Notification-Oriented Software Design Methodology

I. Mendonça, J. Simão, and P. Stadzisz

***Abstract*—The Notification Oriented Paradigm (NOP) is an alternative paradigm of software programming, in which a rules-oriented and notifications approach about factual elements are applied. In Notification Oriented Paradigm, rules and factual elements are decomposed into a set of entities that communicate themselves directly through notifications, and it constitutes a process of active inference about the rules. The paradigm particulars also lead to challenges in the software design. For example, the full use of Unified Modeling Language (UML) is not suitable for the Notification Oriented Paradigm because this paradigm is not object-oriented. This paper proposes the Notification-Oriented software design Methodology (NOM), which applies concepts of software design within a new notation for modeling, defined as Holonic Flow Diagram (HFD). The Notification-Oriented software design Methodology was tried and evaluated in some use cases. In addition, it was conduced a focus group with Notification Oriented Paradigm expert developers that demonstrated the methodology viability.**

***Index Terms*— Modeling, Rule-oriented systems, Software engineering.**

I. Introdução

O desenvolvimento de software é uma atividade de alta complexidade, pois requer diferentes habilidades do desenvolvedor. Essa indústria enfrenta problemas relativos à baixa produtividade, demanda por maior qualidade do software e complexidade crescente [1], [2]. Alguns desses problemas estão relacionados aos paradigmas atuais de programação e desenvolvimento. Os imperativos, como a orientação a objetos (OO), comumente produzem códigos redundantes que causam, por exemplo, processamento desnecessário. Isso ocorre porque as linguagens nesses paradigmas levam a uma estruturação de código usando laços de repetição (e.g. *while* e *for*) sobre elementos passivos (e.g. atribuição de variáveis) e expressões causais (*if-then*), o que muitas vezes resulta em um software que executa atribuições e avaliações de expressões irrelevantes [3]. As referências [4]–[6] aprofundam esse tema.

Uma abordagem diferente é apresentada pelas linguagens do paradigma declarativo, a exemplo dos Sistemas Baseados em Regras (SBR). Nesses, o conhecimento é definido na forma de regras e fatos, e a execução do software é conduzida por um motor de inferência. Deste modo, o sistema não fica suscetível a problemas de códigos redundantes. Por outro lado, existe um custo elevado de processamento dos motores de inferência em termos de buscas necessárias [7].

Neste contexto, propostas de novos paradigmas, técnicas e ferramentas para desenvolvimento de software buscam contribuir para a resolução dos problemas ligados à produção de software. Um desses estudos, de 2001, é sobre uma abordagem de computação denominada Paradigma Orientado a Notificações (PON) [8]. O PON surgiu como alternativa aos atuais paradigmas de desenvolvimento na qual se aplica uma abordagem orientada a regras e notificações sobre elementos factuais. O PON traz contribuições significativas, mas cria desafios no processo de desenvolvimento. Em especial, o PON não é OO e, assim, a aplicação integral da UML não é adequada a ele [9].

Dentro do contexto apresentado, este trabalho propõe uma metodologia para projeto de software PON, denominada NOM (do inglês *Notification-Oriented software design Methodology*). A NOM emprega novos conceitos de concepção de software e aperfeiçoa o desenvolvimento de software PON por ser direcionada às primitivas do paradigma.

II. Paradigma Orientado a Notificações

O PON visa melhorar o desempenho das aplicações e facilitar o seu desenvolvimento. O paradigma apresenta uma perspectiva diferente na concepção de software e suas contribuições buscam minimizar problemas dos paradigmas atuais. O surgimento do PON foi influenciado pela teoria dos Sistemas de Manufatura Holônicos (HMS) [8], [10], nos quais utilizam-se os conceitos de Sistemas Holônicos e de hólons.

Um software desenvolvido no PON é organizado como um conjunto de regras decisionais e entidades factuais. O desenvolvedor elabora a solução orientada a regras. Esta estruturação pode facilitar o desenvolvimento pois permite manter o foco na lógica de tomadas de decisão que produz os resultados pretendidos. Esta é uma maneira mais abstrata de conceber software do que as abordagens algorítmicas usadas nos paradigmas de programação imperativos.

No PON, as entidades factuais representam o conhecimento (i.e., dados ou variáveis) que determina o estado do sistema e são declaradas na forma de classes e relacionamentos, análogo ao paradigma de programação orientado a objetos. Esta forma de declaração das entidades factuais oferece um alto nível de expressividade, útil para definir estas entidades e representar estruturas mais complexas, quando necessário.

Uma inovação do PON, que o difere de outras abordagens baseadas em regras, é que a inferência é realizada pela notificação direta entre as entidades factuais e regras decisionais. Assim, evita-se a necessidade de um motor de

I. T. M. Mendonça, Instituto Federal de Santa Catarina (IFSC), Florianópolis, SC 88015-640, Brasil, igor@ifsc.edu.br.

J. M. Simao, Universidade Tecnológica Federal do Paraná (UTFPR), Curitiba, PR, 80230-901 Brasil, jeansimao@utfpr.edu.br.

P. C. Stadzisz, Universidade Tecnológica Federal do Paraná (UTFPR), Curitiba, PR, 80230-901 Brasil, stadzisz@utfpr.edu.br.

inferência [11] para verificar quais regras estão habilitadas (i.e., *matching*) e quais serão executadas (*i.e., selection*).

As entidades factuais, ao detectarem uma mudança no seu estado (i.e., alteração de um dado ou valor de uma variável), notificam as regras que avaliam o estado destas entidades, conforme pré-estabelecido na concepção do software [12]. Assim, a inferência por notificações evita buscas contínuas sobre a base de regras, contribuindo para a melhoria do desempenho da aplicação. As entidades que compõem o PON estão ilustradas no seu metamodelo (Fig. 1). As entidades factuais são representadas pela classe FBE (*Fact Base Element*), incluindo objetos das classes *Attribute* e *Method*. As regras decisionais são representadas pela classe *Rule*, incluindo um objeto da classe *Condition* e um da classe *Action*, e associações com as classes *Premise* e *Instigation*.

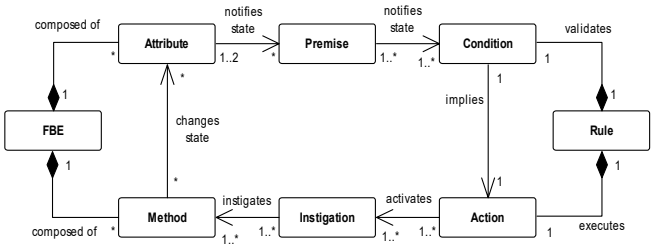

Fig. 1. Metamodelo do PON [10].

O FBE registra fatos do sistema por meio de *Attributes* e pode possuir *Methods* que consultam e modificam esses *Attributes*. A *Premise* é uma expressão lógica relacionando um *Attribute* a um valor (*e.g., Status == "Ligado"*). A *Condition* está associada a uma ou mais *Premises* que, quando verdadeiras, aprovam uma *Rule*. A *Rule* realiza alguma ação quando é aprovada e, para isso, possui uma *Action*. Cada *Action* irá notificar um conjunto de *Instigations* que, por sua vez, irão ativar um ou mais *Methods*, modificando um ou mais *Attributes*. Esta sequência constitui o mecanismo de inferência do PON, ilustrado na Fig. 2. Desse modo, aplicações em PON não dependem de um elemento centralizador, como um motor de inferência.

