Notification Oriented Paradigm (NOP): CTA Simulator

Leonardo F. Pordeus, Adriano F. Ronszcka, Jean M. Simão*, Robson R. Linhares*, Douglas P. B. Renaux*, Paulo C. Stadzisz*, Cleverson A. Ferreira*

Graduate School in Electrical Engineering and Industrial Computer Science (CPGEI), Graduate School of Applied Computing (PPGCA)* - Federal University of Technology - Paraná (UTFPR) - Av. Sete de Setembro, 3165. Curitiba-PR, Brazil. 80-230-901
{leonardopordeus | ronszcka | cleversonferreira}@alunos.utfpr.edu.br, { jeansimao | linhares | douglasrenaux | stadzisz } @ utfpr.edu.br

***Abstract*— The Notification Oriented Paradigm (NOP) presents a new approach to develop software more efficiently compared to the current programming paradigms. These paradigms have drawbacks such as redundant causal-evaluation and strongly coupled entities, decreasing performance and generating greater difficulty in parallelization and distribution. This paper presents the development of an application called CTA Simulator to compare performance and software development complexity using the NOP and the Object-Oriented Paradigm (OOP). The results showed improvement in performance when compared to the same implementation of the application in a standard paradigm. However, with respect to software development complexity, the NOP still presents some difficulties.**

***Keywords*— Notification-Oriented Paradigm, NOP and OPP Comparison**

I. INTRODUÇÃO

A capacidade de processamento dos computadores aumentou de maneira exponencial durante anos. Segundo Moore, a quantidade de transistores em um processador aproximadamente dobra a cada 24 meses [1]. Porém, em entrevista concedida no ano de 2005, Moore afirma que não se sabe até quando essa abordagem será verdadeira, devido às dimensões da estrutura dos átomos [2]. Outras formas adotadas para a melhoria no desempenho dos processadores foram a implementação de múltiplos núcleos (multicore) [3] e por meio da computação distribuída [4].

Devido aos avanços tecnológicos, com relação ao aumento da quantidade de transistores, aumento de frequência do clock, paralelismo e distribuição, muitas vezes não há uma preocupação com o desenvolvimento de software, como consequência, a não utilização dos recursos dos processadores de forma eficiente. Por outro lado, a necessidade de softwares cada vez mais complexos é maior e demanda mais recursos por parte dos processadores.

O Paradigma Orientado a Notificações (PON) é uma nova abordagem para o desenvolvimento de sistemas computacionais, possibilitando melhor desempenho, maior nível de abstração e mais apropriados ao paralelismo/distribuição do que sistemas baseados em paradigmas atuais, como a Programação Procedimental e a Programação Orientada a Objetos (POO) do Paradigma Imperativo (PI). O PON propõe uma solução para os problemas destes paradigmas, que apresentam deficiências com relação a redundâncias estruturais, temporais e forte acoplamento entre suas entidades, diminuindo o desempenho e gerando maior dificuldade de paralelização e distribuição [5][6].

Para o desenvolvimento de softwares fazendo uso do PON, foram realizadas pesquisas com framework C++ [7][8] e uma segunda versão otimizada do framework C++ [9][10][11][12], permitiam a criação de softwares PON sob abordagem de POO.

Essas pesquisas também visaram a implementação de hardware através de lógica reconfigurável [13] seguindo os conceitos do PON. Implementação em lógica reconfigurável de um co-processador PON (CoPON), uma solução híbrida, na qual parte da aplicação é executada em um núcleo von Neumann, e o mecanismo de inferência de notificações, que é executado através de um co-processador baseado em lógica reconfigurável [14]. Uma arquitetura de processador foi desenvolvida de acordo com o modelo do PON, sendo denominada Notification-Oriented Computer Architecture (NOCA) [15][16][17]. Pesquisas recentes visam o desenvolvimento de um compilador e de uma linguagem específica para o PON, denominada LingPon [18][19].

Neste artigo são apresentados os experimentos e resultados de comparações entre os paradigmas POO e PON, sendo efetuados através da análise de tempo entre a implementação de um software simulador de trânsito chamado CTA Simulator [20], na qual a parte de controle deste software é implementado tanto em POO quanto em LingPon, permitindo assim sua comparação.

As próximas seções estão organizadas da seguinte maneira: a Seção II apresenta os conceitos sobre PON. A Seção III descreve a aplicação CTA Simulator. Na Seção IV os experimentos e os resultados são apresentados. As conclusões e os trabalhos futuros são discutidos na Seção V.

