Comparação entre Paradigma orientado a notificações, Paradigma Imperativo orientado a objetos e programação reativa no desenvolvimento de um controlador de semáforos

Fabio Negrini
Universidade Tecnológica Federal do Paraná
Curitiba, Brazil
mail: fnegrini@gmail.com

***Resumo*—No contexto de linguagens de programação atual muito se fala sobre vantagens e desvantagens de cada modelo sobre um problema computacional aplicado à situações específicas. O referito trabalho apresenta uma análise. Aaproveitando-se de um trabalho já desenvolvido anteriormente [1] aplicado à uma problemática específica sobre o prisma de performance: um controlador de tráfego programado para trabalhar com três estratégias distintas: Tráfego independente, controle baseado em congestionamento, controle baseado em tráfego facilitado (onda verde). Este, por sua vez, comparando comparou apenas o paradigma imperativo e o paradigma orientado a objetos, porém este somente sob o prisma de performance,. o O trabalho a seguir vem complementar esta análise fazendo comparativo de três paradigmas: orientado a objetos, orientado a notificações e programação reativa. aplicado à uma problemática específica: um controlador de tráfego programado para trabalhar com três estratégias distintas: Tráfego independente, controle baseado em congestionamento, controle baseado em tráfego facilitado (onda verde). Os comparativos contidos neste trabalho, diferente de seu trabalho base, serão focados em complexidade e tempo para desenvolerdesenvolver dos códigos propostos. O fato de não ser reconhecido como um paradigma, a programação reativa será considerada como tal, pois os pontos aqui analisados trazem as mesmas problemáticas podendo então ser analisado e comparado com os dois modelos já considerados paradigmas. Será possível verificar, então, que o PON mantem uma quantidade de código equivalente aos demais paradigmas comparados mas com um tempo de desenvolvimento consideravelmente menor.**

I. Introdução

O paradigma orientado a notificações (PON) é um novo paradigma para desenvolvimento de *softwares*, ele foi inicialmente concebido por Simão em [2]. O paradigma imperativo (que abrange os paradigmas procedual e orientado a objetos) apresenta alguns problemas, como o uso de *loops* em elementos passivos e expressões causais que causam redundâncias na execução. O paradigma declarativo soluciona algumas das redundâncias, porém faz uso de estruturas de maior custo computacional, e apresenta uma menor flexibilidade que o paradigma imperativo. O PON tem por base a criação de pequenas entidades desacopladas que colaboram entre si, com o uso de notificações, gerando inferências. Seu uso permite uma maneira mais fácil de compor *software*, seja de forma local ou distribuída [3] [4] [5].

O crescimento do PON depende muito de seus resultados frente aos paradigmas com reconhecidos resultados, seja no campo da eficiência bem como na arquitetura do código englobando aí vários aspectos como facilidade de entendimento do código, agilidade no desenvolvimento e verbosidade do mesmo. Este artigo vem apresentar então um comparativo entre o PON e POO em duas formas: POO clássico e programação reativa (PR).

O artigo está organizado na seguinte forma: a seção II irá apresentar o simulador e seu detalhamento. A seção III irá apresentar resumidamente os paradigmas e suas principais características. A seção IV apresentará os resultados obtidos neste experimento e os comparativos entre os paradigmas. A seção V apresentará conclusões finais e propostas para trabalhos futuros.

II. Simulador

O simulador CTA foi construído com o intuito de gerar um ambiente de tráfego intenso, dentro deste ambiente de simulação então é possível gerar eventos que simulem um ambiente muito próximo ao real de tráfego permitindo a simulação de diversas intensidades de automóveis. Desta forma é possível analisar o comportamento dos controladores de tráfego que são o alvo do comparativo deste trabalho. O simulador representa elementos do mundo real, como veículos, ruas, pistas, quadras, sinaleiros, sensores e cruzamentos em uma região de simulação matricial composta por dez ruas verticais e dez ruas horizontais de mão única, formando uma matriz 10x10 com um total de 100 interseções [6]. O simulador permite variar o número de pistas (1 a 4) e o intervalo de tempo que um veículo é adicionado à simulação (0.1, 0.2, 0.3, 0.4 ou 0.5 veículos por segundo). Cada intersecção de ruas horizontal e vertical possui um semáforo que está sempre sincronizado de forma a nunca possuir o sinal verde para ambas as direções ao mesmo tempo. Em todas as quadras são instalados sensores que monitoram a quantidade de veículos que estão

