UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

PROGRAMA DE PÓS-GRADUAÇÃO EM

ENGENHARIA ELÉTRICA E INFORMÁTICA INDUSTRIAL

IGOR THIAGO MARQUES MENDONÇA

METODOLOGIA DE PROJETO DE SOFTWARE ORIENTADO A NOTIFICAÇÕES

TESE DE DOUTORADO

CURITIBA

2020

IGOR THIAGO MARQUES MENDONÇA

METODOLOGIA DE PROJETO DE SOFTWARE ORIENTADO A NOTIFICAÇÕES

Notification Oriented Software Design Metodology

Tese apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial, da Universidade Tecnológica Federal do Paraná (UTFPR), como requisito parcial para obtenção do título de “Doutor em Ciências” – Área de Concentração: Engenharia de Computação.

Orientador: Prof. Dr. Paulo Cézar Stadzisz.

Coorientador: Prof. Dr. Jean Marcelo Simão.

CURITIBA

2020

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Curitiba

IGOR THIAGO MARQUES MENDONCA

METODOLOGIA DE PROJETO DE SOFTWARE ORIENTADO A NOTIFICAÇÕES

Trabalho de pesquisa de doutorado apresentado como requisito para obtenção do título de Doutor Em Ciências da Universidade Tecnológica Federal do Paraná (UTFPR). Área de concentração: Engenharia De Computação.

Data de aprovação: 17 de Dezembro de 2020

Prof Paulo Cezar Stadzisz, Doutorado - Universidade Tecnológica Federal do Paraná
Prof Andrey Ricardo Pimentel, Doutorado - Universidade Federal do Paraná (Ufpr)
Prof Herve Panetto, Doutorado - Universite de Lorraine
Prof Marco Aurelio Wehrmeister, Doutorado - Universidade Tecnológica Federal do Paraná
Prof Robson Ribeiro Linhares, Doutorado - Universidade Tecnológica Federal do Paraná

Documento gerado pelo Sistema Acadêmico da UTFPR a partir dos dados da Ata de Defesa em 07/01/2021.

AGRADECIMENTOS

Agradeço à minha família: meu pai, Delfino e minha mãe Maria de Lourdes, por proporcionarem meus estudos iniciais e incentivarem que eu continuasse estudando e meus irmãos Júnior, Diego e Renan pela amizade de sempre, mesmo todos nós estando distantes fisicamente. Cris, minha esposa e amiga de todas as horas, não há palavras para descrever toda a gratidão pelo apoio e ajuda que me deu durante todas as etapas do doutoramento. Fiji e Robbie, nossos cães, agradeço o apoio moral incondicional.

Agradeço meus orientadores, professores Stadzisz e Simão. Ambos deram todo o apoio necessário para finalização deste trabalho. Os professores ofereceram aprendizados para além do que se pode ler neste documento. O convívio com eles, nos anos que estive em Curitiba, fez com que minha mente se abrisse para questões científicas e profissionais que eu não imaginava. Espero poder retribuir algum dia.

Agradeço a todos do grupo de pesquisa no Paradigma Orientado a Notificações (PON) que me apoiaram quando iniciei os estudos no paradigma. Em especial, agradeço os que puderem participar do grupo focal de avaliação desta tese.

Agradeço aos professores Panetto, Pimentel, Wehrmeister e Linhares por terem aceitado avaliar o trabalho de tese e, também, por terem contribuído com sugestões de melhorias.

Agradeço ao IFSC, UTFPR e CAPES pela oferta do DINTER que possibilitou o meu doutoramento e o de outros colegas.

RESUMO

MENDONÇA, Igor Thiago Marques. **Metodologia de Projeto de Software Orientado a Notificações.** 2020. 221 f. Tese de Doutorado - Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial (CPGEI). Universidade Tecnológica Federal do Paraná (UTFPR). Curitiba, 2020.

Muitas pesquisas vêm sendo apresentadas para mitigar problemas de produtividade, qualidade e complexidade crescente da indústria de software. Uma dessas pesquisas propõe uma nova abordagem de computação denominada Paradigma Orientado a Notificações (PON). O PON surgiu visando melhorar o desempenho e facilitar o desenvolvimento de software. No que diz respeito ao projeto de software PON, a primeira iniciativa do grupo de pesquisa foi o método chamado Desenvolvimento Orientado a Notificações (DON), baseado nas práticas atuais de ES com uso da UML e Redes de Petri. Porém, o DON aplica uma abordagem convencional de modelagem orientada a objetos, o que não favorece a modelagem de software PON. Este trabalho complementa os anteriores pois empregou esforços para estabelecer uma metodologia iterativa e incremental, denominada Metodologia de Projeto de Software Orientado a Notificações (NOM), focada no paradigma PON, ou seja, orientada a concepção de regras, fatos e notificações. A NOM define um conjunto abrangente de atividades para apoiar o projetista na modelagem de software PON a partir de requisitos de software. Adota-se um novo processo de modelagem, denominado Modelagem de Fluxo Holônico (MFH), que se fundamenta nos Sistemas Holônicos e em sua hierarquia de hólons (holarquias), na qual um hólon pode ser decomposto em um subconjunto de hólons que detalham o hólon do nível superior. As principais primitivas de modelagem referem-se a elementos decisionais, factuais, execucionais e de relação desses elementos no PON. Na MFH essas primitivas são chamadas de hólons e relacionam-se para criar o fluxo lógico do software. A versão preliminar da NOM foi avaliada positivamente em um grupo focal com especialistas no PON. A metodologia foi melhorada a partir das sugestões dos especialistas. A NOM foi aplicada em um caso de estudo para demonstrar a sua viabilidade e efetividade e, por fim, foi avaliada em relação aos requisitos e recomendações de qualidade para linguagens de modelagem. Assim, a partir destas avaliações e das propostas apresentadas nesta tese, constatou-se que os esforços empreendidos contribuíram para amadurecer aspectos de modelagem de software para o PON.

Palavras-chave: Paradigma Orientado a Notificações (PON). Metodologia de Projeto de Software Orientado a Notificações (NOM). Modelagem de software PON.

ABSTRACT

MENDONÇA, Igor Thiago Marques. **Notification Oriented Software Design Methodology.** 2020. 221 f. Tese de doutorado - Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial (CPGEI). Universidade Tecnológica Federal do Paraná (UTFPR). Curitiba, 2020.

