
Pade Documentation

Versão 1.0

Lucas Melo

01/10/2015

1	Sistemas Multiagentes para Python!	1
1.1	PADE é divertido!	1
1.2	E fácil de instalar!	1
1.3	Créditos	1
2	Guia do Usuário	3
2.1	Instalação	3
2.2	Alô Mundo	3
2.3	Meu Primeiro Agente	4
2.4	Agentes Temporais	5
2.5	Enviando Mensagens	6
2.6	Recebendo Mensagens	8
2.7	Um momento Por Favor!	9
2.8	Enviando Objetos	11
2.9	Seleção de Mensagens	12
2.10	Interface Gráfica	12
2.11	Protocolos	12
3	Referência da API do PADE	23
3.1	API PADE	23
	Índice de Módulos do Python	29
	Índice	31

Sistemas Multiagentes para Python!

PADE é um framework para desenvolvimento, execução e gerenciamento de sistemas multiagentes em ambientes de computação distribuída.

PADE é escrito 100% em Python e utiliza as bibliotecas do projeto Twisted para implementar a comunicação entre os nós da rede.

PADE é software livre, licenciado sob os termos da licença MIT, desenvolvido no âmbito da Universidade Federal do Ceará pelo Grupo de Redes Elétricas Inteligentes (GREI) que pertence ao departamento de Engenharia Elétrica.

Qualquer um que queira contribuir com o projeto é convidado a baixar, executar, testar e enviar feedback a respeito das impressões tiradas da plataforma.

1.1 PADE é divertido!

```
# este e o arquivo start_ams.py
from pade.misc.common import set_ams, start_loop

if __name__ == '__main__':
    set_ams('localhost', 8000)
    start_loop(list(), gui=True)
```

1.2 E fácil de instalar!

Para instalar o PADE basta executar o seguinte comando em um terminal linux:

```
$ pip install pade
$ python start_ams.py
```

1.3 Créditos

PADE é software livre e é desenvolvido pela equipe do grupo de redes elétricas inteligentes:

Lucas Melo: Ditador Benevolente Vitalício (Benevolent Dictator for Life) do PADE

Professora Ruth Leão: Coordenadora do Grupo de Redes Elétricas Inteligentes da UFC

Professor Raimundo Furtado: Coordenador do Grupo de Redes Elétricas Inteligentes da UFC

Professor Giovanni Cordeiro: Colaborador do projeto PADE

Guia do Usuário

2.1 Instalação

Instalar o PADE em seu computador, ou dispositivo embarcado é bem simples, basta que ele esteja conectado à internet!

2.1.1 Instalação via PIP

Para instalar o PADE via PIP basta digitar em um terminal:

```
$ pip install pade
```

Pronto! O Pade está instalado!

2.1.2 Instalação via GitHub

Se você quiser ter acesso ao fonte do PADE e instala-lo a partir do fonte, basta digitar as seguintes linhas no terminal:

```
$ git clone https://github.com/lucassm/Pade
$ cd Pade
$ python setup.py install
```

Pronto o PADE está pronto para ser utilizado, faça um teste no IPython digitando:

```
from pade.misc.utility import display_message
```

2.2 Alô Mundo

PADE foi implementado com um objetivo central: simplicidade!

Depois de ter o PADE instalado em seu computador fica bem simples começar a trabalhar.

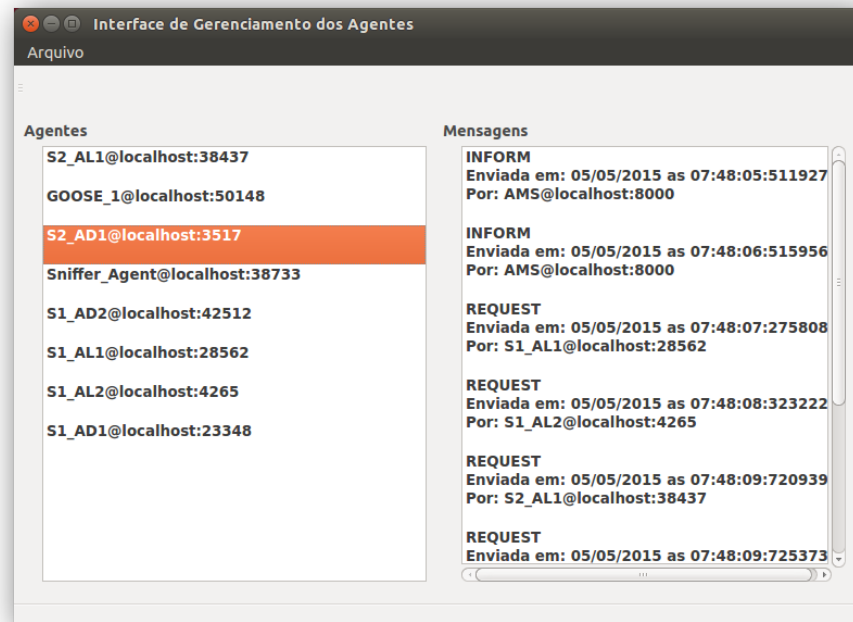
Em uma pasta crie um arquivo chamado `start_ams.py` com o seu editor de texto preferido, e copie e cole o seguinte trecho de código dentro dele:

```
# este e o arquivo start_ams.py
from pade.misc.common import set_ams, start_loop

if __name__ == '__main__':
```

```
set_ams('localhost', 8000)
start_loop(list(), gui=True)
```

A essa altura você já deve estar vendo a interface gráfica de monitoramento dos agentes em Python, assim como essa:



2.3 Meu Primeiro Agente

Agora já está na hora de construir um agente de verdade!

2.3.1 Construindo meu primeiro agente

Para implementar seu primeiro agente abra seu editor de textos preferido e digite o seguinte código:

```
from pade.misc.utility import display_message
from pade.misc.common import set_ams, start_loop
from pade.core.agent import Agent
from pade.acl.aid import AID

class AgenteHelloWorld(Agent):
    def __init__(self, aid):
        super(AgenteHelloWorld, self).__init__(aid=aid, debug=False)
        display_message(self.aid.localname, 'Hello World!')

if __name__ == '__main__':
    set_ams('localhost', 8000, debug=False)
```



```

agents = list()

agente_hello = AgenteHelloWorld(AID(name='agente_hello'))
agente_hello.ams = {'name': 'localhost', 'port': 8000}
agents.append(agente_hello)

start_loop(agents, gui=True)

```

Este já é um agente, mas não tem muita utilidade, não é mesmo! Executa apenas uma vez :(

Então como construir um agente que tenha seu comportamento executado de tempos em tempos?

2.4 Agentes Temporais

Em aplicações reais é comum que o comportamento do agente seja executado de tempos em tempos e não apenas uma vez, mas como fazer isso no PADE? :(

2.4.1 Execução de um agente temporal

Este é um exemplo de um agente que executa indefinidamente um comportamento a cada 1,0 segundos:

```

#!/coding=utf-8
# Hello world temporal in PADE!
#
# Criado por Lucas S Melo em 21 de julho de 2015 - Fortaleza, Ceará - Brasil

from pade.behaviours.protocols import TimedBehaviour
from pade.misc.utility import display_message
from pade.misc.common import set_ams, start_loop
from pade.core.agent import Agent
from pade.acl.aid import AID

class ComportTemporal(TimedBehaviour):
    def __init__(self, agent, time):
        super(ComportTemporal, self).__init__(agent, time)

    def on_time(self):
        super(ComportTemporal, self).on_time()
        display_message(self.agent.aid.localname, 'Hello World!')

class AgenteHelloWorld(Agent):
    def __init__(self, aid):
        super(AgenteHelloWorld, self).__init__(aid=aid, debug=False)

        comp_temp = ComportTemporal(self, 1.0)

        self.behaviours.append(comp_temp)

if __name__ == '__main__':
    set_ams('localhost', 8000, debug=False)

```

```
agents = list()

agente_1 = AgenteHelloWorld(AID(name='agente_1'))
agente_1.ams = {'name': 'localhost', 'port': 8000}

agents.append(agente_1)

start_loop(agents, gui=True)
```

