Revista de Informática Teórica e Aplicada - RITA - ISSN 2175-2745
Vol. 27, Num. 2 (2020) 107-115

RESEARCH ARTICLE

Mosaik and PADE: Multiagents and Co-simulation for smart grids modeling

Mosaik e PADE: Sistemas Multiagentes e Co-Simulação para Modelagem de Redes Elétricas Inteligentes

Lucas S. Melo[1]*, Filipe Saraiva[2], Ruth P. S. Leão[1], Raimundo F. Sampaio[1], Giovanni C. Barroso[1]

Resumo: This paper describes the integration process between two tools in order to perform co-simulation for representation and analysis of dynamic environments in the context of smart grids. The integrated tools are Mosaik, a software to co-simulation management, and PADE, a software to multi-agent systems development. As a study case for demonstrate the integration, a scenary was utilized composed of a low voltage electricity distribution grid with 37 load bus, 20 photo-voltaic distributed generations, randomly connected to load bus, as well as, 20 PADE agents associated to distributed generation, modeling the behavior of electricity storage systems. The simulation results show the integration happening and demonstrate how useful is to model the dynamics of distributed electric resources with multi-agent systems.
Keywords: multi-agent system — co-simulation — smart grids — distributed energetic resources

Resumo: Este artigo descreve o processo de integração entre duas ferramentas para realização de co-simulação na representação e análise de ambientes dinâmicos no contexto de redes elétricas inteligentes. As ferramentas integradas são o software para gerenciamento de co-simulação Mosaik e o software para desenvolvimento de sistemas multiagentes PADE. Como estudo de caso para demonstração da integração foi utilizado um cenário composto de uma rede de distribuição de energia elétrica em baixa tensão com 37 barras de carga, 20 gerações fotovoltaicas distribuídas aleatoriamente nas barras de carga, e 20 agentes PADE associados às gerações distribuídas que modelam o comportamento de um sistema de armazenamento de energia. Os resultados obtidos da simulação comprovam a eficácia da integração e demonstram a utilidade de sistemas multiagentes em modelar comportamentos e dinâmicas em redes elétricas inteligentes.
Palavras-Chave: sistemas multiagentes — co-simulação — redes elétricas inteligentes — recursos energéticos distribuídos

[1] *Departamento de Engenharia Elétrica, Universidade Federal do Ceará, Brazil*
[2] *Faculdade de Computação - Instituto de Ciências Exatas e Naturais, Universidade Federal do Pará, Brazil*
***Corresponding author**: lucassmelo@dee.ufc.br
DOI: http://dx.doi.org/10.22456/2175-2745.95095 • **Received:** 31/07/2019 • **Accepted:** 12/03/2020
CC BY-NC-ND 4.0 - This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

1. Introdução

O sistema elétrico atual vem passando por uma quebra de paradigma devido à inserção de novas tecnologias na rede elétrica, que têm como objetivo produzir energia limpa e renovável, melhorar a qualidade e a confiabilidade da rede elétrica e aumentar sua eficiencia. A inserção de recursos energéticos distribuídos (RED) pode ser apontada como uma das grandes responsáveis pelas mudanças e alterações pelas quais vem passando a rede elétrica [1]. RED são dispositivos conectados à rede elétrica, principalmente em baixa e média tensão, que têm capacidade de gerar energia e/ou armazená-la, por exemplo, mini e micro gerações e sistemas de armazenamento tais como as baterias.

Em se tratando de micro e mini geração distribuída, a tendência atual é que estas sejam do tipo solar fotovoltaica e/ou eólica, ou seja, fontes não despacháveis e que entregam energia à rede de acordo com a disponibilidade de recursos naturais, que por natureza são intermitentes (luz solar e ventos). Esse comportamento característico das fontes solar e eólica, conectadas à rede elétrica, geram grande imprevisibilidade na produção de energia podendo acarretar desequilíbrios entre geração e consumo, o que prejudica a estabilidade do sistema elétrico [2], assim como impactam nos principais indicadores de qualidade de energia, relacionados à nível de tensão, sobrecarga de condutores e transformadores, varia-

ções de frequência, aumento das rampas de carga, entre outros fenômenos indesejáveis do ponto de vista operacional.

Com a crescente evolução na eficiência e redução dos custos de sistemas de armazenamento de energia (SAE), que vem sendo acelerada pelo aumento da participação dos carros elétricos na frota de veículos das grandes cidades, um importante e desejado suporte no controle e operação de RED está sendo inserido nas simulações e estudos que tratam da integração de RED na rede elétrica. Os SAE são essenciais para realizar o equilíbrio entre geração e carga, armazenando o excesso de energia em momentos de geração excedente e entregando a energia armazenada em momentos de geração deficitária, além de poderem auxiliar na implementação de lógicas de controle que visam incentivar ou desestimular o consumo de energia elétrica, auxiliando na operação da rede e postergando investimentos.

Entre os principais fatores necessários para que a transição de uma rede elétrica tradicional para uma rede elétrica moderna aconteça está a utilização de gerenciamento e operação da rede elétrica baseados em sistemas distribuídos e que possibilitem a integração dos diversos RED sem afetar critérios de qualidade e eficiência.