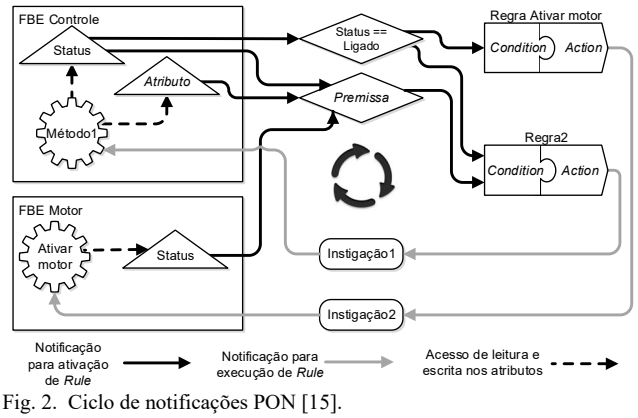

Fig. 2. Ciclo de notificações PON [15].

Nos últimos anos pesquisas sobre o PON têm sido conduzidas, incluindo a proposição de um *framework* [13] e de uma linguagem de programação, a LingPON [14]. Também, estudos sobre uma nova arquitetura de computação denominada NOCA e novas técnicas de implementação do paradigma estão em desenvolvimento [15], [16].

Contudo, essa abordagem apresenta novos desafios na modelagem de sistemas, sobretudo quando o número de regras aumenta. Segundo Moser et al. [17], a falta de suporte para estruturar um modelo de regras impõe limitações no tamanho do sistema. Corradini et al. [18] argumentam que a implementação de aplicativos usando regras é um processo propenso a erros e, portanto, necessita de métodos de análise formais. Para Huang et al. [19], é difícil obter uma base de conhecimento de produção consistente, completa e correta a partir de especialistas, traduzindo suas contribuições para uma representação computacional. Para Ericsson et al. [20], independentemente do sistema-alvo, o comportamento desses sistemas é conhecido por ser difícil de analisar.

No curso do desenvolvimento do PON, recorreu-se a diversas técnicas de modelagem, destacando-se o uso do Modelo de Classes da UML e de Redes de Petri. Em particular, foi desenvolvido o método Desenvolvimento Orientado a Notificações (DON) usando modelos UML e Redes de Petri [21]. O DON é a referência em projeto de software PON e tem sido empregado pelos pesquisadores no desenvolvimento de seus trabalhos. Porém, sentiu-se a necessidade de evoluir essa proposta uma vez que as escolhas para ela implicaram em algumas dificuldades, vivenciadas no uso do método. Por exemplo, o DON aplica uma abordagem UML convencional de modelagem orientada a objetos, o que difere da orientação a regras e notificações do PON e dificulta a modelagem de sistemas de regras e das primitivas do PON em diferentes níveis de abstração. Outra lacuna deixada pelo DON é que a UML não contribui para a descoberta das regras, gerando um intenso processo de síntese e requerendo muito esforço do desenvolvedor. Nota-se a falta de instrumentos (e.g., métodos, ferramentas e artefatos) que auxiliem o desenvolvedor na descoberta dos principais elementos que compõem a aplicação em PON, ou seja, Regras, Fatos e Notificações.

A falta de instrumentos para modelagem de sistemas baseados em regras que pudessem ser adaptados ao PON foi constatado em um mapeamento sistemático da literatura (MSL). Esse mapeamento foi conduzido durante o desenvolvimento da pesquisa e será apresentado resumidamente na próxima seção.

III. MÉTODOS

O problema identificado nesta pesquisa é a dificuldade em desenvolver softwares no PON, e determinou-se como objetivo criar uma metodologia de projeto de software que facilite a concepção de aplicações neste paradigma. A partir deste objetivo, iniciou-se um estudo exploratório para identificar se processos de software tradicionais, ou não, poderiam ser usados. Paralelamente, aprofundou-se no estudo sobre a teoria dos Sistemas Holônicos [22], a qual influenciou a criação do PON, paradigma para o qual a metodologia apresentada neste artigo foi concebida. Com base nessas pesquisas e na experiência adquirida com o uso do método existente (DON), foi elaborada a primeira versão da Metodologia de Projeto de Software Orientado a Notificações. Concomitantemente ao desenvolvimento da metodologia, conduziu-se um Mapeamento Sistemático da Literatura (MSL) para o período de 10 anos, entre 2006 e 2016, visando identificar técnicas, métodos, ferramentas, linguagens,

representações, especificações e instrumentos para a modelagem de regras. O mapeamento teve como questão de pesquisa: "Quais técnicas, métodos, ferramentas, linguagens, representações, especificações e instrumentos são usados para modelagem de sistemas nos quais regras são os elementos lógicos de modelagem?". A partir desta questão, chegou-se na seguinte *string* de busca: "("Causal rules" OR "ECA rules" OR "If-then rules" OR "Situation-action rules" OR "condition-action rules" OR "event-condition-action rules" OR "production rules") AND modeling AND (representation OR specification OR technique OR instrument OR tool OR framework OR LANGUAGE OR method OR methodology) AND NOT fuzzy". As bases consultadas foram Scopus, Science Direct, Engineering Village e IEEE Xplore. Foram considerados somente trabalhos escritos em inglês que tenham sido publicados em revistas, anais de eventos e padrões de especificação. O MSL evidenciou que processos de software em que regras são os elementos de concepção não possuem um padrão *de facto*, assim como há a UML para a OO. Percebeu-se, pelo mapeamento, dificuldades próprias da modelagem de sistemas baseados em regras; ainda assim, a maior parte dos trabalhos faz uso de processos tradicionais de software com alguma adaptação para sistemas de regras.

Para avaliar a NOM, foram criados cenários em que a metodologia pudesse ser aplicada e seus resultados verificados em termos de implementação no PON. Após a experimentação desses cenários, organizou-se um grupo focal exploratório e confirmatório com especialistas em PON para avaliar a qualidade, efetividade e aderência aos princípios do paradigma na metodologia proposta, e, ao mesmo tempo, propor melhorias da proposta apresentada. Participaram do grupo focal oito especialistas em PON em uma sessão que durou três horas. Como resultado, houve um incremento na NOM, apresentado a seguir. Outras considerações da avaliação pelos especialistas estão detalhadas na subseção "Grupo Focal".

IV. Metodologia de Projeto de Software Orientado a Notificações (NOM)

A metodologia proposta neste trabalho foi concebida para ser aderente aos modelos de processos de software tradicionais (e.g. cascata, espiral e incremental) e pode ser enquadrada na etapa de "Análise e Projeto" tanto em processos sequenciais quanto iterativos de desenvolvimento de software. O dado de entrada da NOM é a especificação de requisitos na forma de modelo de casos de uso. Na saída da NOM têm-se artefatos suficientes para que o software modelado possa ser codificado em PON (Fig. 3).

O processo da NOM envolve as três fases ilustradas na Fig. 3: modelagem conceitual, modelagem lógica e verificação e validação dos modelos. Na primeira, usa-se o Diagrama de Fluxo Holônico (DFH), criado para o PON e cujos elementos são inspirados em suas primitivas, direcionando os esforços de modelagem para a arquitetura do paradigma. Esta fase é o principal objeto de exposição e análise deste artigo. Na modelagem lógica considera-se a plataforma-alvo na qual a aplicação será executada; no exemplo, os diagramas ilustram o mapeamento para uma aplicação no *framework* PON [13]. Na terceira fase, faz-se a verificação e validação dos modelos por meio de Redes de Petri ou Diagramas de Estados.