Many studies have been proposed to mitigate problems of productivity, quality and increasing complexity in the software industry. One of these studies proposes a new computing approach called Notification Oriented Paradigm (NOP). NOP aims to improve software performance and to facilitate its development. Regarding NOP software design, the first initiative of the research group was a method called Notification Oriented Design (NOD). NOD is based on the current practices of Software Engineering making use of UML and Petri Nets. However, NOD applies a conventional approach of object-oriented modeling, which does not favor the modeling of NOP software. This research complements the previous ones because it used efforts to establish an iterative and incremental methodology, called Notification Oriented Software Design Methodology (NOM), focused on the NOP, that is, oriented to the design of rules, facts and notifications. The NOM defines a comprehensive set of activities to support designers in NOP software modeling from software requirements. A new modeling process is adopted, called Holonic Flow Modeling (HLM), which is based on Holonic Systems and its holon hierarchy concepts (holarchies), whose approach predicts that a holon can be decomposed into a subset of holons that detail the holon of the upper level. The main modeling primitives refer to decisional, factual, and executional elements and about its relations in the NOP. In HLM these primitives are called holons and they relate to each other creating the software logical flow. The preliminary NOM version was evaluated positively in a focus group with experts in NOP. The focus group allowed an increase on the methodology, based on the experts' suggestions. The NOM was applied on a case study to demonstrate its feasibility and effectiveness and, finally, it was evaluated in relation to the quality requirements and recommendations for modeling languages. Thus, from these evaluations and from the proposals presented in this research, it was evidenced the efforts undertaken contributed to mature aspects of NOP software modeling.

Keywords: Notification Oriented Paradigm (NOP). Notification Oriented Software Design Methodology (NOM). NOP Software Modeling.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS

cf.	conforme
i.e.	id est (isto é, ou seja)
e.g.	exempli gratia (por exemplo)

LISTA DE SIGLAS

ATM	*Automated Teller Machine* - Caixa eletrônico
BPMN	*Business Process Modeling Notation* - Modelo e Notação de Processos de Negócio
CPGEI	Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial
DFD	Diagramas de Fluxos de Dados
DFH	Diagrama de Fluxo Holônico
DSML	*Domain-Specific Modeling Language* - Linguagem de Modelagem de Domínio Específico
ES	Engenharia de Software
FBE	*Fact Base Element* - Elemento da Base de Fatos
FPGA	*Field Programmable Gate Array* - Arranjo de Portas Programáveis em Campo
HD	Hólon Decisional
HE	Hólon de Entidade
HF	Hólon FBE
HT	Hólon Transacional
HX	Hólon de Entidade Externa
IoT	*Internet of Things* - Internet das coisas
MSL	Mapeamento Sistemático da Literatura
OCL	*Object Constraint Language* - Linguagem para Especificação de Restrições em Objetos
OMG	*Object Management Group*
OS	Ordem de Serviço
PA	Projeto Axiomático
PD	Paradigma Declarativo
PI	Paradigma Imperativo
POO	Paradigma Orientado a Objetos
RC	Relação de Chamada
RdP	Redes de Petri
RF	Requisito Funcional
RN	Relação de Notificação
RNAs	Redes Neurais Artificiais
RNF	Requisito Não Funcional

RSL	Revisão Sistemática da Literatura
SBRs	Sistemas Baseados em Regras
SP	Sistemas de Produção
STL	*Standard Template Library* - Biblioteca de Modelos Padrão
TGS	Teoria Geral de Sistemas
UML	*Unified Modeling Language* - Linguagem de Modelagem Unificada

LISTRA DE ACRÔNIMOS

CON	Controle Orientado a Notificações
coPON	Coprocessador PON
DON	Desenvolvimento Orientado a Notificações
GoF	*Gang of Four* - Gangue dos Quatro
GoM	*Guidelines of Modeling* - Recomendações de Modelagem
HeKatE	*Hybrid Knowledge Engineering* - Engenharia de Conhecimento Híbrida
LingPON	Linguagem do PON
MCPON	Método de Compilação para o PON
NOCA	*Notification-Oriented Computer Architecture* - Arquitetura Computacional Orientada a Notificações
NOMSim	NOCA Simulator
NOM	*Notification Oriented Software Design Methodology* - Metodologia de Projeto de Software Orientado a Notificações
PON	Paradigma Orientado a Notificações
POR	Paradigma Orientado a Regras

SUMÁRIO

1 INTRODUÇÃO

Os softwares têm ocupado um papel de destaque crescente na sociedade contemporânea. Eles estão presentes no cotidiano das pessoas de diferentes maneiras, mesmo que muitas vezes não sejam percebidos. Há software em equipamentos domésticos, como fornos de micro-ondas, geladeiras, fogões e máquinas de lavar roupa, e em equipamentos eletrônicos como celulares, tablets e videogames. Os softwares também desempenham papel importante para as pessoas em condições de risco como, por exemplo, em sistemas médico-hospitalares e sistemas de segurança. No setor empresarial, os softwares de gestão oferecem suporte para muitas tarefas administrativas, que otimizam processos e maximizam o uso de recursos humanos. Outros setores, como o de telecomunicações, defesa, aeronáutica e aeroespacial, automação, governo e agronegócios, também fazem uso intenso e crescente de software e ilustram o quão pervasivos estes sistemas são na sociedade atual.

A indústria brasileira de software vem crescendo nos últimos 20 anos a taxas superiores à do PIB nacional. O faturamento desta indústria com a produção e comercialização de software foi superior a US$ 23,5 bilhões, segundo dados de 2018 da Associação Brasileira das Empresas de Software - ABES (2019) e empregou mais de 690 mil pessoas, segundo estimativas da Associação para Promoção da Excelência do Software Brasileiro - SOFTEX (2017). Os dados da ABES e da SOFTEX consideram somente empresas que possuem o desenvolvimento de software como atividade principal, não contabilizando empresas privadas ou públicas que possuam equipes de desenvolvimento internas. Somente o setor público federal emprega mais de 60 mil pessoas na área de Tecnologia da Informação, segundo o último levantamento do Tribunal de Contas da União - TCU (2015). Esses dados ajudam a ilustrar a dimensão da indústria do software e a influência que ela exerce nos segmentos da sociedade.

