Explorando Estilos Arquiteturais para Servir Sistemas Inteligentes

Washington Luiz Meireles de Lima
Ygor Tavela Alves da Silva

Proposta de Trabalho de Formatura Supervisionado
apresentado à disciplina
MAC0499

Orientadores: Prof. Dr. Alfredo Goldman
Coorientador: Renato Cordeiro Ferreira

São Paulo, Abril de 2022

Conteúdo

Capítulo 1

Introdução

Um sistema de software que possui uma inteligência que evolui e melhora com o tempo, particularmente analisando como os usuários interagem com o sistema, é referido como um sistema inteligente (Hulten , 2019). Sistemas inteligentes podem possuir variadas finalidades: uma simples tradução feita no *Google* Tradutor, recomendações de *playlists* com músicas adequadas ao perfil de qualquer pessoa com o *Spotify*, até mesmo revolucionando a pesquisa em biocomputação com o *AlphaFold* (Ewen Callaway , 2022). Tais sistemas são semelhantes à sistemas tradicionais não inteligentes, principalmente no que diz respeito a ter um objetivo e entregar valor aos usuários finais. No entanto, a inteligência oriunda de modelos de aprendizado de máquina caracterizam unicamente tais sistemas.

Dentro do ciclo de vida de um sistema inteligente, podemos dividir uma aplicação de aprendizado de máquina em três eixos de mudança (Sato **et al.** , 2019): dados, modelo, e código. Os dados e modelos, concedem uma característica particular aos sistemas inteligentes. O sistema se encontra em um estado de constante evolução, o que o torna mais complexo, mais difícil de entender e, mais difícil de testar. Tal fato, gera influências em como devemos conceber uma arquitetura de software adequada para um sistema inteligente, isto é, como o eixo de código deve se estruturar para comportar as mudanças promovidas pelo tempo.

Assim como sistemas de software tradicionais, o processo de design de sistemas inteligentes não deve garantir apenas que o sistema funcione com um propósito claro, mas também, que seja um sistema robusto com um longo tempo de vida útil e de baixo custo. Essas características estão diretamente relacionados com a concepção de uma boa arquitetura de software. A arquitetura de software de um sistema é uma *forma* dada para o sistema por aqueles que o constroem. Em que a estrutura desta *forma*, se traduz em como os componentes do sistema são divididos, como os componentes estão dispostos e, como os componentes se relacionam entre si. Tal *forma*, tem como objetivo garantir uma base sólida para o sistema, proporcionando um sistema fácil de entender, de fácil desenvolvimento e também, um fácil processo de servir a aplicação para o cliente (Martin , 2017).

Dentro da arquitetura de um sistema inteligente, surge um importante questionamento relacionado à escalabilidade (Lakshmanan **et al.** , 2020): **Como servir um modelo de tal forma que ele suporte milhões de requisições de predições em um curto período de tempo?** Para abordar tal questionamento, além de ser necessário entender melhor o objetivo do modelo, é fundamental entender como será estabelecido a comunicação entre quem serve e quem consome o modelo em nossa arquitetura de software. Comumente, a principal característica de uma comunicação a se determinar é a sua natureza temporal, síncrona ou assíncrona, e à partir disso escolher o protocolo mais adequado para estabelecer a comunicação.

Dado as peculiaridades de um sistema inteligente e as necessidades de se construir uma boa arquitetura de software, o objetivo deste trabalho de pesquisa, é o de explorar padrões arquiteturais para servir um sistema inteligente. Buscando avaliar diversos cenários e os *trade-offs* para cada padrão de entrada adotado. Consequentemente, encontrar os principais prós e contras para cada abordagem, além de, melhor discutir a aplicabilidade de cada um dos padrões.

De forma geral, neste documento será descrito a proposta de desenvolvimento para explorar alternativas para servir um modelo num sistema inteligente. No Capítulo 2, serão discutidos conceitos relevantes para o que está sendo discutido. No Capítulo 3 é detalhado a proposta deste trabalho. No Capítulo 4, é apresentado um plano de projeto para executar o trabalho.

