Comparações entre duas materializações do Paradigma Orientado a Notificações (PON): Framework PON Prototipal versus Framework PON Primário

Jean M. Simão 1,2, Paulo C. Stadzisz 1,2, Cesar A. Tacla 1,2,
Robson R. Linhares [1,2], Danillo L. Belmonte [1], Roni F. Banaszewski [1]
jeansimao, stadzisz, tacla, linhares {@utfpr.edu.br}

Universidade Tecnológica Federal do Paraná - [1]Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial - [2]Programa de Pós-graduação em Computação Aplicada

Resumo: *Este artigo revisa o Paradigma Orientado a Notificações (PON) e traz uma comparação de um programa desenvolvido segundo os princípios da materialização (Framework) prototipal do PON e depois da materialização (Framework) primária dele. O PON se apresenta como uma alternativa aos Paradigmas de Programação Imperativa (PI) e de Programação Declarativa (PD). Ele se propõe a eliminar deficiências destes no tocante a redundâncias e acoplamento de avaliações causais que impactam no desempenho e paralelismo-distribuição de programas. Nesse âmbito, trabalhos prévios trataram da comparação de performance de materializações do PON para com materializações de PI e PD. Tais trabalhos mostram que materializações do PON são em geral melhores que boas práticas de PD e melhores que PI quando há muitas relações causais e variáveis com baixa ou média variação de estados. Ainda, quanto melhor se dá a materialização do PON, melhor são os resultados. Isto dito, para fins históricos, este artigo contempla o avanço de performance computacional obtido com a elaboração do Framework PON dito Primário em relação ao precedente dito Prototipal. Em suma, nos experimentos com um caso de estudo referente a um jogo Mira-Alvo, o Framework Primário se mostrou em torno de duas vezes melhor que o Framework Prototipal.*

Abstract: *This paper reviews the Notification Oriented Paradigm (NOP) and brings a comparison of a program developed following the principles of the prototypal NOP (Framework) implementation and also the primary NOP (Framework) implementation. NOP presents as an alternative to Imperative Programming (IP) and Declarative Programming (DP) Paradigms. It proposes to eliminate their deficiencies with respect to causal evaluations that impact in performance and parallelism-distribution of the programs. Previous works compared the performance of the NOP (Framework) implementations against some IP and DP implementations. Such works showed that NOP implementations are in general better than suitable PD practices and better than PI when there are so much causal relations and variables with low or medium variation of states. Still, as better is the NOP implementation better the results are. In this sense, for historical reasons, this paper presents the performance gain obtained with elaboration of the so-called NOP Primary Framework with the respect to the NOP Prototypal one. In short, in the experiments carried out with a case of study related to a marksmanship game, the Primary Framework shown to be twice times better than the Prototypal Framework.*

Palavras-chave*:* Paradigma Orientado a Notificações, Comparação de Framework PON Prototipal e Primário.

1. Introdução

A capacidade de processamento computacional tem crescido em função da evolução das tecnologias neste contexto [Linhares et al., 2011][Tanenbaum e Van Steen, 2002]. Entretanto, recursos oferecidos por soluções computacionais modernas, tais como paralelismo e distribuição e particularmente a utilização da capacidade plena de cada processador, nem sempre são devidamente aproveitados em função de limitações das técnicas de programação [Simão e Stadzisz, 2008, 2009][Linhares et al., 2011].

Na verdade, técnicas de programação baseadas no estado da arte, como o chamado Paradigma de Programação Orientada a Objetos (POO) ou os Sistemas Baseados em Regras (SBR), sofrem de limitações intrínsecas de seus paradigmas. Estes paradigmas poderiam ser genericamente classificados como Paradigma Imperativo (PI) e Paradigma Declarativo (PD) que englobam respectivamente o POO e os SBR [Banaszewski, 2009][Linhares et al., 2011].

Particularmente, tais paradigmas levam a forte acoplamento de expressões causais e redundâncias decorrentes das suas avaliações. Estas limitações dificultam a execução distribuída de programas e comprometem o seu desempenho pleno mesmo em sistemas monoprocessados. Assim, existem motivações para buscar alternativas ao PI e PD, com o objetivo de eliminar ou diminuir estas desvantagens [Banaszewski *et al.*, 2007][Banaszewski, 2009][Gabbrielli e Martini, 2010][Roy e Haridi, 2004][Simão e Stadzisz, 2008, 2009][Linhares et al., 2011][Simão et al. 2012].

Uma alternativa é o Paradigma Orientado a Notificações (PON), que foi concebido a partir de uma teoria de Controle Discreto e Inferência [Simão, 2001, 2005][Simão e Stadzisz, 2002, 2008, 2009][Simão, Stadzisz e Tacla, 2009][Simão, Stadzisz e Künzle, 2003]. Ele se propõe a eliminar certas deficiências dos atuais paradigmas no tocante a avaliações causais desnecessárias e acopladas, evitando o processo de inferência monolítico baseado em pesquisas. Isto se dá via um mecanismo baseado no relacionamento de entidades computacionais notificantes [Banaszewski *et al.*, 2007][Banaszewski, 2009][Simão e Stadzisz, 2008, 2009][Simão et al. 2012, 2012b][Linhares et al., 2011][Wiecheteck, 2011] [Wiecheteck, Stadzisz e Simão, 2011].

Trabalhos prévios trataram de comparar, em termos de performance, a materialização primária do PON com materializações de POO e SBR. Tais trabalhos apontam que materializações do PON são melhores que SBRs em geral e melhores que POO onde há muitas relações causais e variáveis com baixa ou média variação de estados [Banaszewski, 2009][Linhares et al., 2011][Batista et al., 2011][Ronskca et al., 2011][Valença et al., 2011][Simão et al., 2012, 2012b]. Ademais, quanto melhor se dá a materialização do PON (em termos de otimização de estruturas de dados, por exemplo), melhor são os resultados do PON [Valença et al., 2011] [Simão et al. 2012b].

