Universidade Federal de Viçosa

Nome: João Victor Magalhães Souza. **Matrícula:** 3483
Lucas Ranieri Oliveira Martins **Matrícula:** 3479
Disciplina: CCF 355 - Sistemas Distribuídos e Paralelos.
Professora: Dra. Thais Regina de Moura.
Ideia: Álbum de figurinhas de Linguagens de Programação.

Programming Language Collection
Implementação utilizando *middleware*

Modificações

Como fizemos todo o processo da Parte 3 (*Sockets*) em Python e, na nossa visão, conseguimos construir um Sistema Distribuído satisfatoriamente funcional e modular, reaproveitamos totalmente os códigos e funções produzidas. A grande diferença entre essa reimplementação e a implementação anterior é que, na forma de comunicação anterior, nós definimos o formato da mensagem com cabeçalho e corpo e, nesta parte, com a utilização do *middleware RMI* **Pyro**, podemos apenas invocar os referidos métodos.

Como introduzido anteriormente, projetamos na parte de *Sockets* uma arquitetura de Cliente e também de servidor o mais modular possível, com métodos bem definidos e não acoplados. Dessa forma, foi necessário mudar apenas as instruções de requisição para tais funções, isto é, como elas são chamadas, visto que agora não são via estrutura de **cabeçalho + mensagem** e sim via **invocação remota.** Portanto, adiantamos que no processo de reimplementação deste Sistema Distribuído não houveram grandes complicações.

Antes:

Figura 1 - Processo com a utilização da API de *Sockets.*

Depois:

Figura 2 - Implementação via *middleware RMI* Pyro.

Arquitetura do Cliente

Para o Cliente, mantemos o mesmo esquema de classe e a mesma simulação das funcionalidades de uma UI (só que via terminal) da parte anterior. Entretanto, o fator modificado aqui foi apenas a parte de estabelecimento de conexão e comunicação (esta será abordada exclusivamente no tópico "**Comunicação**", posteriormente).

Como dito anteriormente, utilizamos o Pyro para poder acessar remotamente os métodos do servidor. Entretanto, precisamos estabelecer, primordialmente, a comunicação com ele. Diferentemente da API de *Sockets,* o Pyro usa uma estrutura de endereço chamada de **URI (Universal Resource Identifier),** que basicamente requer um endereço IP e porta para acesso, semelhante à API de *Sockets.*

```
class Cliente:

    def __init__(self, ip):
        ######## Configurações do servidor e da conexão. ######################
        ipAddressServer = ip
        self.connection = Pyro4.core.Proxy(
            'PYRO:Servidor@' + ipAddressServer + ':9090')
        ########################################################
```

Figura 3 - Instanciação das configurações do Server no Cliente.

Arquitetura do Servidor

Referente ao Servidor, mantemos toda a estrutura e formas de manipulação dos dados no Banco de Dados. As únicas modificações se referem exclusivamente à instanciação do Server (forma própria do Pyro) que é diferente da API de *Sockets.* Como evidenciado na imagem abaixo, o Servidor é iniciado no *localhost (127.0.0.1)* na porta 9090.

```
def startServer(self):
    try:

        Pyro4.Daemon.serveSimple({  # Definindo as configurações do servidor
            Server: 'Servidor',
        }, host="127.0.0.1", port=9090, ns=False, verbose=True)
        """
            Ele cria um loop de serviço automaticamente.
        """
```

Figura 4 - Configurações do Server.

Além disso, agora nós não precisamos mais de uma "**unidade decodificadora**" para poder analisar as mensagens advindas do Cliente, visto que o Cliente agora acessa remotamente os métodos convencionados. Esses métodos podem ser convencionados utilizando-se uma marcação antes de sua definição, como será mostrado a seguir.

```
@Pyro4.expose
def cadastro(self, coins, nickname, password, nome, email):
```

Figura 5 - Exemplo de definição de Método Remotamente Acessível.

Como é possível observar na Figura 4, os métodos remotos são definidos à partir da marcação ***Pyro4.expose*.** Sendo assim, estamos dizendo, no exemplo da figura acima, que o método Cadastro, cujos parâmetros são: *coins, nickname, password, nome* e *email* pode ser acessado remotamente via Cliente. Como é um método remoto, o retorno desse método é passado diretamente para o Cliente pelo Server.