parados num sinal vermelho. Esses sensores apresentam três estados: FEW, MANY e FULL Falar acerca dos três paradigmas, suas semelhanças e porque foram escolhidos [7]. Diante deste ambiente de simulação então, estão definidos três tipos de estratégias de controle: controle independente, controle baseado em congestionamento e controle baseado em tráfego facilitado. No controle independente, cada semáforo possui tempos fixos para cada estado do sinaleiro, não sendo considerados os sensores de quantidade de veículos e tempos de semáforos vizinhos. Na estratégia de controle baseada em congestionamento é avaliado o tempo de cada semáforo e o estado referente à quantidade de veículos parados. Se o sensor detecta que a porcentagem de veículos parados está entre 60% até 100%, e o tempo do sinaleiro em vermelho é menor do que 24 segundos, então o tempo total do sinaleiro no estado vermelho é ajustado para 30 segundos. Caso o sensor detecte que a taxa de ocupação está entre 60% e 100% e o tempo do semáforo vermelho está entre 25 segundos e 39 segundos, o sinaleiro oposto altera imediatamente para o estado amarelo e o tempo restante do sinaleiro no estado vermelho é ajustado para 6 segundos. Se o sensor detectar que a ocupação está entre 60% e 100%, e o tempo do sinaleiro em vermelho for maior do que 39 segundos, não é realizada nenhuma alteração no tempo do sinaleiro. Por fim, a estratégia de controle baseada em tráfego facilitado é uma extensão da estratégia de controle baseado em congestionamento, ou seja, esta possui todos os comportamentos descritos com uma inteligência adicional que é uma intercomunicação entre os semáforos para atuação conjunta. Neste modelo, os semáforos possuem uma indicação de que devem contribuir para o tráfego facilitado em apenas uma direção (horizontal ou vertical) de forma a priorizar o comportamento apenas no sentido indicado. Todos os semáforos conhecem o próximo semáforo na direção indicada para o tráfego facilitado e, quando passa a acelerar a abertura do sinal verde em função do aumento de ocupação, este notifica o próximo semáforo que também acelerará abertura do sinal de forma a garantir uma sequência uniforme de abertura de sinais.

III. Apresentação dos paradigmas

Neste trabalho foram desenvolvidas as seguintes estratégias e paradigmas:

1) Controle independente (CI) em paradigma orientado a objetos (POO)
2) Controle independente (CI) em paradigma orientado a notificações (PON)
3) Controle baseado em congestionamento (CBCL) em paradigma orientado a objetos (POO)
4) notificações (PON)
5) Controle baseado em tráfego facilitado (CBTF) em paradigma orientado a objetos (POO)
6) Controle baseado em tráfego facilitado (CBTF) em paradigma orientado a notificações (PON)
7) Controle independente (CI) em programação reativa (PR)
8) Controle baseado em congestionamento (CBCL) em programação reativa (PR)
9) Controle baseado em tráfego facilitado (CBTF) em programação reativa (PR).

Toda a codificação e compilação foi feita em Microsoft Visual Studio C++ 2017, pois como será verificado nos próximos capítulos esta linguagem consegue interagir com os três modelos propostos. Nos próximos tópicos falaremos brevemente sobre cada um dos paradigmas aqui pesquisados para uma leitura mais concisa dos seus resultados.