2.4.2 Execução de dois agentes temporais

Mas e se eu quiser dois agentes com o mesmo comportamento!? Não tem problema, basta instanciar um outro agente com a mesma classe!

```
if __name__ == '__main__':

    set_ams('localhost', 8000, debug=False)

    agents = list()

    agente_1 = AgenteHelloWorld(AID(name='agente_1'))
    agente_1.ams = {'name': 'localhost', 'port': 8000}

    agente_2 = AgenteHelloWorld(AID(name='agente_2'))
    agente_2.ams = {'name': 'localhost', 'port': 8000}

    agents.append(agente_1)
    agents.append(agente_2)

    start_loop(agents, gui=True)
```

2.5 Enviando Mensagens

Para enviar uma mensagem com o PADE é muito simples! As mensagens enviadas pelos agentes desenvolvidos com PADE seguem o padrão FIPA-ACL e têm os seguintes campos:

- *conversation-id*: identidade única de uma conversa;
- *performative*: rótulo da mensagem;
- *sender*: remetente da mensagem;
- *receivers*: destinatários da mensagem;
- *content*: conteúdo da mensagem;
- *protocol*: protocolo da mensagem;
- *language*: linguagem utilizada;
- *encoding*: codificação da mensagem;
- *ontology*: ontologia utilizada;
- *reply-with*: Expressão utilizada pelo agente de resposta a identificar a mensagem;
- *reply-by*: A referência a uma ação anterior em que a mensagem é uma resposta;
- *in-reply-to*: Data/hora indicando quando uma resposta deve ser recebida..

2.5.1 Mensagens FIPA-ACL no PADE

Uma mensagem FIPA-ACL pode ser montada no pade da seguinte forma:

```
from pade.acl.messages import ACLMessage, AID
message = ACLMessage(ACLMessage.INFORM)
message.set_protocol(ACLMessage.FIPA_REQUEST_PROTOCOL)
message.add_receiver(AID('agente_destino'))
message.set_content('Ola Agente')
```

2.5.2 Enviando uma mensagem com PADE

Uma vez que se está dentro de uma instância da classe *Agent()* a mensagem pode ser enviada, simplesmente utilizando o comando:

```
self.send(message)
```

2.5.3 Mensagem no padrão FIPA-ACL

Realizando o comando *print message* a mensagem no padrão FIPA ACL será impressa na tela:

```
(inform
  :conversationID b2e806b8-50a0-11e5-b3b6-e8b1fc5c3cdf
  :receiver
    (set
      (agent-identifier
        :name agente_destino@localhost:51645
        :addresses
          (sequence
            localhost:51645
          )
      )
    )
  :content "Ola Agente"
  :protocol fipa-request protocol
)
```

2.5.4 Mensagem no padrão XML

Mas também é possível obter a mensagem no formato XML por meio do comando *print message.as_xml()*

```
<?xml version="1.0" ?>
<ACLMessage date="01/09/2015 as 08:58:03:113891">
  <performative>inform</performative>
  <sender/>
  <receivers>
    <receiver>agente_destino@localhost:51645</receiver>
  </receivers>
  <reply-to/>
  <content>Ola Agente</content>
  <language/>
  <encoding/>
  <ontology/>
```

```
<protocol>fipa-request protocol</protocol>
<conversationID>b2e806b8-50a0-11e5-b3b6-e8b1fc5c3cdf</conversationID>
<reply-with/>
<in-reply-to/>
<reply-by/>
</ACLMessage>
```

2.6 Recebendo Mensagens

No PADE para que um agente possa receber mensagens, basta que o método `react()` seja implementado, dentro da classe que herda da classe `Agent()`.

2.6.1 Recebendo mensagens FIPA-ACL com PADE

No exemplo a seguir são implementados dois agentes distintos, o primeiro é o agente 'remetente' modelado pela classe `Remetente()`, que tem o papel de enviar uma mensagem ao agente *destinatario* modelado pela classe `Destinatario`, que irá receber a mensagem enviada pelo agente *remetente* e por isso tem o método `react()` implementado.

```
from pade.misc.utility import display_message
from pade.misc.common import set_ams, start_loop
from pade.core.agent import Agent
from pade.acl.aid import AID
from pade.acl.messages import ACLMessage

class Remetente(Agent):
    def __init__(self, aid):
        super(Remetente, self).__init__(aid=aid, debug=False)

    def on_start(self):
        display_message(self.aid.localname, 'Enviando Mensagem')
        message = ACLMessage(ACLMessage.INFORM)
        message.add_receiver(AID('destinatario'))
        message.set_content('Ola')
        self.send(message)

    def react(self, message):
        pass

class Destinatario(Agent):
    def __init__(self, aid):
        super(Destinatarior, self).__init__(aid=aid, debug=False)

    def react(self, message):
        display_message(self.aid.localname, 'Mensagem recebida')

if __name__ == '__main__':
    set_ams('localhost', 8000, debug=False)

    agentes = list()
```

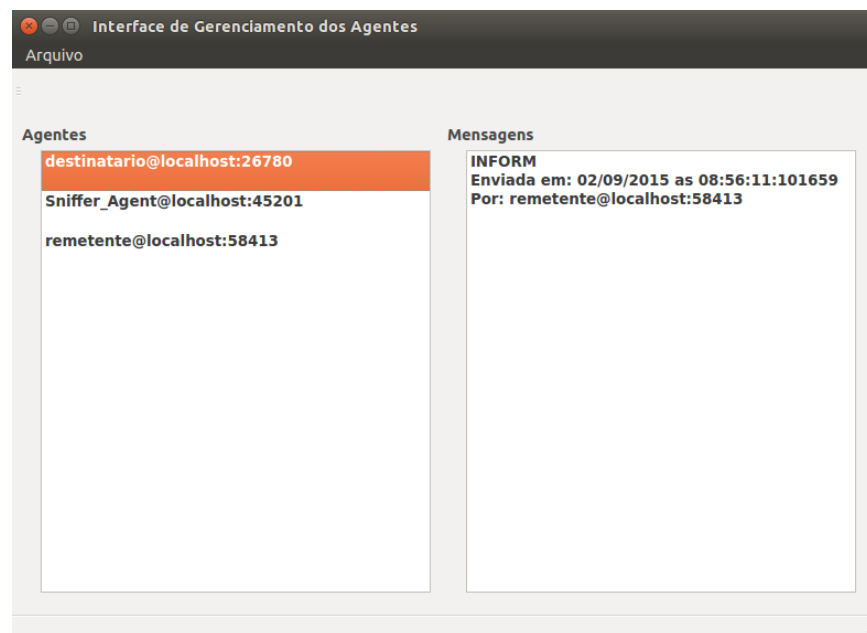
```
destinatario = Destinatario(AID(name='destinatario'))
destinatario.ams = {'name': 'localhost', 'port': 8000}
agentes.append(destinatario)

remetente = Remetente(AID(name='remetente'))
remetente.ams = {'name': 'localhost', 'port': 8000}
agentes.append(remetente)

start_loop(agentes, gui=True)
```

2.6.2 Visualização via Interface Gráfica

A seguir é possível observar a interface gráfica do PADE que mostra os agentes cadastrados no AMS.



Ao clicar na mensagem recebida pelo agente *destinatario* é possível observar todos os dados contidos na mensagem:

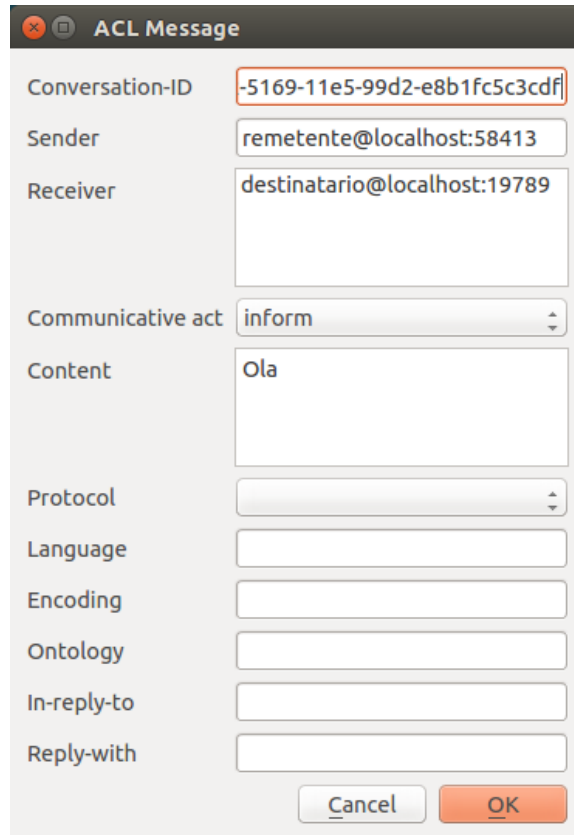
2.7 Um momento Por Favor!