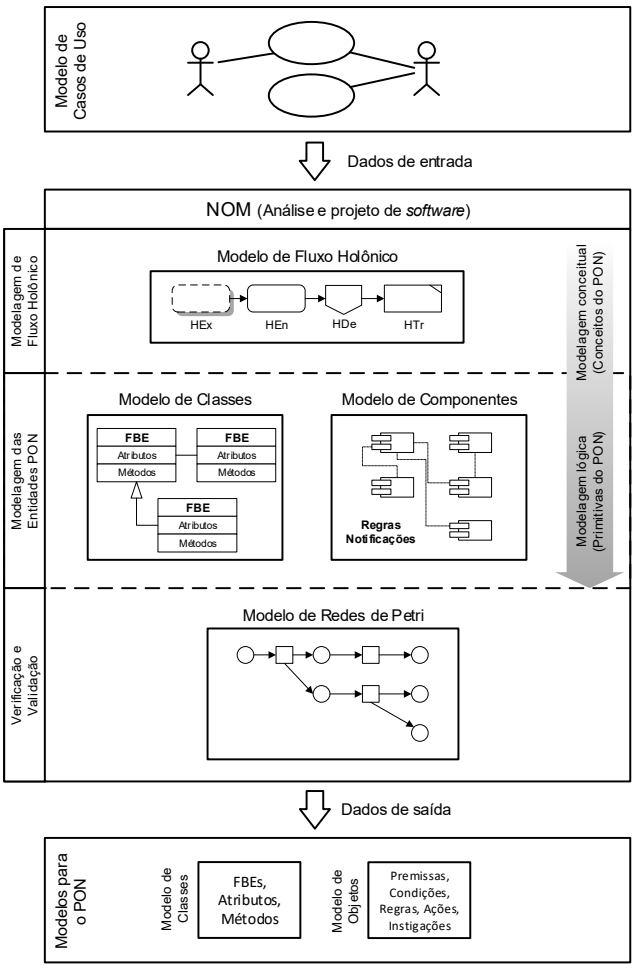

Fig. 3. Processo da NOM.

A. Modelo de Fluxo Holônico (MFH)

O conjunto de DFHs criado pela equipe de projetistas no desenvolvimento de uma aplicação PON define o Modelo de Fluxo Holônico (MFH). Cada DFH é criado a partir de um caso de uso. Pretende-se que, ao final desta modelagem, o projetista possa identificar o conjunto de regras e FBEs que constituirão o software PON. O MFH permite, também, produzir o desenho de arquitetura do software PON determinando a organização dos FBEs e das regras. Com isso, quer-se oferecer ao projetista formas de estruturar o software e lidar com as dificuldades apresentadas anteriormente por Moser et al. [17] e Ericsson et al [20].

A técnica aplicada no MFH, em geral, realiza refinamentos sucessivos a partir de uma visão de alto nível da solução, decompondo-a em níveis mais detalhados de acordo com a abordagem holônica. Desta forma, o MFH adota uma organização hierárquica para o desenvolvimento do modelo tornando-o mais detalhado e completo a cada iteração.

O MFH propõe a construção gradativa do modelo de fluxo lógico do software PON por meio de uma abordagem holônica, hierárquica e iterativa. Os elementos de decisão e transação foram criados a partir da primitiva *Rule* do PON. Do que antecede a *Rule* derivou-se o conceito de decisão e do que sucede a *Rule* derivou-se o conceito de transação. O MFH aplica um princípio de detalhamento dirigido às decisões e

transações. Este princípio determina que uma decisão ou uma transação deve ser decomposta (ou seja, detalhada) em um novo conjunto de decisões e transações mais refinadas. Assim, uma decisão de alto nível é detalhada na forma de decisões e transações de menor granularidade que, em conjunto, produzem a decisão de mais alto nível. Similarmente, uma transação de alto nível é detalhada em decisões e transações de menor granularidade que, em conjunto, produzem a transação de mais alto nível. Dentro de cada nível de detalhamento, os elementos modelados são relacionados por um fluxo, ou seja, um encadeamento das decisões e transações. As decisões e transações são, também, relacionadas a entidades internas e externas nelas envolvidas.

B. Notação Proposta para o DFH

O Diagrama de Fluxo Holônico (DFH) define cinco tipos de hólons e dois tipos de relações. Cada elemento possui semântica e notação próprias. A notação é apresentada na Fig. 4 e a seguir são apresentados os elementos e suas respectivas semânticas:

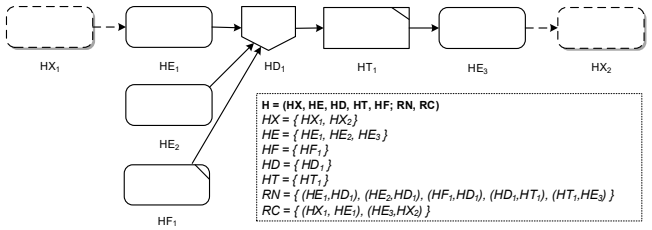

Fig. 4. Representação gráfica e declarativa de um DFH.

1) Hólons de Entidade Externa (HX)

Representam entidades externas ao software, como um usuário, outro software ou dispositivo. O projetista deve usar este tipo de hólon para modelar essas entidades envolvidas no fluxo do software. Este hólon equivale ao ator nos diagramas de casos de uso.

2) *Hólons de Entidade (HE)*

Representam os dados usados nos fluxos do software sendo modelado. O projetista pode usar este tipo de hólon para definir entidades que representem estados ou fatos, armazenem dados de entrada, forneçam dados de saída ou armazenem dados durante a execução do software.

3) Hólons Decisionais (HD)

Representam decisões e direcionam o fluxo de execução do software. O projetista pode usar este tipo de hólon para definir tomadas de decisão baseadas em estados ou fatos do software.

4) Hólons Transacionais (HT)

Representam entidades que realizam ações. O projetista deve usar este tipo de hólon na identificação de ações que são realizadas no contexto de um DFH. Um HT deve realizar uma ação no fluxo sendo modelado e pode atuar criando ou modificando valores em HE.

5) Hólon FBE (HF)

Representa um elemento lógico do PON denominado FBE. O projetista pode utilizar o HF quando entender que a definição prematura desse elemento auxilia na contextualização do problema sendo modelado em termos de PON ou quiser reusar modelos previamente concebidos.

6) Relações de notificação

Representam uma relação de fluxo dirigida entre dois hólons por meio de notificação, ou seja, aquelas que ocorrem por meio do fluxo pré-definido no PON (cf. Fig. 1). Uma notificação entre dois hólons define a habilitação de um hólon seguinte dando continuidade ao fluxo lógico de execução do software.

7) Relações de chamada

Representam, também, uma relação de fluxo dirigida entre dois hólons, sendo ao menos um deles um HX. No nível mais detalhado de modelagem, são os links dos elementos lógicos do PON com entidades externas ao paradigma.

Os hólons no DFH são interligados por arcos direcionais que definem a direção do fluxo modelado e a relação entre eles, seja de notificação ou de chamada. Esses arcos podem conter uma descrição para indicar a semântica da relação.