A. Paradigma Orientado a Objetos

O paradigma orientado a objetos (POO) é um dos mais bem-sucedidas e penetrantes paradigmas das áreas de informática [8]. Teve sua origem na década de 1960 com a Simula 67 [9] [10] [11], mas teve sua maior popularidade atingida na área industrial com o advento do C++ no início da década de 80 [12]. O POO tem como princípio a abstração de conceitos do mundo real em unidades de software chamadas de objetos interagentes e compostas. Segundo Van Roy [8], os principais elementos deste paradigma são:

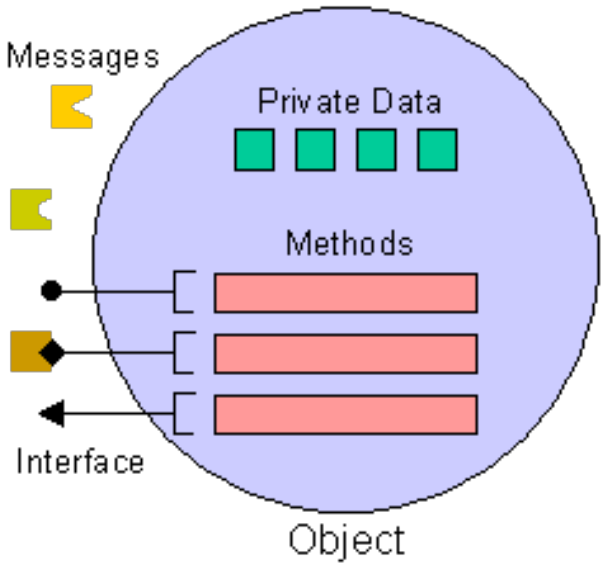

Figura 1. Representação gráfica do paradigma orientado a objetos

- *Classe*: a conjunto de objetos com características afins. A classe define o comportamento de seus objetos através de seus métodos e seus estados através de atributos.
- *Objeto*: a instância de uma classe. O objeto é capaz de armazenar seu próprio estado e interagir com outros objetos mediante mensagens enviadas e recebidas
- *Atributo*: são as características ou estados de um objeto
- *Método*: define a habilidade ou comportamento do objeto conforme definido na classe à qual foi instanciado. Além destes elementos principais, o POO também possui vários

conceitos primordiais, os quais são descritos sucintamente abaixo:

- *Mensagem*: a chamada a um método de um objeto, ativando o comportamento descrito em sua classe.
- *Herança*: o mecanismo pelo qual uma classe pode estender outra classe compartilhando comportamentos e atributos em comum a fim de reaproveitamento de código.
- *Associação*: o mecanismo pelo qual um objeto utiliza recursos de outro objeto. Pode ser simples (usa um) ou acoplamento (todo parte).
- *Abstração*: é a habilidade de concentrar nos aspectos essenciais de um contexto qualquer, ignorando características menos importantes ou acidentais.
- *Polimorfismo*: consiste no princípio pelo qual duas ou mais classes derivadas de uma mesma superclasse podem invocar métodos que têm a mesma identificação (assinatura) mas comportamentos distintos, especializados para cada classe.
- *Interface*: é um contrato entre a classe e o mundo externo. Quando uma classe implementa uma interface, ela está comprometida a fornecer o comportamento publicado pela interface
- *Pacotes ou namespaces*: são referências para organização lógica de classes e interfaces

Para o paradigma orientado a objetos foi utilizado o C++, pois o mesmo é compilável na ferramenta Microsoft Visual Studio 2017 utilizada neste experimento.

B. Paradigma Orientado a Notificações

O PON foi proposto como um novo paradigma de desenvolvimento de software, que apresenta algumas vantagens quando comparado aos paradigmas tradicionais (mais especificamente, o PI Paradigma Imperativo - e o PD Paradigma Declarativo) no que diz respeito ao seu modelo lógico. Tais vantagens são constituídas por uma maior facilidade na concepção de sistemas que apresentem paralelismo ou distribuição, além da redução ou eliminação de alguns dos problemas clássicos de software PI e PD, tais como redundâncias de execução e acoplamento excessivo entre entidades computacionais [15].