Para isso, é necessário que testes e simulações possam ser executados antes da implementação em dispositivos reais conectados à rede elétrica. Uma solução bastante utilizada para esse tipo de análise é o esquema de co-simulação que consiste da integração de vários processos de simulação coordenados entre si em uma mesma base de tempo e com possibilidade de envio e recebimento de informações entre as diferentes instâncias de simulação.

Muitos trabalhos adotam esse esquema para integrar os comportamentos de diferentes dispositivos, simulados em plataformas diversas, em um processo de simulação global. Em [3] os autores propõem a utilização da técnica de high-level architecture (HLA) na integração entre simuladores e SMA para simulação de redes elétricas inteligentes. Em [4] é mostrada a integração entre agentes desenvolvidos em JADE e um ambiente de co-simulação de redes elétricas inteligentes. Em [5] a integração entre o gerenciador de co-simulação Mosaik e o simulador de ambientes de redes de comunicação OMNet++ é apresentada e apontada como essencial para análise do comportamento dos dispositivos comunicantes mediante diferentes cenários de utilização da rede de comunicação.

Este trabalho descreve a integração entre o framework para desenvolvimento de sistemas multiagentes PADE e o framework para realização de co-simulação Mosaik, ambos software livre e desenvolvidos em linguagem de programação Python. Para demonstrar os resultados dessa integração entre softwares um exemplo de aplicação na modelagem da dinâmica de RED em uma rede de distribuição de energia elétrica de baixa tensão é apresentado.

As próximas seções estão organizadas da seguinte forma: a Seção 2 descreve alguns aspectos sobre a dinâmica dos RED, a Seção 3 descreve o processo de co-simulação e uma alternativa para integração de SMA nesse processo, a Seção 4 descreve as APIs do Mosaik, explicando como se dá o processo de integração com a plataforma; a Seção 5 apresenta de que maneira foi realizada a integração com o PADE, sendo a principal contribuição desse artigo; em seguida, a Seção 6 apresenta o estudo de caso proposto, executando sobre a integração do PADE com o Mosaik. Ao final, a Seção 7 conclui o artigo com alguns comentários sobre o estudo.

2. Recursos Energéticos Distribuídos e Co-simulação

Um dos principais avanços tecnológicos que estão guiando a modernização do sistema elétrico, são os Recursos Energéticos Distribuídos (RED). De acordo com [1] é possível dividir os RED em três grandes classes:

- Resposta a demanda: dispositivos ou instalação elétrica capazes de alterarem seu padrão de consumo energético, em tempo real ou próximo do real, em reação a sinais de preço dos operadores de mercado ou da rede elétrica. Essa alteração do consumo pode ser atigida por meio do corte de cargas ou do deslocamento no tempo do consumo de energia elétrica;

- Geração distribuída: unidades de geração de energia elétrica conectadas diretamente à rede de distribuição (de média ou de baixa tensão). Exemplos comuns e cada vez mais populares são os painéis fotovoltáicos e os aerogeradores;

- Armazenamento distribuído: são dispositivos conectados à rede elétrica capazes de armazenar energia, em suas diferentes formas, e entregá-la novamente à rede de acordo com programação pré-estabelecida.

A dinâmica dos RED altera muitos princípios de operação do sistema elétrico de potência tal como concebido atualmente: um número relativamente pequeno de grandes centrais geradoras, conectadas a centros consumidores por meio de linhas de transmissão de energia que cobrem grandes distâncias territoriais [1].

Quanto maior a inserção de RED no sistema elétrico, maior será o impacto da característica de operação distribuída desses dispositivos em detrimento da operação centralizada do modelo tradicional. Analisando um cenário com grande penetração de RED conectados à rede elétrica, percebe-se a complexidade de coordenar e controlar a injeção/absorção de energia ativa/reativa, reagindo às inúmeras flutuações tanto no lado da demanda quanto no lado da geração [6].

Para que esse cenário se concretize alguns fatores são essenciais:

1. Alterações na legislação e no modelo de estruturação no setor elétrico devem ser implementadas;

2. Sistemas de tecnologia da informação que permita comunicação rápida e bidirecional entre as entidades que

participarão do esforço de prover estabilidade e qualidade no fornecimento de energia elétrica;

3. Utilização de esquemas de gerenciamento e operação da rede elétrica baseados em sistemas distribuídos e que possibilitem a integração dos diversos RED sem afetar critérios de qualidade e eficiência.

Em muitos casos, a realização de co-simulação para análise de esquemas de controle e de operação em ambientes de redes elétricas inteligentes (REI) é indispensável uma vez que a grande quantidade de comportamentos e entidades simuladas em diferentes plataformas de software torna o processo de integração bastante complexo.

De acordo com [6] o processo de co-simulação pode ser definido como um tipo específico de simulação utilizado principalmente para analizar sistemas integrados em ambientes de redes elétricas inteligentes em que há grande número de entidades a serem consideradas para que um cenário realista possa ser analisado. São de especial interesse os seguintes tópicos envolvendo as ferramentas e os tipos de co-simulação:

- modelos de simulação;
- *solvers* de simulação;
- infraestrutura de execução;
- sincronização das simulações;

Os modelos de simulação estão relacionados às possíveis maneiras de se descrever os sistemas a serem simulados, por exemplo, sistemas elétricos são geralmente descritos por um conjunto de equações algébricas e equações diferenciais, enquanto sistemas de comunicação são geralmente simulados por meio de sistemas de eventos discretos.