Isto dito, a partir dos experimentos elaborados em [Banaszewski, 2009], este artigo apresenta para fins históricos um caso de estudo sobre o avanço obtido com o *Framework* PON elaborado por Banaszewski (dito Primário) em relação ao *Framework* PON previamente elaborado por Simão (dito Prototipal). O *Framework* Prototipal foi elaborado a partir da derivação da tecnologia de Controle Holônico [Simão, 2001, 2005; Simão e Stadzisz, 2002, 2008, 2009; Simão, Stadzisz e Tacla, 2009; Simão, Stadzisz e Künzle, 2003], o qual atualmente é chamado Controle Orientado a Notificações (CON) e serviu para o vislumbre do PON por parte de Simão e Stadzisz [Simão e Stadzisz, 2008, 2009]. Ao seu turno, o *Framework* PON Primário foi elaborado a partir deste vislumbre e do *Framework* PON Prototipal [Banaszewski, 2009].

Este artigo está organizado como segue: a Seção 2 reflete sobre os atuais Paradigmas de Programação. A Seção 3 apresenta o PON. As Seções 4 e 5 apresentam o caso de estudo e sua implementação usando os dois *frameworks* do PON**.** A Seção 6 discute o caso de estudo e os experimentos. Por fim, a Seção 7 apresenta conclusões e trabalhos futuros.

2. Paradigmas de Programação

A utilização de técnicas de PI, particularmente o POO, costuma atrair os desenvolvedores devido a questões como inércia cultural, riqueza de abstração e flexibilidades algorítmicas. Em PI, em suma, concebem-se programas como seqüências de instruções utilizando-se buscas sobre entidades passivas (dados e comandos) organizadas segundo uma lógica de execução que envolve, inclusive, a avaliação de expressões causais (*e.g.* se-então) [Banaszewski *et al.*, 2007] [Brookshear, 2006][Gabbrielli e Martini, 2010][Linhares et al., 2011][Simão et al. 2012, 2012b].

Algoritmo 1 - Pseudocódigo do Paradigma Imperativo

```
Enquanto ( verdade ) faça
   se ( ( objeto1.atributo1 = 1   ) e ( objeto2.atributo1 = 1 ) )   então  objeto1.método1(); objeto2.método2();
fim-se
   . . .
   se ( ( objeto1.atributoN = N ) e ( objeto2.atributoN = N ) )  então  objeto1.métodoN(); objeto2.métodoN();
fim-se
fim-enquanto
```

Particularmente, estas expressões são frequentemente avaliadas sem necessidade, degradando desempenho computacional. Isto pode ser exemplificado com um conjunto de expressões se-então que avaliam os estados de objetos em um laço de repetição 'infinito'. Cada expressão condicional avalia estados de atributos de objetos e, se aprovada, chama alguns métodos destes que podem mudar estados de atributos. [Brookshear, 2006][Gabbrielli e Martini, 2010][Simão e Stadzisz, 2008, 2009][Simão et al. 2012, 2012b]. Isto é apresentado no Algoritmo 1 [Linhares et al., 2011].

Observa-se no exemplo que o laço de repetição força seqüencialmente a avaliação (inferência) de todas as condições. Entretanto, muitas destas são desnecessárias porque somente alguns objetos têm o valor de atributo modificado. Isto pode ser considerado pouco importante neste exemplo simples e pedagógico, sobretudo se o número *N* de expressões causais for pequeno. Entretanto, se considerado um sistema complexo, com várias partes como aquela, pode-se ter grande diferença de desempenho [Simão e Stadzisz, 2008][Linhares et al., 2011][Simão et al., 2012].

Por sua vez, em PD, enfatizando SBR, existe a vantagem da programação em alto nível. Primeiro se define uma Base de Fatos composta por entidades como objetos com atributos e métodos. Depois se define a Base de Regras com relações causais relativas às entidades da Base de Fatos. Estas duas bases são processadas por meio de uma Máquina de Inferência que automaticamente compara regras e fatos (*e.g.* estados de atributos) gerando novos fatos e, portanto, um ciclo de inferência. Não obstante a organização e algoritmos eficientes de inferência, a programação em PD normalmente é computacionalmente cara em termos de estruturas de dados processadas [Scott, 2000][Simão e Stadzisz, 2008, 2009][Linhares et al., 2011][Simão et al. 2012, 2012b].

Entretanto, em uma análise mais profunda, PI e PD são similares no tocante à inferência que normalmente se dá por entidades monolíticas baseadas em pesquisas sobre entidades (quase) passivas que conduzem a programas com passos de execução interdependentes. Estas características contribuem para a existência de sobreprocessamento e forte acoplamento entre expressões causais e estrutura de fatos/dados, o que dificulta a execução dos programas de maneira otimizada, bem como paralela ou distribuída [Gabbrielli e Martini, 2010][Simão e Stadzisz, 2008, 2009][Linhares et al., 2011][Simão et al. 2012, 2012b]. Ainda que haja outras alternativas de programação, como orientações a eventos e mesmo a dados, elas apenas atenuam ou fatoram o problema, não o resolvendo conforme discutido em [Banaszewski, 2009;

Simão e Stadzisz, 2008, 2009][Linhares et al., 2011][Simão et al. 2012, 2012b].

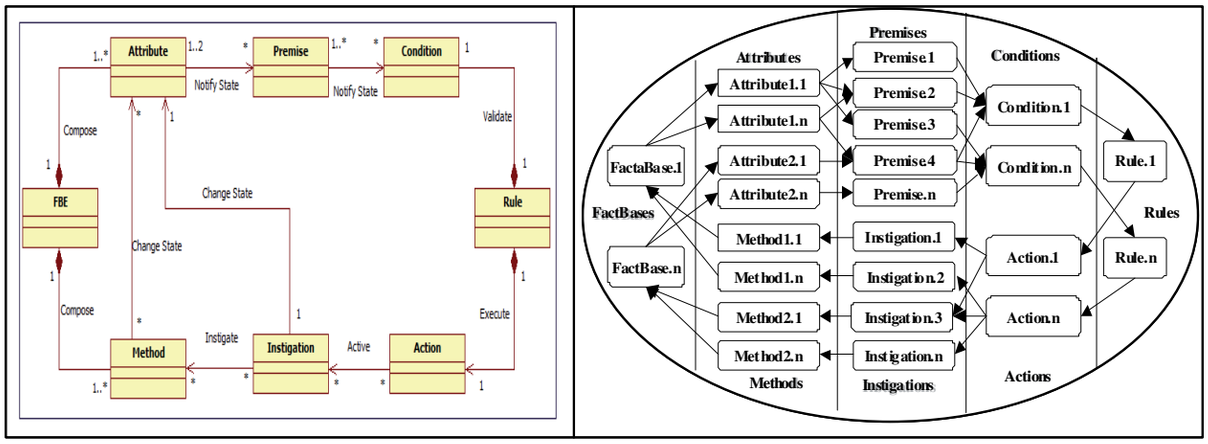


Figura 1. Entidades PON [Linhares et al., 2011] & Inferência por Notificações [Simão e Stadzisz, 2008, 2009].