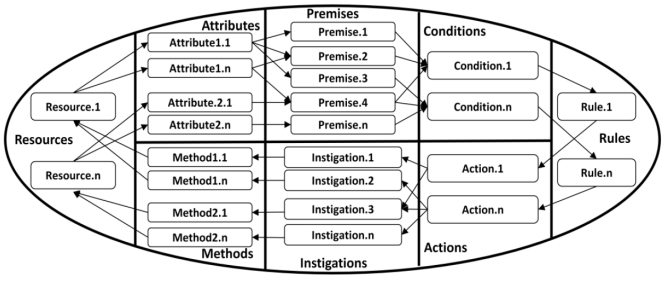

Figura 2. Fluxo de ativação do paradigma orientado a notificações

Estruturalmente, o software PON é representado na forma de Base de Fatos (FBE Fact Base Element) e as Regras (Rules). Os elementos FBE são utilizados para representar objetos do mundo real em um sistema computacional, por meio de estados (atributos) e serviços (métodos). Os elementos Rules, por sua vez, definem o cálculo lógico causal a ser efetuado sobre os estados dos FBEs, controlando a execução dos seus serviços. A colaboração entre estes elementos ocorre por meio de notificações diretas, que é um processo de inferência essencialmente distinto dos processos utilizados em software PI e em Sistemas Baseados em Regras (SBR) do PD [13]. Desta forma, o PON proporciona uma execução livre de avaliações redundantes e desnecessárias criando uma estrutura com um alto grau de desacoplamento ao mesmo tempo que permite um paralelismo natural dos eventos.

Para o experimento deste artigo, foi utilizado o LingPON[14], uma versão ainda experimental do paradigma PON. Esta linguagem passou por várias versões e evoluções[23] que permite que sejam gerados diversos tipotipos se saídas como resultado de sua compilação. Para este experimento foi utilizada a materialização em C++ OO pelo fato de que desta forma o produto deste código possa então ser aproveitado ao restante do projeto que, como já foi falado, está desenvolvido em Visual Studio C++. Contudo, a materialização em C++ OO não pode ser diretamente aproveitada, pois foi necessário uma adaptação manual desta para que então pudesse se integrar ao projeto. Importante ressaltar que, para fins de comparativo não foi considerado o tempo de adaptação da compilação ao projeto não foi considerada para o comparativo, pois esta atividade se dá somente pela baixa maturidade desta versão de compilação.

C. Programação Reativa

Se apresentando como uma nova forma de pensar sistemas, a programação reativa é a mais nova das tendências que vêm influenciando o dia a dia dos desenvolvedores. A Programação reativa baseia-se em fluxos de dados e na propagação de mudanças, com o modelo de execução de uma linguagem de programação repercutindo automaticamente alterações através do fluxo de dados. Com a popularidade de arquiteturas orientadas a eventos, escaláveis e interativas, tanto do lado do cliente quanto do servidor, o conceito de "reatividade"está ganhando cada vez mais atenção. O paradigma reativo é fundamentado em quatro pilares[15]:

- *Elástico*: Reage à demanda/carga: aplicações podem fazer uso de múltiplos núcleos e múltiplos servidores;
- *Resiliente*: Reage às falhas; aplicações reagem e se recuperam de falhas de software, hardware e de conectividade;
- *Orientado a mensagens*: Reage aos eventos (event driven): em vez de compor aplicações por múltiplas threads síncronas, sistemas são compostos de gerenciadores de eventos assíncronos e não bloqueantes;
- *Responsivo*: Reage aos usuários: aplicações que oferecem interações ricas e tempo real com usuários.

Para este experimento foi utilizada a biblioteca Reactive.io para C++[16] que possui materialização compilável em C++. Estes fontes foram agregados ao projeto e então foi possível sua compilação junto ao projeto.