Os *solvers* estão presentes tanto em simulações de sistemas elétricos quanto de sistemas de comunicação, sendo os *solvers* de sistemas elétricos geralmente implementações de métodos númericos. Já os *solvers* de sistemas de comunicação são implementados como loops que tratam eventos assim que estes são disparados.

A infraestrutura de execução diz respeito ao mecanismo de coordenação e troca de mensagens entre os simuladores, podendo ser centralizada ou distribuída. Quando distribuída é possível que os processos a serem executados durante a co-simulação ocorram em hardware paralelo integrado, ou seja com latência de microssegundos entre as trocas de mensagens ou em hardware distribuído com troca de mensagens com latência de alguns milissegundos.

No quesito tipos de sincronização de simulação, é possível categorizar os esquemas de co-simulação em três tipos diferentes de sincronização: *sincronização de barreira*, quando o próximo passo da simulação só é dado quando todos os simuladores envolvidos terminam de processar suas análises; *sincronização ótima*, que permite a identificação e correção do processamento de eventos fora de ordem em tempo de execução e *sincronização baseada em web* que faz uso de *web services* para prover a sincronização e a interoperabilidade entre os diferentes simuladores.

A Figura 1, baseada em [6], mostra um esquemático de classificação dos possíveis tipos de co-simulação adotados em diferentes trabalhos pela comunidade acadêmica:

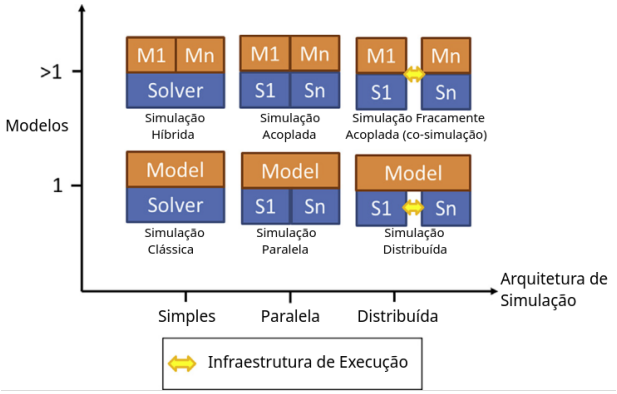

Figura 1. Tipos de co-simulação adotados em diferentes trabalhos [6].

Coordenar os diferentes processos de simulação é uma tarefa complexa que exige a coordenação entre os simuladores, além do gerenciamento das dependências de dados interrelacionados.

3. PADE e Mosaik: SMA integrado ao ambiente de co-simulação

Considerada uma linguagem de fácil aprendizagem e com muitos recursos para computação numérica, Python vem sendo cada vez mais utilizada pela comunidade acadêmica em diversas áreas de conhecimento [7]. Algumas ferramentas voltadas à realização de análises científicas vêm sendo desenvolvidas em Python. No contexto de redes elétricas inteligentes e de co-simulações, o *framework* Mosaik [8] é uma ferramenta de especial interesse. O Mosaik é desenvolvido por pesquisadores do instituto alemão de pesquisa e desenvolvimento OFFIS vinculado à universidade Carl von Ossietzky em Oldenburg.

A principal funcionalidade provida pelo Mosaik é a integração de processos de simulação existentes em um contexto comum, com a finalidade de representar a dinâmica não só dos sistemas elétricos mas de cada um dos componentes que integram o ambiente das redes elétricas inteligentes, como infraestrutura de comunicação, dispositivos de controle, mecanismos de mercados, entre outros [9]. Ou seja, cada um dos simuladores são executados separadamente, com seu próprio loop de eventos. Mosaik sincroniza os processos de simulação e gerencia a troca de dados entre eles, provendo as seguintes funcionalidades:

- API para possibilitar a comunicação Mosaik/Simulador/Mosaik;
- Implementação de *handlers* para diferentes tipo de processos de simulação;

- Composição de cenários de simulação;
- Geração de uma base de dados comum, coordenando a troca de dados entre os simuladores.

Diversas estratégias de controle podem ser utilizadas em simulações envolvendo instâncias que modelam o comportamento de RED. Sistemas multiagentes (SMA) vêm sendo utilizados para prover funções de controle distribuído em sistemas que integram RED em ambientes de REI [10].

O framework Python Agent DEvelopment (PADE) [11], foi desenvolvido inicialmente pelo Grupo de Redes Elétricas Inteligentes da Universidade Federal do Ceará e inspirado pelo tradiconal framework para SMA JADE. O PADE é uma alternativa para implementação e execução de sistemas de agentes reativos em Python, sendo construído no topo do framework para desenvolvimento de sistemas assíncronos Twisted [12], disponibilizando aos agentes módulos de programação, interface de gerenciamento e comunicação de acordo com os padrões especificados pela *Foundation for Interoperable Physical Agents* (FIPA).

Na Figura 2 é mostrada a arquitetura do framework PADE com todas as camadas de integração com o sistema em que os agentes são executados, passando pelo hardware, sistema operacional, implementação em Python com Twisted e os recursos disponibilizados aos usuários para desenvolvimento de SMA: ambiente de execução, biblioteca com módulos de desenvolvimento e gerenciamento de agentes.