3. Paradigma Orientado a Notificações (PON)

O PON encontra inspirações no PI, como a flexibilidade algorítmica e a abstração em forma de classes/objetos da POO. O PON também aproveita conceitos próprios do PD, como facilidade de programação em alto nível e a representação do conhecimento em regras dos SBR. Assim, o PON permite o uso (de parte) de ambos os estilos de programação em seu modelo, ainda que os evolua e mesmo os revolucione no tocante ao processo de inferência ou cálculo lógico-causal [Simão e Stadzisz, 2008, 2009][Banaszewski, 2009][Linhares et al., 2011][Simão et al. 2012, 2012b].

O PON apresenta resposta a problemas destes paradigmas, como repetição de expressões lógicas e reavaliações desnecessárias delas (*i.e.* redundâncias estruturais e temporais) e, particularmente, o acoplamento forte de entidades quanto às avaliações ou cálculo lógico-causal. Justamente, o PON apresenta outra forma de realizar tais avaliações ou inferências via entidades computacionais de pequeno porte, ativas e desacopladas que colaboram por meio de notificações pontuais e são criadas a partir do 'conhecimento' de regras [Linhares et al., 2011][Simão et al. 2012, 2012b].

Mais precisamente, o PON possui objetos que tratam dos elementos da base de fatos, que são genericamente modelados pela classe *FBE* (*Fact Base Element*), conforme a Figura 1. Cada *FBE* trata de seus atributos por meio de objetos da classe *Attribute* e seus serviços por meio de objetos da classe *Method*. Os objetos *FBE*, por meio de seus *Attributes* e *Methods*, são passíveis de correlação causal por meio de *Rules*, as quais se constituem em elementos fundamentais do PON [Simão e Stadzisz, 2008][Linhares et al., 2011][Simão et al. 2012, 2012b].

A Figura 2 apresenta um exemplo de *Rule*, na forma de uma regra causal. Na verdade, a *Rule* é uma entidade computacional composta por outras entidades, cf. Figura 1, as quais podem ser vistas como objetos. Por exemplo, a *Rule* apresentada é composta por um objeto *Condition* e um objeto *Action*. A *Condition* trata da decisão da *Rule*, enquanto a *Action* trata da execução das ações da *Rule*. Assim sendo, *Condition* e *Action* trabalham juntas para realizar o conhecimento lógico e causal da *Rule* [Simão e Stadzisz, 2008][Linhares et al., 2011][Simão et al. 2012, 2012b].

A *Rule* mostrada faz parte do caso de estudo Mira-Alvo detalhado na próxima seção. A *Condition* da *Rule* refere-se à decisão sobre a interação entre um *FBE* 'Maça' (*Apple*) e um *FBE* 'Arqueiro' (*Archer*). Esta *Condition* tem três objetos *Premises*. Estes realizam as seguintes verificações no tocante aos *FBEs*: *a) a Maça está vermelha? b) a Maça está pronta? c) o Arqueiro está pronto?* Assim, conclui-se (em geral) que os estados de atributos de *FBEs* compõem os fatos avaliados pelas *Premises* [Simão e Stadzisz, 2008][Linhares et al., 2011][Simão et al. 2012].

Cada *Premise* avalia o estado de um ou dois *Attributes* de *FBE*, sendo que para cada mudança de estado de *Attribute* da *FBE*, ocorrem automaticamente avaliações (lógicas) somente nas *Premises* relacionadas com eventuais mudanças nos seus estados. Igualmente, a partir da mudança de estado das *Premises*, ocorrem automaticamente avaliações (causais) somente nas *Conditions* relacionadas com eventuais mudanças de seus estados. Isto se dá por uma cadeia de notificações entre objetos inteligentes, o que se constitui no fundamento do PON conforme modelado e esboçado na Figura 1 [Simão e Stadzisz, 2008][Linhares et al., 2011][Simão et al. 2012, 2012b].

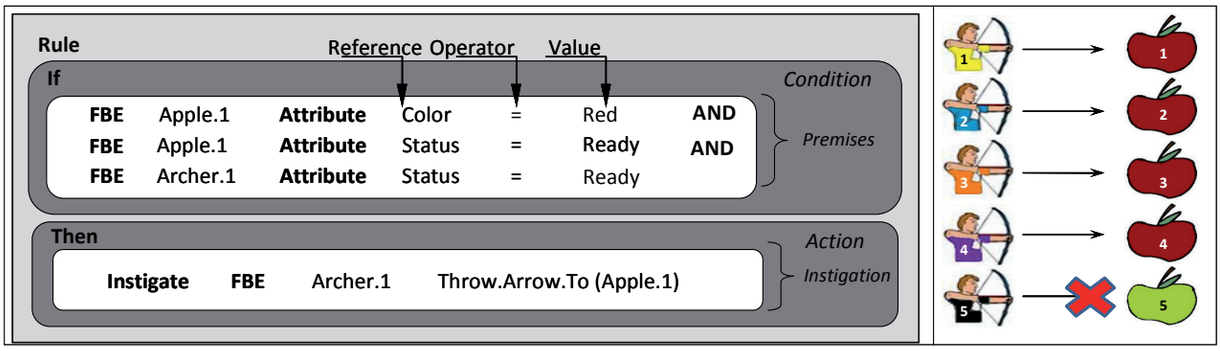


Figura 2. Exemplo de uma *Rule* em PON [Banaszewski, 2009] [Simão et al. 2012].