Com PADE é possível adiar a execução de um determinado trecho de código de forma bem simples! É só utilizar o método *call_later()* disponível na classe *Agent()*.

2.7.1 Como utilizar o método *call_later()*

Para utilizar *call_later()*, os seguintes parâmetros devem ser especificados: tempo de atraso, metodo que deve ser chamado após este tempo e argumento do metodo utilizado, *call_later(tempo, metodo, *args)*.

No código a seguir utiliza *call_later()* é utilizado na classe *Remetente()* no método *on_start()* para assegurar que todos os agentes já foram lançados na plataforma, chamando o método *send_message()* 4,0 segundos após a inicialização dos agentes:



The image shows a dialog box titled "ACL Message". It contains several input fields and a dropdown menu. The "Conversation-ID" field is highlighted with a red border and contains the text "-5169-11e5-99d2-e8b1fc3cdf". The "Sender" field contains "remetente@localhost:58413". The "Receiver" field contains "destinatario@localhost:19789". The "Communicative act" dropdown menu is set to "inform". The "Content" field contains "Ola". The "Protocol" dropdown menu is empty. The "Language", "Encoding", "Ontology", "In-reply-to", and "Reply-with" fields are empty. At the bottom right, there are "Cancel" and "OK" buttons.

```

from pade.misc.utility import display_message
from pade.misc.common import set_ams, start_loop
from pade.core.agent import Agent
from pade.acl.aid import AID
from pade.acl.messages import ACLMessage

class Remetente(Agent):
    def __init__(self, aid):
        super(Remetente, self).__init__(aid=aid, debug=False)

    def on_start(self):
        self.call_later(4.0, self.send_message)

    def send_message(self):
        display_message(self.aid.localname, 'Enviando Mensagem')
        message = ACLMessage(ACLMessage.INFORM)
        message.add_receiver(AID('destinatario'))
        message.set_content('Ola')
        self.send(message)

    def react(self, message):
        pass

class Destinatario(Agent):
    def __init__(self, aid):
        super(Destinatario, self).__init__(aid=aid, debug=False)

```

```

def react(self, message):
    display_message(self.aid.localname, 'Mensagem recebida')

if __name__ == '__main__':

    set_ams('localhost', 8000, debug=False)

    agentes = list()

    destinatario = Destinatario(AID(name='destinatario'))
    destinatario.ams = {'name': 'localhost', 'port': 8000}
    agentes.append(destinatario)

    remetente = Remetente(AID(name='remetente'))
    remetente.ams = {'name': 'localhost', 'port': 8000}
    agentes.append(remetente)

    start_loop(agentes, gui=True)

```

2.8 Enviando Objetos

Nem sempre o que é preciso enviar para outros agentes pode ser representado por texto simples não é mesmo!

Para enviar objetos encapsulados no content de mensagens FIPA-ACL com PADE basta utilizar o módulo nativo do Python *pickle*.

2.8.1 Enviando objetos serializados com pickle

Para enviar um objeto serializado com pickle basta seguir os passos:

```
import pickle
```

pickle é uma biblioteca para serialização de objetos, assim, para serializar um objeto qualquer, utilize *pickle.dumps()*, veja:

```

dados = {'nome' : 'agente_consumidor', 'porta' : 2004}
dados_serial = pickle.dumps(dados)
message.set_content(dados_serial)

```

Pronto! O objeto já pode ser enviado no conteúdo da mensagem.

2.8.2 Recebendo objetos serializados com pickle

Agora para receber o objeto, basta carregá-lo utilizando o comando:

```

dados_serial = message.content
dados = pickle.loads(dados_serial)

```

Simples assim ;)

2.9 Seleção de Mensagens

Com PADE é possível implementar filtros de mensagens de maneira simples e direta, por meio da classe Filtro:

```
from pade.acl.filters import Filter
```

2.9.1 Filtrando mensagens com o módulo filters

Por exemplo para a seguinte mensagem:

```
from pade.acl.messages import ACLMessage
from pade.acl.aid import AID

message = ACLMessage(ACLMessage.INFORM)
message.set_protocol(ACLMessage.FIPA_REQUEST_PROTOCOL)
message.set_sender(AID('remetente'))
message.add_receiver(AID('destinatario'))
```

Podemos criar o seguinte filtro:

```
from pade.acl.filters import Filter

f.performative = ACLMessage.REQUEST
```

Em uma sessão IPython é possível observar o efeito da aplicação do filtro sobre a mensagem:

```
In [12]: f.filter(message)
Out[12]: False
```

Ajustando agora o filtro para outra condição:

```
f.performative = ACLMessage.INFORM
```

E aplicando o filtro novamente sobre a mensagem, obtemos um novo resultado:

```
In [14]: f.filter(message)
Out[14]: True
```

2.10 Interface Gráfica

Para ativar a funcionalidade de interface gráfica do PADE é bem simples, basta passar o parâmetro *gui=True* na função

```
start_loop(agentes, gui=True)
```

A interface gráfica do PADE ainda está bem simples e sem muitas funcionalidades, mas para a versão 2.0 prometemos um interface mais completa e funcional, ok :)

2.11 Protocolos

O PADE tem suporte aos protocolos mais utilizados estabelecidos pela FIPA, são eles:

- *FIPA-Request*
- *FIPA-Contract-Net*

- *FIPA-Subscribe*

No PADE qualquer protocolo deve ser implementado como uma classe que estende a classe do protocolo desejado, por exemplo para implementar o protocolo FIPA-Request, deve ser implementada uma classe que implementa a herança da classe FipaRequestProtocol:

```
from pade.behaviours.protocols import FipaRequestProtocol

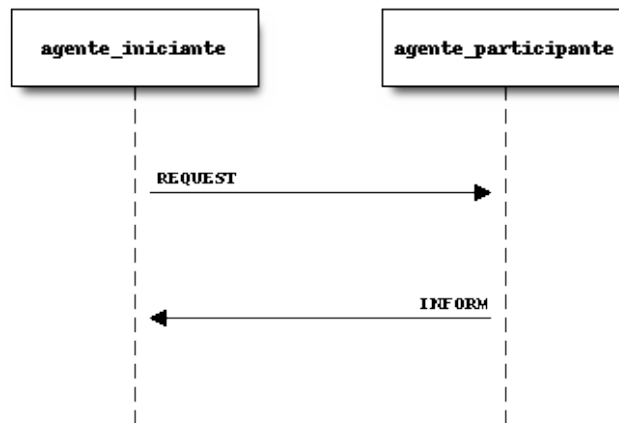
class ProtocoloDeRequisicao(FipaRequestProtocol):

    def __init__(self):
        super(ProtocoloDeRequisicao, self).__init__()
```

2.11.1 FIPA-Request

O protocolo FIPA-Request é o mais simples de se utilizar e constitui uma padronização do ato de requisitar alguma tarefa ou informação de um agente iniciador para um agente participante.

O diagrama de comunicação do protocolo FIPA-Request está mostrado na Figura abaixo:



Para exemplificar o protocolo FIPA-Request, iremos utilizar como exemplo a interação entre dois agentes, um agente relógio, que a cada um segundo exibe na tela a data e o horário atuais, mas com um problema, o agente relógio não sabe calcular nem a data, e muito menos o horário atual. Assim, ele precisa requisitar estas informações do agente horário que consegue calcular estas informações.