Um DFH é formalmente definido por:

(a) Definição. Uma sétupla *H = (HX, HE, HD, HT, HF; RN, RC)* é chamada de DFH se e somente se:

(i) *HX, HE, HD, HT e HF* são conjuntos disjuntos (os elementos de HX são chamados Hólons de Entidade Externa, os elementos de HE são chamados Hólons de Entidade, os elementos de HD de Hólons Decisionais, os elementos de HT são chamados Hólons Transacionais e os elementos de HF são chamados Hólons FBE).

(ii) *RN* é uma relação binária, dada pelo produto cartesiano dos conjuntos definidos em (i) excluindo as relações $(HX \times HX) \cup (HX \times HE) \cup (HE \times HX) \cup (HX \times HD) \cup (HD \times HX) \cup (HT \times HX) \cup (HX \times HT) \cup (HF \times HX) \cup (HX \times HF)$, denominada relação de fluxo holônico por **notificação** de *RN*.

(iii) *RC* é uma relação binária, dada pelo produto cartesiano dos seguintes conjuntos definidos em (i) $(HX \times HX) \cup (HX \times HE) \cup (HE \times HX) \cup (HX \times HD) \cup (HD \times HX) \cup (HX \times HT) \cup (HT \times HX) \cup (HX \times HF) \cup (HF \times HX)$, denominada relação de fluxo holônico por **chamada** de *RC*.

Graficamente, os elementos HX, HE, HD, HT e HF são representados conforme os símbolos da Fig. 4. A relação de fluxo holônico por notificação RN é representada por arcos direcionais contínuos entre os elementos do diagrama de fluxo holônico e a relação de fluxo holônico por chamada RC é representada por arcos direcionais tracejados entre os elementos do diagrama de fluxo holônico.

(b) Notação. Seja *H = (HX, HE, HD, HT, HF; RN, RC)* um DFH. Os sete componentes HX, HE, HD, HT, HF, RN e RC são denotados HX_H, HE_H, HD_H, HT_H, HF_H, RN_H e RC_H, respectivamente.

A Fig. 4 ilustra um exemplo hipotético de DFH na forma gráfica e declarativa. Conforme descrito no início desta seção, os hólons podem ser detalhados, com exceção do HX e HF. Assim, quando há o detalhamento, o hólon tem sua representação gráfica modificada para possuir uma cor de preenchimento (por padrão cinza) e um ícone com um sinal de mais (+). Adicionalmente, o projetista pode definir hólons iniciais do diagrama, que destacam os elementos com bordas mais grossas. Assim, essas referências visuais chamam a

atenção para o início do fluxo apresentado no diagrama.

O elemento HF, por ser um elemento lógico do PON, possui propriedades que representam seus *Attributes* e *Methods*. Essas propriedades são usadas para associar o hólon às relações do DFH. Por exemplo, um hipotético HF chamado "Controle" pode ter uma propriedade chamada "Botão" que se relaciona com um HD "Abrir portão?".

V. Avaliação da NOM

A. Cenário de Estudo

No intuito de avaliar e demonstrar a viabilidade de uso da NOM, alguns cenários de estudo foram elaborados. Optou-se por apresentar neste trabalho um exemplo de modelagem de uma operação de um caixa eletrônico (ATM), pois ele apresenta certa complexidade de operações e seu funcionamento geral é de amplo conhecimento. Do ponto de vista de modelagem e programação de sistemas baseados em regras, softwares que possuem diversos fluxos que se entrelaçam, como o escolhido neste cenário de estudo, são especialmente difíceis de criar devido à característica de desacoplamento definida pela estruturação do software por regras, ou seja, o fluxo de encadeamento lógico do software se dá por meio da avaliação e execução de um conjunto de regras e não de uma sequência de instruções. A operação escolhida foi a de "Sacar dinheiro". O caso de uso dessa operação possui o mesmo nome e seu fluxo principal e um dos fluxos de exceção são apresentados na Tabela 1.

TABELA I
FLUXOS PRINCIPAL E DE EXCEÇÃO DO CASO DE USO SACAR DINHEIRO

Fluxo principal
1. O cliente verifica se o ATM está operando normalmente.
2. O cliente insere o seu cartão no ATM.
3. O ATM efetua a validação do cartão.
4. O ATM solicita a senha do cliente.
5. O cliente digita sua senha.
6. O ATM cria uma sessão para o cliente.
7. O ATM mostra as opções para o cliente.
8. O cliente escolhe a opção "Sacar dinheiro".
9. O ATM solicita o valor desejado.
10. O cliente digita o valor desejado.
11. O ATM verifica se o cliente tem saldo suficiente.
12. O ATM verifica se possui saldo suficiente na máquina.
13. O ATM dispensa o dinheiro solicitado.
14. O ATM mostra a opção de imprimir recibo da transação.
15. O cliente escolhe não imprimir o recibo da transação.
16. O ATM fecha a sessão do cliente.
Fluxo de exceção: Cartão bloqueado
1. O cliente verifica se o ATM está operando normalmente.
2. O cliente insere o seu cartão no ATM.
3. O ATM efetua a validação do cartão.
Exceção: [Cartão bloqueado].
4. O ATM registra a tentativa de uso de cartão bloqueado.
5. O ATM exibe uma mensagem de tentativa com cartão bloqueado.
6. Retorna ao passo 2.

DFHs são criados para cada caso de uso, realizando refinamentos sucessivos em uma organização holônica. Porém, isso não significa que há uma única forma de conceber esses diagramas. Como se trata de um processo de engenharia, cada projetista fará uso do diagrama conforme seu raciocínio lógico, mas respeitando as convenções elaboradas para o modelo. Além disso, a organização do diagrama com a abordagem holônica permite que o projetista faça uso dele em largura ou em profundidade, ou seja, ele pode criar elementos mais abstratos e depois realizar detalhamentos para cada um deles ou criar elementos mais detalhados e posteriormente agrupá-los em elementos mais abstratos. Neste trabalho optou-se por apresentar a abordagem em largura.

Cada caso de uso tem a função de descrever um uso de valor que o sistema deverá ter. Assim, o projetista precisa compreender o caso de uso antes de iniciar a criação de um DFH. Para iniciar a modelagem do DFH para o Caso de Uso "Sacar dinheiro" criou-se um Hólon Transacional de mesmo nome que representa o fluxo holônico inteiro desse caso de uso. A este HT foram relacionados os fluxos com as entidades externas ao software, neste diagrama representadas pelos HXs. A representação visual dessa primeira iteração no DFH é ilustrada na Fig. 5. O HT "Sacar dinheiro" está preenchido de cinza pois ele será detalhado na sequência.

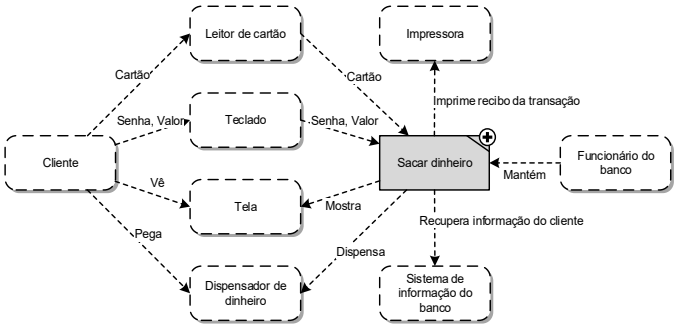

Fig. 5. Primeiras iterações no DFH "Sacar dinheiro".