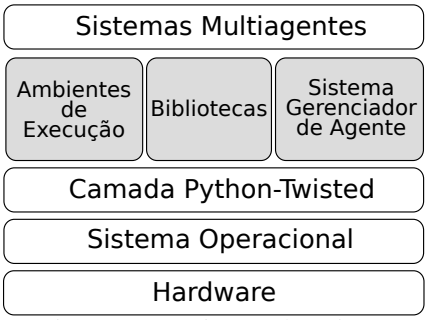

Figura 2. Arquitetura PADE integrada à sistemas computacionais [11].

Utilizado em conjunto com o framework para co-simulação Mosaik, o PADE pode se tornar uma ferramenta indispensável na modelagem dos comportamentos e na dinâmica de controle dos dispositivos simulados no ambiente de redes elétricas inteligentes.

4. API Mosaik para integração de simuladores

Um dos componentes mais importantes disponibilizados pelo Mosaik para prover o ambiente de co-simulação é sua API para permitir a integração de diferentes processos de simulação em uma base de tempo comum. Existem duas opções de utilização da API disponibilizada pelo Mosaik: API de alto nível e API de baixo nível.

Conforme mostrado na Figura 3, tanto na API de alto nível quanto na API de baixo nível, o Mosaik provê comunicação com cada uma das instâncias de simulação enviando mensagens padronizadas no formato *JavaScript Object Notation* (JSON) por meio de *sockets* TCP/IP. A API de alto nível é utilizada para integrar simuladores que não possuem loop de execução próprio e que podem ser modelados em linguagem de programação Python. Essa implementação da API de alto nível do Mosaik trata-se de um módulo Python que pode ser importado e utilizado em um arquivo de código fonte. Suas classes implementam a comunicação direta com o núcleo de simulação do Mosaik e deixam transparente ao usuário os processos necessários à integração.

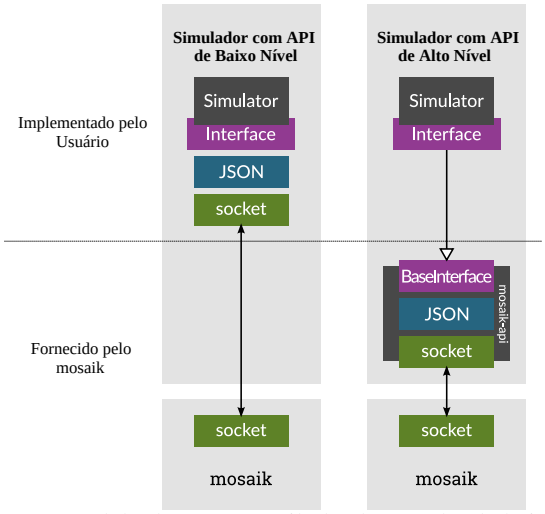

Figura 3. Modelo de API Mosaik de alto nível e de baixo nível [8].

A API de baixo nível deve ser utilizada em simuladores que possuem loop de execução próprio, ou ainda, que não possam ser implementados utilizando Python. Nesse caso, a implementação dos sockets TCP/IP e o tratamento das mensagens JSON enviadas pelo Mosaik devem ser realizadas no simulador. Essa foi a abordagem desenvolvida neste trabalho.

O Mosaik realiza os procedimentos de criação de cenário, controle do fluxo de informações e compartilhamento de base de dados comum, como já mencionado anteriormente, por meio de mensagens padronizadas para cada um dos simuladores gerenciados no processo de co-simulação. Na Figura 4 são apresentas as mensagens mais comuns utilizadas pelo Mosaik, destacando-se:

- *init(sid, sim_params)*: Inicializa o simulador com o identificador sid e recebe parâmetros adicionais passados pela API do Mosaik;
- *create(num, model, model_params)*: Cria *num* instâncias do modelo especificado com os parâmetros enviados pela API do Mosaik;
- *step(time, inputs)*: Executa um passo de simulação no tempo especificado pelo mecanismo de sincronismo

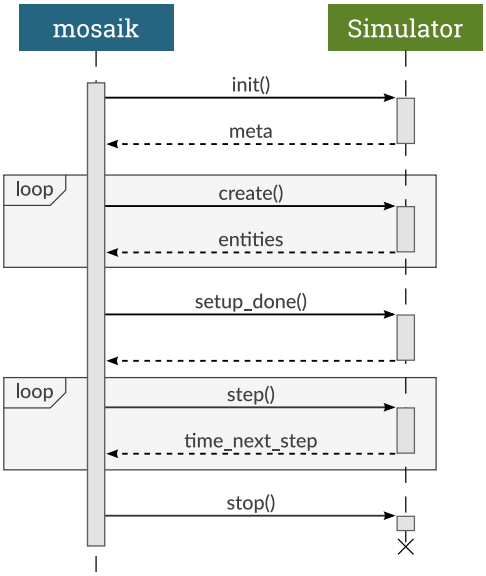

Figura 4. Sequência de mensagens enviadas entre Mosaik e processo de simulação [8].

do Mosaik com os valores de entrada enviados por outro simulador previamente conectado à instância de simulação e retornando à API do Mosaik o tempo para a próxima chamada de execução do simulador;

- *get_data(outputs)*: Recebe requisição para envio de dados, previamente especificados, à API do Mosaik que cuidará em direcioná-los ao simulador que os receberá como parâmetro *input* do método *setup*.