Cada *Attribute* notifica as *Premises* relevantes sobre seus estados apenas quando necessário. Cada *Premise* similarmente notifica as *Conditions* relevantes dos seus estados. Baseado nestes estados notificados cada *Condition* pode ser aprovada. Se sim, a respectiva *Rule* pode ativar sua *Action*. Ainda, o conhecimento de quem se deve notificar se dá na composição das *Rules* [Banaszewski, 2009][Simão e Stadzisz, 2008, 2009][Linhares et al., 2011][Simão et al. 2012].

Por sua vez, uma *Action* também é um objeto que se conecta a outros objetos, as *Instigations*. No exemplo, a *Action* contém uma *Instigation* que instiga o Arqueiro a disparar na respectiva Maça. Cada *Instigation* instiga um ou mais métodos para realizarem serviços de um *FBE*. Ainda, cada método de *FBE* é tratado por um *Method* cuja execução geralmente muda o estado de um ou mais *Attributes*. Estes conceitos de *Attribute* e *Method* seriam uma evolução dos conceitos similares da OO. A diferença é o desacoplamento explícito da classe proprietária e a colaboração pontual para com *Premises* e *Instigations* [Simão e Stadzisz, 2008][Linhares et al., 2011][Simão et al. 2012].

Isto considerado, nota-se que a essência da computação no PON está organizada e distribuída entre entidades autônomas e reativas que colaboram por meio de notificações pontuais. Este arranjo forma o mecanismo de notificações, o qual determina o fluxo de execução das aplicações. Por meio deste mecanismo, as responsabilidades de um programa são divididas entre os objetos do modelo, o que permite execução otimizada e 'desacoplada' (i.e. minimamente acoplada) útil para o aproveitamento correto de mono-processamento, bem como para o processamento distribuído [Simão e Stadzisz, 2008][Linhares et al., 2011][Simão et al. 2012, 2012b]..

A natureza do PON leva a uma nova maneira de compor *software*, onde os fluxos de execução são distribuídos e colaborativos nas entidades. Ainda, muito embora o PON permita compor *software* em alto nível na forma de regras, sem o conhecimento de sua essência, este conhecimento é deveras importante. Por exemplo, é importante saber dos impactos de desempenho e das estratégias de distribuição, como o agrupamento de elementos de maior fluxo de notificações juntos. Assim sendo, o PON permite uma nova maneira de estruturar, executar e conceber os artefatos de *software* [Simão e Stadzisz, 2008][Linhares et al., 2011][Simão et al. 2012, 2012b].

Atualmente, o PON está materializado na forma de um conjunto de *Frameworks/Wizards* em C++ enquanto uma linguagem-compilador PON permanece um trabalho futuro. Embora um paradigma possa se materializar em outro (e.g. um programa POO feito em linguagem procedimental) e isto seja natural em paradigmas emergentes, seria mais confortável e apropriado um ambiente efetivamente voltado para o PON [Banaszewski, 2009][Valença et al., 2011][Simão et al. 2012b].

Neste ínterim, a materialização do PON dita primária foi comparada em termos de performance com implementações PI/POO e PD/SBR. No caso de PI/POO houve comparações com programas C++/OO usuais [Banaszewski, 2009][Simão et al. 2012]. No caso de PD/SBR, houve comparações com dois *shells*, *CLIPS* e *RuleWorks*, que usam o eficiente motor de inferência RETE. Estas comparações apresentaram resultados a favor do PON, ainda que sobre *toy problems* [Banaszewski, 2009]. Também houve comparações qualitativas e assintóticas favoráveis ao PON em relação a motores de inferência como *RETE*, *TREAT*, *LEAPS* e *HAL* [Banaszewski, 2009].

Todavia, outros testes sobre aplicações reais se fizeram necessários para verificar a eficiência e eficácia do PON, em termos de performance, particularmente em relação ao dominante POO. Tais testes mostram que a materialização do PON é melhor que POO onde há muitas relações causais e variáveis com baixa ou média variação de estados [Banaszewski, 2009][Linhares et al., 2011] [Batista et al., 2011][Ronskca et al., 2011][Valença et al., 2011][Simão et al. 2012, 2012b].

Ademais, quanto melhor for a materialização do PON (e.g. em termos de otimização de estruturas de dados), melhor são os resultados de performance do PON. Este afirmação advém da comparação entre o *Framework* PON Prototipal e o Primário [Banaszewski, 2009], bem como de comparação entre o Primário e a dita nova versão [Valença et al., 2011][Simão et al. 2012b].

Isto dito, a partir dos experimentos elaborados em [Banaszewski, 2009], este artigo apresenta, para fins históricos, um caso de estudo sobre o avanço obtido com a elaboração do *Framework* PON dito Primário em relação ao *Framework* PON precedente dito Prototipal, Outrossim, isto permite definir a curva evolutiva entre cada materialização ou *Framework* PON.

4. Caso de Estudo

Esta seção descreve o caso de estudo usado para comparar as materializações do PON. Ele consiste em um jogo que simula a interação entre miras e alvos, chamado de Mira-Alvo. Neste ambiente, ambas as entidades são posicionadas a uma dada distância, sendo que a mira tenta atingir o alvo com o arremesso de um projétil [Banaszewski, 2009][Simão et al. 2012].

Na verdade, este artigo traz um cenário baseado neste jogo onde as entidades miras e alvos são representadas por arqueiros e maçãs, havendo uma maçã por arqueiro posicionados frente a frente cf. Figura 2. Na implementação, cada entidade é representada por um objeto (i.e. um elemento da base de fatos) capaz de interagir de acordo com a validação de expressões causais pertinentes, as quais fazem menção a atributos destes objetos e seus estados [Banaszewski, 2009].

