HLA e Microsserviços: Uma Aplicação para Simulação Distribuída

Oswaldo Segundo da Costa Neto, Carlos Magno Oliveira de Abreu, Marcelo Alexandre Martins da Conceição, André Benzi Baccarin e Adilson Marques da Cunha
Instituto Tecnológico de Aeronáutica (ITA), São José dos Campos/SP - Brasil

***Resumo*— Ao promover interoperabilidade entre sistemas de simulação, a abordagem de arquitetura distribuída vem sendo compreendida como uma alternativa promissora, trazendo benefícios ao processo de desenvolvimento de aplicações que compõem o universo *Modeling and Simulation* (M&S).**

O *framework High Level Architecture* (HLA) vem se mantendo como uma das principais arquiteturas entre sistemas de simulação distribuída, por aproveitar a normatização de mecanismos que buscam facilitar o intercâmbio de dados entre aplicações.

Visando proporcionar um ambiente heterogêneo por meio do *framework* HLA e se beneficiar de características da computação em nuvem, a ferramenta Docker vem se tornando um elemento promissor para viabilizar a criação de ecossistemas de simulação construídos sob a ótica de microsserviços.

Este artigo apresenta um estudo de caso envolvendo a implementação de uma simulação distribuída HLA obtida por meio do uso de microsserviços com potencial redução de custo de desenvolvimento e aumento de velocidade na construção de ambientes de simulação distribuída.

***Palavras-Chave*— simulação distribuída, HLA, microsserviços**

I. Introdução

A utilização de arquiteturas de simulação distribuída vem beneficiando largamente instituições militares e civis nas últimas décadas [1], [2]. A abordagem de uma infraestrutura que possibilite a comunicação entre sistemas construídos inicialmente para trabalhar de forma independente vem resultando não apenas em melhorias de capacidades de treinamentos, como também impulsionando entregas de valor em processos decisórios.

Partindo das premissas de facilitação de desenvolvimentos e de manutenção de sistemas simulados que compartilham o mesmo ecossistema, a serialização de softwares que compõem esse ambiente, por meio do uso de microsserviços, vem representando um aspecto relevante e trazendo pontos positivos para o gerenciamento de processos de simulação distribuída.

Neste trabalho de pesquisa, a Seção II apresenta uma breve descrição do *framework* de arquitetura *High Level Architecture* (HLA) para simulação distribuída e de aspectos básicos sobre microsserviços e *containers*, com foco na utilização da ferramenta Docker [3].

A Seção III inicia com a abordagem central do artigo, sintetizando os principais trabalhos correlatos envolvendo o tema em pauta. Por fim, as Seções IV e V trazem informações acerca de um sistema denominado Arcanjo, que utiliza as ferramentas descritas nas seções anteriores, especificando as principais características que o destacam, além de uma abordagem para o emprego do conceito de microsserviços aplicado a um ambiente de simulação distribuída.

II. Fundamentação Teórica

A. High Level Architecture

O *framework High Level Architecture* (HLA) envolve um conjunto de padrões internacionais publicados pelo Instituto de Engenheiros Elétricos e Eletrônico (*Institute of Electrical and Electronics Engineers* - IEEE) desenvolvido pela Organização de Padrões de Simulação de Interoperabilidade (*Simulation Interoperability Standards Organization* - SISO).

As especificações dos padrões HLA descrevem os elementos, as interfaces e as propriedades que devem ser utilizadas no ecossistema compartilhado pelas aplicações que têm o objetivo de participar deste ecossistema.

Por ter se tornado uma padronização consolidada, o *framework* HLA é atualmente prescrito e recomendado pela Organização do Tratado do Atlântico Norte (OTAN), promovendo políticas de facilitação de interoperabilidade de diferentes sistemas a qualquer tempo, organização ou país. O governo brasileiro, através do Ministério da Defesa (MD), vem buscando obter cada vez mais interoperabilidade de dados de simulação por meio do *framework* HLA [4].

O *framework* HLA utiliza uma topologia baseada em um barramento de serviços que possibilita o tráfego de dados entre as aplicações por meio de interfaces bem definidas. Essas interfaces fornecem os serviços de gerenciamento de execução, de tempo e de outras atividades essenciais para a obtenção de simulações HLA, permitindo trocas de dados entre aplicações em tempo de execução.

Seguindo a topologia descrita na Fig. 1, comumente chamada de topologia de pirulito (*lollypop topology*), outros componentes básicos do padrão HLA são os Federados e a Federação.