Na próxima Seção será descrito qual dessas APIs foi selecionada para implementação da integração com o PADE, além de detalhar como foi realizada essa integração.

5. Implementação de API PADE para integração com Mosaik

Para integrar os agentes do ambiente de execução do PADE com o gerenciador de simulação do Mosaik, é necessário utilizar a API de baixo nível disponibilizada pois, apesar do código fonte dos agentes PADE estar em Python, o PADE possui seu próprio loop de execução e portanto não consegue integrar em alto nível com o Mosaik. A API de baixo nível irá enviar uma sequência de mensagens padronizadas em formato JSON para um socket TCP/IP previamente informado nas configurações do Mosaik, que deverá receber, processar e responder as mensagens que o Mosaik enviar.

As mensagens enviadas pelo Mosaik ao simulador integrado via API de baixo nível seguem o padrão estabelecido na Figura 5, em que o campo *Header* indica o número de bytes, codificado em unsigned integer (uint32), contidos no campo *Payload*, que é codificado em UTF-8 e contém uma lista no formato JSON com três campos:

- *Type*: codificação para classificar tipo de mensagem sendo 0 para requisição, 1 para resposta e 2 para falha;
- *ID*: Identificador único para cada requisição enviada pelo Mosaik para os simuladores. As respostas precisam ter o mesmo identificador que a mensagem de requisição;
- *Content*: Conteúdo da mensagem, padronizado como uma lista JSON e que depende do tipo de mensagem.

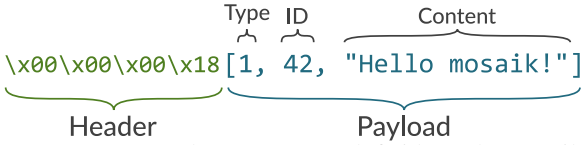

Figura 5. Estrutura das mensagens definidas pelo Mosaik para coordenação e sincronismo no ambiente de co-simulação [8].

O Mosaik faz extensivo uso da estrutura de dados Python chamada dicionário, que trata-se de um conjunto de pares (chave, valor) para configurar seus modelos de simulação. Dessa forma, o primeiro procedimento para que o PADE possa ser integrado com o Mosaik é a inclusão do nome do modelo que irá representar os agentes PADE, assim como seu endereço na rede (endereço IP e porta de execução do agente) no dicionário que fornece informações a respeito dos simuladores que serão executados durante o ciclo de simulação. Um dicionário contendo essas configurações é apresentado na Figura 6, em que é possível visualizar as declarações de simuladores que utilizam a API de alto nível e o modelo que representa um agente PADE utilizando sua API de baixo nível.

```
1    sim_config = {
2        "ExampleSim": {"python": "example_sim:
     ExampleSim"},
3        "PadeSim": {"connect":
     "127.0.0.1:20000"}
4    }
5
```

Figura 6. Configuração de simulador Mosaik e agente PADE integrados em co-simulação.

Na Figura 6 a palavra chave `connect` indica um endereço e uma porta na qual o Mosaik tentará estabelecer uma conexão após ser inicializado. Portanto, o agente PADE já deve estar sendo executado no instante que o processo do Mosaik for inicializado.

No lado do PADE, o agente precisa implementar uma classe que irá tratar cada uma das requisições enviadas pelo Mosaik. O diagrama de classes UML mostrando a relação da classe `MosaikSim` com a classe `Agent` do PADE é apresentado na Figura 7.

Após estabelecida a conexão entre Mosaik e PADE, o processo de gerenciamento de simulações do Mosaik inicia o envio de mensagens de controle para o agente PADE. Essas

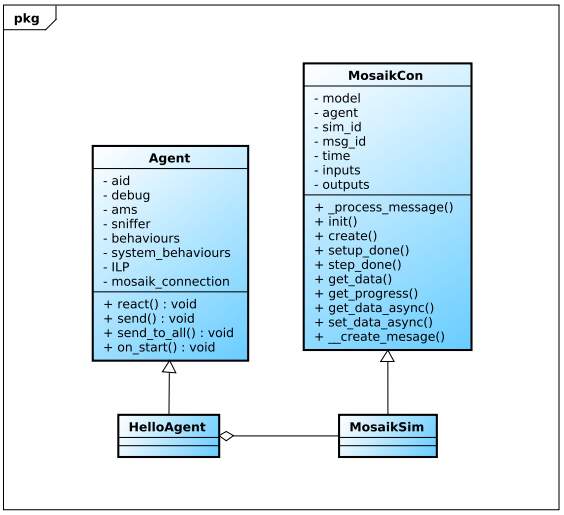

Figura 7. Diagrama de classes UML.

mensagens são identificadas pelo PADE e processadas separadamente pelos métodos desenvolvidos na classe `MosaikCon` do PADE, conforme mostrado na Figura 7.