II. PARADIGMA ORIENTADO A NOTIFICAÇÕES (PON)

Os elementos principais do PON são as Bases 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, através de estados (atributos) e serviços (métodos). No FBE os estados são representados por meio de objetos da classe Attribute, enquanto os serviços dessas entidades são representados pela classe Method.

O elemento que representa uma Regra é composto por uma condição, representada pelo objeto Condition e uma Ação, pelo objeto Action. O objeto Condition pode se relacionar com uma ou mais premissas, representadas pelo objeto Premise, responsáveis por verificar os atributos de um FBE. Cada premissa é composta por uma referência, um operador lógico e um valor, representados pelas classes Reference, Operator e Value respectivamente. O objeto Reference guarda a referência de um objeto Attribute de uma FBE, que notifica sobre a mudança de seu estado. O elemento Operator representa um operador lógico, para realizar comparações entre os valores dos objetos Reference e Value, o qual pode ser uma constante ou

uma referência para outro atributo. Quando todas as premissas são satisfeitas, a Regra é aprovada e notifica uma Ação. O objeto Action apresenta referências para um ou mais objetos Instigation, fazendo associação com os métodos dos elementos das Bases de Fatos.

Cada vez que o valor em um Attribute é modificado, o próprio Attribute notifica as entidades Premise relacionadas a ele. As Premises, por sua vez, são reavaliadas e, através de uma operação lógica é comparada ao novo valor do Attribute com uma constante ou um valor notificado por outro Attribute. Caso o resultado lógico da reavaliação da entidade Premises seja alterado, a Premise notifica um conjunto de entidades Conditions relacionadas a ela. Em seguida, as Conditions também têm seus estados lógicos reavaliados de acordo com os resultados lógicos das Premises. Assim, quando todas as entidades Premises que compõem uma entidade Condition apresentam seus valores lógicos verdadeiros, a entidade Condition também é satisfeita, aprovando a execução da sua respectiva Rule. Com isso, a entidade Action agregada a esta Rule é executada, invocando os Methods necessários através das entidades Instigations [7].

Na Fig. 3 é apresentado um exemplo de regra escrita através da linguagem LingPon. A Condition desta Rule contém as Premisses prSeconds e prSemaphoreState que avaliam os Atributtes atSeconds (tempo do semáforo) e atSemaphoreState (estado do semáforo). Para que a Condition seja verdadeira, o atributo atSeconds deve ser igual a 2 segundos e o atributo atSemaphoreState deve ser igual a 5. Ao aprovar as duas Premisses, é ativada a Instigation inHTLG que por sua vez instiga o Method mtHorizontalTrafficLightGreen do FBE Semaphore_NOP, apresentado na Fig. 2 e alterando o atributo atSemaphoreState para 0, abrindo o semáforo correspondente.

```
rule rlHorizontalTrafficLightGreen
    condition
        subcondition sbHorizontalTrafficLightGreen
            premise imp prSeconds semaphore_NOP.atSeconds == 2  and
            premise prSemaphoreState semaphore_NOP.atSemaphoreState == 5
        end_subcondition
    end_condition
    action
        instigation inHTLG semaphore_NOP.mtHorizontalTrafficLightGREEN();
    end_action
end_rule
```

Figura 1. Exemplo de Rule para controle independente.

```
fbe Semaphore_NOP
    attributes
        integer atSeconds 0
        integer atSemaphoreState 5
    end_attributes

    methods
        method mtResetTimer(atSeconds = 0)
        method mtHorizontalTrafficLightGREEN(atSemaphoreState = 0)
        method mtHorizontalTrafficLightYELLOW(atSemaphoreState = 1)
        method mtHorizontalTrafficLightRED(atSemaphoreState = 2)
        method mtVerticalTrafficLightGREEN(atSemaphoreState = 3)
        method mtVerticalTrafficLightYELLOW(atSemaphoreState = 4)
        method mtVerticalTrafficLightRED(atSemaphoreState = 5)
    end_methods
end_fbe
```

Figura 2. Exemplo de FBE para controle independente.

A partir deste mecanismo de notificações é possível desenvolver programas com melhor desempenho, menor número de redundâncias estruturais e temporais. Estes programas são mais apropriados para paralelismo e distribuição do que os sistemas computacionais desenvolvidos por meio das soluções baseadas em paradigmas atuais.

A Fig. 3 apresenta o mecanismo de colaboração por notificações das entidades presentes no PON.