Capítulo 2

Revisão de Literatura

2.1 Sistemas Inteligentes

Sistemas Inteligentes são aqueles em que existe alguma inteligência (aprendizado de máquina) aprendendo e evoluindo com dados. Por este motivo, a implementação de um sistema inteligente impõem diferentes exigências que os diferenciam de sistemas não inteligentes. O ciclo de vida de tais sistemas dependem: em como criar a inteligência, como mudar a inteligência, como organizar a inteligência, e como lidar com erros ao longo do tempo. Para isso, construir um sistema inteligente efetivo requer balancear cinco componentes principais (Hulten , 2019):

- Como definir um objetivo que seja claro;
- Como apresentar a saída dos modelos de aprendizado aos usuários;
- Como executar a inteligência;
- Como criar uma inteligência que cumpra com o seu objetivo;
- Como orquestrar o ciclo de vida da inteligência.

2.2 Aprendizado de Máquina

A área da inteligência artificial, ou IA, assim como na filosofia e psicologia, busca entender o funcionamento de agentes inteligentes, e formas de contruí-los também (Russel e Norvig , 2012). Algumas definições de IA podem se agrupar em quatro categorias de sistemas:

1. que pensam como humanos,
2. que pensam racionalmente,
3. que agem como humanos, e
4. que agem racionalmente.

As definições 1 e 3 se baseiam na capacidade humana e em estudos empíricos, envolvendo hipóteses e experimentos. Enquanto que, 2 e 4, baseiam-se no conceito ideal de inteligência, tido como racionalidade, no qual combinam-se matemática e engenharia. Dentre essas definições, a quarta é onde se enquadra o Aprendizado de Máquina.

O Aprendizado de Máquina é uma área de IA que se preocupa em construir algoritmos através de dados, os quais podem vir da natureza, criados pelos humanos ou gerados por outros algoritmos. Pode ser definido também pelo processo de coleta de um conjunto de dados e pelo treinamento de um modelo estatístico usando esse conjunto através de algum algorítimo de aprendizado.

2.3 Arquitetura de Software

A arquitetura de software define as partes que compõem o software, como são as sua estruturas e como elas se relacionam entre si. A estrutura de um software, é o que permite que ele seja flexível o suficiente para que rapidamente evolua e mude o seu comportamento para atender uma dada necessidade, ou seja, é o que o torna *soft* o suficiente para deixar o maior número de opções disponíveis pelo maior tempo possível (Martin , 2017).

2.3.1 Padrões de Design

Padrões de design é uma maneira barata, rápida, e eficiente de resolver problemas comuns em programação (Beck , 2007). Cada padrão engloba um problema em específico, com a discussão de fatores que afetam o problema e um conselho de como resolver o problema de forma rápida para criar uma solução satisfatória. No design de arquitetura de software, os padrões são uma importante ferramenta para os desenvolvedores, contribuindo para reduzir o gasto de tempo e energia para definir uma forma de resolver um problema.

2.3.2 Estilos Arquiteturais

Um estilo de arquitetura é a inspiração que está por trás da ideia da arquitetura. Tal inspiração é definida como um conjunto de padrões (2.3.1) a serem adotados para construir uma estrutura que atenda os requisitos funcionais e não funcionais do sistema. Mais especificamente, um estilo arquitetural determina o *vocabulário* de componentes e as formas de interação entre eles que podem ser usadas nesse estilo, em um conjunto de restrições (GARLAN e SHAW , 1993).

2.3.3 Arquitetura Hexagonal

A arquitetura hexagonal é um estilo arquitetural cuja motivação é a de separar as regras de negócio da aplicação dos detalhes de implementação, tais como o *framework*, a interface de usuário, banco de dados, etc (Cockburn , 2005). Essa arquitetura segue o fortemente o princípio *Dependency Rule* (Martin , 2017), na qual as dependências sempre devem apontar para camadas mais internas, que por sua vez devem ser completamente ignorantes em relação às camadas externas.

2.3.4 Por quê uma boa arquitetura é importante?

Além do próprio valor entregue pela solução do software em si, uma outra ótica a se avaliar o valor de um software se da pela sua estrutura (Martin , 2017). Muitas vezes por não ser algo aparente aos usuários finais ou, pelo custo de tempo e esforço, a arquitetura acaba sendo deixada de lado no processo de desenvolvimento do software. Desta forma, é importante notar que o design de arquitetura não se refere apenas ao processo inicial de desenvolvimento, mas sim à todo ciclo de vida de um sistema.

Via de regra um maior tempo de vida de sistema sempre irá beneficiar o uso de boas práticas de desenvolvimento, independentemente de todo gasto com tempo e esforço para o desenvolvimento de um software de alta qualidade (Martin Fowler , 2019). Com isso, a principal finalidade de um bom design de arquitetura é o de reduzir custos não de desenvolvimento, aumentando a compressão do código pelos programadores, melhorando a manutenibilidade do sistema, facilitando o entendimento da divisão entre requisitos importantes e requisitos que são detalhes ao sistema, etc.