A demanda por software é crescente, porém nem sempre é atendida por sua indústria (BAMBINI *et al.*, 2013; SILVA; PITASSI, 2013). Alguns dos fatores determinantes para o não atendimento dessa demanda são problemas enfrentados pela indústria de software relativos à baixa produtividade, baixa qualidade do

software e complexidade crescente (STADZISZ, 2014), conforme argumentado a seguir.

- A produtividade no desenvolvimento de software é algo difícil de alcançar, pois trata-se de uma atividade inerentemente humana e que, além de envolver aspectos de qualidade da mão de obra e aspectos sociais como a satisfação no trabalho (LAL; PATHAK; KUMAR, 2015), exige esforço intelectual intenso para criar lógicas ou algoritmos que requerem inventividade e capacidade de abstração (JACKSON, 2012; SELIC, 2003). Estes aspectos tornam o desenvolvimento de software difícil e moroso levando à baixa produtividade que, por sua vez, causa o encarecimento dos produtos de software e a ampliação dos prazos de desenvolvimento.
- A baixa qualidade do software diz respeito à sua não conformidade com os requisitos do cliente e falhas nas características implícitas (e.g., problemas de desempenho do software ou de usabilidade) que são esperadas de um software desenvolvido profissionalmente (GALIN, 2004). Geralmente, o software é entregue ao cliente ainda com problemas que precisam ser corrigidos. Estima-se que 90% do custo de projeto de software advém da manutenção e correção de erros (GOUES *et al.*, 2012). Porém, aumentar a qualidade de software significa mais tempo e custos, pois se exige maior rigor em todas as etapas do desenvolvimento e isso impacta diretamente a produtividade.
- Com o passar dos anos os softwares estão ficando cada vez mais complexos. O mercado exige dos softwares maior número de funções que são mais intrincadas ou mais evoluídas, consequentemente tornando-os maiores e mais complicados, diminuindo ainda mais a produtividade no seu desenvolvimento (MOORE; WERMELINGER, 2013).

Os softwares oferecidos ao mercado são, na verdade, produtos que não trazem o máximo em qualidade. Trata-se de uma relação entre o que o mercado está disposto a aceitar e a qualidade que a indústria de software consegue produzir. Segundo Baskerville *et al.* (2001), clientes sérios vão aceitar certo grau de falta de confiança ou dificuldade operacional no software em troca de inovação, enquanto um produto mais apropriado não for desenvolvido. De forma análoga, o alto custo de software pode impelir os consumidores a adquirirem software de menor qualidade, mas com custos acessíveis. Em resumo, este desencontro entre o que o mercado

necessita e a oferta de software tem limitado o crescimento desta indústria e afetado o uso mais intensivo de software na sociedade.

Outra dificuldade enfrentada pela indústria de software é a falta de mão de obra. A SOFTEX estimou uma escassez de 408.000 postos de trabalho em TI no Brasil para o período de 2011 a 2022 e prevê que a perda de receita líquida em virtude dessa escassez será na casa de R$ 220 bilhões no mesmo período (SOFTEX, 2013). Esse fator, aliado aos problemas relatados anteriormente, fazem com que a indústria de software necessite abordagens inovadoras e eficazes para dominar a complexidade das atividades de desenvolvimento de software, reduzir os riscos e garantir a confiabilidade das soluções de software (FUGGETTA; NITTO, 2014).

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. Muitas pesquisas científicas e tecnológicas têm sido conduzidas, incluindo os estudos realizados na UTFPR desde 2001 sobre uma nova abordagem (ou paradigma) de computação que, ao longo do tempo, passou a ser denominado Paradigma Orientado a Notificações - PON (SIMÃO, 2001).

O PON surgiu como uma abordagem para desenvolvimento de software visando melhorar o desempenho das aplicações e facilitar o desenvolvimento delas, contribuindo, potencialmente, para o aumento da produtividade, melhoria da qualidade e redução da complexidade em engenharia de software. O PON apresenta um novo conceito de concepção e execução de softwares, mas que deriva em certo grau dos paradigmas convencionais imperativos e declarativos (VAN ROY, 2009). Analogamente ao paradigma declarativo, o PON faz uso de uma lógica declarativa (notadamente na forma de **regras**) na composição das soluções de software. Como no paradigma imperativo, mais especificamente do subparadigma orientado a objetos, o PON faz uso da abstração de **classes e objetos** para a composição dos elementos informacionais. Adicionalmente, o PON agrega novos instrumentos, particularmente a definição dos fluxos lógicos do software por meio de **notificações**.

O PON é recente, tendo sido proposto como paradigma no ano de 2008 (SIMÃO; STADZISZ, 2011), após sua forma embrionária do que atualmente é chamado Controle Orientado a Notificações (CON) e, outrora, de Controle Holônico

e afins (SIMÃO, 2001, 2005) (SIMAO; TACLA; STADZISZ, 2009). Sua evolução está sendo conduzida pelo grupo de pesquisa em Engenharia de Software da UTFPR liderado pelos professores Paulo Cézar Stadzisz e Jean Marcelo Simão. A maior parte dos esforços de pesquisa atuais sobre o paradigma se concentram na comprovação dos seus benefícios e na evolução de sua teoria e técnicas, além de estudos sobre seu uso prático.

Particularmente para o projeto de software PON, foi proposto um método denominado Desenvolvimento Orientado a Notificações - DON (WIECHETECK, 2011). O DON é baseado nas práticas atuais de ES com uso da UML e, também, de Redes de Petri. O DON foi a primeira iniciativa desenvolvida pelo grupo de pesquisa sobre um método de projeto de software PON. Entretanto, o DON aplica uma abordagem convencional de modelagem orientada a objetos, o que não necessariamente favorece a modelagem de software PON, pois não permite orientar adequadamente o projeto aos conceitos do paradigma. Isso se deve, principalmente, ao fato de a UML não ter sido desenvolvida para esse fim.

Dentro do contexto apresentado, o grupo de pesquisa almeja dispor de uma metodologia para desenvolvimento de software PON que esteja orientada às particularidades do paradigma. Com tal metodologia, o grupo de pesquisa pretende compor e oferecer os instrumentos necessários para o desenvolvimento de software PON. Este quadro fornece a motivação para empreender pesquisas por alternativas de métodos, técnicas e, mesmo, linguagens de modelagem próprias ao PON.