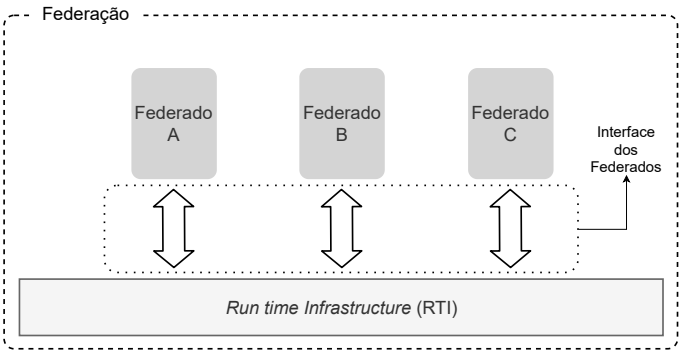

Fig. 1: Exemplo de Topologia do *framework* HLA composta por três Federados

Um Federado pode ser definido como uma aplicação-membro de um ecossistema compartilhado, cumprindo todas as especificações de requisitos previstas no *framework* HLA. Tecnicamente, um Federado também pode ser caracterizado como uma conexão unitária ao barramento de comunicações,

que é nomeado *Runtime Infrastructure* (RTI) [5].

Dessa forma, um Federado pode ser considerado como um software de simulação de voos virtuais como o X-*Plane* [6] ou o *Microsoft Flight Simulator* [7]. Cada Federado pode servir para modelar qualquer número de objetos em uma simulação. Pode, por exemplo, servir para modelar uma aeronave ou centenas delas. Outros exemplos de Federados são as ferramentas gerais como *data loggers* ou visualizadores 3D [8].

Uma Federação representa um conjunto de todos os Federados sob sua responsabilidade. O elemento que identifica e conecta as aplicações de Federados à identidade de uma Federação é uma especificação de comunicação de dados denominada *Federate Object Model* (FOM).

Geralmente um FOM é escrito em *eXtensible Markup Language* (XML), servindo para padronizar todos os modelos de intercâmbio de dados entre os Federados que, por sua vez, devem possuir um documento denominado *Simulation Object Model* (SOM), que especifica quais informações o Federado vai alimentar ou consumir (*publish/subscribe*) via barramento de comunicações RTI. Esses componentes encontram-se mostrados na Fig. 2.

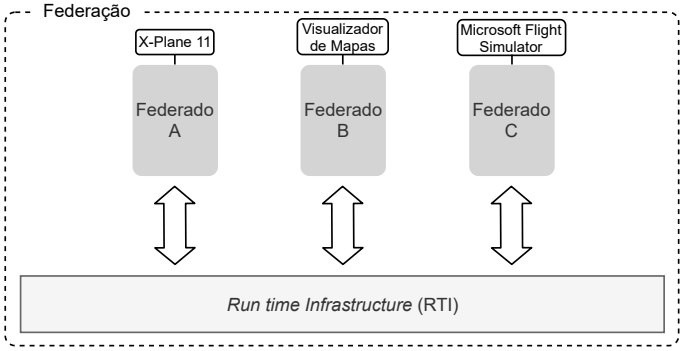

Fig. 2: Exemplo de Topologia *framework* HLA - Federação e Federados

FOM e SOM são modelos de objetos altamente padronizados pelas políticas do *framework* HLA. As suas especificações de requisitos se encontram dispostas em um documento chamado *Object Model Template* (OMT), que estabelece o formato a ser obedecido por esses modelos, para que possam ser compreendidos como uma Classe Objeto do *framework* HLA ou (HLA *Object Class*). Há ainda um terceiro modelo de objeto denominado *Management Object Model* (MOM), que tem a função de fornecer um grupo de construtores para suporte, monitoramento e controle da execução da Federação [9].

Buscando promover uma padronização em ambientes de simulação distribuída, a organização SISO divulgou um FOM de referência. O *Real-time Platform-level Reference Federation Object Model* (RPR-FOM v2.0) define a hierarquia de objetos e a interação com as classes utilizadas em simulações HLA. Ele foi criado também para conectar simulações de entidades físicas discretas dentro de um complexo mundo virtual [10].

Aplicações que se utilizam de outros padrões de arquitetura, como o *Distributed Interactive Simulation* (DIS), recorrem ao RPR-FOM (ou a uma de suas variantes) quando decidem ingressar em um ecossistema HLA.

B. SOA[1], Microsserviços e MSaaS

Partindo de uma abordagem moderna para o desenvolvimento de software e de soluções, de acordo com [11], um serviço pode ser definido como "[...] valor entregue através de uma interface bem definida e disponível para uma comunidade". Nesse sentido, o surgimento do conceito de microsserviços atrelado às Arquiteturas Orientadas a Serviços (*Service-Oriented Architectures* - SOA) se tornaram uma real tendência no meio tecnológico.

Microsserviços podem ser definidos como uma capacidade independente e autônoma projetada como processo isolado ou que se comunica com outros microsserviços através de padrões e protocolos de comunicação, como o *Hypertext Transfer Protocol* (HTTP) e RESTful *Web Services* (REST) [12].

Dessa forma, uma arquitetura baseada em microsserviços pode ser entendida como um grupo de aplicações independentes que oferecem, individualmente, um serviço específico e com funcionalidades bem definidas por meio de um protocolo estabelecido, objetivando suplantar o funcionamento de todo o sistema.