Cada arqueiro e cada maçã recebe um identificador numérico, sendo que um arqueiro somente pode flechar uma maçã que apresente o mesmo identificador numérico que o seu. Cada arqueiro ainda apresenta um atributo que denota o seu estado de pronto ou não (*status*) para agir sobre o cenário (flechar a respectiva maçã). Cada maçã também apresenta um atributo de estado (*status*) no sentido de informar o seu estado de pronto para ser flechada. Cada maçã apresenta ainda um atributo que explicita a sua coloração (*color*), vermelha ou verde [Banaszewski, 2009].

Neste cenário, cada arqueiro pode interagir com a sua respectiva maçã se: (a) a cor da maçã posicionada em sua frente é vermelha, (b) esta maçã está pronta para ser atingida e (c) ela é identificada pelo seu número apropriado. Se tais premissas são satisfeitas, o arqueiro pode atingir a respectiva maçã com a projeção da flecha. Assim, percebe-se que para cada par de arqueiros e maçãs deve haver uma expressão causal para comparar os seus estados [Banaszewski, 2009].

Conforme ilustra a Figura 2, os arqueiros e as maçãs identificados pelos números 1, 2, 3 e 4 confirmam a veracidade das três premissas e esses flecham as respectivas maçãs. Porém, o mesmo não ocorre com o arqueiro identificado pelo número 5, uma vez que a sua respectiva maçã é verde, o que não satisfaz uma das sub-condições ou premissas [Banaszewski, 2009].

No entanto, este mesmo arqueiro terá outras oportunidades para interagir com a sua maçã, uma vez que o cenário é composto por várias iterações. Uma iteração consiste nas avaliações das premissas pertinentes para cada arqueiro sendo que as maçãs são perfuradas quando as condições são satisfeitas. Ainda, ao final de uma iteração, as maçãs perfuradas são substituídas por novas maçãs e todo o processo de interação entre os personagens é reinicializado [Banaszewski, 2009].

Os experimentos realizados sobre este cenário varia os estados dos personagens para que apenas uma porcentagem pré-definida de expressões causais seja satisfeita. Esta porcentagem cresce na escala de 10% até atingir a totalidade das expressões causais. Assim, um experimento pode ser dividido em escalas, chamadas de fases. Cada fase é executada uma dada quantidade de vezes de forma a evitar que variações de ambiente afetem a análise [Banaszewski, 2009].

5. Implementação

Para realizar o estudo comparativo entre as duas versões do *Framework* (Prototipal e Primário), o cenário apresentado na Seção 4 é usado na implementação de dois experimentos, os quais se diferem na quantidade de iterações (dez mil e cem mil) em que 100 arqueiros e 100 maçãs interagem. Nestes experimentos, os arqueiros são representados pela classe *Archer* e as maçãs pela classe *Apple*, ambas derivadas da classe *FBE* [Banaszewski, 2009].

```
/*****************************
 **** FRAMEWORK PROTOTIPAL ****
 *****************************/

 ...
for(int i = 0; i < 100; i++)
{
    //Premises
    prAppleColorRed = new AgentePremissa(appleList->at(i)->atAppleColor, True);
    prAppleColorRed->conectaPredBoleano(&comparaBoleanos);
    prAppleStatusTrue = new AgentePremissa(appleList->at(i)->atAppleStatus, True);
    prAppleStatusTrue->conectaPredBoleano(&comparaBoleanos);
    prArcherStatusTrue = new AgentePremissa(archerList->at(i)->atArcherStatus, True);
    prArcherStatusTrue->conectaPredBoleano(&comparaBoleanos);

    //Condition
    AgenteCondicao *cdFireApple;
    cdFireApple = new AgenteCondicao();
    cdFireApple->conectaAgentePremissa(prAppleColorRed);
    cdFireApple->conectaAgentePremissa(prAppleStatusTrue);
    cdFireApple->conectaAgentePremissa(prArcherStatusTrue);

    //Action
    AgenteAcao *acFireApple;
    acFireApple = new AgenteAcao;
    acFireApple->conectaAgenteOrdem(appleList->at(i)->mtChangeToGreen);

    //Method
    mtChangeToGreen = new AgenteMetodoGen<Apple>(&Apple::ChangeToGreen);

    //Instigation
    itChangeToGreen = new AgenteOrdem(mtChangeToGreen);

    //Rule
    AgenteRegra *rlFireApple;
    rlFireApple = new AgenteRegra(cdFireApple, acFireApple);
}
 ...
```

```
/*****************************
 **** FRAMEWORK PRIMÁRIO ****
 *****************************/
 ...
for(int i = 0; i < 100; i++){
    Rule* rlFireApple = new Rule(Condition::CONJUNCTION);
    rlFireApple->addPremise(appleList->at(i)->atAppleColor, Boolean::TRUE, Premise::EQUAL);
    rlFireApple->addPremise(appleList->at(i)->atAppleStatus, Boolean::TRUE, Premise::EQUAL);
    rlFireApple->addPremise(archerList->at(i)->atArcherStatus, Boolean::TRUE, Premise::EQUAL);
    rlFireApple->addInstigation(appleList->at(i)->atAppleColor, false);
    rlFireApple->end();
}
 ...
```

```
...
void SimpleApplication::initFacts(){
  ...
  for( int i = 0; i < 100; i++ ){
    Archer *archerTmp;
    archerTmp = new Archer();
    archerTmp->atArcherStatus->setValue(true);
    archerList->push_back(archerTmp);

    Apple *appleTmp;
    appleTmp = new Apple();
    appleTmp->atAppleStatus->setValue(true);
    appleTmp->atAppleColor->setValue("GREEN");
    appleList->push_back(appleTmp);
  }
}
void SimpleApplication::notificationExperiment(){
 for(int i = 0; i < iterations; i++){
   for(int j = 0; j < percentage; j++){
     appleList->at(j)->atAppleColor->setValue("RED");
   }
 }
}
```

```
void SimpleApplication::codeApplication(){

  long iterations  = 0;
  clock_t start, finish;
  double duration;

  cout << "Quantidade de Iterações: ";
  cin >> iterations;
  cout << "Porcentagem de Maçãs Vermelhas: ";
  cin >> percentage;

  start = clock();

  notificationExperiment();

  finish = clock();
  duration = finish - start;

  cout << "\n Tempo transcorrido: \n";
  cout << (durationNop / CLOCKS_PER_SEC) * 1000;
  cout << " ms" << "\n";
}
```

Figura 3. Cenário do jogo & Representação das *Rules* nos *Frameworks* PON [Banaszewski, 2009].