1.1 PROBLEMAS TRATADOS NA PESQUISA E HIPÓTESES

Um conceito fundamental do PON é a organização da lógica das aplicações por meio de “regras”. Assim, os processos de desenvolvimento de software PON deveriam ser adaptados a este conceito. Porém, as propostas de métodos de desenvolvimento baseados em regras, em geral, não evoluíram expressivamente, limitando-se a poucas abordagens (NALEPA; LIGĘZA, 2010). Na falta de metodologias de desenvolvimento mais específicas para sistemas baseados em regras, os desenvolvedores desses sistemas empregam processos tradicionais de desenvolvimento de software (ZACHARIAS, 2008, 2009).

O problema considerado nesta pesquisa é a dificuldade em desenvolver softwares no Paradigma Orientado a Notificações (PON), mais precisamente no que se refere à atividade de concepção de software para o paradigma. A mitigação deste problema iniciou-se com a proposta do método DON (WIECHETECK, 2011). Entretanto, o desenvolvimento do DON seguiu o caminho tradicional em que se especializou a UML para a modelagem em projetos de software PON apoiado na abordagem orientada a objetos. Desta forma, criou-se um método que permite a modelagem de projetos de software PON, porém com uma capacidade limitada de apoio ao projetista no que se refere à concepção da solução, isto é, em transformar as necessidades por software em elementos funcionais e não funcionais do software. Some-se a isto o fato que o DON não é alinhado especificamente com as particularidades do PON. Assim, a busca por métodos mais aderentes aos conceitos do PON motiva novas pesquisas por alternativas.

Na medida em que a teoria e a técnica do PON evoluem, as aplicações desenvolvidas usando o paradigma se tornam cada vez mais extensas e complexas. O desenvolvimento de software de qualidade e que atenda às necessidades dos clientes requer processos de software adequados. Este trabalho de pesquisa se dedica a avançar neste sentido e visa propor uma metodologia de projeto de software adequada ao Paradigma Orientado a Notificações (PON).

A hipótese considerada, a partir do problema exposto, é que uma nova metodologia de projeto de software explicitamente orientada aos conceitos do PON possa minimizar as dificuldades na concepção e desenvolvimento de aplicações PON.

1.2 OBJETIVOS DO TRABALHO DE PESQUISA

O objetivo geral deste trabalho de pesquisa é conceber uma nova metodologia de projeto de software para o Paradigma Orientado a Notificações visando facilitar a concepção de aplicações neste paradigma.

A metodologia desenvolvida foi denominada Metodologia de Projeto de Software Orientado a Notificações (NOM - *Notification Oriented software design Methodology)* e buscou atender às seguintes diretrizes:

- Ser orientada aos conceitos e primitivas do PON.

- Cobrir as fases de análise e projeto de software.
- Produzir artefatos suficientes para que o software possa ser implementado no PON.
- Considerar os métodos e técnicas de engenharia de software desenvolvidos pelo grupo de pesquisas sobre o PON.

A partir do objetivo geral, relacionam-se, a seguir, os objetivos específicos da pesquisa.

1. **Determinar métodos de Engenharia de Software compatíveis com o PON**

   Identificar, a partir de uma revisão bibliográfica, métodos ou técnicas da Engenharia de Software que possam ser usados, ou adaptados para uso, na metodologia de projeto de software para o PON sendo proposta neste trabalho.

2. **Estabelecer os dados de entrada e saída da metodologia NOM**

   A metodologia NOM se concentra na análise e projeto de software para o PON, assim, os dados de entrada e saída da metodologia deverão ser estabelecidos.

3. **Estipular o ciclo de vida, os modelos e as fases da NOM**

   Este objetivo específico visa formular a metodologia NOM propriamente dita, descrevendo suas atividades e fornecendo um guia para os projetistas de software PON.

4. **Demonstrar o uso da metodologia NOM**

   Aplicar a NOM no desenvolvimento de software para o PON, projetando e implementando aplicações.

5. **Avaliar a metodologia NOM**

   - Avaliar, a partir dos dados de entrada e de saída da metodologia, se a NOM cobre o escopo definido para ela, produzindo artefatos necessários e suficientes para implementação do software projetado.
   - Avaliar a aderência da NOM em relação às orientações para criação de processos de software.
   - Avaliar com o grupo de pesquisas em PON em um grupo focal se a NOM condiz com os preceitos de PON, bem como sua viabilidade de uso.

1.3 MOTIVAÇÃO E ORIGINALIDADE DA PESQUISA

O Paradigma Orientado a Notificações (PON) mostra-se promissor para o desenvolvimento de sistemas computacionais, como apontam pesquisas realizadas e em andamento (BANASZEWSKI, 2009; BATISTA, 2013; FERREIRA, 2015; KERSCHBAUMER, 2018; LINHARES, 2015; NEGRINI, 2019; NOVAES, 2019; OLIVEIRA, 2019; RONSZCKA, 2012, 2019; SCHÜTZ, 2019; SIMÃO *et al.*, 2012; VALENÇA *et al.*, 2011; WITT *et al.*, 2011; XAVIER, 2014). Devido ao estágio ainda exploratório de desenvolvimento do paradigma, as aplicações criadas para validar os conceitos do PON foram relativamente simples, sendo a maior delas, com 74 regras, desenvolvida por Leonardo Araújo Santos em sua dissertação de mestrado (SANTOS, 2017). Para se ter ideia, em aplicações com outras abordagens orientadas a regras, a base de regras dos sistemas pode ter milhares ou, mesmo, milhões de regras (NALEPA *et al.*, 2011). As aplicações desenvolvidas para o PON até o momento não ultrapassaram uma ou algumas dezenas de regras e, possivelmente, por esse motivo, a necessidade de uma metodologia de projeto de software não tenha sido extensivamente explorada.

Na evolução do PON, usaram-se técnicas e ferramentas convencionais, geralmente oriundas do Paradigma Orientado a Objetos, na criação das aplicações. Dentre as técnicas e ferramentas usadas estão diagramas de classe, de sequência, de atividades e outros da UML, além de Redes de Petri. Wiecheteck (2011) organizou essas técnicas e ferramentas, e propôs outras, em uma primeira iniciativa de método de Engenharia de Software para desenvolvimento de aplicações para o PON, chamado de Desenvolvimento Orientado a Notificações (DON). O método obteve certo êxito, porém, como será relatado neste documento, o uso quase exclusivo da UML não beneficiou a concepção e *design* dos elementos fundamentais do PON. O DON é carente, também, de técnicas de verificação e validação do software modelado. O método propõe o uso de Redes de Petri, porém o assunto foi pouco explorado. Na prática, poucos projetos que seguiram o DON criaram o modelo de Redes de Petri proposto.