A serialização dos serviços oferecidos por um sistema mostra aspectos positivos e relevantes em comparação com sistemas monolíticos. Dentre essas vantagens, pode-se citar: *a*) simplicidade, obtida por meio de modularização e isolamento dos serviços; *b*) escalabilidade; *c*) velocidade no desenvolvimento de novos serviços, através de reusabilidade; e *d*) descentralização de dados.

Entretanto, também existem alguns aspectos negativos na implementação desse modelo de arquitetura. Dentre eles, pode-se citar: *a*) dificuldade de identificação de falhas, pois a difusão do sistema causa impacto no seu gerenciamento; *b*) aumento de latência, decorrente do processo de comunicação entre os serviços; e *c*) crescimento de complexidade operacional. Estes aspectos negativos podem ser entendidos como problemas intrínsecos às aplicações que utilizam soluções distribuídas e, portanto, teriam impacto negativo minimizado no emprego em arquiteturas de simulação que utilizem o *framework* HLA.

Com o objetivo de suplantar as limitações descritas em [13] e oriundas de características de sistemas monolíticos de Modelagem e Simulação (M&S) para simulação distribuída, este artigo propõe uma solução baseada em microsserviços e inspirada no paradigma *Modeling and Simulation as a Service* (MSaaS) [14], além de viabilizar os benefícios e vantagens de aplicações que utilizam Computação em Nuvem (*Cloud Computing*) e Arquitetura Orientada a Serviços (SOA). Esse tipo de abordagem pode, além das vantagens anteriormente mencionadas, reduzir custos na implementação de sistemas de M&S.

C. Docker

A iniciativa da comunidade tecnológica para a criação de uma plataforma de código aberto, que buscasse resolver os problemas de implantação e de escalabilidade oriundos da separação de aplicativos e dependências de sua infraestrutura, culminou com o surgimento de uma tendência atual no âmbito de desenvolvimento de sistemas computacionais: a plataforma Docker.

O advento dessa plataforma proporcionou a sedimentação do conceito de virtualização em conjunto com a infraestrutura de

[1] Arquitetura Orientada a Serviços

Linux *Containers* [15], transformando a noção de provisionamento de máquina virtual de um modelo demorado e centrado na administração do sistema em outro modelo que atribui maior foco no fluxo de trabalho orientado ao desenvolvedor [16].

A utilização da plataforma Docker, diferentemente de máquinas virtuais baseadas em *hypervisor*, não apenas fornece uma abstração da camada de hardware, mas virtualiza as chamadas do sistema operacional, proporcionando que múltiplos *containers* compartilhem a mesma instância de *kernel* do sistema operacional. Este compartilhamento ocorre porque os *containers* utilizam recursos de isolamento do *kernel* Linux, como grupos de controle (*cgroups*) e *namespaces* para permitir que processos independentes sejam executados dentro de uma única instância Linux [12]. Portanto, *containers* Docker são aplicações leves e que utilizam os recursos de hardware de uma maneira muito eficiente.

Dessa maneira, a infraestrutura do Docker fornece os principais serviços para a completude do fluxo de trabalho de aplicações. Sendo assim, o conjunto de ferramentas do ecossistema Docker propiciam o gerenciamento e a implementação dos *containers*, que são obtidos através de requisições pelo Docker *Client* com o uso de Docker *Application Programming Interfaces* (APIs) específicas, que dão as instruções para que o Docker *Daemon* obtenha as Docker *Images* a partir de um repositório público ou privado, o Docker *Registry*.

Os comandos subsequentes enviados pelo *Client* promovem a alocação de recursos de hardware para a disponibilização das aplicações que funcionarão internamente aos Docker *Containers*. Todo o conjunto de ferramentas e elementos componentes do ecossistema Docker são executados sob a égide do sistema operacional do servidor hospedeiro, que compreende toda essa infraestrutura de virtualização por meio do encapsulamento no Docker *Host*.

Outro ponto a ser destacado da infraestrutura Docker é o fato de que, depois de ser realizada a implementação de imagens, apesar da autonomia existente entre cada um dos *containers*, existem camadas na arquitetura Docker compartilhadas. Por exemplo, ao estruturar os Federados a partir de *containers*, o Docker disponibiliza uma rede interna em formato *bridge*, para que possa haver a comunicação interna e o fluxo de dados que a RTI necessitar na estrutura HLA, conforme mostrado na Fig. 3.