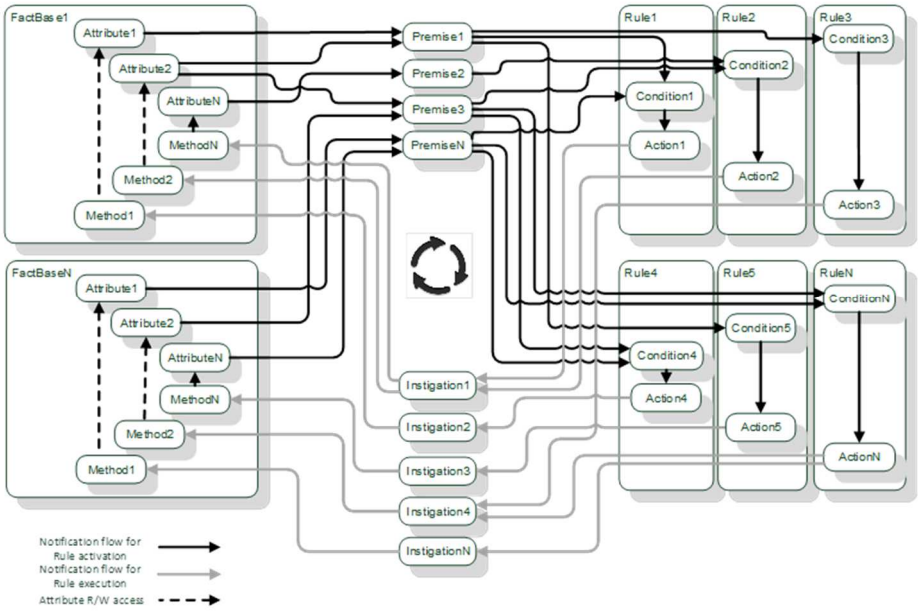

Figura 2. Colaboração por notificações das entidades do PON [12]

III. DESCRIÇÃO DO CTA SIMULATOR

Nesta Seção são apresentados os objetivos e características do software CTA Simulator.

Os objetivos do projeto CTA Simulator são desenvolver estratégias de controle de semáforos, simular regiões de tráfego em uma área urbana, comparar o desempenho das estratégias, comparar o desempenho e complexidade quando as estratégias de controles forem implementadas em paradigmas diferentes, como o paradigma imperativo e o paradigma orientado a notificações.

Para realizar as comparações entre os paradigmas, o CTA Simulator foi dividido em dois módulos desacoplados, sendo um de simulação da região urbana, e outro módulo para implementação das estratégias de controle em diferentes paradigmas.

A. Simulador

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 10 ruas horizontais, formando uma matriz 10x10 com um total de 100 interseções.

Cada interseção da região simulada é composta por dois sinaleiros opostos, os quais têm os mesmos estados de um sinaleiro do mundo real: verde, amarelo e vermelho. A união de um sinaleiro para a rua vertical e outro para a rua horizontal é chamada de semáforo e, por questões de segurança, os sinaleiros não podem ser verdes simultaneamente.

Cada quadra possui o comprimento de 100 metros e capacidade de 25 veículos por pista, sendo que cada rua pode possuir de 1 a 4 pistas. Os veículos são criados de forma constante, no intervalo de 0.1, 0.2, 0.3, 0.4 ou 0.5 veículos por segundo. Quando um veículo é criado em uma entrada, ele começa a se movimentar a uma velocidade de 20m/s, e deve parar de se movimentar quando o sinal da quadra em que se encontra está vermelho ou a próxima quadra esteja cheia. Em cada cruzamento, uma porcentagem de veículos poderá virar. Esta porcentagem pode variar de 5% até 35%.

Em todas as quadras são instalados sensores que monitoram a quantidade de veículos que estão parados. Esses sensores apresentam três estados: FEW, MANY e FULL. Para o sensor estar no estado FEW a taxa de ocupação da rua deve ser menor do que 60%, para o estado MANY, a taxa de ocupação deve estar entre 60% e 99% e para o estado FULL, a taxa de ocupação deve ser 100%.

B. Estratégias de controle

Para o simulador CTA, foram levantadas três alternativas de estratégia de controle de semáforos: 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 em 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.

No controle baseado em tráfego facilitado, para cada semáforo, um de seus sinaleiros possui uma flag sinalizando que a rua possui tráfego facilitado. Neste método de controle, o semáforo conhece o estado dos sensores de quantidade de tráfego e o tempo do semáforo anterior.