Dessa forma, será utilizado o protocolo FIPA-Request, para que estas informações sejam trocadas entre os dois agentes, sendo o agente relógio o iniciante, no processo de requisição e o agente horário, o participante, segue o código do exemplo:

```
from pade.misc.common import start_loop, set_ams
from pade.misc.utility import display_message
from pade.core.agent import Agent
from pade.acl.messages import ACLMessage
from pade.acl.aid import AID
from pade.behaviours.protocols import FipaRequestProtocol
from pade.behaviours.protocols import TimedBehaviour

from datetime import datetime
```

```
class CompRequest(FipaRequestProtocol):
    """Comportamento FIPA Request
    do agente Horario"""
    def __init__(self, agent):
        super(CompRequest, self).__init__(agent=agent,
                                           message=None,
                                           is_initiator=False)

    def handle_request(self, message):
        super(CompRequest, self).handle_request(message)
        display_message(self.agent.aid.localname, 'mensagem request recebida')
        now = datetime.now()
        reply = message.create_reply()
        reply.set_performative(ACLMessage.INFORM)
        reply.set_content(now.strftime('%d/%m/%Y - %H:%M:%S'))
        self.agent.send(reply)

class CompRequest2(FipaRequestProtocol):
    """Comportamento FIPA Request
    do agente Relogio"""
    def __init__(self, agent, message):
        super(CompRequest2, self).__init__(agent=agent,
                                           message=message,
                                           is_initiator=True)

    def handle_inform(self, message):
        display_message(self.agent.aid.localname, message.content)

class ComportTemporal(TimedBehaviour):
    """Comportamento FIPA Request
    do agente Relogio"""
    def __init__(self, agent, time, message):
        super(ComportTemporal, self).__init__(agent, time)
        self.message = message

    def on_time(self):
        super(ComportTemporal, self).on_time()
        self.agent.send(self.message)

class AgenteHorario(Agent):
    """Classe que define o agente Horario"""
    def __init__(self, aid):
        super(AgenteHorario, self).__init__(aid=aid, debug=False)

        self.comport_request = CompRequest(self)

        self.behaviours.append(self.comport_request)

class AgenteRelogio(Agent):
    """Classe que define o agente Relogio"""
    def __init__(self, aid):
        super(AgenteRelogio, self).__init__(aid=aid)
```

```

    # mensagem que requisita horario do horario
    message = ACLMessage(ACLMessage.REQUEST)
    message.set_protocol(ACLMessage.FIPA_REQUEST_PROTOCOL)
    message.add_receiver(AID(name='horario'))
    message.set_content('time')

    self.comport_request = ComptRequest2(self, message)
    self.comport_temp = ComportTemporal(self, 1.0, message)

    self.behaviours.append(self.comport_request)
    self.behaviours.append(self.comport_temp)

def main():
    agentes = list()
    set_ams('localhost', 8000, debug=False)

    a = AgenteHorario(AID(name='horario'))
    a.ams = {'name': 'localhost', 'port': 8000}
    agentes.append(a)

    a = AgenteRelogio(AID(name='relogio'))
    a.ams = {'name': 'localhost', 'port': 8000}
    agentes.append(a)

    start_loop(agentes, gui=True)

if __name__ == '__main__':
    main()

```

Na primeira parte do código são importadas todos módulos e classes necessários à construção dos agentes, logo em seguida as classes que implementam o protocolo são definidas, as classes `ComptRequest` e `ComptRequest2` que serão associadas aos comportamentos dos agentes `horario` e `relogio`, respectivamente. Como o agente `relogio` precisa, a cada segundo enviar requisição ao agente `horario`, então também deve ser associado a este agente um comportamento temporal, definido na classe `ComportTemporal` que envia uma solicitação ao agente `horario`, a cada segundo.

Em seguida, os agente propriamente ditos, são definidos nas classes `AgenteHorario` e `AgenteRelogio` que estendem a classe `Agent`, nessas classes é que os comportamentos e protocolos são associados a cada agente.

Na ultima parte do código, é definida uma função `main` que indica a localização do agente `ams`, instancia os agentes e dá inicio ao loop de execução.

2.11.2 FIPA-Contract-Net

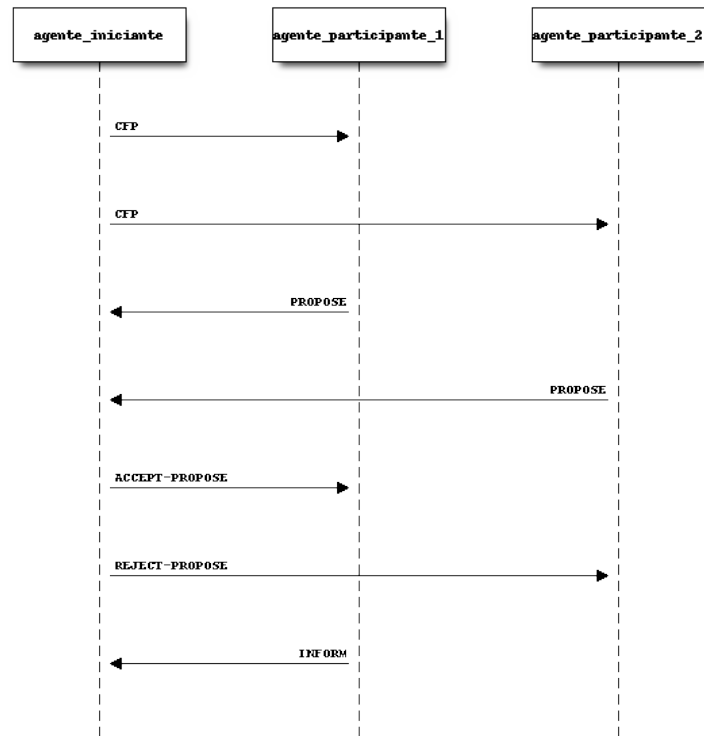
O protocolo `FIPA-Contract-Net` é utilizado para situações onde é necessário realizar algum tipo de negociação entre agentes. Da mesma forma que no protocolo `FIPA-Request`, no protocolo `FIPA-ContractNet` existem dois tipos de agentes, um agente que inicia a negociação, ou agente iniciante, fazendo solicitação de propostas e um ou mais agentes que participam da negociação, ou agentes participantes, que repondem às solicitações de propostas do agente iniciante. Veja:

Um exemplo de utilização do protocolo `FIPA-ContractNet` na negociação é mostrado abaixo, com a solicitação de um agente iniciante por potência elétrica a outros dois agentes participantes:

```

from pade.misc.common import start_loop, set_ams
from pade.misc.utility import display_message
from pade.core.agent import Agent
from pade.acl.messages import ACLMessage

```



```

from pade.acl.aid import AID
from pade.behaviours.protocols import FipaContractNetProtocol

class CompContNet1(FipaContractNetProtocol):
    '''CompContNet1

    Comportamento FIPA-ContractNet Iniciante que envia mensagens
    CFP para outros agentes alimentadores solicitando propostas
    de restauração. Este comportamento também faz a análise das
    das propostas e analisa-as selecionando a que julga ser a
    melhor'''

    def __init__(self, agent, message):
        super(CompContNet1, self).__init__(
            agent=agent, message=message, is_initiator=True)
        self.cfp = message

    def handle_all_proposes(self, proposes):
        """
        """

        super(CompContNet1, self).handle_all_proposes(proposes)

        melhor_propositor = None
        maior_potencia = 0.0
        demais_propositores = list()
        display_message(self.agent.aid.name, 'Analisando propostas...')
  