Os arqueiros e maçãs são agrupados em estruturas de dados do tipo vetor chamadas de *ArcherList* e *AppleList*. Estas estruturas são percorridas em um laço de repetição para a criação das 100 expressões causais pertinentes, as quais são representadas por *Rules* no contexto dos *Frameworks* (na verdade chamadas de *AgenteRegra* no Prototipal e *Rules* efetivamente no Primário). A Figura 3 apresenta a implementação das *Rules* no *Framework* Prototipal (linha 1) e no *Framework* Primário (linha 45), as quais se diferem em suas expressões [Banaszewski, 2009].

Dentre as três *Premises* de cada *Rule*, duas sempre apresentam estado verdadeiro e uma varia de estado em cada iteração. Esta *Premise* se refere à avaliação da cor da *Apple* (linhas 9 e linha 48), a qual se torna falsa quando a cor da *Apple* referenciada é verde e se torna verdadeira quando a cor da *Apple* referenciada é vermelha. Se ela se torna verdadeira no início de cada iteração então ela se tornará falsa depois pela própria execução da *Rule* que a contém, a qual altera a coloração da *Apple* referenciada para verde. Ainda, as outras duas *Premises* se referem aos estados de pronto (*status*) dos respectivos *Archers* (linhas 13 e 55) e *Apples* (linhas 11 e 53) [Banaszewski, 2009].

A Figura 3 também apresenta outros códigos (comuns a ambos os *Frameworks*) necessários para a execução dos experimentos. Conforme estes códigos, para verificar o desempenho dos dois *Frameworks* em várias situações, a quantidade de *Rules* aprovadas varia em cada fase do experimento na escala crescente de 10% pela variação da quantidade de *Apples* vermelhas. Na medida em que a quantidade de *Rules* aprovadas aumenta nas fases, verifica-se o desempenho dos *Frameworks* em relação ao aumento de notificações, dado que as mudanças de estados se tornam mais freqüentes [Banaszewski, 2009].

Outrossim, os experimentos comparativos com a aplicação Mira-Alvo foram executados em um computador com processador Pentium 4 de 1,8 GHz e 512 MB de memória e o tempo foi medido em milissegundos Ainda, para que os experimentos apresentassem resultados confiáveis, eles foram executados com o ambiente livre de preempções via MS-DOS (*Microsoft - Disk Operating System*) versão 7.0 incluso no Sistema Operacional *Windows 98* [Banaszewski, 2009].

Ainda, para que os experimentos pudessem executar neste ambiente, fez-se necessário ter acesso ilimitado aos recursos computacionais oferecidos no Modo-Protegido, principalmente aos recursos de memória. Para isto, os códigos-fonte foram compilados para executarem em Modo-Protegido com o compilador *DJGPP*, que é uma versão do compilador *GCC* para *MS-DOS*, usando uma interface chamada *DPMI* (*DOS Protected Mode Interface*) que permite ao processador acessar toda a memória disponível [Delorie, 2003][Banaszewski, 2009].

6. Discussão

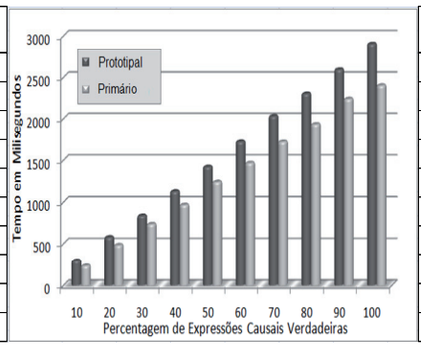
No 1º experimento, os *Frameworks* foram comparados sobre as mesmas condições com 10.000 iterações. O resultado disto está ilustrado na Figura 4 sendo que os dois *Frameworks* quase se equivalem em termos de desempenho. Todavia, mesmo com pouca diferença, o *Framework* Primário ainda se apresenta mais eficiente, mesmo sendo mais complexo pela inclusão de novas funcionalidades, como o método *addPremise* que automaticamente cria *Premise* e a associa à *Rule*. Este tipo de funcionalidade permite que a composição das *Rules* ocorra de maneira mais sucinta, cf. Figura 3, o que se constitui em uma melhora qualitativa [Banaszewski, 2009].

No entanto, o *Framework* Primário pode se apresentar ainda mais eficiente quando a quantidade de iterações aumenta. Assim, outro experimento foi realizado apenas aumentando a quantidade de *Rules* aprovadas, onde os *Archers* e *Apples* interagem em 100.000 iterações, apresentando diferença maior cf. Figura 4. As diferenças se mantêm constantes quando 30% de *Rules* são aprovadas, mas quando a porcentagem de aprovações aumenta o tempo de execução do *Framework* Prototipal se eleva em maior escala do que o tempo do Primário [Banaszewski, 2009].

Este aumento no tempo de execução do *Framework* Prototipal se deve a problemas relacionados à alocação dinâmica de memória, o que causa a extrapolação da memória principal para que dados sejam salvos temporariamente em memória virtual. O *Framework* Prototipal empregava uma lista encadeada artesanal, feita segundo o uso de *templates* em C++, para cada entidade armazenar os endereços das demais entidades a serem notificadas. Ao seu turno, o *Framework* Primário empregou para tal o *list* do *Standard Template Library* (*STL*) do C++. Isto, dentro outras melhorias, permitiu um melhor gerenciamento da memória [Banaszewski, 2009].

Como apresenta o 2º gráfico, quando as 100 *Rules* apresentam estado lógico verdadeiro, o desempenho do *Framework* Prototipal corresponde aproximadamente a três vezes o desempenho do *Framework* dito Primário. Obviamente, se a quantidade de iterações ainda fosse aumentada, o que demandaria mais memória, esta diferença seria ainda maior [Banaszewski, 2009]. Desta forma, o *Framework* Primário se mostra mais eficiente do que o *Framework* Prototipal, tendo por isto sido adotado o Primário para comparação com os demais paradigmas, o que é disponível em [Banaszewski, 2009] e em [Simão et al., 2012], ambas referências disponível *on line*, cf. Seção 8.