IV. Resultados

Para a apresentação deste trabalho foram desenvolvidos os controladores de semáforos em um ambiente de simulação de tráfego. Para o experimento foram apresentadas três estratégias de desenvolvimento: Controle independente, controle baseado congestionamento e controle baseado em tráfego facilitado. Para cada uma das estratégias foram desenvolvidos três modelos de codificação: Programação orientada a objetos representada em C++; programação orientada a notificações, representada pelo LingPON em sua materialização em C++ OO; por fim foi desenvolvido o paradigma orientado a objetos com programação reativa. Para este experimento, o autor possuía conhecimentos anteriores em C++ e LingPON. Não havia, portanto, nenhum conhecimento anterior referente à programação reativa. Portanto, para uma tentativa de maior objetividade (visto que não há como ter total objetividade dos valores num experimento deste), não foi considerado o tempo de estudo e aprendizado sobre programação reativa. De qualquer forma, alguma influência negativa pela pouca experiência neste modelo afetou os resultados, porém não há como excluir totalmente o nível de aprendizado do autor nos resultados.

Durante o desenvolvimento do experimento, muitas variáveis passíveis de comparação foram surgindo. Contudo, sua grande maioria gerava uma grande subjetividade em seus valores de comparação de forma a não trazer fatos objetivos, mas sim interpretação ou opinião do autor frente aos modelos abordados. Optou-se, então, focar em duas variáveis objetivas e de bastante relevância para o tema proposto neste experimento: quantidade de código e tempo de desenvolvimento. A quantidade de código, apesar de não ser a única, representa um comparativo quanto à complexidade encontrada para resolução do problema em cada modelo proposto. O tempo de desenvolvimento também é uma representação comparativa de complexidade entre os modelos. Apesar de existir uma tendência de crescimento proporcional entre a quantidade de código produzido e o tempo, será possível visualizar que esta tendência não foi reproduzida neste experimento. Serão apresentados primeiramente as conclusões do ponto de vista de complexidade de cada um dos modelos estudados, expressa em quantidade de código produzido e tempo de desenvolvimento.

A. Quantidade de código

Para avaliar a complexidade considerando-se a quantidade de código desenvolvido, foram contados o número de palavras e caracteres. Com o comparativo de contagem de palavras é possível identificar o quanto de esforço é necessário em cada modelo para desenvolver cada uma das estratégias propostas. Abaixo são apresentadas as contagens de palavras e caracteres para cada modelo implementado.

No comparativo de complexidade apresentado do ponto de vista de contagem de palavras e contagem de caracteres, foi possível verificar que a menor complexidade se deu no POO, seguido pelo PR. Este fato é possível se justificar porque tanto POO quanto PR são representações do mesmo paradigma apenas com pequenas variações focadas na forma de ativação das lógicas, mas não houve diferença das lógicas em si. O PON, no entanto, apresentou a maior quantidade de palavras. A grande diferença ocorre no comparativo entre o POO e o PON, pois há uma grande diferença de conceitos de programação entre os dois, o que deriva então o próximo tópico que virá a seguir.

Tabela I
Contagem de palavras por experimento

	CI	CBCL	CBTF
POO	130	434	610
PON	229	672	943
PR	195	514	672

Tabela II
Contagem de caracteres por experimento

	CI	CBCL	CBTF
POO	1878	6502	8771
PON	2592	6465	9053
PR	2387	7039	9316

B. Tempo de Desenvolvimento

Para este experimento o desenvolvedor possuía conhecimento prévio em POO e PON e não tinha conhecimentos anteriores sobre PR, entretanto o tempo de aprendizado foi desconsiderado para não gerar desvio nos comparativos. A tabela a seguir demonstra o tempo gasto do desenvolvimento de cada modelo em horas. Como os desenvolvimentos ocorreram em várias etapas (conforme disponibilidade do autor), a contagem é aproximada e os valores foram arredondados, pois o tempo exato em segundos não foi controlado.

Como pode ser visto na tabela anterior, apesar de apresentar uma verbosidade maior que os outros comparativos, a programação em PON foi consideravelmente mais rápida que os outros dois modelos em todas as estratégias desenvolvidas. Para o autor, o motivo desta redução no tempo de desenvolvimento se deu pela organização lógica mais concisa em relação à lógica imperativa sequencial representada nos outros modelos, ambos imperativos.