Tendo em vista a necessidade de continuidade na evolução de métodos de Engenharia de Software para o desenvolvimento de aplicações no PON, a principal motivação deste trabalho é justamente contribuir no desenvolvimento de técnicas, ferramentas e processos adaptados ao Paradigma Orientado a Notificações. Ou

seja, deve ser possível que uma aplicação seja concebida, desde o início, levando em consideração os conceitos fundamentais que regem o PON. Além disso, deve ser possível fazer a verificação e validação do software, especialmente em relação aos seus requisitos.

Esforços no campo da Engenharia de Software para o PON são importantes para consolidá-lo como paradigma de desenvolvimento de software. Esses esforços são particularmente importantes se facilitarem a concepção de software no paradigma, bem como permitirem que o software seja verificado e validado. Adicionalmente, mesmo que não sendo objetivo desta pesquisa, a metodologia resultante poderia ser explorada para aplicação em outras abordagens que fazem uso de regras, como os Sistemas Baseados em Regras. Ou seja, seria pertinente o seu uso no agora chamado Paradigma Orientado a Regras (POR) em geral (SIMÃO; STADZISZ; WIECHETECK, 2015).

A originalidade desta pesquisa está em focar nos aspectos de concepção de software para o PON, ou seja, na proposição de uma nova metodologia que seja orientada às principais primitivas do paradigma para concepção de software. Essas primitivas referem-se a elementos decisionais, factuais, execucionais e de relação entre eles (chamada notificação e dá nome ao paradigma). Nesta pesquisa são definidas primitivas de modelagem dentro de uma técnica própria para o PON. Adicionalmente, desenvolve-se uma proposta de linguagem que se apoia na visão de sistemas holônicos (KOESTLER, 1969). Assim, os elementos dessa linguagem poderão propiciar o desenvolvimento de projetos com características fundamentais dessa visão que são características de autonomia, facilidade de distribuição, simplicidade, interatividade e organização hierárquica (WALLACE, 2008), sem perder de vista as primitivas do PON. Portanto, sistemas desenvolvidos nessa abordagem tendem a ter certo grau de coesão e estabilidade e herdar características de escalabilidade, robustez e simplicidade de controle dos sistemas holônicos (WALLACE, 2008).

1.4 CARACTERIZAÇÃO DA PESQUISA

Esta pesquisa de doutorado pode ser caracterizada em quatro critérios: natureza, abordagem do problema, objetivos almejados e procedimentos técnicos

adotados (SILVA; MENEZES, 2005). As escolhas para essa caracterização visam evidenciar as estratégias metodológicas usadas no decorrer da pesquisa de doutorado.

Em relação ao critério **natureza**, ou seja, a finalidade deste projeto, a pesquisa classifica-se como “aplicada”. Isto porque ela abrange estudos elaborados com a finalidade de resolver problemas identificados no âmbito do Paradigma Orientado a Notificações. Ainda sob esse critério e usando a classificação da *Adelaide University*, a pesquisa pode ser caracterizada como “desenvolvimento experimental” (GIL, 2010), pois visa produzir novos materiais, ou seja, um método de desenvolvimento de software PON visando a melhoria dos processos de desenvolvimento de sistemas neste paradigma e utilizando conhecimentos derivados da pesquisa e experiências práticas.

Quanto à **abordagem do problema**, esta pesquisa se caracteriza como “hipotético-dedutiva” (MARCONI; LAKATOS, 2003, p. 106) por usar a abordagem de pesquisa no qual uma lacuna do conhecimento acerca do PON foi identificada (i.e., falta de métodos de concepção e desenvolvimento apropriados) e foi formulada a hipótese (i.e., um novo método) para então ser avaliada e testada.

Quanto aos **objetivos almejados**, esta pesquisa de doutorado pode ser classificada como “exploratória”. Ela é exploratória pois busca maior familiaridade com o problema de desenvolvimento de sistemas segundo esse novo paradigma a fim de torná-lo explícito e possibilitar a formulação de hipóteses. Além disso, o próprio PON, sob o qual este trabalho de pesquisa se fundamenta, está ainda sendo investigado e desenvolvido, muito embora com indicativos e resultados animadores (GIL, 2010).

Quanto aos **procedimentos técnicos adotados**, esta pesquisa engloba “pesquisas bibliográficas”, “pesquisa experimental” e “grupo focal”. As pesquisas bibliográficas ocorrem durante grande parte deste trabalho de pesquisa e buscam fundamentá-lo, bem como identificar trabalhos correlatos a ele (SILVA; MENEZES, 2005). Este trabalho de pesquisa se classifica como “experimental”, pois propõe-se uma nova metodologia de projeto de software PON e realiza-se experimentos para determinar sua viabilidade prática (MARCONI; LAKATOS, 2003). Por fim, esta pesquisa adota o grupo focal com especialistas em PON como técnica de evolução e avaliação da metodologia de concepção e desenvolvimento de software para o paradigma (DRESCH; LACERDA; ANTUNES, 2014).

1.5 MÉTODO DA PESQUISA

A sequência de atividades do método de pesquisa planejado para este trabalho é ilustrada na Figura 1 na forma de um diagrama usando a notação *Business Process Modeling Notation* (BPMN). As atividades estão organizadas em três fases: “Estudo Exploratório”, “Desenvolvimento” e “Refinamento e Avaliação”.

A fase de Estudo Exploratório abrange as atividades que auxiliaram na definição dos objetivos da pesquisa e dos procedimentos técnicos planejados para o seu desenvolvimento. Esta fase envolveu as seguintes atividades:

- **Definir tema de pesquisa**

  O tema do trabalho de pesquisa definido e apresentado no pré-projeto de tese foi “Investigação sobre métodos para desenvolvimento de software usando o Paradigma Orientado a Notificações”.

- **Realizar revisão inicial da literatura**

  O tema da pesquisa serviu de base para o início do trabalho de revisão inicial da literatura no qual se buscou um entendimento inicial dos conceitos e uma visão geral da área de conhecimento. Esta atividade subsidiou as decisões sobre os temas que são apresentados na fundamentação teórica deste trabalho.