Um exemplo de classe que herda a classe `MosaikCon` no código fonte de um agente PADE pode ser visualizada na Figura 8, que mostra a classe `MosaikSim` e cada um dos métodos da classe `MosaikCon` que estão sendo sobrescritos de acordo com o comportamento desejado nos procedimentos da simulação.

A próxima seção apresentará o estudo de caso utilizando a integração entre o PADE e o Mosaik.

```
class MosaikSim(MosaikCon):

    def __init__(self, agent):
        super(MosaikSim, self).__init__(
MOSAIK_MODELS, agent)
        self.entities = list()
        self.loc_name = self.agent.aid.localname

    def create(self, num, model, init_val,
medium_val):
        entities_info = list()
        for i in range(num):
            self.entities.append(init_val)
            display_message(self.loc_name, str(
init_val))
            entities_info.append(
                {'eid': self.sim_id + '.' + str(
i),
                 'type': model,
                 'rel': []})
        return entities_info

    def step(self, time, inputs):
        if time % 501 == 0 and time != 0:
            display_message(self.loc_name, '{:4d
}'.format(time))
        return time + self.time_step

    def get_data(self, outputs):
        response = dict()
        for model, list_values in outputs.items
():
            response[model] = dict()
            for value in list_values:
                response[model][value] = 1.0
        return response
```

Figura 8. Código exemplo de implementação da classe MosaikCon.

6. Estudo de Caso

Para demonstrar a potencialidade da integração entre os *frameworks* Mosaik e PADE na implementação de modelos computacionais para simulações de ambientes de redes elétricas inteligentes, será realizado um estudo de caso para simulação da dinâmica dos RED tipicamente presentes em uma rede de distribuição de baixa tensão, ou seja, carga residencial, geração fotovoltaica e sistema de armazenamento de energia. Os principais elementos do estudo de caso são:

- Rede de distribuição secundária (baixa tensão) com um transformador de distribuição e 37 barras associadas à cargas residenciais e RED;
- 20 gerações distribuídas solar-fotovoltaicas ao longo da rede;
- Para cada geração distribuída, está associado um agente PADE, denominado *Device Agent* que simulará o comportamento de um sistema de armazenamento de energia que armazenará 10% da energia gerada pelo sistema fotovoltaico;
- Integração com ferramenta de cálculo de fluxo de carga para verificação do estado da rede, com especial atenção no critério de nível de tensão;
- Integração com ferramenta para visualização de resultados via interface *web*.

A topologia do sistema elétrico pode ser observada na Figura 9, em que é possível identificar cada um dos nós que compõem a rede de energia elétrica em nível secundário de distribuição, assim como suas interconexões. A rede é composta por um transformador e 37 nós que podem ter cargas associadas ou não à geração e sistema de armazenamento de energia elétrica.

Antes de analisar os resultados obtidos da simulação é importante descrever o cenário geral, já que o principal objetivo desta seção é demonstrar a integração entre as ferramentas Mosaik e PADE e não os aspectos de qualidade de energia ou de estabilidade do sistema, parâmetros geralmente analisados neste tipo de aplicação.

O estudo de caso foi estruturado tomando como base um exemplo de utilização do Mosaik disponível publicamente em repositório online [13] e modificado para a inclusão dos agentes PADE. Um repositório on-line de livre acesso está disponível com o código fonte deste estudo de caso[1].

[1] <https://github.com/grei-ufc/mosaik-demo-wesaac-2019>

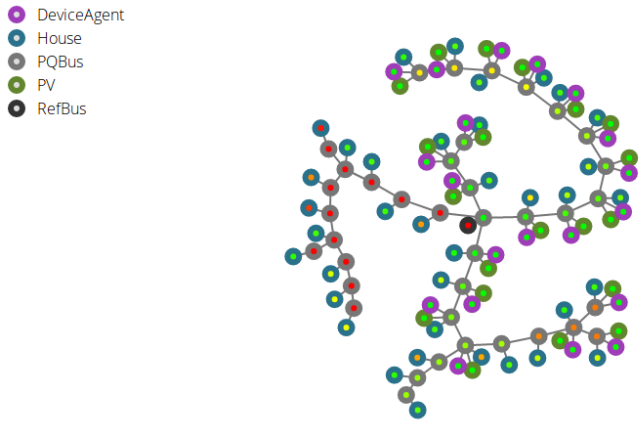

Figura 9. Representação gráfica da rede elétrica do estudo de caso.

No exemplo são utilizados plugins para integração do Mosaik com ferramentas de armazenamento de dados, acesso a arquivos CSV, ferramenta de análise de fluxo de carga e visualização via interface *web*. Na Figura 10 é possível visualizar a declaração de cada um dos simuladores utilizados na co-simulação:

```
sim_config = {
    'CSV': {
        'python': 'mosaik_csv:CSV',
    },
    'DB': {
        'cmd': 'mosaik-hdf5 %(addr)s',
    },
    'HouseholdSim': {
        'python': 'householdsim.mosaik:
HouseholdSim',
    },
    'PyPower': {
        'python': 'mosaik_pypower.mosaik:
PyPower',
    },
    'WebVis': {
        'cmd': 'mosaik-web -s 0.0.0.0:8000
%(addr)s',
    },
}

```

Figura 10. Configuração dos simuladores no Mosaik.