V. Conclusões e trabalhos futuros

Estudos comparativos entre linguagens e paradigmas de programação em uma problemática específica e delimitada ajudam na identificação de qual paradigma se mostra mais vantajoso frente a outros modelos. Afinal, para cada problemática há um paradigma mais adequado a solução [8]. Do ponto de vista do autor, apesar de a modelagem

Tabela III
Tempo de desenvolvimento em horas

	CI	CBCL	CBTF
POO	6:00	7:20	8:10
PON	3:50	5:10	5:50
PR	6:30	7:50	8:30

em programação orientada a objetos ser mais adequada, o LingPON apresentou uma diferença considerável no tempo de desenvolvimento. Ainda na percepção do autor, esta diferença se deu justamente pela forma mais concisa e natural de desenvolvimento com uma maior abstração das lógicas causais através de suas Rules.

Como proposta para trabalhos futuros fica a possibilidade de codificação dos controladores em outros paradigmas ou modelos de programação. Outra proposta para comparativo seria o desenvolvimento dos mesmos modelos aplicados neste experimento por um grupo de desenvolvedores a fim de auferir os valores de forma estatística. Com isto os valores refletiriam melhor a realidade de um conjunto e não apenas um indivíduo.

REFERÊNCIAS

[1] L. F. Pordeus, "Comparação entre paradigma orientado a notificações e paradigma imperativo sobre um simulador de tráfego," *CPGEI/UTFPR, Curitiba, Brazil*, 2015.

[2] J. M. Simão, "A Contribution To The Development Of A HMS Simulation Tool And Proposition Of A Meta-Model For Holonic Control," Tese. Doutorado em Engenharia Elétrica e Informática Industrial - CPGEI. Universidade Tecnológica Federal do Paraná (UTFPR), Curitiba, p. 168, 2005.

[3] J. M. Simão, C. A. Tacla, P. C. Stadzisz, and R. F. Banaszewski, "Notification oriented paradigm (nop) and imperative paradigm: A comparative study," *Journal of Software Engineering and Applications*, vol. 5, no. 6, pp. 402–416, 2012.

[4] D. L. Belmonte, A. F. Ronszcka, R. R. Linhares, R. F. Banaszewski, C. A. Tacla, P. C. Stadzisz, and M. Batista, "Notification oriented and object oriented paradigms comparison via sale system," *Journal of Software Engineering and Applications*, vol. 5, no. 9, pp. 695–710, 2012.

[5] D. L. Belmonte, M. V. Batista, R. R. Linhares, R. F. Banaszewski, C. A. Tacla, P. C. Stadzisz, and A. F. Ronszcka, "A game comparative study: Object-oriented paradigm and notification-oriented paradigm," *Journal of Software Engineering and Applications*, vol. 5, no. 9, pp. 722–736, 2012.

[6] D. P. B. Renaux, R. R. Linhares, J. M. Simão, and P. C. Stadzisz. Cta conops.

[7] J. G. Brookshear, *Computer Science: An Overview*.

[8] P. V. Roy and S. Haridi, *Concepts, Techniques, and Models of Computer Programming*.

[9] P. Naur, J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. L. McCarthy, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. H. Wegstein, A. van Wijngaarden, and M. Woodger, *Revised report on the algorithmic language ALGOL 60*.

[10] K. Nygaard and O. J. Dahl, *The Development of the SIMULA Languages*.

[11] R. J. Pooley, *An Introduction to Programming in SIMULA*.

[12] B. Stroustrup, *The C++ Programming Language, 3rd edition*.

[13] R. R. Linhares, J. M. Simao, and P. C. Stadzisz, "Noca - a notification-oriented computer architecture," *IEEE Latin America Transactions*, vol. 13, no. 5, pp. 1593–1604, May 2015.

[14] C. A. Ferreira, "Linguagem e compilador para o paradigma orientado a notificações (PON): avanços e comparações," Dissertação. Mestrado em Computação Aplicada - PPGCA. Universidade Tecnológica Federal do Paraná (UTFPR) , Curitiba, p. 227, Agosto 2015.

[15] The reactive manifesto. [Online]. Available: http://www.reactivemanifesto.org/pt-BR

[16] Reactive extensions for c++. [Online]. Available: https://github.com/Reactive-Extensions/RxCpp