Se o nível de congestionamento for menor do que 60% o atraso com relação ao semáforo anterior é de 5 segundos, ou seja, o sinaleiro abre 5 segundos após o sinaleiro do semáforo anterior abrir. Se o nível de congestionamento estiver entre 60% e 99% não há diferença no tempo de abertura com os semáforos anteriores. Se o nível de congestionamento for igual a 100% o atraso com relação ao semáforo anterior é de -5 segundos, ou seja, o semáforo abre 5 segundos antes do que o semáforo anterior.

C. Desenvolvimento do CTA

O software foi dividido em dois módulos: um módulo de simulação e outro de controle de estratégias, de forma que seja possível implementar o controle dos semáforos em paradigmas de programação diferentes. Para o desenvolvimento do simulador, foi utilizada a linguagem C++ com o ambiente de desenvolvimento Visual Studio 2013 Professional. As estratégias de controle foram desenvolvidas em PI, com a mesma linguagem do simulador, e em PON através da geração de código C++ a partir do LingPon.

Para lógica do controle independente foi utilizado o diagrama de estados como mostra na Fig. 4.

Figura 4. Diagrama de estados para controle independente.

Para implementação da mesma estratégia utilizando PON, utilizou-se da linguagem LingPon. Cada transição do diagrama da Fig. 4 foi traduzida para uma regra, totalizando 6 regras para a estratégia de controle independente, na qual cada regra contém duas premissas, o estado e o tempo do semáforo. A Fig. 1 e a Fig. 2 apresentam as sintaxes de uma regra e uma FBE escritas através da linguagem LingPon, com base no diagrama da Fig. 4.

Ao compilar o código escrito na linguagem LingPon é gerado um código correspondente em C++. Para fazer a integração da estratégia desenvolvida em PON e o simulador desenvolvido em PI, mais especificamente em orientação a

objetos, foram adicionados ao projeto as classes geradas a partir da compilação do código em LingPon para C++. A classe Semaphore_NOP, gerada a partir da compilação do código LingPon, foi manualmente alterada para ser derivada da classe Sempahore presente no simulador e suas regras instanciadas a partir da instância de cada semáforo da região de simulação. Além das alterações no código gerado, foi utilizado o padrão de projeto Factory [20] para instanciar um semáforo corretamente, a partir da escolha da estratégia a ser analisada.

Na implementação da estratégia de controle baseada em congestionamento foi usada como referência a descrição da estratégia de controle descrita na Seção III B, no qual o tempo de abertura de um semáforo passa a depender do estado do sensor de tráfego e do tempo do semáforo. Para separar as estratégias de controle independente com a de controle baseado em congestionamento, foi desenvolvida uma nova aplicação em LingPon, sendo adicionada ao projeto de forma idêntica, através dos padrões de projeto. A Fig. 5 mostra a implementação da FBE para o controle baseado em congestionamento. Na FBE Semaphore_NOP_CBCL contém os mesmos atributos da FBE Semaphore_NOP e foram adicionados os atributos dos sensores de congestionamento para os sinaleiros horizontais e verticais, além dos métodos que alteram seus estados. Por questões de falhas na compilação do código LingPon, foi necessário diminuir o tamanho dos nomes de atributos e métodos.

```
fbe Semaphore_NOP_CBCL
    attributes
        integer atSeconds 0
        integer atSemaphoreState 5
        integer atHVSS 0
        integer atVVSS 0
    end_attributes

    methods
        method mtRT(atSeconds = 0)
        method mtHTLG(atSemaphoreState = 0)
        method mtHTLY(atSemaphoreState = 1)
        method mtHTLR(atSemaphoreState = 2)
        method mtVTLG(atSemaphoreState = 3)
        method mtVTLY(atSemaphoreState = 4)
        method mtVTLR(atSemaphoreState = 5)
        method mtHTLGCBCL(atSemaphoreState = 6)
        method mtHTLYCBCL(atSemaphoreState = 7)
        method mtVTLGCBCL(atSemaphoreState = 8)
        method mtVTLYCBCL(atSemaphoreState = 9)
    end_methods
end_fbe
```

Figura 5. Exemplo de *FBE* para controle baseado em congestionamento.

```
rule r1CBCL18
    condition
        subcondition sbCBCL18
                premise imp prSeconds10Full semaphore_NOP.atSeconds >= 18 and
                premise imp prSeconds210Full semaphore_NOP.atSeconds < 32 and
                premise prSemaphoreState10Full semaphore_NOP.atSemaphoreState == 3 and
                premise imp prVehicleSensorState10Full semaphore_NOP.atVVSS == 2
        end_subcondition
    end_condition
    action
        instigation inCBCL30 semaphore_NOP.mtVTLYCBCL();
        instigation inCBCL31 semaphore_NOP.mtRT();
    end_action
end_rule
```

Figura 6. Exemplo de *Rule* para controle baseado em congestionamento.