```

```

i = 1

# lógica de seleção de propostas pela maior potência disponibilizada
for message in proposes:
    content = message.content
    potencia = float(content)
    display_message(self.agent.aid.name,
                    'Analisando proposta {i}'.format(i=i))
    display_message(self.agent.aid.name,
                    'Potencia Ofertada: {pot}'.format(pot=potencia))

    i += 1
    if potencia > maior_potencia:
        if melhor_propositor is not None:
            demais_propositores.append(melhor_propositor)

        maior_potencia = potencia
        melhor_propositor = message.sender
    else:
        demais_propositores.append(message.sender)

display_message(self.agent.aid.name,
                'A melhor proposta foi de: {pot} VA'.format(
                    pot=maior_potencia))

if demais_propositores != []:
    display_message(self.agent.aid.name,
                    'Enviando respostas de recusa...')
    resposta = ACLMessage(ACLMessage.REJECT_PROPOSAL)
    resposta.set_protocol(ACLMessage.FIPA_CONTRACT_NET_PROTOCOL)
    resposta.set_content('')
    for agente in demais_propositores:
        resposta.add_receiver(agente)

    self.agent.send(resposta)

if melhor_propositor is not None:
    display_message(self.agent.aid.name,
                    'Enviando resposta de aceitacao...')

    resposta = ACLMessage(ACLMessage.ACCEPT_PROPOSAL)
    resposta.set_protocol(ACLMessage.FIPA_CONTRACT_NET_PROTOCOL)
    resposta.set_content('OK')
    resposta.add_receiver(melhor_propositor)
    self.agent.send(resposta)

def handle_inform(self, message):
    """
    """
    super(CompContNet1, self).handle_inform(message)

    display_message(self.agent.aid.name, 'Mensagem INFORM recebida')

def handle_refuse(self, message):
    """
    """
    super(CompContNet1, self).handle_refuse(message)

    display_message(self.agent.aid.name, 'Mensagem REFUSE recebida')

```

```
def handle_propose(self, message):
    """
    """
    super(CompContNet1, self).handle_propose(message)

    display_message(self.agent.aid.name, 'Mensagem PROPOSE recebida')

class CompContNet2(FipaContractNetProtocol):
    '''CompContNet2

    Comportamento FIPA-ContractNet Participante que é acionado
    quando um agente recebe uma mensagem do Tipo CFP enviando logo
    em seguida uma proposta e caso esta seja selecionada realiza as
    análises de restrição para que seja possível a restauração'''

    def __init__(self, agent):
        super(CompContNet2, self).__init__(agent=agent,
                                           message=None,
                                           is_initiator=False)

    def handle_cfp(self, message):
        """
        """
        self.agent.call_later(1.0, self._handle_cfp, message)

    def _handle_cfp(self, message):
        """
        """
        super(CompContNet2, self).handle_cfp(message)
        self.message = message

        display_message(self.agent.aid.name, 'Mensagem CFP recebida')

        resposta = self.message.create_reply()
        resposta.set_performative(ACLMessage.PROPOSE)
        resposta.set_content(str(self.agent.pot_disp))
        self.agent.send(resposta)

    def handle_reject_propose(self, message):
        """
        """
        super(CompContNet2, self).handle_reject_propose(message)

        display_message(self.agent.aid.name,
                        'Mensagem REJECT_PROPOSAL recebida')

    def handle_accept_propose(self, message):
        """
        """
        super(CompContNet2, self).handle_accept_propose(message)

        display_message(self.agent.aid.name,
                        'Mensagem ACCEPT_PROPOSE recebida')

        resposta = message.create_reply()
        resposta.set_performative(ACLMessage.INFORM)
        resposta.set_content('OK')
```

```

        self.agent.send(resposta)

class AgenteIniciante(Agent):

    def __init__(self, aid):
        super(AgenteIniciante, self).__init__(aid=aid, debug=False)

        message = ACLMessage(ACLMessage.CFP)
        message.set_protocol(ACLMessage.FIPA_CONTRACT_NET_PROTOCOL)
        message.set_content('60.0')
        message.add_receiver(AID('AP1'))
        message.add_receiver(AID('AP2'))

        comp = CompContNet1(self, message)
        self.behaviours.append(comp)
        self.call_later(2.0, comp.on_start)

class AgenteParticipante(Agent):

    def __init__(self, aid, pot_disp):
        super(AgenteParticipante, self).__init__(aid=aid, debug=False)

        self.pot_disp = pot_disp

        comp = CompContNet2(self)

        self.behaviours.append(comp)

if __name__ == "__main__":

    set_ams('localhost', 5000, debug=False)

    aa_1 = AgenteIniciante(AID(name='AI1'))
    aa_1.ams = {'name': 'localhost', 'port': 5000}

    aa_2 = AgenteParticipante(AID(name='AP1'), 150.0)
    aa_2.ams = {'name': 'localhost', 'port': 5000}

    aa_3 = AgenteParticipante(AID(name='AP2'), 100.0)
    aa_3.ams = {'name': 'localhost', 'port': 5000}

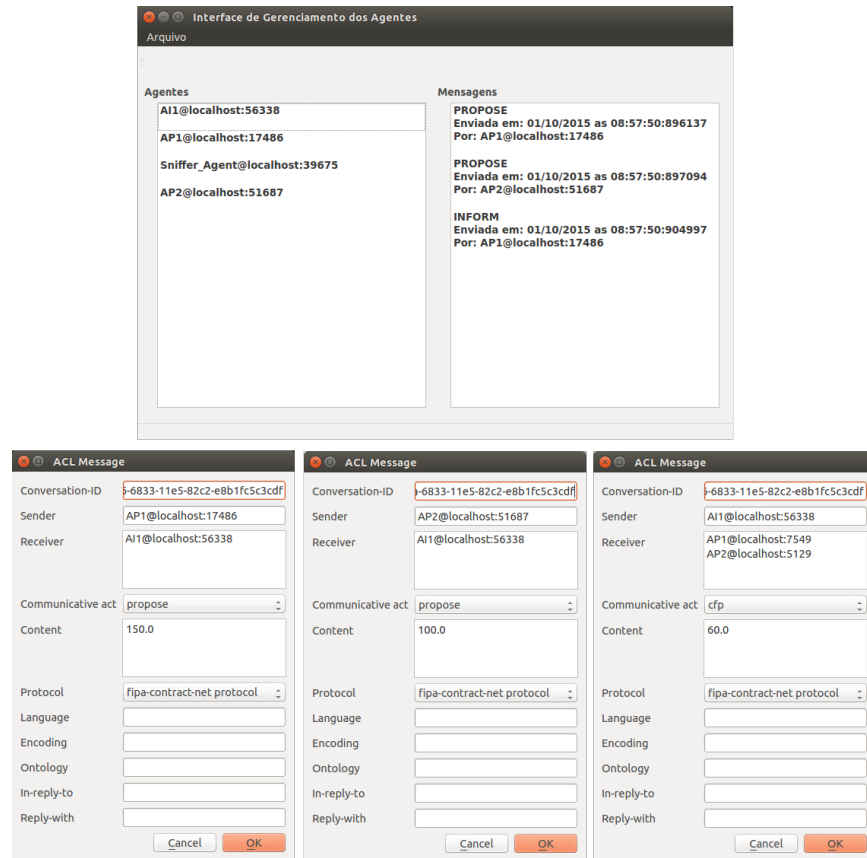
    agents_list = list([aa_1, aa_2, aa_3])

    start_loop(agents_list, gui=True)

```

O código que implementa os agentes que se comunicam utilizando o protocolo FIPA-ContractNet, define as duas classes do protocolo, a primeira implementa o comportamento do agente Iniciante (CompContNet1) e a segunda implementa o comportamento do agente participante (CompContNet2). Note que para a classe iniciante é necessário que uma mensagem do tipo CFP (call for proposes) seja montada e o método on_start() seja chamado, isso é feito dentro da classe que implementa os agente iniciante, AgenteIniciante(), já a classe AgenteParticipante(), implementa os agentes que participarão da negociação como propositores.

É possível observar as mensagens da negociação na interface gráfica do PADE, veja:



2.11.3 FIPA-Subscribe

O protocolo FIPA-Subscribe, implementa o comportamento de editor-assinante, que consiste na presença de um agente editor que pode aceitar a associação de outros agentes interessados, agentes assinantes, em algum tipo de informação que este agente possua, assinando a informação e recebendo mensagem sempre que esta informação for disponibilizada pelo agente editor. Veja:

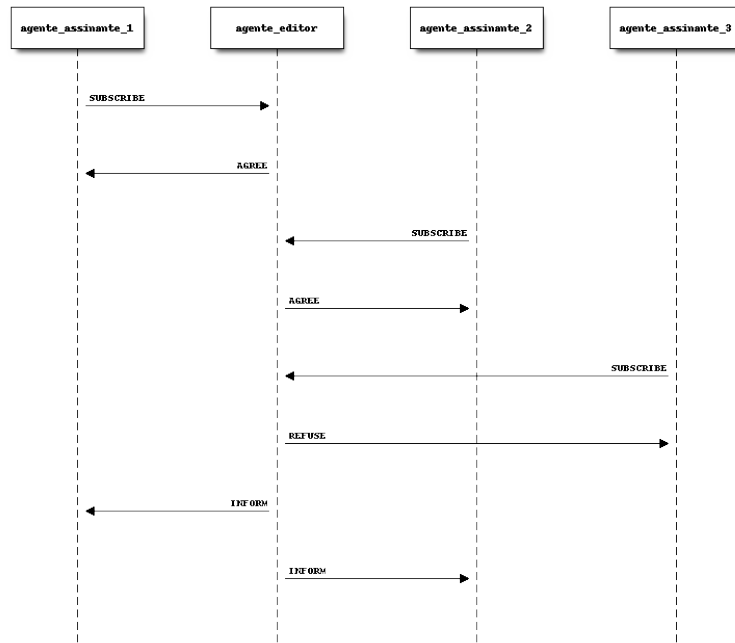
Para assinar a informação o agente precisa enviar uma mensagem SUSBCRIBE para o agente editor. Que por sua vez pode aceitar ou recusar a assinatura (AGREE/REFUSE). Quando uma informação é atualizada, então o editor publica esta informação para todos os seus assinantes, enviando-os mensagens INFORM.