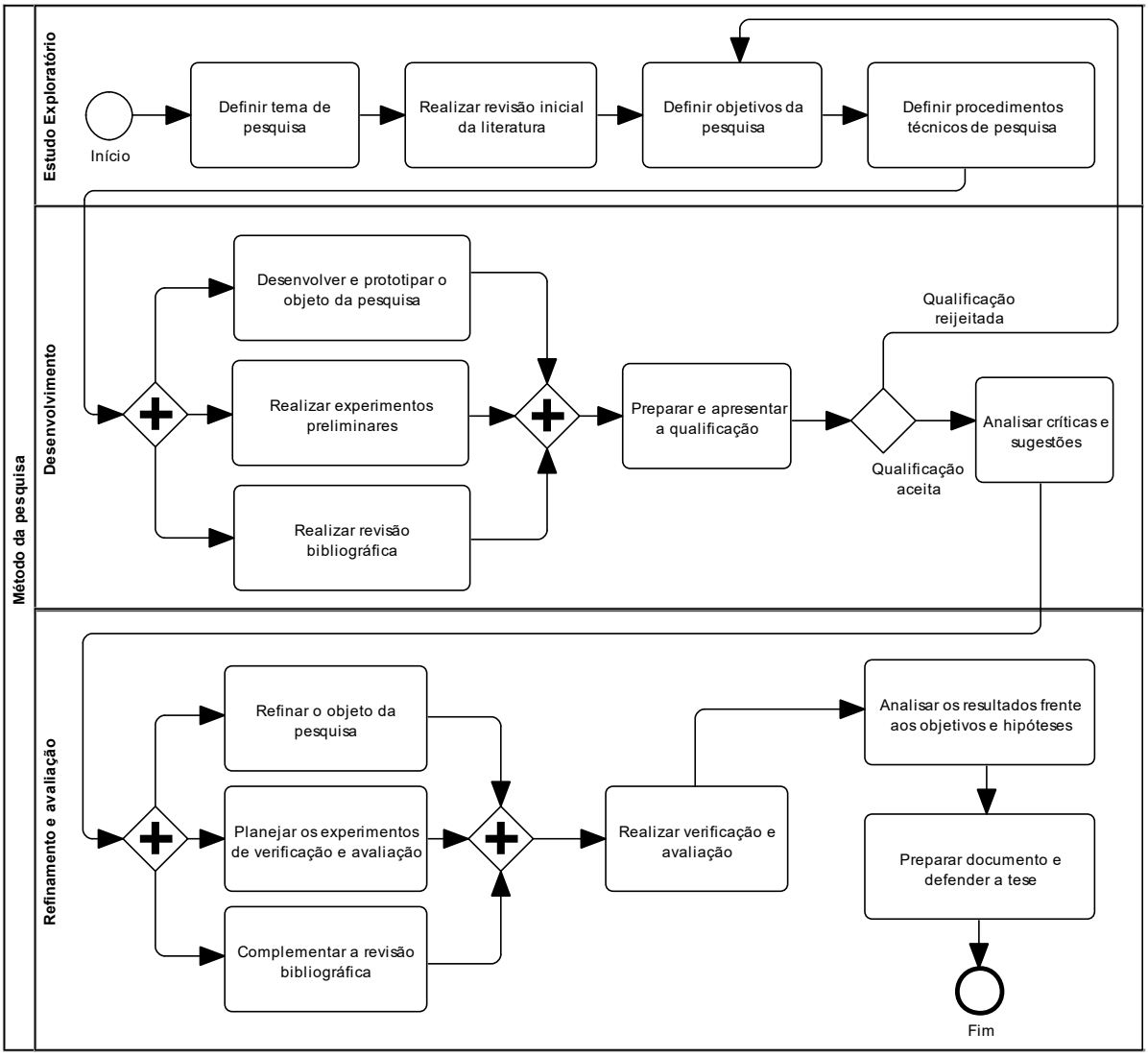
- **Definir o objetivo da pesquisa**

  Na sequência, ocorreu uma série de reuniões de orientação com o objetivo de realizar um estudo especial em Engenharia de Software. Esse estudo, além de alinhar os conceitos acerca de Engenharia de Software entre o doutorando e os orientadores, auxiliou na definição do objetivo do trabalho de pesquisa.

- **Definir os procedimentos técnicos da pesquisa**

  A partir da definição do objetivo da pesquisa, formularam-se os procedimentos técnicos adotados para a condução dos trabalhos. Os procedimentos foram apresentados na seção anterior, caracterização da pesquisa, e serão melhor detalhados nos capítulos em que foram usados.

Figura 1 - Etapas do método de pesquisa adotado



Fonte: Autoria própria

A fase de Desenvolvimento envolveu cinco atividades, das quais as três primeiras foram executadas em paralelo, conforme descrito a seguir.

- **Desenvolver e prototipar o objeto da pesquisa**

  As atividades de desenvolvimento e prototipação do objeto da pesquisa concentraram os primeiros esforços na definição da metodologia de projeto de software para o PON. Essa metodologia de projeto apoia o projetista no processo de desenvolvimento de software para o PON.

- **Realizar experimentos preliminares**

  Os experimentos preliminares, realizados em simultaneamente com a prototipação do objeto de pesquisa, permitiram o melhoramento gradativo da metodologia de projeto de software PON até a proposta final.

- **Realizar revisão bibliográfica**
  A atividade de revisão bibliográfica envolveu o Paradigma Orientado a Notificações (PON), com especial atenção ao método de Desenvolvimento Orientado a Notificações (DON). Na revisão bibliográfica buscaram-se, também, trabalhos relacionados aos processos de software mais comuns e alternativos. A busca por processos de software alternativos objetivou encontrar soluções, preferencialmente, não baseadas na UML e que, potencialmente, poderiam modelar Sistemas Orientados a Regras.
- **Preparar e apresentar a qualificação**
  A fundamentação e situação intermediária do trabalho de pesquisa foram compiladas e apresentadas como pré-requisito para a qualificação de doutorado.
- **Analisar críticas e sugestões**
  Na sequência de atividades previstas para o trabalho de pesquisa, após a apresentação da qualificação, foram analisadas as críticas e sugestões de melhorias para a metodologia de projeto e iniciou-se o refinamento da mesma.

