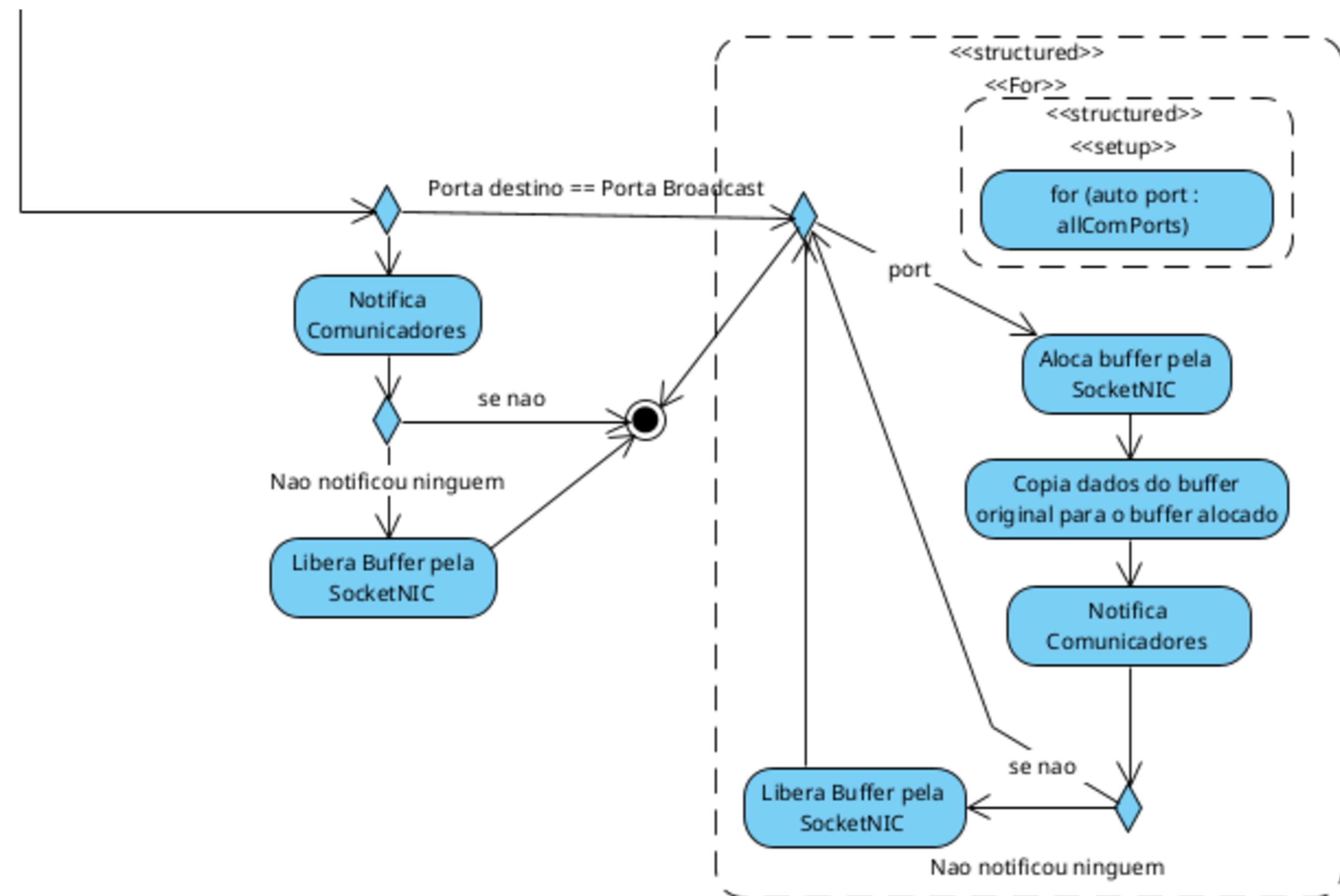


act [protocol.update(...)]

Esse metodo é chamado quando uma NIC notifica o protocolo de uma nova mensagem

O protocolo armazena um SysID em seu interior que representa o Identificador unico do veiculo





act [SharedEngine Flow]