O código que implementa um agente editor e dois agentes assinantes utilizando PADE pode ser visualizado abaixo:

```
from pade.misc.common import start_loop, set_ams
from pade.misc.utility import display_message
from pade.core.agent import Agent
from pade.acl.aid import AID
from pade.acl.messages import ACLMessage
from pade.behaviours.protocols import FipaSubscribeProtocol, TimedBehaviour
from numpy import sin

class SubscribeInitiator(FipaSubscribeProtocol):

    def __init__(self, agent, message):
        super(SubscribeInitiator, self).__init__(agent,
                                                message,
```

is_initiator=True)

```
def handle_agree(self, message):
    display_message(self.agent.aid.name, message.content)
```

```
def handle_inform(self, message):
    display_message(self.agent.aid.name, message.content)
```

```
class SubscribeParticipant(FipaSubscribeProtocol):
```

```
def __init__(self, agent):
    super(SubscribeParticipant, self).__init__(agent,
                                              message=None,
                                              is_initiator=False)
```

```
def handle_subscribe(self, message):
    self.register(message.sender)
    display_message(self.agent.aid.name, message.content)

    resposta = message.create_reply()
    resposta.set_performative(ACLMessage.AGREE)
    resposta.set_content('Pedido de subscricao aceito')
    self.agent.send(resposta)
```

```
def handle_cancel(self, message):
    self.deregister(self, message.sender)
    display_message(self.agent.aid.name, message.content)
```

```
def notify(self, message):
    super(SubscribeParticipant, self).notify(message)
```

```

class Time(TimedBehaviour):

    def __init__(self, agent, notify):
        super(Time, self).__init__(agent, 1)
        self.notify = notify
        self.inc = 0

    def on_time(self):
        super(Time, self).on_time()
        message = ACLMessage(ACLMessage.INFORM)
        message.set_protocol(ACLMessage.FIPA_SUBSCRIBE_PROTOCOL)
        message.set_content(str(sin(self.inc)))

        self.notify(message)
        self.inc += 0.1

class AgenteInitiator(Agent):

    def __init__(self, aid, message):
        super(AgenteInitiator, self).__init__(aid)
        self.protocol = SubscribeInitiator(self, message)
        self.behaviours.append(self.protocol)

class AgenteParticipante(Agent):

    def __init__(self, aid):
        super(AgenteParticipante, self).__init__(aid)

        self.protocol = SubscribeParticipant(self)
        self.timed = Time(self, self.protocol.notify)

        self.behaviours.append(self.protocol)
        self.behaviours.append(self.timed)

if __name__ == '__main__':

    set_ams('localhost', 5000, debug=False)

    editor = AgenteParticipante(AID('editor'))
    editor.ams = {'name': 'localhost', 'port': 5000}

    msg = ACLMessage(ACLMessage.SUBSCRIBE)
    msg.set_protocol(ACLMessage.FIPA_SUBSCRIBE_PROTOCOL)
    msg.set_content('Pedido de subscriçao')
    msg.add_receiver('editor')

    ass1 = AgenteInitiator(AID('assinante_1'), msg)
    ass1.ams = {'name': 'localhost', 'port': 5000}

    ass2 = AgenteInitiator(AID('assinante_2'), msg)
    ass2.ams = {'name': 'localhost', 'port': 5000}

    agentes = [editor, ass1, ass2]

    start_loop(agentes, gui=True)

```

Referência da API do PADE

3.1 API PADE

Aqui estão todos os módulos que são de interesse para utilização do PADE, tais como o módulo que contém a classe `Agent`, módulo de construção de mensagens e de construção de comportamentos.

3.1.1 Módulo de Implementação de agentes

Este módulo Python faz parte da infraestrutura de comunicação e gerenciamento de agentes que compõem o framework para construção de agentes inteligentes implementado com base na biblioteca para implementação de sistemas distribuídos em Python Twisted

@author: Lucas S Melo

class `pade.core.agent.Agent` (*aid, debug=False*)

A classe `Agente` estabelece as funcionalidades essenciais de um agente como: 1. Conexão com o AMS 2. Configurações iniciais 3. Envio de mensagens 4. Adição de comportamentos 5. metodo abstrato a ser utilizado na implementação dos comportamentos iniciais 6. metodo abstrato a ser utilizado na implementação dos comportamentos dos agentes quando recebem uma mensagem

on_start ()

Metodo que define os comportamentos iniciais de um agente

react (*message*)

Este metodo deve ser SobreEscrito e será executado todas as vezes que o agente em questão receber algum tipo de dado

Parâmetros *message* – `ACLMessage` mensagem recebida

send (*message*)

Envia uma mensagem ACL para os agentes especificados no campo `receivers` da mensagem ACL

send_to_all (*message*)

Envia mensagem de broadcast, ou seja envia mensagem para todos os agentes com registro na tabela de agentes

Parâmetros *message* – mensagem a ser enviada a todos

os agentes registrados na tabela do agente

class `pade.core.agent.AgentFactory` (*aid, ams, debug, react, on_start*)

Esta classe implementa as ações e atributos do protocolo `Agent` sua principal função é armazenar informações importantes ao protocolo de comunicação do agente

buildProtocol (*addr*)

Este metodo inicializa o protocolo Agent

clientConnectionFailed (*connector, reason*)

Este método é chamado quando ocorre uma falha na conexão de um cliente com o servidor

clientConnectionLost (*connector, reason*)

Este método chamado quando a conexão de um cliente com um servidor é perdida

class `pade.core.agent.AgentProtocol` (*fact*)

Esta classe implementa o protocolo que será seguido pelos agentes no processo de comunicação. Esta classe modela os atos de comunicação entre agente e agente AMS, agente e agente Sniffer e entre agentes.

Esta classe não armazena informações permanentes, sendo esta função delegada à classe AgentFactory

connectionLost (*reason*)

Este método executa qualquer coisa quando uma conexão é perdida

Parâmetros *reason* – Identifica o problema na perda de conexão

connectionMade ()

Este método é executado sempre que uma conexão é executada entre um agente no modo cliente e um agente no modo servidor

lineReceived (*line*)

Este método é executado sempre que uma nova mensagem é recebida pelo agente, tanto no modo cliente quanto no modo servidor

Parâmetros *line* – mensagem recebida pelo agente

sniffer_message (*message*)

Este método trata a mensagem enviada pelo agente Sniffer e cria uma mensagem de resposta ao agente Sniffer

Parâmetros *message* – mensagem recebida pelo agente, enviada pelo agente Sniffer

3.1.2 Módulo de Definição do AMS

Neste módulo estão definidos os comportamentos do agente AMS.

class `pade.core.ams.AgentManagementFactory` (*port, debug*)

Esta classe implementa as ações e atributos do protocolo AMS sua principal função é armazenar informações importantes ao protocolo de comunicação do agente AMS

broadcast_message (*message*)

Este método é utilizado para o envio de mensagens de atualização da tabela de agentes ativos sempre que um novo agente é conectado.

connection_test_send ()

Este método é executado ciclicamente com o objetivo de verificar se os agentes estão conectados

class `pade.core.ams.AgentManagementProtocol` (*fact*)

Esta classe implementa os comportamentos de um agente AMS

A principal funcionalidade do AMS é registrar todos os agentes que estão conectados ao sistema e atualizar a tabela de agentes de cada um deles sempre que um novo agente se conectar.