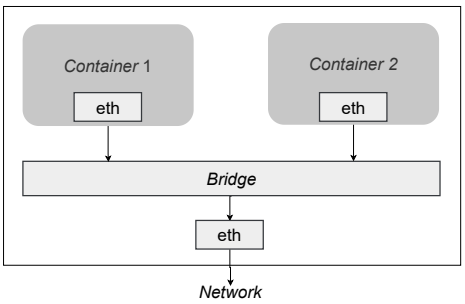

Fig. 3: Rede Docker em Formato *Bridge*

A união da plataforma Docker com a infraestrutura proporcionada pela especificação de arquitetura de simulação distribuída HLA permite a conteinerização de aplicações até então obtidas de forma monolítica. Portanto, a aplicabilidade das tecnologias mencionadas formam um contexto de exposição de uma estrutura de interoperabilidade de sistemas de simulação, a partir de microsserviços.

III. Os Principais Trabalhos Correlatos Encontrados

Utilizando o método de revisão literária por conveniência [17], a realização deste trabalho de pesquisa foi acompanhada de um estudo e levantamento dos principais autores que abordaram o tema de simulação distribuída sob a perspectiva da utilização de microsserviços.

Os filtros de pesquisa utilizados para obter os principais trabalhos correlatos foram aplicados em 5 passos extraídos de [18]: *i*) Pesquisa por palavras-chave; *ii*) Aplicação de critérios de aceitação e rejeição; *iii*) Filtragem por título; *iv*) Filtragem por resumo; e *v*) Filtragem por conteúdo. A aplicação destas técnicas nestes 5 passos possibilitou a obtenção de um conjunto de trabalhos utilizados para as comparações com o estudo de caso descrito neste artigo.

Desse grupo, destacou-se a referência [9], que descreve modelos para obtenção de uma simulação distribuída estruturada no *framework* HLA, nas ferramentas do ecossistema Docker e utilizando o serviço *payware* da Pitch Technologies [19], empresa de referência no ramo de simulação distribuída. Essa abordagem prevê a utilização de *containers* Docker para abrigar as aplicações dos Federados do ecossistema HLA.

Entretanto, diferentemente do proposto neste artigo, [9] elenca um *container* específico para ser o gerenciador da RTI, conforme mostrado na Fig. 4.

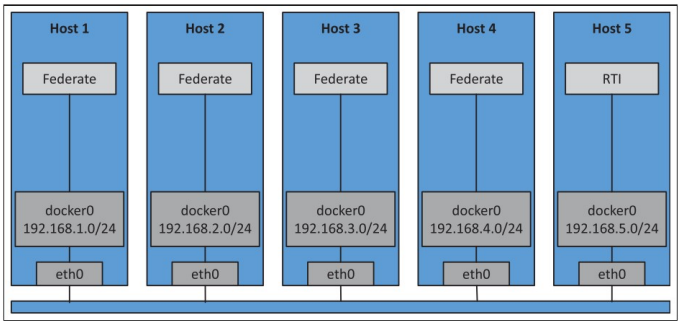

Fig. 4: A Estrutura de Rede Aplicada em [9]

Essa abordagem de RTI centralizada traz consigo a necessidade de ferramentas de orquestração dos *containers*, pois o Componente Central de RTI (*Central RTI Component* - CRC) necessita estar em operação para que existam os serviços básicos de gerenciamento HLA.

De forma diversa, uma RTI descentralizada, ou seja, estruturada apenas a partir de Componentes Locais de RTI (*Local RTI Components* - LRCs), permite mais flexibilidade na composição dos microsserviços, visto que cada Federado possui o mesmo nível de responsabilidade de gerenciamento dos serviços básicos de simulação, podendo haver a composição dos *containers* através de mecanismos de coreografia [20].

IV. O Projeto Arcanjo

Buscando endereçar uma alternativa de solução para uma plataforma de simulação que permaneça em conformidade com os padrões de recomendação da OTAN [21], o Projeto Arcanjo foi construído também com base nos requisitos do Ministério da Defesa [4] e proporciona uma solução de integração para

os sistemas de simulação que as Forças Armadas brasileiras utilizam.

Tendo como predominância a linguagem Java para sintaxe, o Projeto Arcanjo contém uma união entre tecnologias e plataformas atuais que promovem a integração de todo o sistema. A seguir encontra-se uma lista com as principais tecnologias de integração:

- *Ubuntu* - sistema operacional no qual operam os demais componentes do servidor;
- *Docker* - plataforma que abriga os *containers* para a produção dos Federados;
- *Spring Boot* - *framework open source* para a plataforma Java utilizado para acelerar o desenvolvimento de microsserviços [22];
- *Cesium* - biblioteca *open source* para manipulação de mapas 3D; e
- *Portico* - infraestrutura *open source* para implementação de arquiteturas HLA com RTI descentralizada [23].

Como exemplo de integração, segue uma prova de conceito utilizando o X-*Plane*, como um simulador virtual, que utiliza normatização de modelos proprietários e que deseja ingressar no ecossistema de simulação distribuído promovido pelo Projeto Arcanjo. Para isto, será criado um Federado na Federação chamado X-Plane *Proxy*.