As atividades da fase Refinamento e Avaliação concluem as proposições da tese, as avaliam e determinam se cumprem aos objetivos e hipóteses propostas. Esta fase envolveu as seguintes atividades.

- **Refinar o objeto da pesquisa**
  Nesta etapa, a metodologia de projeto foi completada, isto é, os artefatos e ciclos da metodologia foram definitivamente propostos. Esses refinamentos foram feitos observando a avaliação da banca de qualificação.
- **Planejar os experimentos de verificação e avaliação**
  Nessa atividade, criou-se os instrumentos para avaliar a metodologia de desenvolvimento com especialistas, neste caso, os pesquisadores em PON. Em linhas gerais, estabeleceu-se um experimento permitindo uma análise aprofundada da aplicação da metodologia proposta. Assim, optou-se pela aplicação de um grupo focal com os pesquisadores em

PON. Adicionalmente, a linguagem de modelagem criada no âmbito da metodologia proposta foi avaliada segundo critérios de qualidade para linguagens de modelagem (FRANK, 2013).

- **Complementar a revisão bibliográfica**
  A busca por trabalhos relacionados continuou durante todo o desenvolvimento do trabalho de pesquisa. Além disso, nessa fase em que o objeto da pesquisa foi refinado e completado, a revisão bibliográfica auxiliou a fundamentação das escolhas deste trabalho.
- **Realizar verificação e avaliação**
  Nesta atividade, os experimentos planejados foram realizados e documentados. A documentação desta atividade foi a base para a comprovação das hipóteses e conclusão dos objetivos planejados para este trabalho de doutorado.
- **Analisar os resultados frente aos objetivos e hipóteses**
  Nesta atividade foram analisados os resultados alcançados com os experimentos e avaliações sobre a metodologia de projeto de software para PON. A partir de casos de estudo, do grupo focal com especialistas PON e da avaliação da metodologia segundo critérios de qualidade para linguagens de modelagem, determinou-se as contribuições deste trabalho. Além disso, a avaliação com especialistas contribuiu com aspectos não percebidos durante o desenvolvimento do trabalho de pesquisa e que puderam ser incorporados à metodologia de projeto.
- **Preparar documento e defender a tese**
  A análise dos resultados frente aos objetivos e hipóteses finalizou o trabalho de pesquisa. O documento de tese foi preparado para avaliação e apresentação para a banca final de doutorado.

1.6 ORGANIZAÇÃO DO DOCUMENTO

Este documento está organizado em 5 capítulos. O Capítulo 2 descreve a fundamentação teórica do trabalho de pesquisa para elaboração deste documento, abrangendo a teoria e evolução do PON e métodos de modelagem. Em especial, o capítulo apresenta um Mapeamento Sistemático da Literatura (MSL) realizado no

decorrer da pesquisa, pelo qual buscou-se técnicas, métodos, ferramentas, linguagens, representações, especificações e instrumentos usados na modelagem de regras dos Sistemas Baseados em Regras (SBR) para, além de fazer um levantamento do estado da arte, identificar trabalhos correlatos.

O capítulo 3 apresenta os resultados da tese, ou seja, a Metodologia de Desenvolvimento de Software Orientado a Notificações (NOM). O capítulo apresenta o modelo descritivo da metodologia e, para ilustrar, um exemplo de uso.

O capítulo 4 apresenta a avaliação da NOM. Inicialmente, conclui-se a modelagem do caso de estudo iniciado do capítulo anterior e faz-se discussões sobre o uso da NOM baseando-se nessa experiência. Na sequência são apresentados os resultados da avaliação da metodologia com o grupo focal realizado com os pesquisadores no PON e, por fim, apresenta-se um panorama da NOM no que se refere a requisitos de qualidade para linguagens de modelagem de software.

O Capítulo 5 apresenta as conclusões da pesquisa e as propostas de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA DA PESQUISA

Este capítulo apresenta o embasamento teórico do trabalho de pesquisa desenvolvido para esta tese de doutorado. Os principais referenciais teóricos foram agrupados em quatro grandes áreas: (i) Paradigma Orientado a Notificações na seção 2.1; (ii) Abordagens para desenvolvimento de software orientado a regras na seção 2.2; (iii) Recomendações para criação de linguagens de modelagem na seção 2.3 e (iv) Sistemas Holônicos na seção 2.4. A seção 2.5 encerra a fundamentação teórica trazendo algumas discussões sobre o capítulo.

2.1 PARADIGMA ORIENTADO A NOTIFICAÇÕES

O Paradigma Orientado a Notificações (PON) define novos conceitos para a concepção, programação e execução de softwares em sistemas computacionais. Esses conceitos visam melhorar o desempenho dos softwares e facilitar a sua concepção e programação. Todavia, entender o PON pode não ser simples, sobretudo por ele diferir de paradigmas de programação mais comuns, como o Paradigma Orientado a Objetos englobado pelo Paradigma Imperativo. Esta seção apresenta os conceitos gerais do PON e a evolução do estado da técnica até o momento.

A subseção “2.1.1 Contextualização do Paradigma Orientado a Notificações” relaciona o PON aos outros paradigmas existentes e justifica a sua criação. A subseção “2.1.2 Da computação tradicional ao Paradigma Orientado a Notificações” introduz os conceitos do PON e apresenta os elementos estruturantes do PON. Em seguida, são realizados detalhamentos da visão geral até que se identifiquem alguns dos elementos da arquitetura do PON. O termo “notificação”, que aparece no nome do paradigma, é inicialmente abordado nesta seção. A subseção “2.1.3 Fundamentos sobre o Paradigma Orientado a Notificações”, por sua vez, apresenta os elementos da arquitetura do PON. Nessa seção, o conceito de notificações é detalhado. Esta seção resume as qualidades do PON apresentadas em trabalhos anteriores que evidenciam o seu potencial enquanto paradigma de programação. A subseção “2.1.4 Estado de desenvolvimento do PON” apresenta os esforços de pesquisa na evolução do paradigma e, por fim, a seção “2.1.5 Desenvolvimento

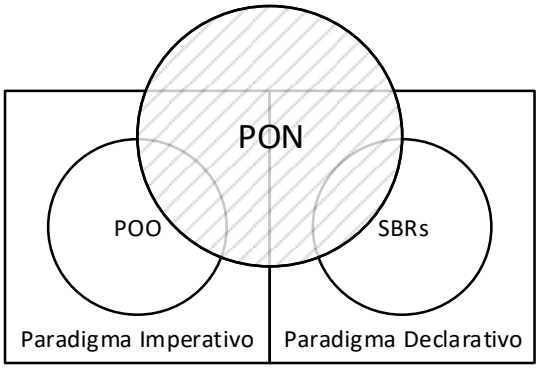
Orientado a Notificações (DON)” apresenta com maiores detalhes o método já existente para modelagem de softwares PON.