O cerne do MFH são os elementos decisionais e transacionais que, em conjunto, definem a lógica do software. O elemento decisional adiciona ao modelo os possíveis caminhos e alternativas de fluxos que se deseja modelar. Assim, para este caso de uso que possui diversos fluxos (dos quais três serão demonstrados) optou-se pela criação das decisões que determinam primeiro o fluxo de sucesso do caso de uso, ou seja, o fluxo principal. Após análise do Caso de Uso "Sacar dinheiro", e algumas iterações, chegou-se ao detalhamento do HT "Sacar dinheiro" apresentado na Fig. 6. Em cada ponto do caso de uso em que há um fluxo alternativo, ou de exceção, criou-se um Hólon Decisional (HD) que, ao mesmo tempo em que representa o fluxo de sucesso, será detalhado para representar o fluxo alternativo. No interior do HD descreve-se textualmente o que ele representa, de preferência com perguntas, e as relações de saída são rotuladas com a condição que a satisfaz para gerar aquele fluxo ou com informações do fluxo que ela representa. Os HDs apresentados na Fig. 6 foram criados a partir dos passos 1, 2, 3, 5, 8, 11, 12 e 15 da Tabela 1, respectivamente. Ao detalhar um hólon, as relações que existem para ele estarão disponíveis para serem associadas ao detalhamento, bem como novas relações podem ser criadas para representar os fluxos detalhados. É possível visualizar esta situação comparando a Fig. 5 com a Fig. 6. Por exemplo, o HX "Leitor cartão" passa a se relacionar com o HD "O cartão foi inserido?".

Na sequência do processo inicia-se o detalhamento dos hólons para refletir a especificação de caso de uso dada.

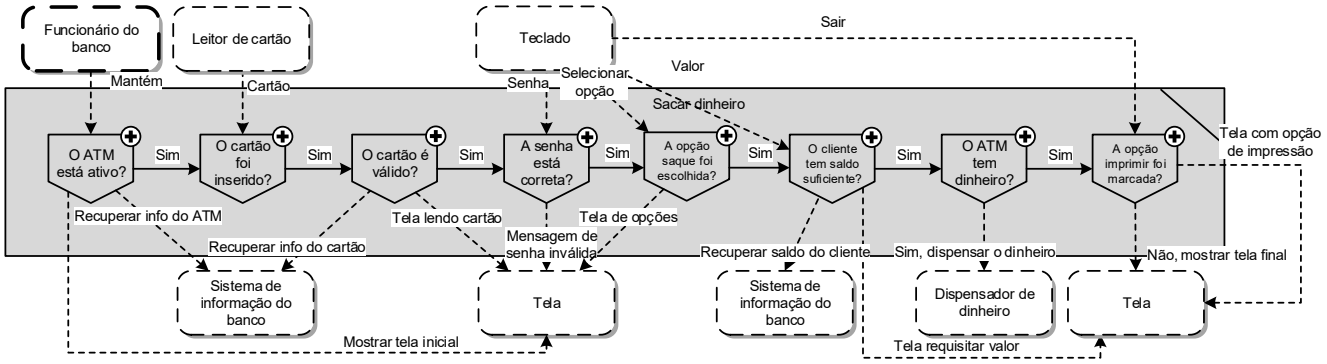

Fig. 6. Detalhamento do HT "Sacar dinheiro".

O fluxo de exceção apresentado no caso de uso da Tabela 1 reflete uma ocorrência em que se tenta acesso com um cartão inválido. Os passos 1 a 3 são os mesmos do fluxo principal e o HD "O cartão é válido?" representa a verificação do cartão apresentado no passo 3. Assim, detalhou-se o HD "O cartão é válido?" (Fig. 7) para continuar refletindo o sucesso na validação do cartão e atuar no fluxo de exceção.

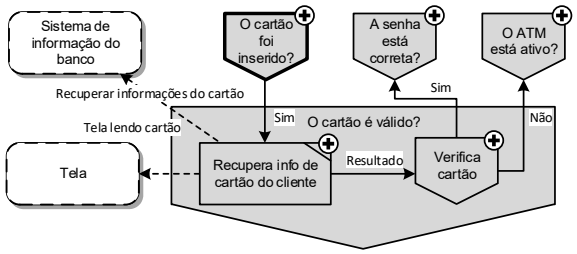

Fig. 7. Detalhamento do HD "O cartão é válido?".

No detalhamento do HD "O cartão é válido?", os 2 HX e 2 HD que já se relacionavam com ele continuam no diagrama detalhado. Um novo hólon HD e uma relação de notificação foi identificada pois, quando ocorre a exceção que se está modelando, o fluxo deve retornar para o início do caso de uso, conforme indicado no passo 6 do fluxo de exceção ilustrado na Tabela 1. O fluxo que inicia este hólon detalhado vem do HD "O cartão foi inserido?". Esse fluxo foi relacionado com um HT "Recupera info de cartão de cliente", recém-criado, pois entende-se que o ATM precisa, primeiro, recuperar as informações do cartão nele inserido. Como se trata de "ações" realizadas pelo sistema e há interações com elementos externos, optou-se pelo uso do hólon transacional. Este HT recupera a informação no HX "Sistema de informação do banco" e exibe na tela a informação de que está lendo o cartão. Neste ponto, a característica intrínseca de execução paralela do PON aparece, pois o DFH permite que o projetista defina uma ordem de execução dos fluxos de saída em seus rótulos. No caso desse DFH, a omissão significa que ambos podem executar simultaneamente. O detalhamento do HT "Recupera info de cartão de cliente" é apresentado na Fig. 8.

Ainda no HD detalhado, ilustrado pela Fig. 7, foi criado um HD para modelar o próximo passo do caso de uso. Esse HD, chamado "Cartão verificado?", recebe fluxo do HT, já mencionado, e com as informações recebidas decide pela continuação do fluxo principal para o HD "A senha está correta?" ou, caso verifique inconsistência, decide pelo fluxo de exceção, voltando para a tela inicial.

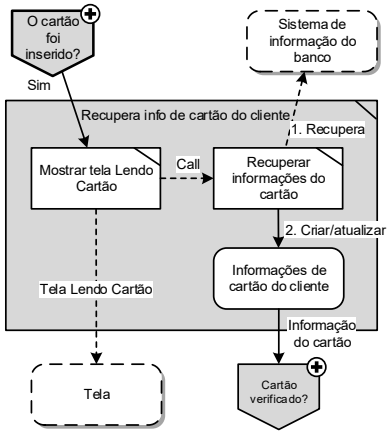

Fig. 8. Detalhamento do HT "Recupera info de cartão do cliente".

O detalhamento continua: o HT "Recupera info de cartão do cliente", ilustrado na Fig. 8, mostra uma mensagem na tela e recupera as informações do cartão do cliente em um hólon externo, criando um hólon entidade para representar essas informações (HE "Informações de cartão do cliente"). O HE é responsável por desencadear a continuidade para o HD "Cartão verificado?" que está detalhado na Fig. 9.

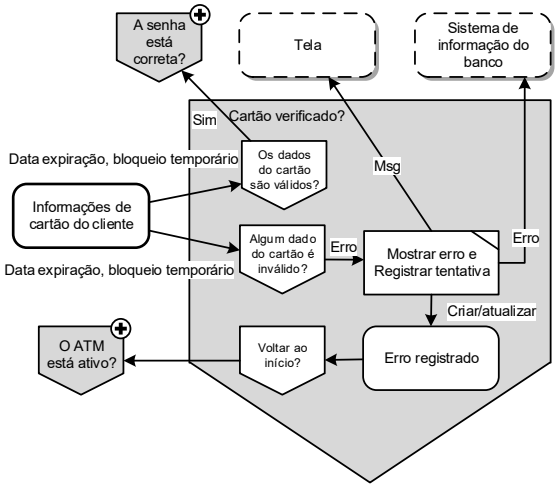

Fig. 9. Detalhamento do HD "Cartão verificado?".