No desenvolvimento desta estratégia também foi necessário criar novas regras para implementação do controle baseado em congestionamento em PON, sendo 18 o total de regras implementadas e 8 dessas regras possuem avaliações de congestionamento através dos sensores. A Fig. 6 apresenta a sintaxe de uma regra desenvolvida segundo abordagem de estratégia de controle baseado em congestionamento. Porém, há duas avaliações do atributo atSeconds para que esteja com valor maior ou igual a 18 e menor do que 32 segundos, e o valor do atributo atVVSS igual a 2, representando o sensor de congestionamento vertical igual a FULL e o atributo atSemaphoreState que representa o estado atual, deve possuir valor 3. Quando todas as premissas forem verdadeiras, é efetuada a instigação do método mtVTLYCBCL, que é responsável pela transição para o estado 9 e do método mtRT que zera o tempo do semáforo.

Por questões de desempenho, as premissas que avaliam o tempo e o estado dos sensores só são pertinentes [9] caso esteja no estado desejado para que ocorra a transição. Assim, essas premissas só serão notificadas e avaliadas, caso a premissa que avalia o estado esteja verdadeira. Ao compilar as aplicações desenvolvidas em LingPon foi necessário modificar manualmente os códigos equivalentes gerados em C++ para que fizessem uso da impertinência dos atributos de tempo e sensor. Outra alteração necessária para o código PON apresentar melhora de desempenho foi alterar para que cada premissa seja única e compartilhada entre regras. Desta forma foi possível eliminar redundâncias na avaliação do estado das premissas.

Para implementação da estratégia baseada em tráfego facilitado foi desenvolvido um novo diagrama de estados baseado no diagrama da Fig. 4 e na descrição da estratégia na Seção III B. Durante a implementação desta estratégia de controle, foram encontradas dificuldades no desenvolvimento devido à estrutura da implementação do código equivalente em C++.

Caso duas Premises diferentes dependam do mesmo Attribute, e este valor é modificado, as entidades Premises relacionadas a ele também serão notificadas para serem reavaliadas. Esta notificação acontece de forma que essas entidades serão avaliadas em sequencia, porém, caso a primeira Premisse tenha seu valor lógico verdadeiro, ela irá notificar e executar a sua Condition. Se está também for verdadeira, haverá uma instigação para execução de sua respectiva Rule através da entidade Action, que realizará a instigação de um Method. A execução deste método pode alterar o valor do mesmo Attribute, reiniciando o fluxo de notificações de forma que a segunda premissa nunca seja reavaliada, portanto ela sempre estará desatualizada. Desta forma, é necessário implementar no LingPon um mecanismo de determinismo [15][22][23] que garanta que todas as avaliações das entidades do PON, tenham a mesma oportunidade de serem avaliadas, assegurando que todos os objetos notificantes estejam atualizados.

IV. RESULTADOS

Os experimentos realizados tiveram como objetivo analisar o desempenho das estratégias de controle implementadas em PON, comparadas as implementações em PI. Para isso, foram desenvolvidas as estratégias de controle independente e baseado em congestionamento descrito anteriormente, em ambos os paradigmas. Para executar os experimentos foi utilizado um notebook com processador core i7 – 4500U 1.8GHz – 2.4GHz e 8GB de memória RAM, com sistema operacional Windows 8.1. O sistema operacional Windows 8.1 foi utilizado devido ao ambiente de desenvolvimento escolhido.

Para análise dos experimentos de desempenho, a aplicação foi executada alterando o número de repetições: 90, 200, 500, 1000, 1500 e 2000, na qual cada repetição representa um segundo do semáforo. A Tabela I e a Fig. 7 apresentam os resultados dos experimentos executados, que correspondem ao

tempo de execução em milissegundos das duas estratégias implementadas, alterando o número de repetições.

TABELA I. RESULTADOS RELATIVOS AO EXPERIMENTO 1.