O simulador de nome *CSV* é utilizado para leitura de dados de medição da potência de saída de geradores solar fotovoltaicos e utilizados para integrar o processo de simulação. O simulador *DB* irá receber os resultados de parâmetros elétricos da simulação e armazenar os dados em um arquivo de dados *hdf5*. O simulador *HouseHoldSim* gera uma série temporal que representa a energia consumida em uma residência com número arbitrário de moradores. O simulador *PyPower* integra a ferramenta de fluxo de carga *PyPower* com Mosaik para realizar as análises a cada passo de simulação estabelecido. Por fim o simulador *WebVis* é uma ferramenta para visualização gráfica das dinâmicas de simulação, gerando a topologia gráfica da rede e gráficos de nível de tensão e potência gerada/consumida em cada um dos nós.

Para integrar os agentes PADE ao processo de simulação é necessário declarar simuladores que irão representar os agentes, da mesma forma que os demais simuladores foram declarados.

O código que integra os agentes ao processo de co-simulação do Mosaik é apresentado na Figura 11.

```
data = json.load(open('data/demo_lv_grid.json'))
agent_names = [i[0] for i in data['bus']][2:]

agent_names = random.sample(agent_names, PV_QTD)

port = 1234
device_agent_sim_names = dict()
for i in agent_names:
    name = 'DeviceAgentSim{}'.format(i)
    device_agent_sim_names[i] = name
    sim_config[name] = {'connect': 'localhost:'
    + str(port)}
    port += 1
```

Figura 11. script de integração dos agentes Pade ao Mosaik.

Na Figura 11 são mostrados o carregamento dos nomes dos agentes que serão executados, a declaração dos simuladores no dicionário de configuração `sim_config`, e a definição do endereço de cada agente, assim como a porta em que cada um será executado.

Após a definição dos agentes, é necessário associá-los aos geradores e as barras da rede de distribuição. Isso é feito utilizando o método `connect()`, disponibilizado pela API do Mosaik para que os simuladores possam trocar dados entre si.

Realizada a definição dos modelos de simulação e de seus respectivos simuladores, bem como a criação das instâncias de modelo que serão executadas nos simuladores e do estabelecimento das conexões entre estes, o ambiente de co-simulação está pronto para ser executado por meio do comando `run(until=END)` que inicializa o processo de simulação e estabelece o tempo total que ela deve durar, definido em segundos na variável `END`.

Durante o processo de co-simulação é possível ter acesso aos dados gerados utilizando uma interface gráfica *web*, bastando inserir em um navegador de internet o endereço IP e a porta do plugin `WebVis`, que disponibiliza visualmente os dados de cada uma das instâncias de simulação previamente configuradas. Na Figura 12 são mostrados os valores de demanda da residência, a potência de saída do sistema fotovoltaico, a parcela de 10% da potência gerada armazenada na bateria e o nível de tensão na barra considerada.

Na Figura 12 é possível também observar a semelhança entre as formas de onda que representam a potência gerada pelo sistema fotovoltaico e a energia armazenada pelo sistema de armazenamento, como era de se esperar já que existe uma correspondência direta entre os dois valores, diferindo

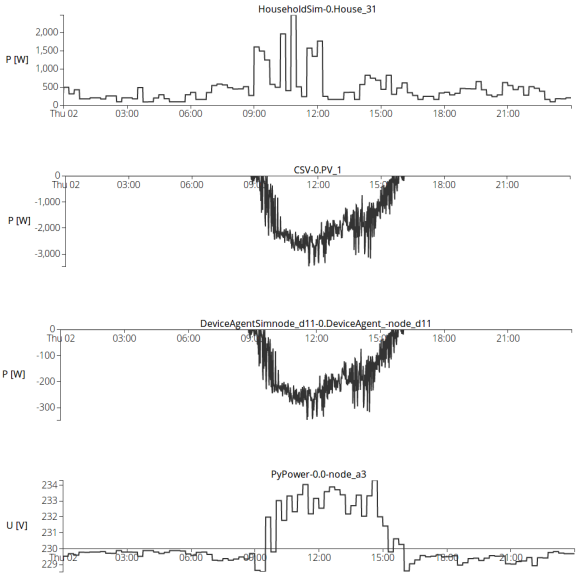

Figura 12. Resultados da simulação.

somente por um fator de escala de 10%.

O agente PADE que modela o comportamento do sistema de armazenamento recebe os dados de geração, aplica o fator de escala de 10% e envia este valor de energia armazenada para o simulador responsável pelo fluxo de carga que irá agregar os valores de potência gerada/consumida e descontar o valor de energia armazenada para o cálculo das tensões em cada uma das barras do sistema.

Assim, esta simulação apresentou o estudo de caso utilizando Python e Mosaik para identificação do estado do sistema elétrico de distribuição, levando-se em conta a presença de 37 cargas, 20 geradores fotovoltaicos distribuídos, e 20 dispositivos de armazenamento que guardam 10% da energia produzida. Mesmo com todos esses elementos presentes e tendo os armazenadores modelados como agentes, foi possível verificar o impacto dos mesmos no sistema elétrico a partir da integração com o Mosaik.