Os objetos proprietários do X-*Plane* devem ser mapeados para o modelo previsto no RPR-FOM v2.0, para que possam se comunicar de forma homogênea com os outros sistemas computacionais que também estejam conectados à RTI, conforme mostrado na Fig. 5.

Este processo de mapeamento é executado pelo Federado responsável pela interpretação das informações do sistema proprietário, neste caso o X-*Plane*, realizando a mediação entre a RTI e o sistema.

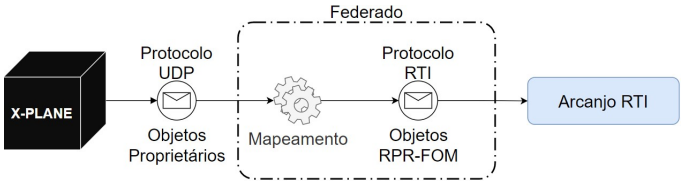

Fig. 5: O Processo de Mapeamento de Objetos para RTI

Para tornar eficaz a padronização dos modelos utilizados no ecossistema, a arquitetura virtualizada do Projeto Arcanjo é disposta de tal maneira que os arquivos de modelagem para o RPR-FOM v2.0 ficam localizados em volumes fora do ambiente Docker, garantindo eficiência no acesso de todos os *containers* para o mapeamento, conforme mostrado na Figura 6.

Compartilhando o ambiente virtual e concluindo a prova de conceito, foram utilizados dados de aeronaves do Flightradar24[2] para realizar a injeção de objetos no mundo virtual. Dessa forma, foi criado o Flightradar *Proxy*, como Federado componente da Federação em pauta.

Portanto, considerando que todo o serviço utiliza computação em nuvem e que a hospedagem ocorre em uma rede pública ou privada, um participante externo que deseje visualizar a simulação poderá fazer o uso do Federado

[2]Domínio *web* que fornece informações e visualização de aeronaves em rota por todo o planeta em tempo real [24].

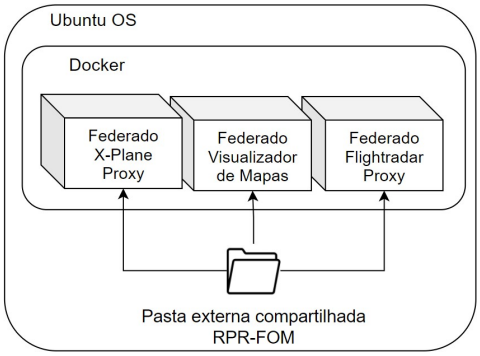

Fig. 6: O Compartilhamento de Arquivos RPR-FOM

Visualizador de Mapas, conforme mostrado na Figura 7, acessando-o através de um navegador, que fará requisições de serviços para verificar a execução da simulação por meio de uma página *web*.

O uso das bibliotecas de manipulação de mapas a partir da injeção de objetos relativos às aeronaves - pertencentes aos Federados X-Plane *Proxy* e Flightradar *Proxy* - fornece as informações de posicionamento das entidades existentes na simulação, tornando possível a visualização delas na página *web*.

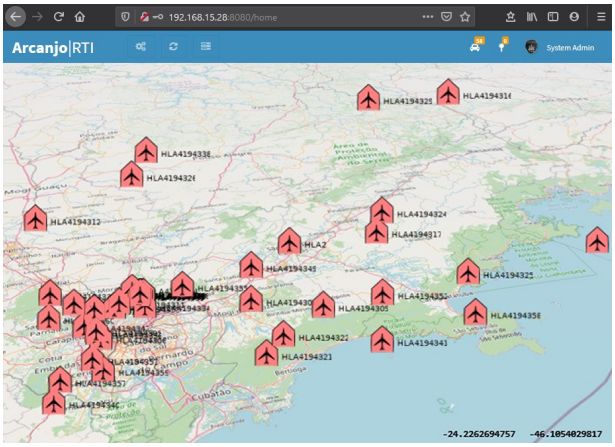

Fig. 7: Visão Geral do Federado Visualizador de Mapas do sistema Arcanjo

A infraestrutura de simulação distribuída HLA com Federados estruturados em microsserviços pode ter conexão LAN ou WAN, possibilitando gerenciamento, manipulação e utilização da infraestrutura de modo remoto. Uma representação deste contexto é mostrada na Figura 8.

V. Contribuições

Um sistema que deseja se enquadrar nos moldes do *framework* HLA deve estar em concordância com as regras estabelecidas em [25]. Esta referência é composta por 10 regras que delineiam as responsabilidades dos Federados e Federações da HLA com o objetivo de garantir uma implementação consistente. Isto posto, de acordo com a regra número 3 dessa referência: "Durante a execução da Federação, todas as trocas de dados de FOM entre Federados deverão ocorrer através da RTI".