Strategy	90	200	500	1000	1500	2000
Independent_Strategy	17.243	38.380	104.793	201.136	301.194	399.688
Independent_Strategy_NOP	20.854	50.355	132.532	270.337	408.743	550.617
Congestion_Level_Strategy	24.413	55.197	154.076	306.681	459.726	610.568
Congestion_Level_Strategy_NOP	32.467	74.002	199.991	406.324	605.500	805.553

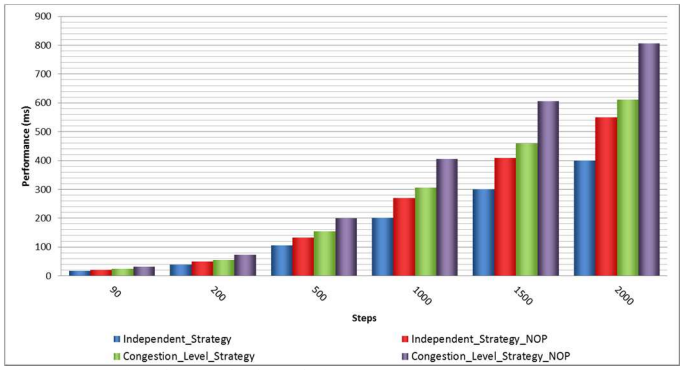

Figura 7. Gráfico dos resultados relativos ao experimento 1.

A partir dos dados da Tabela I e da Fig. 7, observou-se que a implementação em PON teve desempenho inferior quando comparado ao da implementação em PI. Porém, isso ocorreu devido ao fato de que a implementação da pertinência não estava sendo feita da forma correta, além do código equivalente gerado em C++ ainda apresentar redundâncias estruturais. Desta forma, o código gerado foi alterado manualmente, seguindo os princípios do PON para eliminar redundâncias estruturais e implementar o conceito de premissas pertinentes de maneira correta. Após as alterações, os experimentos foram executados novamente, e gerados os dados da Tabela II e da Fig. 8.

TABELA II. RESULTADOS RELATIVOS AO EXPERIMENTO 2.

Strategy	90	200	500	1000	1500	2000
Independent_Strategy	17.243	38.380	104.793	201.136	301.194	399.688
Independent_Strategy_NOP	20.854	48.355	117.233	237.906	385.907	477.710
Congestion_Level_Strategy	24.413	55.197	154.076	306.681	459.726	610.568
Congestion_Level_Strategy_NOP	22.490	50.112	128.372	271.875	388.382	513.299

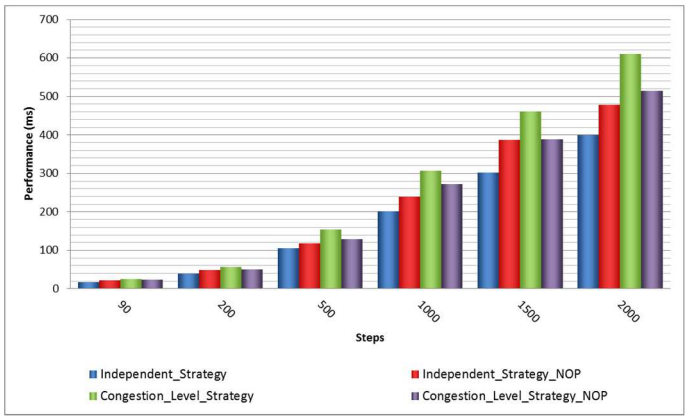

Figura 8. Gráfico dos resultados relativos ao experimento 2.

Após as alterações feitas a partir do código equivalente gerado em LingPon, os resultados da estratégia baseada em congestionamento obtiveram melhor desempenho em relação à implementação da mesma estratégia desenvolvida em PI. Esse resultado deve-se pela eliminação das redundâncias estruturais que ainda estão presentes nas entidades Premisses e na avaliação dessas entidades quando envolvem atributos de tempo ou sensores de congestionamento, que não são pertinentes.

V. CONCLUSÕES E TRABALHOS FUTUROS

O PON apresenta uma abordagem para o desenvolvimento mais eficiente de softwares em comparação aos paradigmas atuais. Este paradigma se propõe a resolver problemas tais como redundâncias estruturais, temporais e forte acoplamento entre as suas entidades, visando facilitar o desenvolvimento de softwares paralelos e distribuídos. Este trabalho apresentou o desenvolvimento de uma aplicação para a comparação de desempenho através da análise de tempo e, avaliar a complexidade de desenvolvimento sob este novo paradigma.