7. Conclusão

Neste artigo foi demonstrado a integração entre duas ferramentas que podem ser de grande ajuda na modelagem e análise de sistemas de energia elétrica em um contexto de redes elétricas inteligentes com presença de recursos energéticos distribuídos.

A primeira ferramenta chama-se Mosaik que é um software implementado na linguagem de programação Python com o objetivo de prover um serviço de integração de simuladores em co-simulação, oferecendo ferramentas para descrição de cenário, gerando uma base de tempo comum entre os simuladores e provendo serviço de troca de mensagens entre estes.

A segunda ferramenta analisada chama-se PADE, um framework em Python para modelagem de sistemas multiagentes baseados nos padrões estabelecidos pela FIPA, que provê bibliotecas para modelagem dos comportamentos dos agentes, ambiente de execução e interface gráfica para gerenciamento.

As duas ferramentas foram integradas com êxito e foi demonstrada por meio de um estudo de caso que apresenta uma rede de distribuição com RED em que os agentes PADE simulam o comportamento de um sistema de armazenamento de energia. Os resultados demonstram a troca de informações dos simuladores com os agentes e a possibilidade de implementação de esquemas de controle mais complexos em que seja necessário a utilização de modelos de inteligência distribuída.

Vale ressaltar que as duas ferramentas são software livre e disponíveis gratuitamente em repositórios *on-line*.

Agradecimentos

Os autores do artigo agradecem à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pelo apoio financeiro e à Universidade Federal do Ceará (UFC) e Universidade Federal do Pará (UFPA) pela disponibilização do espaço físico e da utilização de seus laboratórios.

Contribuição dos Autores

Lucas Silveira Melo contribuiu no desenvolvimento das simulações computacionais e escrita inicial; Filipe Saraiva na revisão e sugestões nos algoritmos das simulações e revisões na escrita; Ruth Pastora Saraiva Leão com sugestões e correções na sessão 2 e 6; Raimundo Furtado Sampaio contribuiu com sugestões e correções na sessão 2 e 3; Giovanni Cordeiro Barroso com sugestões e correções na sessão 2 e 6.

Referências

[1] KOK, K.; WIDERGREN, S. A Society of Devices: Integrating Intelligent Distributed Resources with Transactive Energy. *IEEE Power and Energy Magazine*, v. 14, n. 3, p. 34–45, 2016.

[2] SOARES, J. *et al.* A stochastic model for energy resources management considering demand response in smart grids. *Electric Power Systems Research*, Elsevier B.V., v. 143, p. 599–610, 2017. Disponível em: <http://dx.doi.org/10.1016/j.epsr.2016.10.056>.

[3] ALBAGLI, A. N.; FALCÃO, D. M.; De Rezende, J. F. Smart grid framework co-simulation using HLA architecture. *Electric Power Systems Research*, Elsevier B.V., v. 130, p. 22–33, 2016. Disponível em: <http://dx.doi.org/10.1016/j.epsr.2015.08.019>.

[4] DUAN, Y. *et al.* Co-simulation of distributed control system based on JADE for smart distribution networks with distributed generations. *IET Generation, Transmission &*

Distribution, v. 11, n. 12, p. 3097–3105, 2017. Disponível em: <http://digital-library.theiet.org/content/journals/10.1049/iet-gtd.2016.1382>.

[5] CULLEN, M. F.; MILLER, S. F.; HORD, R. M. A model-based system for object recognition in aerial scenes. *Proceedings of SPIE - The International Society for Optical Engineering*, v. 726, p. 530–538, 1987.

[6] VOGT, M.; MARTEN, F.; BRAUN, M. A survey and statistical analysis of smart grid co-simulations. *Applied Energy*, v. 222, p. 67 – 78, 2018. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0306261918304616>.

[7] BROWN, T.; HÖRSCH, J.; SCHLACHTBERGER, D. PyPSA: Python for Power System Analysis. 2017. Disponível em: <http://arxiv.org/abs/1707.09913{\%}0Ahttp://dx.doi.org/10.5334/jors.>

[8] MOSAIK. *Mosaik Software*. [S.l.]: BitBucket, 2018. <https://bitbucket.org/mosaik/mosaik>.

[9] SCHÜTTE, S.; SCHERFKE, S.; TRÖSCHEL, M. Mosaik: A framework for modular simulation of active components in Smart Grids. In: *2011 IEEE 1st International Workshop on Smart Grid Modeling and Simulation, SGMS 2011*. [S.l.: s.n.], 2011. p. 55–60.

[10] Kumar Nunna, H. S. V. S.; DOOLLA, S. Multiagent-based distributed-energy-resource management for intelligent microgrids. *IEEE Transactions on Industrial Electronics*, v. 60, n. 4, p. 1678–1687, 2013.

[11] MELO, L. S. *et al.* Python-based multi-agent platform for application on power grids. *International Transactions on Electrical Energy Systems*, v. 29, n. 6, p. e12012, 2019. E12012 ITEES-18-0867.R2. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/2050-7038.12012>.

[12] TWISTED. *Twisted Matrix Labs*. [S.l.]: GitHub, 2018. <https://github.com/twisted/twisted>.

[13] MOSAIK. *Mosaik Demo*. [S.l.]: BitBucket, 2017. <https://bitbucket.org/mosaik/mosaik-demo>.