Essa regra se aplica de forma natural para aplicações monolíticas e mais tradicionais de sistemas que utilizam o *fra-*

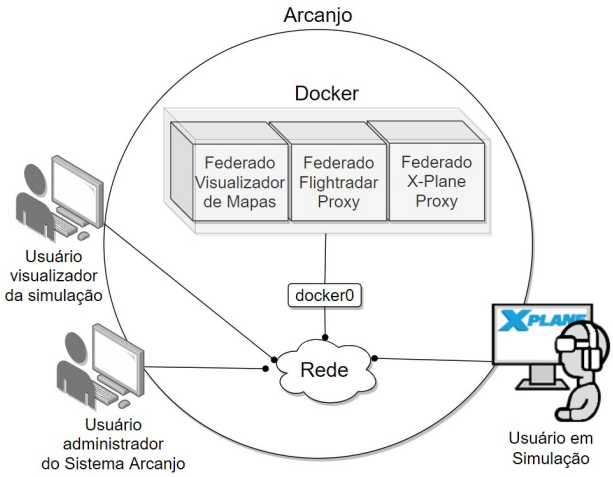

Fig. 8: Simulação HLA Utilizando Microsserviços.

mework HLA, onde a simulação necessita de vários computadores, cujo acesso para enviar comandos e dados aos Federados ocorre por terminal remoto ou físico, conforme mostrado na Figura 9.

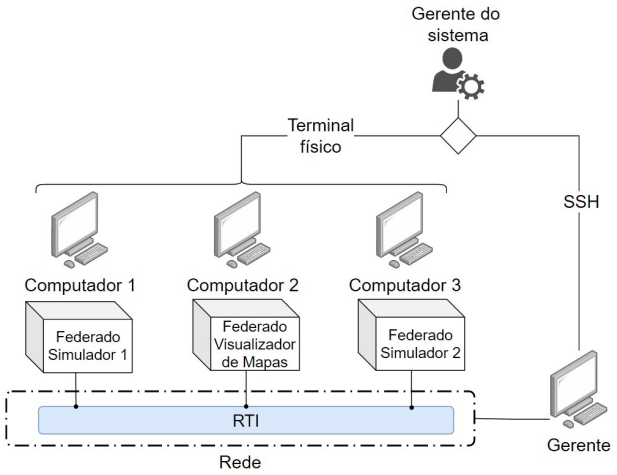

Fig. 9: Abordagem Tradicional de Comunicação HLA

Entretanto, ao utilizar os benefícios de computação em nuvem e de microsserviços, o Projeto Arcanjo possui a abordagem distinta mostrada na Figura 10, na qual o gerente comanda as ações do ecossistema HLA através de *web services*, utilizando as especificações estruturais REST [26].

Isso permite que os dados não relacionados diretamente à simulação trafeguem na rede, de forma descentralizada, isto porque a troca de informações relacionadas diretamente à simulação devem trafegar necessariamente no interior da RTI, conforme a regra número 3 de [25].

Esses dados, em formato de comandos REST, representam *inputs* de controle, tais como iniciar ou interromper uma simulação, criar ou destruir um objeto de uma classe de Federado, ou realizar alterações climáticas a partir de um Federado que controle variáveis de caráter ambiental na simulação.

A abordagem do uso de *web services* para gerenciar e manusear a simulação permitiu rapidez no desenvolvimento de componentes da simulação que, aliada aos benefícios obtidos através do uso de microsserviços, possibilitou alavancar de forma ágil a adaptação de sistemas simulados já existentes ao ecossistema gerido por uma simulação do tipo HLA.

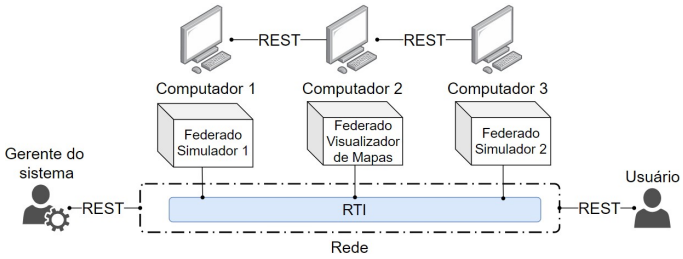

Fig. 10: Abordagem de Comunicação HLA no Arcanjo

Outro aspecto importante de divergência verificado em [9] é o fato de não haver um Federado exclusivo que contenha a infraestrutura de RTI. A inexistência de um CRC permite que, caso não haja uma RTI disponível no momento em que um Federado deseja ingressar na simulação, ele tenha autonomia suficiente para abrir um novo barramento RTI.

Posteriormente, outros Federados que também decidam ingressar na simulação podem utilizar a infraestrutura disponibilizada pelo Federado precursor. Este fato é relevante pois traz simplicidade na arquitetura do sistema e, devido às características de modularização e reusabilidade observadas na perspectiva de sistemas que utilizam o paradigma SOA, não acarreta complexidade na implementação desta funcionalidade no desenvolvimento de novos Federados.