2.4 Padrões de comunicação

A integração entre diferentes aplicações lida com diversos desafios: conexão não confiável e lenta, diferenças nas aplicações e mudanças inevitáveis. Algumas das formas de integração são: arquivo de transferência, banco de dados compartilhado, invocação de procedimento remoto e mensagem. Todas essas abordagens têm as suas vantagens e desvantagens, por isso, não é incomum que uma

aplicação use múltiplas formas de integração afim de minimizar essas desvantagens (Hohpe e Woolf , 2012).

2.4.1 *API REST*

API é o acrônimo de *Application Programming Interface*, sendo um conjunto de definições e protocolos usados no desenvolvimento e na integração de aplicações. Pode ser entendida como um contrato entre o provedor e o consumidor, em que o primeiro estabelece padrão de comunicação e o segundo usa esse padrão para executar ações, obter ou transferir informações.

O estilo *REST*, *Representational State Transfer*, é definido por um conjunto de restrições de arquitetura (Fielding , 2000): interface uniforme, cliente-servidor, *stateless*, *cache*, sistema em camadas e código por demanda. Dentre essas restrições, a interface uniforme é que faz a distinção da arquitetura *REST* das demais. Nela são aplicados os princípios de generalização da engenharia de software para os componentes de interface, promovendo uma simplificação na arquitetura do sistema e maior visibilidade das interações. Para garantir a uniformidade da interface, quatro restrições são impostas: identificação dos recursos, manipulação dos recursos por representações, mensagens auto descritivas e, hipermídia como motor do estado da aplicação.

Uma importante característica de uma *API REST* é a sua natureza síncrona de comunicação, o que cria um grande acoplamento entre o cliente e o servidor. Um grande benefício, no entanto, é a sua baixa complexidade para implementação.

2.4.2 Mensageria

Um sistema de mensageria é responsável por coordenar e gerir o envio e recebimento de mensagens, assim como a leitura e persistência delas (Hohpe e Woolf , 2012). Um dos padrões de design usados é *publish-subscribe* na qual um *publish* publica alguma mensagem categorizada, sem destinatários específicos, e um *subscribe* pode receber essa mensagem caso tenha interesse nessa categoria. Neste padrão, tanto quem publica e quem se inscreve para receber a publicação não tem conhecimento um do outro, apenas das categorias. O sistema responsável por manter, filtrar ou rotear as publicações costumam é comumente referido como um *message broker*.

Nesse padrão, a comunicação é naturalmente assíncrona, tendo em vista que o *publisher* e o *subscriber* não estabelecem uma comunicação direta. Tal fato, contribuí para a redução do acoplamento, o que pode melhorar por exemplo, a questão de disponibilidade do sistema. Mas uma desvantagem é de que para estabelecer uma comunicação via mensagens, é necessário o uso de um *message broker*.

Capítulo 3

Proposta

Neste trabalho, o objeto de estudo será concentrado em um dos cinco componentes de um sistema inteligente: como executar a inteligência de um sistema inteligente. Mais precisamente, tentar responder empiricamente: qual a forma mais adequada para servir sistema inteligente.

É pretendido criar um sistema inteligente de *benchmark*, preferencialmente usando o estilo arquitetural Hexagonal (2.3.3), para que seja diminuído o acoplamento entre os componentes e seja fácil suportar modos distintos para servir um dado modelo de aprendizado de máquina. Tal modelo à princípio será um *mock* já que o objetivo do trabalho não é focar na inteligência de um sistema inteligente, possibilitando a criação de cenários com características específicas para simular diferentes modelos que possam existir no mundo real.

3.1 Sistema de *benchmark*

O objetivo é de planejar e desenvolver um sistema inteligente (2.1) de *benchmark* adotando o estilo arquitetural Hexagonal (2.3.3). Definindo aspectos essenciais para criar um sistema inteligente pragmático aos detalhes de implementação, como por exemplo, o padrão de comunicação a ser utilizado (2.4) ou o modelo de aprendizado (2.2). Além disso, serão criados testes a serem utilizados para realizar um posterior estudo comparativo (3.2).