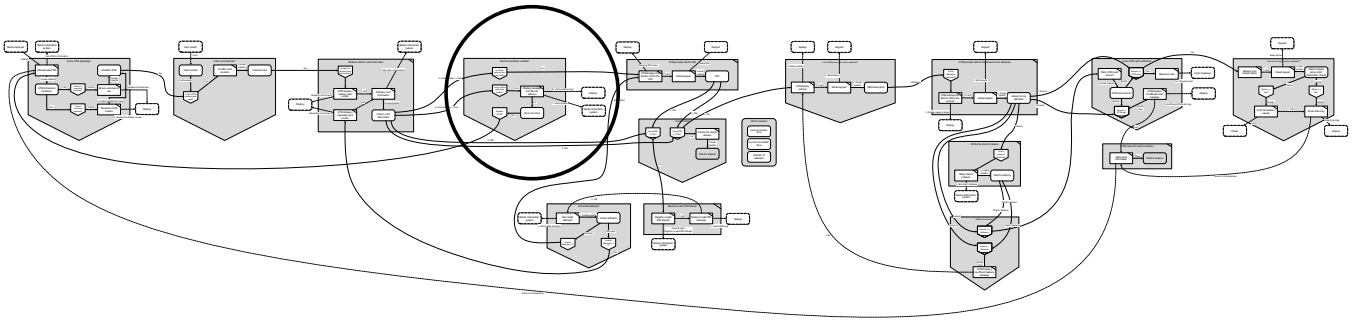

Fig. 10. Visão planificada do DFH "Sacar dinheiro", destaque no HD "Cartão Verificado?".

Caso as informações estejam corretas, o fluxo principal segue para o HD "A senha está correta?"; caso contrário é exibida uma mensagem na tela e realizado o registro de tentativas, retornando para o hólon que apresenta a tela inicial do ATM. No detalhamento do HD "Cartão verificado?" foram criadas relações com dois HX; a apresentação dessas relações nos níveis anteriores é opcional, mas pode auxiliar na compreensão do modelo.

O detalhamento do modelo deve continuar até que os cenários do caso de uso sejam atendidos. Aqui, a apresentação de algumas de suas partes visa ilustrar, além de um caso real de modelagem usando o DFH, os passos de concepção desse modelo. Ao final do processo é possível criar uma visão integrada do modelo, em que os hólons que possuem relações são apresentados em um modelo planificado. A Fig. 10 apresenta o modelo planificado do DFH criado a partir do caso de uso "Sacar dinheiro". Nesta figura é destacada, ainda, a posição em que o HD "Cartão verificado?" está no modelo. Adicionalmente, a modelagem planificada permite que o projetista tenha uma referência visual da quantidade de notificações e relações entre os elementos concebidos.

B. Verificação e Validação dos Modelos e Geração de Códigos

O processo de concepção usando o DFH, a partir dos casos de uso, permite ao projetista conceber uma solução PON com elementos em um fluxo dando origem às primitivas (regras e notificações) que comporão o código do software. Assim, para verificar e validar se o modelo possui essa compatibilidade, foram mapeadas as relações permitidas de forma que um algoritmo possa realizar as verificações. Por exemplo, no nível mais detalhado não é permitida a relação de notificação partindo de um HE para um HT, pois na lógica do PON as transações são executadas em decorrência das decisões. O algoritmo pode indicar essa inconsistência e, inclusive, sugerir como o modelo pode ser modificado para possuir uma lógica compatível com o PON. A Fig. 11 apresenta a parte do algoritmo que faz essa verificação (em português estruturado) e sua visualização gráfica. Outras verificações possíveis são relativas à lógica do PON, como a garantia de determinismo. Assim, o algoritmo identificaria os locais em que o PON não garante o determinismo e o projetista poderia melhorar a modelagem.

O projeto concebido na NOM deve ser compatível com o dado de entrada, ou seja, o caso de uso. Assim, a partir do DFH pode-se mapear a solução para outras ferramentas ou técnicas de validação, como, por exemplo, mapear o DFH para um diagrama de estados ou para Redes de Petri e verificar a compatibilidade com os dados de entrada.

Algoritmo:

```
01  Para cada relação de notificação RN partindo de um HE
02    Se a relação RN não for com um HD
03      Erro!
```

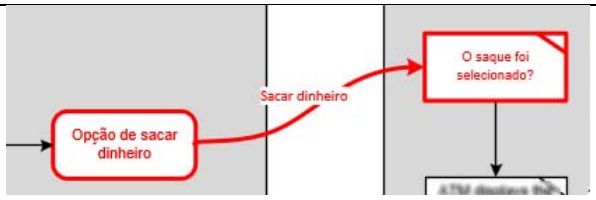

Fig. 11. Apresentação de um pequeno fragmento do algoritmo de verificação e a respectiva visualização gráfica destacada de incompatibilidade encontrada.

A partir do DFH criado percebe-se que os elementos HD são os gatilhos que provocam as mudanças de estado. Assim, é possível criar um diagrama de estados no qual cada HD é representado por uma ou mais transições de estados. Alguns experimentos foram realizados com o mapeamento do DFH para um diagrama de estados em que foi possível, visualmente, comparar e validar os estados com o caso de uso.

É importante ressaltar que no atual estágio de desenvolvimento da NOM não há uma ferramenta de modelagem própria. Assim, os diagramas apresentados neste trabalho foram elaborados com auxílio da ferramenta Microsoft Visio. A ferramenta permite a criação de elementos personalizados, porém não suporta a modelagem de níveis hierárquicos e o detalhamento dos elementos segundo a teoria dos sistemas holônicos, tampouco permite a definição de metamodelos para validação dos modelos. Pretende-se desenvolver uma ferramenta própria para modelagem de software PON usando a NOM que comporte as etapas de verificação e validação. Atualmente essas etapas são realizadas visualmente pelo projetista.

A partir dos modelos DFH detalhados, verificados e validados, pode-se mapear a solução para o framework PON. Foram realizados experimentos utilizando a LingPON[1] como camada intermediária de geração de código para o framework PON. Nesse processo, usam-se somente os hólons que não possuem detalhamentos, ou seja, aqueles que representam o nível mais baixo de abstração na modelagem e que são apresentados no DFH na cor branca.

[1] LingPON é a linguagem de programação do PON, em que seus elementos podem ser definidos. Ela possui um compilador que transforma o código PON para códigos alvo, por exemplo, C, C++ ou Framework PON., que podem, por fim, ser compilados para criar os arquivos executáveis.

Para cada HE ou HT foram criados FBEs com um *Attribute* ou um *Method*, respectivamente.