Exp. Verd.	Prototipal (ms)	Primário (ms)
10%	297	234
20%	578	485
30%	843	734
40%	1125	969
50%	1422	1234
60%	1720	1468
70%	2031	1719
80%	2297	1935
90%	2594	2235
100%	2906	2406



Exp. Verd.	Prototipal (ms)	Primário (ms)
10%	2828	2434
20%	5531	4906
30%	8281	7344
40%	12297	9782
50%	16375	12234
60%	22047	14718
70%	30781	17188
80%	37860	19578
90%	50937	22078
100%	64109	24125

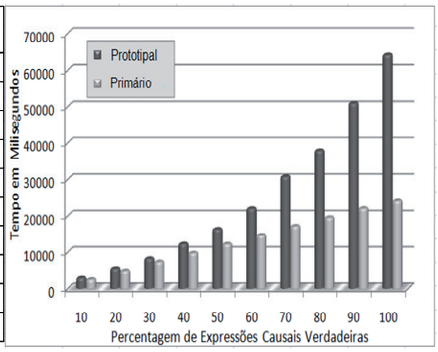


Figura 4. Resultados do 1º e 2º experimento entre as versões do *Framework* PON [Banaszewski, 2009].

7. Conclusões e Trabalhos Futuros

O PON se mostra promissor dado as suas características, o que se expressa em uma diversidade de publicações e ramificações de pesquisa. Entretanto, é necessário avançar com sua materialização para que se tenha em termos práticos o que se constata na teória, inclusive assintoticamente. Neste âmbito, o PON foi primeiramente materializado como um *Framework* Prototipal sendo subseqüentemente substituído por outro *Framework* chamado de Primário [Banaszewski, 2009]. Isto dito, os experimentos apresentados neste artigo demonstram que, em média, o *Framework* Primário apresenta performance duas vezes melhor em relação ao precedente.

Outrossim, um novo *Framework* mais recente (também em C++) substitui o *Framework* Primário, com novas melhorias como estruturas de dados otimizadas/dedicadas e otimizações algorítmicas pontuais [Valença et al., 2011][Simão et al., 2012b]. Este novo *Framework* apresentaria média de performance duas vezes melhor do que o Primário, se bem aplicado. Ainda, o novo *Framework* vem acompanhado de um *Wizard* que permite escrever *Rules* em alto nível (na forma de regras se-então) e mesmo compor *FBEs* em alto nível permitindo gerar código no formato do *Framework*.

Entretanto, o *Framework* em C++ poderia ser ainda mais otimizado até que atingisse um dado limite de otimização. Ainda, poderia-se mesmo compor um *Framework* em linguagem *assembly* para um conjunto de arquiteturas monoprocessadas particulares que sigam as arquiteturas microprocessadas usuais como *Von Neumann* e *Harvard*. Depois, o próximo passo seria elaborar uma linguagem-compilador para o PON que gerasse (em um primeiro momento) código *assembly* apropriado e otimizado para este mesmo conjunto de arquiteturas monoprocessadas.

Com uma linguagem-compilador específica, os programas PON poderiam apresentar melhor desempenho, em geral, que os construídos com outros paradigmas (e.g. POO/PI) em arquitetura monoprocessadas. Na falta de tal tecnologia, entretanto, o PON nem sempre apresenta melhor desempenho devido às sobrecargas de processamento das atuais materializações [Simão et al. 2012b]. Ainda assim, quando as redundâncias são significativas, a inferência do PON compensa as deficiências da atual materialização, mostrando-se mais eficiente [Banaszewski, 2009].

Por fim, na verdade, a materialização do PON sobre as tradicionais arquiteturas monoprocessadas prejudicaria suas aplicações uma vez que tais arquiteturas são sequencializadas enquanto o PON é naturalmente concorrente e distribuído. Mesmo que haja multi-processamento em arquitetura microprocessadas usuais e o PON venha a ser um meio para bem explorá-lo, cada unidade é monoprocessada o que gera a dicotomia citada. Isto dito, salienta-se que há pesquisas visando inclusive o PON via *FPGA*s (*Field Programmable Gate Arrays*) [Witt et al., 2010].

Referências bibliográficas

[Banaszewski et al., 2007] Banaszewski, R. F.; Stadzisz, P. C.; Tacla, C. A.; Simão, J. M. "Notification Oriented Paradigm (NOP): A software development approach based on artificial intelligence concepts," in Proceedings of the VI Congress of LAPTEC, Santos, Brazil, 2007

[Banaszewski, 2009] Banaszewski, R. F. "Paradigma orientado a notificações: avanços e comparações". Dissertação de Mestrado - CPGEI/UTFPR, Curitiba-PR, Brasil, 2009 – Disponível em: http://arquivos.cpgei.ct.utfpr.edu.br/Ano_2009/dissertacoes/Dissertacao_500_2009.pdf.

[Batista et al., 2011] Batista, M. V.; Banaszewski, R. F.; Ronszcka, A. F.; Valença, G. Z.; Linhares, R. R.; Stadzisz, P. C.; Tacla, C. A.; Simão J. M.; "Uma comparação entre o Paradigma Orientado a Notificações (PON) e o Paradigma Orientado a Objetos (POO) realizado por meio da implementação de um Sistema de Vendas". III Congreso Internacional de Computación y Telecom. COMTEL, Lima, Peru, Outubro de 2011.

[Brookshear, 2006] Brookshear, J. G. "Computer Science: An Overview". Addison Wesley, 2006.

[Gabbrielli e Martini, 2010] Gabbrielli, M., Martini, S. "Programming Languages: Principles and Paradigms. Series: Undergraduate Topics in Computer Science". 1st Edition, 2010, XIX, 440 p., Softcover.

[Delorie, 2003] Delorie, D; "DJGPP", 2003.

http://www.delorie.com/djgpp/ 2/2/2009.