Quando realizados os primeiros experimentos comparando a implementação da aplicação CTA Simulator em ambos os paradigmas, a implementação em PI se demonstrou mais eficiente do que a implementação equivalente em PON. Porém, a partir das alterações realizadas no código gerado a partir do LingPon, foi possível verificar melhora no desempenho do PON, com relação a mesma implementação em PI. Além da melhora de desempenho, percebeu-se uma facilidade para incluir e alterar o código gerado em PON em um software que já havia sido desenvolvido utilizando outro paradigma. No caso deste artigo, foi inserido no CTA Simulator, sob paradigma orientado a objetos.

Com relação à complexidade de desenvolvimento, no PON as regras do sistema podem ser visualizadas de maneira mais fácil, pois o nível de abstração das regras é maior, quando comparado às abstrações das mesmas transições desenvolvidas em linguagem imperativa. Porém, o estado da técnica do compilador LingPon ainda está em desenvolvimento, logo ocorreram algumas dificuldades para compilação do código, como por exemplo o tamanho dos nomes das entidades presentes no PON, que variava de acordo com o tamanho do código, identificação de erros ao compilar, falta de suporte da linguagem a estruturas de dados e a declaração de uma FBE, como atributo de outra FBE.

Mesmo apresentando algumas dificuldades o uso da linguagem LingPon é promissor, pois com o avanço no estado da técnica desta linguagem, como otimizações do compilador, inclusão de estruturas de dados, determinismo e identificação de erros pelo compilador será possível gerar código de maneira mais eficiente, com possível distribuição e paralelismo, facilidade de programação e menor tempo de desenvolvimento, independente da plataforma na qual será executado. No estado da técnica atual já é possível gerar códigos C, C++ e Framework C. Além das linguagens já implementadas, é possível gerar códigos para outras linguagens, como Java, C#, NOCA e VHDL.

REFERÊNCIAS

[1] G. Moore, Cramming More Components onto Integrated Circuits. Eletronics Magazine , 1965.

[2] G. Moore, Excerpts from A Conversation with Gordon Moore: Moore's Law, 2005. Available in: < http://large.stanford.edu/courses/2012/ph250/lee1/docs/Excepts_A_Conversation_with_Gordon_Moore.pdf>. Acessed in: Jun. 09th, 2015.

[3] K. Bousias, L. Guang, C. R. Jesshope and M. Lankamp, "Implementation and evaluation of a microthread architecture". Journal of Systems Architecture, 55(3), 2009, 149–161. DOI 10.1016/j.sysarc.2008.07.001.

[4] G. Coulouris, J. Dollimore, and T. Kindberg. "Distributed Systems – Concepts and Designs". Bookman, 2013.

[5] J. M. Simão and P. C. Stadzisz, “Inference Process Based on Notifications: The Kernel of a Holonic Inference Meta-Model Applied to Control Issues”. IEEE Trans. on Systems, Man and Cybernetics. Part A, Systems and Humans, V.39, I.1, 238-250, 2009. DOI 10.1109/TSMCA.2008.2006371.

[6] J. M. Simão and P. C. Stadzisz, “Paradigma Orientado a Notificações (PON) - Uma Técnica de Composição e Execução de Software Orientada a Notificações”. Patent pending submitted to INPI/Brazil in 2008 and UTFPR Innovation Agency 2007. INPI Number: PI0805518-1. http://www.patentesonline.com.br/paradigma-orientado-anotificacoes--uma-tecnica-de-composicao-e-execucao-de-software-234943.html.

[7] R. F. Banaszewski, "Notification Oriented Paradigm: Advances and Comparisons". Original title: "Paradigma Orientado a Notificações: Avanços e Comparações". M. Sc. Thesis, CPGEI/UTFPR. Curitiba- PR, Brazil 2009. (Advisors: Prof. C. A. Tacla and Prof. J. M. Simão) http://arquivos.cpgei.ct.utfpr.edu.br/Ano-2009/dissertacoes/Dissertacao-500-2009.pdf

[8] J. M. Simão, R. F. Banaszewski, C. A. Tacla and P. C. Stadzisz, “Notification Oriented Paradigm (NOP) and Imperative Paradigm: A Comparative Study”, Journal of Software Engineering and Applications (JSEA), Vol. 5, No. 6, 402-416, 2012. DOI 10.4236/jsea.2012.56047.

[9] A. F. Ronszcka, Contribuição Para a Concepção de Aplicações no Paradigma Orientado a Notificações (PON) Sob o Viés de Padrões. 2012. Dissertação de Mestrado, CPGEI, UTFPR. Curitiba, Brasil, 2012.