Assim, foi possível representar as entidades factuais do modelo e como elas serão modificadas durante o fluxo de execução do PON. Para os FBEs de HT podem ser criados *Methods* PON e imperativos. Para os casos de relação de notificação do HT com um HE foram criados *Methods* PON e para os casos de relações de chamada entre HT e HX foram criados *Methods* do paradigma imperativo; estes últimos o desenvolvedor deverá programar na linguagem imperativa do framework, pois, em geral, trata-se das entidades externas, que são as interfaces com o usuário. Os HF são mapeados diretamente como FBE já que eles são a própria representação lógica desse tipo de elemento. Os HD transformam-se em *Rules*, sendo as relações de notificação que chegam a eles suas *Premises* e sua *Condition* e as relações de notificações de saída para o HT sua *Action* e as *Instigations*. A partir do modelo criado para o caso de uso "Sacar dinheiro" foi possível gerar o código e, após compilações, executá-lo.

C. Grupo Focal

Para avaliar a proposta de metodologia apresentada neste trabalho, aproveitou-se do grupo de pesquisa em PON, especialistas no paradigma, para condução de um grupo focal exploratório e confirmatório conforme procedimentos apresentados por Dresch et al. [23]. Participaram oito especialistas, identificados como E1 a E8. Outras duas pessoas fizeram parte: o proponente da NOM como moderador, conduzindo os trabalhos, e uma ouvinte, para realizar anotações de detalhes que poderiam passar despercebidos pelo moderador. Além disso, foi solicitado e autorizado pelos participantes que a sessão fosse gravada em áudio para posterior transcrição e análises.

O grupo focal foi planejado para ter duração de três horas e estruturado na forma de uma apresentação da NOM e de sua aplicação em um caso de uso. Usou-se o exemplo "Sacar dinheiro" do ATM, completamente modelado pelo DFH, e, à medida que o diagrama e sua modelagem foram sendo apresentados, abria-se tempo para discussão e contribuições.

Das principais contribuições práticas dos participantes que foram incorporadas à metodologia, uma é a adição do elemento HF ao DFH. Percebeu-se que a existência desse elemento lógico do PON permitiria um maior poder de modelagem e reuso, pois, muitas vezes, o projetista já tem em mente como esses elementos serão organizados, ou poderá usar elementos recuperados de projetos anteriores. Outra contribuição importante foi no aspecto visual do DFH. Os especialistas perceberam certa dificuldade em identificar quais hólons possuem detalhamentos. Assim, foi incorporado um sinal de mais (+) aos hólons que possuem detalhamentos.

Durante o grupo focal, os especialistas, por vezes, sentiram a necessidade de elementos adicionais no DFH. Porém, após diversas iterações, chegou-se ao consenso que os elementos apresentados são suficientes e que somente a adição do elemento HF seria útil, conforme descrito anteriormente.

Dentre os aspectos da NOM que foram avaliados positivamente pelos especialistas, está a apresentação em um nível mais abstrato do fluxo principal do caso de uso (Fig. 6). O E1 enalteceu como interessante que o modelo apresente claramente o caminho base, sendo ele o norteador dos próximos passos da modelagem, adicionando que isso permite uma visão mais clara para o projetista. Ainda sobre o que é apresentado na Fig. 6, o E2 ressaltou que esse exemplo é uma possibilidade de modelagem a questionamentos, concluindo que o vê como uma contribuição interessante na abordagem. Um *insight* que se pode ter é que a modelagem a regras começa pela modelagem por questionamentos. O E3 fez uma crítica em relação à dificuldade em implementar software PON se comparado a outras linguagens, mas o E1 destacou que, por permitir a visualização dos fluxos, o DFH facilita o desenvolvimento já que a lógica do PON não é sequencial.

O E4 mostrou preocupação nos modelos criados com o DFH questionando se não ficariam demasiadamente complexos, mas concordou que não foi o caso para a modelagem apresentada "Sacar dinheiro" (Fig. 10). Nos modelos criados até o momento não foi identificada essa característica. Diferente disso, percebe-se que a organização por meio de holarquias (as hierarquias de hólons) colabora para que os hólons se agrupem por suas relações. O E5 sugeriu marcar, de alguma forma, quando os elementos do DFH passam a representar entidades do PON, como regras, pois nesse momento o conhecimento que veio do caso de uso se transforma em regra. De fato, essa transição não ficou devidamente destacada na metodologia, porém, conforme descrito na seção "Verificação e Validação dos Modelos e Geração de Códigos", pode-se, a partir do modelo detalhado, realizar o mapeamento para os elementos do PON.

O grupo focal, além da validação pelos especialistas, permitiu um incremento importante na metodologia NOM; aqui foi apresentada parte das contribuições ou o que foi entendido como consenso. Outras discussões e seus desdobramentos estão sendo tratados pelo grupo de pesquisa. É importante ressaltar que a participação dos membros do grupo de pesquisa em PON no grupo focal pode constituir um risco à validação da metodologia. Porém, devido ao estágio de maturidade do paradigma, os especialistas são os próprios pesquisadores. Ademais, esta seção demonstrou a efetividade do grupo focal uma vez que a sua aplicação resultou em incrementos importantes na metodologia.

IV. Discussões e Conclusões

A NOM se fundamenta na teoria dos sistemas holônicos para delinear o seu principal modelo, o Modelo de Fluxo Holônico (MFH), usado na concepção de software PON. O MFH define elementos (hólons) baseados nas primitivas do PON. Assim, o projetista é direcionado a soluções aderentes ao PON desde os primeiros estágios do processo de modelagem. Essa teoria também favorece a organização dos elementos no processo de modelagem já que os dispõe em holarquias, atuando na dificuldade de organização desse tipo de sistema apresentada por Moser et al. [17]. O MFH difere de outras abordagens mais tradicionais nas quais somente atividades ou processos são detalhados, pois permite o detalhamento dos elementos decisionais. Assim, abrem-se possibilidades para modelagem de sistemas nos quais as decisões possuem papel de destaque.

A metodologia apresentada para concepção de software PON não desconsidera o uso de modelos existentes. Por exemplo, o modelo de casos de uso da UML é o dado de entrada da NOM. Adicionalmente, apesar de não ser o foco

deste artigo, a metodologia sugere o uso de modelos da UML na modelagem lógica, especialmente quando a plataforma-alvo do software é o framework PON [13].

Os algoritmos de análise propostos e apresentados em seção anterior auxiliam o desenvolvedor ainda na fase de projeto do sistema. Com isso, mitigam-se problemas apresentados sobre SBRs e a necessidade de métodos de análise formais [18] [20].

Adicionalmente, foi possível gerar parte do código LingPON (não exibido no artigo por questões de espaço). Assim, percebe-se que foi cumprido o objetivo da metodologia de entregar artefatos suficientes para que o programador possa codificar em linguagem PON o que projetou.

REFERÊNCIAS

[1] D. Longstreet, "History of Software Productivity," 2006. [Online]. Available: http://www.softwaremetrics.com/Articles/history.htm. [Accessed: 21-Sep-2015].

[2] G. A. Liebchen and M. Shepperd, "Software productivity analysis of a large data set and issues of confidentiality and data quality," *Proc. - Int. Softw. Metrics Symp.*, vol. 2005, no. Metrics, pp. 399–401, 2005.

[3] S. H. Kaisler, *Software paradigms*, 1st ed. Hoboken: John Wiley & Sons, Inc., 2005.

[4] R. F. Banaszewski, "Paradigma Orientado a Notificações : Avanços e Comparações," Dissertação (Mestrado), Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial, Universidade Tecnológica Federal do Paraná, Curitiba, 2009.