VI. Desempenho

A aplicabilidade de conceitos e técnicas que permitam a modularização de sistemas monolíticos traz consigo os benefícios citados nas Seções anteriores. Entretanto, a separação de uma arquitetura originalmente monolítica em microsserviços acrescenta camadas entre os módulos, no caso os Federados. Em consequência disso, há uma queda de desempenho do fluxo de dados refletida em métricas bem definidas, como taxa de latência e *throughput*.

A conteinerização é uma estratégia de virtualização de aplicações. Neste sentido, há estudos como [27], [28] e [29] que projetaram experimentos para encontrar qual a perda de desempenho causada pela virtualização. Eles descobriram que os aplicativos estruturados em máquinas virtuais ou *containers* têm a perda máxima de 3,33% de desempenho em relação aos mesmos aplicativos estruturados monoliticamente em máquinas físicas [30].

O desenvolvimento de simulações distribuídas que se valham de sistemas de tempo real, que necessitem de desempenho adequado para uma boa experiência do usuário, requer estratégias que reduzam a latência entre as aplicações.

As medições de desempenho nos testes realizados no estudo de caso com o Projeto Arcanjo seguiram as mesmas características relatadas em [9]: a troca de pacotes de dados entre os Federados com tamanho de 30 a 60 bytes, realizando a medição do tempo absoluto de 20 trocas completas desses pacotes.

Os resultados dessas medições são comparadas com a referência, que é uma infraestrutura de simulação HLA sem o uso de microsserviços.

Em [9], os testes de comunicabilidade entre os Federados proporcionaram resultados de degradação do desempenho de 16% a 26%, dependendo das configurações de rede para obtenção de uma RTI em modo LAN e WAN. Essa queda de desempenho reflete o ambiente *multi-host* dos *containers*

Docker, acrescentando camadas extras na comunicabilidade das aplicações.

Neste artigo, o estudo de caso utilizou uma configuração de *host* único, obtendo uma degradação de desempenho (em comparação com a mesma referência) de 7,6%. O objetivo de conteinerizar os Federados com a estratégia de *host* único evitou o acréscimo de barreiras de comunicação na Camada 3 do modelo OSI de rede [31] em comparação com [9], proporcionando menor queda de desempenho geral nos testes realizados.

VII. Conclusão

Este trabalho de pesquisa objetivou apresentar um levantamento sobre o uso de microsserviços aliado à infraestrutura de simulação distribuída em ambiente de computação em nuvem. Ao utilizar como parâmetro a arquitetura prevista nas regras e documentações do HLA, nele, mostrou-se o desenvolvimento de uma prova de conceito capaz de unir aplicações antes monolíticas e transformá-las em subsistemas de um mundo virtual mais complexo.

A implementação de uma RTI que não dependa de um Federado central para existir (CRC) tornou possível a redução da complexidade arquitetural, minimizando os prejuízos na implantação de uma HLA através da utilização do paradigma SOA e de microsserviços.

As diferenças entre os resultados obtidos neste artigo e o principal trabalho correlato [9] refletem uma abordagem distinta para obtenção da infraestrutura de simulação distribuída, colocando em um plano secundário a obtenção das características de MSaaS e focando no melhor desempenho para a simulação de sistemas de tempo real em uma arquitetura estruturada sob o *framework* HLA e obtida através de microsserviços.

Agradecimentos

Os autores deste artigo agradecem ao Comando da Aeronáutica (COMAER), ao Instituto Tecnológico de Aeronáutica (ITA) e ao Programa de Pós Graduação em Aplicações Operacionais (PPGAO), pelo suporte e infraestrutura oferecidos durante todo o desenvolvimento deste trabalho de pesquisa.

Referências

[1] A. G. Bruzzone and M. Massei, "Simulation-based military training," in *Guide to Simulation-Based Disciplines*, S. Mittal, U. Durak, and T. Ören, Eds. Springer, Cham, 2017, ch. 14, pp. 315–350.

[2] M. M. Gunal, *Simulation for Industry 4.0. Past, Present, and Future*. Springer, 2019.

[3] Docker. [Online]. Available: https://www.docker.com/

[4] Brasil, "Portaria Normativa Nº 1.814/MD, de 13 de junho de 2013," *Diário Oficial da União*, 2013.

[5] Okan and H. Oğuztüzün, *Guide to Distributed Simulation with HLA*. Springer, 2017. [Online]. Available: http://link.springer.com/10.1007/978-3-319-61267-6

[6] X-Plane 11 Flight Simulator. [Online]. Available: https://www.x-plane.com/

[7] Microsoft Flight Simulator: Standard - Microsoft Store. [Online]. Available: https://www.microsoft.com/pt-br/p/microsoft-flight-simulator-standard/9nxn8gf8n9ht?activetab=pivot:overviewtab

[8] B. P. T. Moller, "The HLA Tutorial," p. 101, 2012. [Online]. Available: http://www.pitchtechnologies.com/wp-content/uploads/2014/04/TheHLAtutorial.pdf

[9] T. van den Berg, B. Siegel, and A. Cramp, "Containerization of high level architecture-based simulations: A case study," *Journal of Defense Modeling and Simulation*, vol. 14, no. 2, pp. 115–138, 2017.