Vale destacar que será utilizado um modelo *mock* que simula os efeitos de um modelo real. A implementação não está definida, mas ela deverá ser capaz de aceitar parâmetros que irão influenciar sua execução e resposta. O uso do modelo *mock* têm algumas motivações. Primeiro, a construção de um modelo real não é o objetivo do trabalho e isso poderia consumir um tempo considerável. Segundo, a abstração do modelo permite um estudo não centrado nas especificidades ou vieses que ele possa trazer. Por último, através da simulação, podemos definir e controlar os parâmetros de configuração, entrada e saída do modelo, ou seja, um ambiente controlado para experimentos.

3.2 Estudo comparativo

O objetivo do estudo comparativo é o de elaborar, coletar e interpretar métricas importantes relacionadas à como os diferentes padrões de comunicação (2.4) se comportam no sistema de *benchmark* (3.1). Analisando e comparando os resultados dos testes criados será possível responder uma importante questão de escalabilidade relacionada ao design de sistemas inteligentes.

Capítulo 4

Plano de Projeto

Nesta seção é destacado cada uma das etapas para cumprir com o objetivo deste trabalho. Além disso, é apresentado um cronograma esperado para realizarmos cada uma das etapas apresentadas.

4.1 Etapas

4.1.1 Elaboração da Proposta

Apresentação da proposta para o Trabalho de Formatura Supervisionado, destacando o que será feito e os seus objetivos. Tal etapa, tem como resultado este documento de proposta em questão.

4.1.2 Revisão de Literatura

De forma enxuta, é apresentado conceitos importantes para o desenvolvimento deste trabalho. Tal etapa será realizado ao longo de todo trabalho, e o resultado será reunido no Capítulo 2.

4.1.3 Planejamento e implementação do sistema de *benchmark*

Nesta etapa serão cumpridos os objetivos propostos em 3.1.

4.1.4 Estudo Comparativo

Nesta etapa serão cumpridos os objetivos propostos em 3.2.

4.1.5 Elaboração da Monografia

Processo de escrita e revisão da monografia do trabalho. Tal etapa será realizada ao longo do ano como uma tentativa de reduzir a fricção e a dificuldade inerente ao processo de escrita. Destacando que a monografia já está sendo materializado com boa parte do conteúdo deste documento de proposta.

4.2 Cronograma

Etapas	Abr	Maio	Jun	Jul	Ago	Set	Out	Nov	Dez
4.1.1	X								
4.1.2	X	X	X	X	X	X	X	X	X
4.1.3		X	X	X	X	X	X		
4.1.4				X	X	X	X	X	X
4.1.5	X	X	X	X	X	X	X	X	X

Bibliografia

Beck (2007) Kent Beck. **Implementation Patterns**. Citado na pág. 4

Cockburn (2005) Alistair Cockburn. Hexagonal Architecture, 2005. Citado na pág. 4

Ewen Callaway (2022) Ewen Callaway. What's next for AlphaFold and the AI protein-folding revolution. **https://www.nature.com/articles/d41586-022-00997-5**. Citado na pág. 1

Fielding (2000) Roy Thomas Fielding. **Architectural Styles and the Design of Network-based Software Architectures**. Tese de Doutorado. Citado na pág. 5

GARLAN e SHAW (1993) DAVID GARLAN e MARY SHAW. AN INTRODUCTION TO SOFTWARE ARCHITECTURE. doi: 10.1142/9789812798039{_}0001. Citado na pág. 4

Hohpe e Woolf (2012) Gregor Hohpe e Bobby Woolf. **Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions (Google eBook)**. Citado na pág. 5

Hulten (2019) Geoff Hulten. **Building Intelligent Systems**. Apress, Berkeley, CA. ISBN 978-1-4842-3933-9. doi: 10.1007/978-1-4842-3933-9. Citado na pág. 1, 3

Lakshmanan et al. (2020) Valliappa Lakshmanan, Sara Robinson e Michael Munn. **Machine Learning Design Patterns: Solutions to Common Challenges in Data Preparation, Model Building, and MLOps** . Citado na pág. 1

Martin (2017) Robert C Martin. **Clean Architecture: A Craftsman's Guide to Software Structure and Design**. Citado na pág. 1, 4

Martin Fowler (2019) Martin Fowler. Is High Quality Software Worth the Cost?, 2019. Citado na pág. 4

Russel e Norvig (2012) Stuart Russel e Peter Norvig. **Artificial intelligence—a modern approach 3rd Edition**. doi: 10.1017/S0269888900007724. Citado na pág. 3

Sato et al. (2019) Danilo Sato, Arif Wider e Christoph Windheuser. Continuous Delivery for Machine Learning. **Martin Fowler**. Citado na pág. 1