[5] A. F. Ronszcka, "Contribuição para a concepção de aplicações no Paradigma Orientado a Notificações (PON) sob o viés de padrões," Dissertação (Mestrado), Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial, Universidade Tecnológica Federal do Paraná, Curitiba, 2012.

[6] R. D. Xavier, "Paradigmas de desenvolvimento de software: comparação entre abordagens Orientada a Eventos e Orientada a Notificações," Dissertação (Mestrado), Programa de Pós-Graduação em Computação Aplicada, Universidade Tecnológica Federal do Paraná, Curitiba, 2014.

[7] E. Friedman-Hill, *Jess in Action: Rule Based System in Java*. Greenwich, CT: USA: Manning Publications Co, 2003.

[8] J. M. Simão, "Proposta de uma Arquitetura de Controle para Sistemas Flexíveis de Manufatura Baseada em Regras e Agentes," Dissertação (Mestrado), Universidade Tecnológica Federal do Paraná - UTFPR, Curitiba, 2001.

[9] J. M. Simão, P. C. Stadzisz, and L. V. B. Wiecheteck, "Perfil UML para o Paradigma Orientado a Notificações (PON), Perfil UML para o Paradigma Orientado a Regras (POR), Método de Desenvolvimento Orientado a Notificações (DON) e Método de Desenvolvimento Orientado a Regras (DOR)," INPI Provisory Number: BR 10 2012 026430 7, 2012.

[10] J. M. Simão, "A contribution to the development of a HMS simulation tool and proposition of a meta-model for holonic control," Tese (Doutorado), Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial, Centro Federal de Educação Tecnológica do Paraná, Curitiba, 2005.

[11] G. J. Nalepa, S. Bobek, A. Ligęza, and K. Kaczor, "Algorithms for rule inference in modularized rule bases," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6826 LNCS, no. June 2015, pp. 305–312, 2011.

[12] 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. Syst. Man Cybern. Part A, Syst. Humans*, vol. 39, no. 1, pp. 238–250, 2009.

[13] A. F. Ronszcka, G. Z. Valença, R. R. Linhares, J. A. Fabro, P. C. Stadzisz, and J. M. Simão, "An Implementation Based On Design Patterns," *IEEE Lat. Am. Trans.*, vol. 15, no. 11, pp. 2220–2231, 2017.

[14] C. A. Ferreira, "Linguagem e compilador para o paradigma orientado a notificações (PON): Avanços e comparações," Dissertação (Mestrado), Programa de Pós-Graduação em Computação Aplicada, Universidade Tecnológica Federal do Paraná, Curitiba, 2015.

[15] R. R. Linhares, J. M. Simão, and P. C. Stadzisz, "NOCA - A Notification-Oriented Computer Architecture," *IEEE Lat. Am. Trans.*, vol. 13, no. 5, pp. 1593–1604, 2015.

[16] R. Kerschbaumer, J. M. Simao, J. A. Fabro, C. R. Erig Lima, and R. Ribeiro Linhares, "A Tool for Digital Circuits Synthesis Based on Notification Oriented Paradigm," *IEEE Lat. Am. Trans.*, vol. 16, no. 6, pp. 1574–1586, Jun. 2018.

[17] R. Moser, S. Dustdar, J. Gutleber, and L. Orsini, "A Fresh Look at Modeling Distributed Reactive Systems," in *2013 International Conference on Cloud and Green Computing*, 2013, pp. 494–501.

[18] F. Corradini, R. Culmone, L. Mostarda, L. Tesei, and F. Raimondi, "A Constrained ECA Language Supporting Formal Verification of WSNs," *Proc. - IEEE 29th Int. Conf. Adv. Inf. Netw. Appl. Work. WAINA 2015*, pp. 187–192, 2015.

[19] H. Huang, S. Huang, and T. Zhang, "A formal method for verifying production knowledge base," *Proc. - 4th Int. Conf. Internet Comput. Sci. Eng. ICICSE 2009*, pp. 19–23, 2010.

[20] A. Ericsson, M. Berndtsson, P. Pettersson, and L. Pettersson, "Verification of an industrial rule-based manufacturing system using REX," in *1st International Workshop on Complex Event Processing for Future Internet*, 2008, pp. 1–10.

[21] L. V. B. Wiecheteck, "Método para projeto de software usando o Paradigma Orientado a Notificações – PON," Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial, Universidade Tecnológica Federal do Paraná, Curitiba, 2011.

[22] A. Koestler, *O fantasma da máquina*. Rio de Janeiro: Zahar Editores, 1969.

[23] A. Dresch, D. P. Lacerda, and J. A. V. J. Antunes, *Design Science Research*, 1st ed. Porto Alegre: Bookman, 2014.

Igor Thiago Marques Mendonça was born in Cambé – PR, Brazil. He received in 2005 the Diploma in Computer Engineering from the University of North Paraná (UNOPAR), Brazil, and in 2011 the M.Sc. degree in automation and systems engineering from Graduate Program in Automation and Systems Engineering (PPGEAS) at Federal University of Santa Catarina (UFSC). He is currently Ph.D. student in computer engineering from Graduate Program in Electrical Engineering and Industrial Informatics (CPGEI), UTFPR, and he is Assistant Professor at Federal Institute of Santa Catarina (IFSC), Florianópolis, Brazil, teaching in a set of educational programs.

Jean Marcelo Simão was born in Ponta Grossa – PR, Brazil, in 1976. He received in 1998 the B.Sc. degree in Computer Science from State University of Ponta Grossa (UEPG). In 2001, he obtained the M.Sc. degree from Graduate School in Electrical Engineering and Industrial Computer Science (CPGEI) at University of Technology - Paraná (UTFPR). In 2005, he obtained the Ph.D. degree, after a double thesis, in the domains of Industrial Computer Science at CPGEI/UTFPR and Computer Engineering & Automatics at Research Center for Automatic Control of Nancy (CRAN) - Henry Poincaré University (UHP) - University of Lorraine (UL), in France. Subsequently, in 2005/06, he developed teaching and research

activities in a Master at UHP and at CRAN in a post-doctoral context.

Since august 2006, he is affiliated at UTFPR, in which nowadays he is affiliated as Professor to the Department of Informatics, Centre of Technology Innovation (CITEC), and CPGEI. Still, his teaching activities concern computer science, whereas his research ones include artificial intelligence, software/system engineering, and development/computer paradigm, namely the Notification Oriented Paradigm (NOP) of his authorship. http://lattes.cnpq.br/3593420323268103.

Paulo Cézar Stadzisz was born in Blumenau – SC, Brazil, in 1963. He received in 1987 the Diploma in Data Processing Technologies from the Federal University of Paraná (UFPR), Brazil, and in 1990 the M.Sc. degree in industrial computer science from Graduate Program in Electrical Engineering and Industrial Computer Science (CPGEI) at Federal Center for Technological Education of Parana (CEFET-PR). He received in 1997 the Ph.D. degree from the Franche-Comté University (France).

He is currently Professor at Federal University of Technology - Paraná (UTFPR) teaching in a set of educational programs. He namely teaches and coordinates researches in the CPGEI/UTFPR doctoral program associated to CNPq (National Council for Scientific and Technological Development) and CAPES (Coordination for the Improvement of Higher Education Personnel) of Brazil. He also coordinates the Laboratory of Intelligent Manufacturing Systems (LSIP) and the Laboratory of Innovation in Technology (LIT) at UTFPR. His publications and researches include systems modeling and analysis, renewable energy, and software engineering.