[10] SISO-STD-001.1-2015: Standard for Real-time Platform Reference Federation Object Model. [Online]. Available: https://www.sisostds.org/DesktopModules/Bring2mind/DMX/API/Entries/Download?Command=Core_Download&EntryId=30823&PortalId=0&TabId=105

[11] Object Management Group Acronyms. [Online]. Available: https://www.omg.org/gettingstarted/terms_and_acronyms.htm

[12] P. S. Kocher, *Microservices and Containers*. Pearson, 2018.

[13] P. Bocciarelli, A. D'Ambrogio, E. Paglia, T. Panetti, and A. Giglio, "A cloud-based service-oriented architecture for business process modeling and simulation," *CEUR Workshop Proceedings*, vol. 2010, pp. 87–93, 2017.

[14] J. E. Hannay, T. Van Den Berg, and B. Siegel, "The NATO MSG-136 Reference Architecture for M&S as a Service," *2016 Simulation Innovation Workshop, SIW 2016*, no. October, 2016.

[15] Linux Containers. [Online]. Available: https://linuxcontainers.org/

[16] J. Fink, "Docker: a Software as a Service, Operating System-Level Virtualization Framework," 2014. [Online]. Available: https://journal.code4lib.org/articles/9669?utm{%}257B{%}255C{_}{%}257Dsource=feedburner{%}257B{%}255C{&}{%}257Dutm{%}257B{%}255C{_}{%}257Dmedium=feed{%}257B{%}255C{&}{%}257Dutm{%}257B{%}255C{_}{%}257Dcampaign=Feed{%}257B{%}255C{%}25{%}257D3A+c4lj+

[17] M. C. B. Galvão and I. L. M. Ricarte, "Revisão sistemática da Literatura: Conceituação, Produção e Publicação," *Logeion: Filosofia da Informação*, vol. 6, no. 1, pp. 57–73, 2019.

[18] M. d. S. Dias, "MEGA-RCD : Um Método de Gerenciamento de Ameaças em Rede de Comunicacão de Dados," *Simpósio de Aplicações Operacionais em Áreas de Defesa (SIGE)*, 2020.

[19] Pitch Technologies – Make Your Systems Work Together. [Online]. Available: http://pitchtechnologies.com/

[20] M. Richards, *Microservices versus SOA*, 1st ed. O'Reilly Media, 2015.

[21] STANREC 4800 - AMSP-04 NATO Education and Training Network Federation Agreement and FOM Design. [Online]. Available: https://www.sto.nato.int/publications/STO%20Meeting%20Proceedings/STO-MP-MSG-159/MP-MSG-159-12.pdf

[22] Spring Boot Framework. [Online]. Available: https://spring.io/

[23] The portico project. [Online]. Available: http://porticoproject.org/

[24] Live Flight Tracker - Real-Time Flight Tracker Map. Flightradar24. [Online]. Available: http://www.flightradar24.com/

[25] IEEE, "Std 1516: IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules," *IEEE Std 1516-2010 (Revision of IEEE Std 1516-2000)*, pp. 1–38, 2010.

[26] Web Services Architecture. 3.1.3 Relationship to the World Wide Web and REST Architectures. World Wide Web Consortium. [Online]. Available: https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest

[27] X. Liu, Q. He, X. Qiu, B. Chen, and K. Huang, "Cloud-based computer simulation: Towards planting existing simulation software into the cloud," *Simulation Modelling Practice and Theory*, vol. 26, pp. 135–150, 2012. [Online]. Available: http://dx.doi.org/10.1016/j.simpat.2012.05.001

[28] A. Menon, J. R. Santos, Y. Turner, G. Janakiraman, and W. Zwaenepoel, "Diagnosing performance overheads in the xen virtual machine environment," *Proceedings of the First ACM/USENIX International Conference on Virual Execution Environments, VEE 05*, pp. 13–23, 2005.

[29] D. Gupta, R. Gardner, L. Cherkasova, H.-p. Laboratories, and P. Alto, "XenMon : QoS Monitoring and Performance Profiling Tool," *Architecture*, pp. 1–13, 2005.

[30] X. Song, Y. Ma, and D. Teng, "A load balancing scheme using federate migration based on virtual machines for cloud simulations," *Mathematical Problems in Engineering*, vol. 2015, 2015.

[31] J. D. Day and H. Zimmermann, "The OSI Reference Model," in *Proceedings of The IEEE*, vol. 71, no. 12. IEEE, 1983, pp. 1–10.