connectionLost (*reason*)

Este método é executado sempre que uma conexão é perdida com o agente AMS

connectionMade ()

Este método é executado sempre que uma conexão é realizada com o agente AMS

handle_identif (*aid*)

Este método é utilizado para cadastrar o agente que esta se identificando na tabela de agentes ativos.

lineReceived (*line*)

Quando uma mensagem é enviada ao AMS este método é executado. Quando em fase de identificação, o AMS registra o agente em sua tabela de agentes ativos

3.1.3 Módulo de Definição do Sniffer

Neste módulo estão definidos os comportamentos do agente Sniffer.

class pade.core.sniffer.**Sniffer** (*fact*)

Esta classe implementa o agente Sniffer que tem o objetivo de enviar mensagens para os agentes ativos e exibir suas mensagens por meio de uma GUI. Protocolo do Agente GUI paramonitoramento dos agentes e das mensagens dos agentes

connectionMade ()

Este método é executado sempre que uma conexão é executada entre um cliente e um servidor

lineReceived (*line*)

Este método é executado sempre que uma mensagem é recebida pelo agente Sniffer

show_messages (*messages*)

Este método exibe a lista de mensagens que estão em na lista de mensagens do agente selecionado

class pade.core.sniffer.**SnifferFactory** (*aid, ams, ui*)

Esta classe implementa o factory do protocolo do agente Sniffer

3.1.4 Módulo de criação e manipulação de mensagens FIPA-ACL

Este módulo contém a classe que implementa um objeto do tipo ACLMessage, que é a mensagem padronizada pela FIPA utilizada na troca de mensagens entre os agentes.

class pade.acl.messages.**ACLMessage** (*performative=None*)

Classe que implementa uma mensagem do tipo ACLMessage

add_receiver (*aid*)

Método utilizado para adicionar receptores para a mensagem que está sendo montada

Parâmetros *aid* – objeto do tipo AID que identifica o agente que receberá a mensagem

add_reply_to (*aid*)

Método utilizado para adicionar agentes que devem receber a resposta desta mensagem

Parâmetros *aid* – objeto do tipo AID que identifica o agente que receberá a resposta desta mensagem

create_reply ()

Creates a reply for the message Duplicates all the message structures exchanges the ‘from’ AID with the ‘to’ AID

set_performative (*performative*)

Método que seta o parâmetro Performative da mensagem ACL

Parâmetros *performative* – tipo da performative da mensagem, podendo ser qualquer um dos atributos da classe ACLMessage.

set_sender (*aid*)

Método utilizado para definir o agente que irá enviar a mensagem

Parâmetros `aid` – objeto do tipo AID que identifica o agente que enviará a mensagem

3.1.5 Módulo de protocolos

Este modulo implementa os protocolos padronizados pela FIPA: 1. FipaRequestProtocol 2. FipaContractNetProtocol 3. FIPA_Subscribe_Protocol

class `pade.behaviours.protocols.Behaviour` (*agent*)

Classe que declara os metodos essenciais de um comportamento, todo comportamento deve herdar esta classe.

execute (*message*)

Executa o comportamento propriamente dito do protocolo para cada tipo de mensagem recebida

on_start ()

Executado sempre que o protocolo e Inicializado

timed_behaviour ()

Utilizado quando o protocolo implementado possui restrições de tempo, como por exemplo utilização de timeout

class `pade.behaviours.protocols.FipaContractNetProtocol` (*agent*, *message=None*, *is_initiator=True*)

Classe que implementa o protocolo FipaContractNetProtocol herdando a classe Behaviour e implementando seus métodos

execute (*message*)

Este método sobrescreve o metodo execute da classe Behaviour. Nele esta implementado a seleção de qual método será executado logo após o recebimento de uma mensagem

Parâmetros `message` – Mensagem FIPA-ACL

execute_on_timeout ()

Este método executa o método handle_all_proposes caso nem todas as mensagens FIPA_CFP enviadas pelo agente obtenham resposta

handle_accept_propose (*message*)

Este método deve ser sobrescrito quando na implementação do protocolo, sendo executado sempre que o agente receber uma mensagem do tipo FIPA_ACCEPT_PROPOSE

Parâmetros `message` – Mensagem FIPA-ACL

handle_all_proposes (*proposes*)

Este método deve ser sobrescrito quando na implementação do protocolo, sendo executado em um dos dois casos: * A quantidade de respostas recebidas for igual a quantidade de solicitações recebidas * O timeout for atingido

Parâmetros `proposes` – lista com as respostas das solicitações envidas pelo Initiator

handle_cfp (*message*)

Este método deve ser sobrescrito quando na implementação do protocolo, sendo executado sempre que o agente receber uma mensagem do tipo FIPA_CFP

Parâmetros `message` – Mensagem FIPA-ACL

handle_inform (*message*)

Este método deve ser sobrescrito quando na implementação do protocolo, sendo executado sempre que o agente receber uma mensagem do tipo FIPA_INFORM

Parâmetros `message` – Mensagem FIPA-ACL

handle_propose (*message*)

Este método deve ser sobrescrito quando na implementação do protocolo, sendo executado sempre que o agente receber uma mensagem do tipo FIPA_PROPOSE

Parâmetros *message* – Mensagem FIPA-ACL

handle_refuse (*message*)

Este método deve ser sobrescrito quando na implementação do protocolo, sendo executado sempre que o agente receber uma mensagem do tipo FIPA_REFUSE

Parâmetros *message* – Mensagem FIPA-ACL

handle_reject_propose (*message*)

Este método deve ser sobrescrito quando na implementação do protocolo, sendo executado sempre que o agente receber uma mensagem do tipo FIPA_REJECT_PROPOSE

Parâmetros *message* – Mensagem FIPA-ACL

on_start ()

Este método sobrescreve o método on_start da classe Behaviour e implementa configurações adicionais à inicialização do protocolo FipaContractNetProtocol

timed_behaviour ()

Este método é implementado sempre que o protocolo a ser implementado necessitar de restrições temporais, como é o caso do FipaContractNetProtocol. Neste caso ele faz uso do método callLater do twisted que recebe como parâmetro um método e o atraso de tempo para que este método seja executado

```
class pade.behaviours.protocols.FipaRequestProtocol (agent, message=None,  
                                                    is_initiator=True)
```

Classe que implementa o protocolo FipaRequestProtocol herdando a classe Behaviour e implementando seus métodos

execute (*message*)

Este método sobrescreve o método execute da classe Behaviour. Nele está implementado a seleção de qual método será executado logo após o recebimento de uma mensagem

Parâmetros *message* – Mensagem FIPA-ACL

handle_agree (*message*)

Este método deve ser sobrescrito quando na implementação do protocolo, sendo executado sempre que o agente receber uma mensagem do tipo FIPA_AGREE

Parâmetros *message* – Mensagem FIPA-ACL

handle_failure (*message*)

Este método deve ser sobrescrito quando na implementação do protocolo, sendo executado sempre que o agente receber uma mensagem do tipo FIPA_FAILURE

Parâmetros *message* – Mensagem FIPA-ACL

handle_inform (*message*)

Este método deve ser sobrescrito quando na implementação do protocolo, sendo executado sempre que o agente receber uma mensagem do tipo FIPA_INFORM

:param *message* : Mensagem FIPA-ACL

handle_refuse (*message*)

Este método deve ser sobrescrito quando na implementação do protocolo, sendo executado sempre que o agente receber uma mensagem do tipo FIPA_REFUSE

Parâmetros *message* – Mensagem FIPA-ACL

handle_request (*message*)

Este método deve ser sobrescrito quando na implementação do protocolo, sendo executado sempre que o agente receber uma mensagem do tipo FIPA_REQUEST

Parâmetros *message* – Mensagem FIPA-ACL

on_start ()

Este método sobrescreve o método on_start da classe Behaviour e implementa configurações adicionais à inicialização do protocolo FipaRequestProtocol

class pade.behaviours.protocols.**FipaSubscribeProtocol** (*agent*, *message=None*,
is_initiator=True)

Classe que implementa o protocolo FipaSubscribeProtocol herdando a classe Behaviour e implementando seus métodos

deregister (*aid*)

execute (*message*)