[10] G. Z. Valença, Contribuição para Materialização do Paradigma Orientado a Notificações (PON) Via Framework e Wizard. 2012. Dissertação de Mestrado, Programa de Pós-Graduação em Computação Aplicada (PPGCA), UTFPR. Curitiba, Brasil, 2012.

[11] J. M. Simão, D. L. Belmonte, A. F. Ronszcka, R. R. Linhares, G. Z. Valença, R. F. Banaszewski; J. A. Fabro, C. A. Tacla, P. C. Stadzisz, M. V. Batista. Notification Oriented and Object Oriented Paradigms comparison via Sale System. Journal of Software Engineering and Applications (print), v. 5, p. 695-710, 2012.

[12] J. M. Simão, D. L. Belmonte, G. Z. Valença, M. V. Batista, R. R. Linhares, R. F. Banaszewski, J. A. Fabro, C. A. Tacla, P. C. Stadzisz, A. F. Ronszcka. A Game Comparative Study: Object-Oriented Paradigm and Notification-Oriented Paradigm. Journal of Software Engineering and Applications (print), v. 5, p. 722-736, 2012.

[13] J. M. Simão, R. R. Linhares, F. A. Witt, C. R. E. Lima and P. C. Stadzisz, “Paradigma Orientado a Notificações em Hardware Digital”. Patent pending submitted to INPI/Brazil in 2012 and UTFPR Innovation Agency (2012). INPI Provisory Number: BR 10 2012026429 3.

[14] E. Peters, Co-processor to Speed up of Application developed under the Notification Oriented Paradigm. Original title in Portuguese: Coprocessador para Aceleração de Aplicações Desenvolvidas utilizando Paradigma Orientado a Notificações. M. Sc. Thesis, CPGEI/UTFPR. Curitiba-PR, Brazil 2012

[15] R. R. Linhares. Contribution to development of a computing architecture proper to the notification oriented paradigm. Tese de Doutorado - Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial (CPGEI). Universidade Tecnológica Federal do Paraná (UTFPR). Curitiba, 2015.

[16] R. R. Linhares, J. M. Simão and P. C. Stadzisz, "NOCA A Notification-Oriented Computer Architecture," Latin America Transactions, IEEE (Revista IEEE America Latina), vol.13, no.5, pp.1593,1604, May 2015. Doi: 10.1109/TLA.2015.7112020

[17] R. R. Linhares, P. C. Stadzisz and J. M. Simão, “Arquitetura de Computador Orientada a Notificações - ARQPON”. Patent Application submitted to UTFPR Innovation Agency, 2013.

[18] R. D. Xavier, Software development paradigms: comparison between Event Oriented Paradigm and Notification Oriented Paradigm approaches.. Original title in Portuguese Paradigmas de desenvolvimento de software: comparação entre abordagens orientada a eventos e orientada a notificações. M. Sc. Thesis, PPGCA/UTFPR. Curitiba-PR, Brazil 2014

[19] C. A. Ferreira. Linguagem e compilador para o paradigma orientado a notificações (PON): Avanços e comparações. Seminário de Acompanhamento, PPGCA, UTFPR. Curitiba, Brasil, 2014.

[20] D. P. B. Renaux, R. R. Linhares, J. M. Simão, P. C. Stadzisz , “CTA_CONOPS”, Available in: < http://www.dainf.ct.utfpr.edu.br/~douglas/CTA_CONOPS.pdf>, 2014,Accessed in July10th, 2015.

[21] Gamma, Erich. Padrões de projeto: soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2000 xii,364 p. ISBN 8573076100.

[22] Simão, J. M.; Stadzisz, P. C. Mecanismo de Resolução de Conflito e Garantia de Determinismo para o Paradigma Orientado a Notificações (PON) Pedido de Patente .Submetido ao INPI/Brasil (Instituto Nacional de Propriedade Industrial) e a Agência de Inovação da UTFPR em 2010. Número INPI Temporário: 015100000477 – Data 02/2010. Número INPI Efetivo: PI1000296-0.

[23] J. M. Simão, R. F. Banaszewski, C. A. Tacla, P. C. Stadzisz .Mecanismo de Inferência Otimizado do Paradigma Orientado a Notificações (PON) e Mecanismos de Resolução de Conflitos para Ambientes Monoprocessados e Multiprocessados Aplicados ao PON. 2010, Brasil. Patente: Privilégio de Inovação. Número do registro: PI10037365, data de depósito: 25/03/2010. Instituição(ões) financiadora(s): Universidade Tecnológica Federal do Paraná, 2010.