[Linhares et al., 2011] Linhares, R. R.; Ronszcka, A. F. ; Valença, G. Z.; Batista, M. V.; Erig Lima, C. R.; Witt, F. A.; Stadzisz, P. C.; Simão J. M.; "Comparações entre o Paradigma Orientado a Objetos e o Paradigma Orientado a Notificações sob o

contexto de um simulador de sistema telefônico". III Congresso Intern. de Computación y Telecom. - COMTEL, Lima, Peru, Outubro de 2011.

[Ronszcka et al., 2011] Ronszcka, A. F.; Belmonte, D. L.; Valença, G. Z.; Batista, M. V.; Linhares, R. R.; Tacla, C. A.; Stadzisz, P. C.; Simão, J. M.; "Comparações quantitativas e qualitativas entre o Paradigma Orientado a Objetos e o Paradigma Orientado a Notificações sobre um simulador de jogo". III Congresso Intern. de Computación y Telecom. - COMTEL, Lima, Peru, Outubro de 2011.

[Valença et al., 2011] Valença, G. Z.; Banaszewski, R. F.; Ronszcka, A. F.; Batista, M. V.; Linhares, R. R.; Fabro, J. A.; Stadzisz, P. C.; Simão, J. M.; "*Framework* PON, Avanços e Comparações". III Simpósio de Computação Aplicada, Passo Fundo - RS, Brasil, 2011.

[Roy e Haridi, 2004] Roy, P. V.; Haridi, S. "Concepts, Techniques, and Models of Computer Programming". MIT Press, 2004.

[Scott, 2000] Scott, M. L. "Programming Language Pragmatics", 2º Edition, p. 8, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc, 2000.

[Simão, 2001] Simão, J. M. "Proposta de uma Arquitetura de Controle para Sistemas Flexíveis de Manufatura Baseada em Regras e Agentes". Dissertação M.Sc. CPGEI/UTFPR, Brasil, 2001.

[Simão, 2005] Simão, J. M. "A Contribution to the Development of a HMS Simulation Tool and Proposition of a Meta-Model for Holonic Control". Tese de Doutorado, CRAN-UHP (França) & CPGEI/UTFPR (Brasil) 2005.

http://arquivos.cpgei.ct.utfpr.edu.br/Ano_2005/teses/Tese_012_2005.pdf.

[Simão et al., 2012] Simão, J. M.; Banaszewski, R. F.; Tacla, C. A.; Stadzisz, P. C.; "Notification Oriented Paradigm (NOP) and Imperative Paradigm: A Comparative Study," Journal of Software Engineering and Applications – JSEA (V. 5 N. 6, 2012, pp. 402-416) doi:10.4236/jsea.2012.56047.

http://www.scirp.org/journal/PaperInformation.aspx?paperID=19842#abstract.

[Simão et al., 2012b] Simão, J. M.; Belmonte, D. L.; Ronszcka, A. F.; Linhares, R. R.; Valença, G. Z.; Banaszewski, R. F.; Fabro, J. A; Tacla, C. A.; Stadzisz, P. C.; Batista, M. V; "Notification Oriented and Object Oriented Paradigm comparison via Sale System". Journal of Software Engineering and Applications – JSEA, Paper accepted in July and foresaw to September 2012.

[Simão e Stadzisz, 2002] Simão, J. M.; Stadzisz, P. C. "An Agent-Oriented Inference Engine applied for Supervisory Control of Automated Manufacturing Systems". In: J. Abe & J. Silva Filho, Advances in Logic, Artificial Int. and Robotics (V. 85, p. 234-241). IOS Press Books, Amsterdam. 2002.

[Simão, Stadzisz e Künzle, 2003] Simão, J. M.; Stadzisz, P. C.; Künzle, L. "Rule and Agent-oriented Architecture to Discrete Control Applied as Petri Net Player". 4th Congress of Logic Applied to Technology, LAPTEC, 101, p. 217, 2003.

[Simão e Stadzisz, 2008] Simão, J. M.; Stadzisz, P. C. "Paradigma Orientado a Notificações (PON) – Uma Técnica de Composição e Execução de Software Orientado a Notificações". Pedido de Patente junto ao INPI/Brazil em 2008 e a Agência de Inovação/UTFPR em 2007. Nº INPI Efetivo PI0805518-1.

[Simão e Stadzisz, 2009] Simão, J. M.; Stadzisz, P. C. "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, doi:10.1109/TSMCA.2008.20066371, 2009.

[Simão, Stadzisz e Tacla, 2009] Simão, J. M., Stadzisz, P. C.; Tacla, C. A. "Holonic Control Meta-Model". IEEE Trans. on Systems, Man and Cybernetics. Part A, Systems and Humans, V.39, N.5, 1126-1139, 2009.

[Tanenbaum, Steen, 2002] Tanenbaum, A. S.; Van Steen, M. "Distributed Systems: Principles and Paradigms", Prentice Hall, 2002.

[Watanabe et al., 1997] Watanabe, H.; Tokuoka, H.; Wu, W.; Saeki, M. "A Technique for Analysing and Testing Object-oriented Software Using Coloured Petri Nets", IPSJ SIGNotes Software Eng. N.117, 1997.

[Wiecheteck, 2011] Wiecheteck L. V. B.; "Método para Projeto de Software usando o Paradigma Orientado a Notificações – PON". Dissertação M. Sc., CPGEI/UTFPR Curitiba-PR Brasil, 2011.

[Wiecheteck, Stadzisz e Simão, 2011] Wiecheteck, L. V. B.; Stadzisz, P. C.; Simão, J. M.; "Um Perfil UML para o Paradigma Orientado a Notificações (PON)". III COMTEL, Lima, Peru, Outubro de 2011.

[Witt et al., 2010] Witt, F. A. De ; Simão, J. M.; Linhares, R. R.; Stadzisz, P. C.; Lima, C. R. E.; "Comparação entre o Paradigma Orientado a Objetos (POO) e o Paradigma Orientado a Notificações (PON) em um Controle Discreto em Lógica Reconfigurável". XVI Seminário de Iniciação Científica e Tecnológica da UTFPR. Ponta Grossa-PR Brasil, Setembro 2011.