Este método sobrescreve o método execute da classe Behaviour. Nele está implementado a seleção de qual método será executado logo após o recebimento de uma mensagem

Parâmetros *message* – Mensagem FIPA-ACL

handle_agree (*message*)

handle_cancel (*message*)

handle_inform (*message*)

handle_refuse (*message*)

handle_subscribe (*message*)

notify (*message*)

on_start ()

Este método sobrescreve o método on_start da classe Behaviour e implementa configurações adicionais à inicialização do protocolo FipaContractNetProtocol

register (*aid*)

class pade.behaviours.protocols.**TimedBehaviour** (*agent*, *time*)

Classe para implementação de comportamentos temporizados

on_start ()

Este método sobrescreve o método on_start da classe Behaviour e implementa configurações adicionais à inicialização do comportamento TimedBehaviour

on_time ()

Este método executa o método handle_all_proposes caso nem todas as mensagens FIPA_CFP enviadas pelo agente obtenham resposta

timed_behaviour ()

Este método é implementado sempre que o comportamento a ser implementado necessitar de restrições temporais. Neste caso ele faz uso do método callLater do twisted que recebe como parâmetro um método e o atraso de tempo para que este método seja executado

p

- `pade.acl.messages`, [25](#)
- `pade.behaviours.protocols`, [26](#)
- `pade.core.agent`, [23](#)
- `pade.core.ams`, [24](#)
- `pade.core.sniffer`, [25](#)

A

ACLMessage (classe em `pade.acl.messages`), 25
 add_receiver() (método `pade.acl.messages.ACLMessage`), 25
 add_reply_to() (método `pade.acl.messages.ACLMessage`), 25
 Agent (classe em `pade.core.agent`), 23
 AgentFactory (classe em `pade.core.agent`), 23
 AgentManagementFactory (classe em `pade.core.ams`), 24
 AgentManagementProtocol (classe em `pade.core.ams`), 24
 AgentProtocol (classe em `pade.core.agent`), 24

B

Behaviour (classe em `pade.behaviours.protocols`), 26
 broadcast_message() (método `pade.core.ams.AgentManagementFactory`), 24
 buildProtocol() (método `pade.core.agent.AgentFactory`), 23

C

clientConnectionFailed() (método `pade.core.agent.AgentFactory`), 24
 clientConnectionLost() (método `pade.core.agent.AgentFactory`), 24
 connection_test_send() (método `pade.core.ams.AgentManagementFactory`), 24
 connectionLost() (método `pade.core.agent.AgentProtocol`), 24
 connectionLost() (método `pade.core.ams.AgentManagementProtocol`), 24
 connectionMade() (método `pade.core.agent.AgentProtocol`), 24
 connectionMade() (método `pade.core.ams.AgentManagementProtocol`), 24
 connectionMade() (método `pade.core.sniffer.Sniffer`), 25
 create_reply() (método `pade.acl.messages.ACLMessage`), 25

D

deregister() (método `pade.behaviours.protocols.FipaSubscribeProtocol`), 28

E

execute() (método `pade.behaviours.protocols.Behaviour`), 26
 execute() (método `pade.behaviours.protocols.FipaContractNetProtocol`), 26
 execute() (método `pade.behaviours.protocols.FipaRequestProtocol`), 27
 execute() (método `pade.behaviours.protocols.FipaSubscribeProtocol`), 28
 execute_on_timeout() (método `pade.behaviours.protocols.FipaContractNetProtocol`), 26

F

FipaContractNetProtocol (classe em `pade.behaviours.protocols`), 26
 FipaRequestProtocol (classe em `pade.behaviours.protocols`), 27
 FipaSubscribeProtocol (classe em `pade.behaviours.protocols`), 28

H

handle_accept_propose() (método `pade.behaviours.protocols.FipaContractNetProtocol`), 26
 handle_agree() (método `pade.behaviours.protocols.FipaRequestProtocol`), 27
 handle_agree() (método `pade.behaviours.protocols.FipaSubscribeProtocol`), 28
 handle_all_proposes() (método `pade.behaviours.protocols.FipaContractNetProtocol`), 26
 handle_cancel() (método `pade.behaviours.protocols.FipaSubscribeProtocol`), 28

[handle_cfp\(\)](#) (método [pade.behaviours.protocols.FipaContractNetProtocol](#)), [26](#)
[handle_cfp\(\)](#) (método [pade.behaviours.protocols.FipaRequestProtocol](#)), [28](#)
[handle_failure\(\)](#) (método [pade.behaviours.protocols.FipaRequestProtocol](#)), [27](#)
[handle_identif\(\)](#) (método [pade.core.ams.AgentManagementProtocol](#)), [24](#)
[handle_inform\(\)](#) (método [pade.behaviours.protocols.FipaContractNetProtocol](#)), [26](#)
[handle_inform\(\)](#) (método [pade.behaviours.protocols.FipaRequestProtocol](#)), [27](#)
[handle_inform\(\)](#) (método [pade.behaviours.protocols.FipaSubscribeProtocol](#)), [28](#)
[handle_propose\(\)](#) (método [pade.behaviours.protocols.FipaContractNetProtocol](#)), [26](#)
[handle_refuse\(\)](#) (método [pade.behaviours.protocols.FipaContractNetProtocol](#)), [27](#)
[handle_refuse\(\)](#) (método [pade.behaviours.protocols.FipaRequestProtocol](#)), [27](#)
[handle_refuse\(\)](#) (método [pade.behaviours.protocols.FipaSubscribeProtocol](#)), [28](#)
[handle_reject_propose\(\)](#) (método [pade.behaviours.protocols.FipaContractNetProtocol](#)), [27](#)
[handle_request\(\)](#) (método [pade.behaviours.protocols.FipaRequestProtocol](#)), [27](#)
[handle_subscribe\(\)](#) (método [pade.behaviours.protocols.FipaSubscribeProtocol](#)), [28](#)

L
[lineReceived\(\)](#) (método [pade.core.agent.AgentProtocol](#)), [24](#)
[lineReceived\(\)](#) (método [pade.core.ams.AgentManagementProtocol](#)), [25](#)
[lineReceived\(\)](#) (método [pade.core.sniffer.Sniffer](#)), [25](#)

N
[notify\(\)](#) (método [pade.behaviours.protocols.FipaSubscribeProtocol](#)), [28](#)

O
[on_start\(\)](#) (método [pade.behaviours.protocols.Behaviour](#)), [26](#)
[on_start\(\)](#) (método [pade.behaviours.protocols.FipaContractNetProtocol](#)), [27](#)

P

[pade.acl.messages](#) (módulo), [25](#)
[pade.behaviours.protocols](#) (módulo), [26](#)
[pade.core.agent](#) (módulo), [23](#)
[pade.core.ams](#) (módulo), [24](#)
[pade.core.sniffer](#) (módulo), [25](#)

R

[react\(\)](#) (método [pade.core.agent.Agent](#)), [23](#)
[register\(\)](#) (método [pade.behaviours.protocols.FipaSubscribeProtocol](#)), [28](#)

S

[send\(\)](#) (método [pade.core.agent.Agent](#)), [23](#)
[send_to_all\(\)](#) (método [pade.core.agent.Agent](#)), [23](#)
[set_performative\(\)](#) (método [pade.acl.messages.ACLMessage](#)), [25](#)
[set_sender\(\)](#) (método [pade.acl.messages.ACLMessage](#)), [25](#)
[show_messages\(\)](#) (método [pade.core.sniffer.Sniffer](#)), [25](#)
[Sniffer](#) (classe em [pade.core.sniffer](#)), [25](#)
[sniffer_message\(\)](#) (método [pade.core.agent.AgentProtocol](#)), [24](#)
[SnifferFactory](#) (classe em [pade.core.sniffer](#)), [25](#)

T

[timed_behaviour\(\)](#) (método [pade.behaviours.protocols.Behaviour](#)), [26](#)
[timed_behaviour\(\)](#) (método [pade.behaviours.protocols.FipaContractNetProtocol](#)), [27](#)
[timed_behaviour\(\)](#) (método [pade.behaviours.protocols.TimedBehaviour](#)), [28](#)
[TimedBehaviour](#) (classe em [pade.behaviours.protocols](#)), [28](#)