Comunicação

Conseguintemente, a comunicação que une as duas entidades previamente estabelecidas, se dá de uma forma extremamente mais simples quando comparada com o modelo de comunicação da API de *Sockets*. Isso se deve ao fato de, como mencionado, com o *middleware RMI* Pyro nós podemos acessar remotamente os métodos que o Servidor especificou como remotamente acessíveis e, dessa forma, não é preciso pensar na arquitetura de troca de mensagens em baixo nível como na API de *Sockets.*

Primeiramente, através da simulação de UI via terminal, pedimos ao usuário que escolha a funcionalidade que quer executar. Após isso, fazemos uma chamada ao método do Servidor que é responsável por fazer as consultas e modificações no Banco de Dados acerca da requisição especificada.

```
if (escolha == "1"):
    print(
        "#####################################################################")
    print(
        "#                    INFORMAÇÕES DE CADASTRO                        #")
    print(
        "#####################################################################")
    nick = input("Digite seu nickname: ")
    senha = input("Digite sua senha: ")
    nome = input("Digite seu nome: ")
    email = input("Digite seu email: ")
    """
        Agora a gente chama diretamente o método.
    """
    print(self.connection.cadastro(
        "200", nick, senha, nome, email))
    print("\n\n")
```

Figura 6 - Estabelecimento da conexão do Cliente com o Server.

Na imagem acima, é possível observar um exemplo acerca da instrução de Cadastro de usuário: primeiramente nós levantamos as informações acerca desse usuário como *nickname, senha, nome* e *e-mail* e, posteriormente, requisitamos através da invocação remota de métodos do servidor o método “*cadastro*” passando como parâmetro *200 coins (*quantidade inicial padrão de *coins* para um novo usuário), *nickname, senha, nome* e *e-mail*. Depois disso, o método no Servidor processará as informações e retornará um resultado dizendo se é possível ou não realizar o cadastro a partir das credenciais informadas.

Conclusões

Por fim, gostaríamos de levantar alguns pontos interessantes na nossa visão grupal e que são úteis para fazermos uma análise ponto a ponto entre a implementação utilizando a API de *Sockets* e o Pyro:

- **Redigibilidade:** há uma diferença gritante com a utilização do *middleware*: aqui, nós apenas invocamos os métodos remotos. Com a utilização da API de *Sockets,* nós precisávamos sempre especificar cabeçalhos e corpo da mensagem, além de codificar e decodificar tal mensagem de/para bytes sempre. Logo, no quesito de redigibilidade, o Pyro possui um grande diferencial ;
- **Curva de Aprendizado:** Em ambas implementações, depois que a conexão é estabelecida, cabe ao programador decidir como enviará e receberá as informações. Na API de *Sockets* temos diversos fatores envolvidos como a estrutura da mensagem, codificar a mensagem para bytes no cliente, enviá-la, decodificar a mensagem no servidor, entender a mensagem, processar, montar outra mensagem de resposta, codificar ela para bytes e enviar de volta para o cliente. Já utilizando o *middleware*, nós apenas fazemos a chamada do método remoto e passamos os parâmetros necessários. Logo, na nossa visão, a curva de aprendizado inicial é bem pária mas, nos processos posteriores, a utilização de mais baixo nível (*Sockets*) demanda mais recursos, enquanto a implementação em mais alto nível (*Middleware*) fornece uma maior praticidade.

As nossas dificuldades nesta etapa de utilização de um *middlewar*e RMI se deram principalmente na estrutura de conexão das entidades. Isso se deve ao fato de que cada modelo possui um formato próprio de conexão e, sendo assim, precisamos ir um pouco mais a fundo em termos de implementação. Em relação a utilização do Pyro, não tivemos grandes dificuldades uma vez que a tecnologia já nos fornece uma enorme praticidade e, além disso, o grupo se encontra perfeitamente tangenciado aos conteúdos sobre Invocação Remota vistos na disciplina.

Logo, concluímos que a utilização do *middlewar*e RMI Pyro nos dá uma enorme praticidade na implementação de um Sistema Distribuído, visto que ele fornece uma forma de comunicação que adota com maior intensidade o paradigma “direto ao ponto”. Além disso, na nossa visão, é singularmente importante entender como as ferramentas operam em mais baixo nível mas, ao mesmo tempo, tentar evitá-las visto que à medida em que descemos cada vez mais o nível de abstração precisamos codificar mais para alcançarmos um mesmo objetivo que pode ser feito em poucas linhas em mais alto nível.

Referências

DE JONG, Irmen. ***Python Remote Objects - 4.81.*** Disponível em: <https://pyro4.readthedocs.io/en/stable/>. Acesso em: 01/10/2021.