2.1.1 Contextualização do Paradigma Orientado a Notificações

O Paradigma Orientado a Notificações (PON) foi idealizado e desenvolvido a partir de 2001 por um grupo de pesquisa da UTFPR liderado pelos professores Jean Marcelo Simão e Paulo Cézar Stadzisz. Os conceitos originais surgiram de pesquisas sobre soluções de Controle Orientado a Notificações (CON), para sistemas automatizados de manufatura (SIMÃO, 2001) (SIMÃO, 2005). A partir de 2007, o PON foi formulado (BANASZEWSKI *et al.*, 2007) (SIMÃO; STADZISZ, 2011) e vem sendo desenvolvido e explorado em vários outros trabalhos de pesquisa do grupo.

O PON diferencia-se dos paradigmas atuais de programação são classificados em imperativos (PI) e declarativos (PD) (KAISLER, 2005). Fazem parte do PI as abordagens procedimentais e orientada a objetos e do PD as abordagens lógicas e funcionais. Isto em uma visão simplista, sendo que mais precisamente, naturalmente, por haver interseções entre as abordagens, salientando o caso da programação funcional (BANASZEWSKI, 2009) (VAN ROY, 2009). Em uma visão crítica, o PI indubitavelmente apresenta problemas de redundância de código e acoplamento devido a forma como os códigos são escritos em suas abordagens. Nelas usam-se estruturas de condicionais, ou avaliações causais, que são muitas vezes verificadas e não executadas. O PD, por sua vez, apresenta problemas com acoplamento e sobrecarga de processamento nos mecanismos de inferência, pois mesmo algoritmos otimizados dos mecanismos de inferência usam estruturas de dados pouco eficientes (SIMÃO *et al.*, 2012) (SIMÃO *et al.*, 2014).

Figura 2 - Relação entre o PON e paradigmas atuais



Fonte: Adaptado de Banaszewski (2009)

O PON reaproveita, em termos, elementos do Paradigma Orientado a Objetos (POO) e dos Sistemas Baseados em Regras (SBRs), conforme ilustra a Figura 2, mas na verdade os evolui dando conotação outra, como será descrito nas próximas seções. O PON foi discutido enquanto paradigma nos trabalhos de Banaszewski (2009) e Ronszcka (2012) e classificado como um paradigma de programação, no trabalho de Xavier (2014), segundo a taxonomia de paradigmas de programação de Van Roy (2009).

Mais precisamente, do Paradigma Imperativo, o PON faz uso de conceitos de abstração e agregação na forma de classes/objetos e, em algum medida, da reatividade da programação dirigida a eventos (BANASZEWSKI, 2009). Do Paradigma Declarativo, o PON faz uso de conceitos de representação de conhecimento na forma de regras e facilidades da programação declarativa em alto nível, que é tida como mais próxima da compreensão humana (BANASZEWSKI, 2009). Contudo, o PON apresenta uma nova perspectiva na concepção de software e suas contribuições visam minimizar problemas dos paradigmas atuais e estabelecer-se como um novo paradigma (BANASZEWSKI, 2009).

2.1.2 Da computação tradicional ao Paradigma Orientado a Notificações

O surgimento do primeiro computador digital, na década de 40, iniciou a era atual dos computadores. O ENIAC[1] podia realizar 5.000 adições por segundo (STALLINGS, 2010) e isso representava um grande avanço se comparado ao que

[1] *Electronic Numerical Integrator and Computer* (ENIAC) foi o primeiro computador digital eletrônico de grande escala.

os computadores analógicos[2] podiam fazer. Porém, apesar de ser um computador de uso geral, a tarefa de criar e alterar programas no ENIAC era custosa. Isso só mudou com a possibilidade de os programas poderem ser representados e armazenados na memória, junto aos dados. Essa ideia, conhecida por “programa armazenado” é atribuída ao matemático John von Neumann (STALLINGS, 2010).

O processamento de dados é uma atividade que já ocorria, mas que, com a invenção dos computadores, passou a poder ser feita em uma velocidade muito maior. O conceito de processamento de dados é ilustrado na Figura 3, na qual existe uma Entrada, um Processamento e uma Saída. Comumente, esse processamento é chamado de “processamento de dados em lote” (em inglês *batch processing*) (TANENBAUM, 2009). A Entrada fornece os dados para o processamento e a Saída é o resultado do processamento. Esse conceito é a base para os sistemas computacionais.

Figura 3 - Processamento de dados.

Fonte: Autoria própria

Os dados de entrada representam informações oriundas de entidades interessadas nesse processamento. Por exemplo, no processamento de faturas de energia elétrica, cada cliente pode ser considerado uma entidade. Caso se pretenda calcular o consumo mensal de cada entidade, os dados de entrada deveriam conter a identificação única de cada cliente e os valores extraídos de seu medidor de consumo.

Outra forma de interpretar os dados de entrada é considerar que eles representam fatos a respeito das entidades (clientes). Fatos são feitos, circunstâncias, acontecimentos, eventos ou, ainda, ocorrências (WEISZFLOG, 1998) (7GRAUS, 2015) caracterizados por dados. Se o cliente consumiu 100kWh em um mês, os dados são a identificação do cliente, a data e o valor 100kWh. Estes dados, também, representam um fato (ou seja, um acontecimento) que é o consumo

[2] É uma forma de computador que usa fenômenos elétricos, mecânicos ou hidráulicos para modelar o problema a ser resolvido.

medido daquele cliente em um determinado mês. No mês anterior, o mesmo cliente gastou 95kWh e, assim, outro fato é o consumo de 95kWh deste cliente no mês anterior.

Os dados de entrada (ou fatos) precisam estar disponíveis para que ocorra o processamento. Deve-se perceber que há uma relação de dependência entre eles, ou seja, os dados de entrada são requisitos para o processamento, conforme ilustra a Figura 4. O processamento poderá ocorrer apenas se houver dados de entrada para processar.

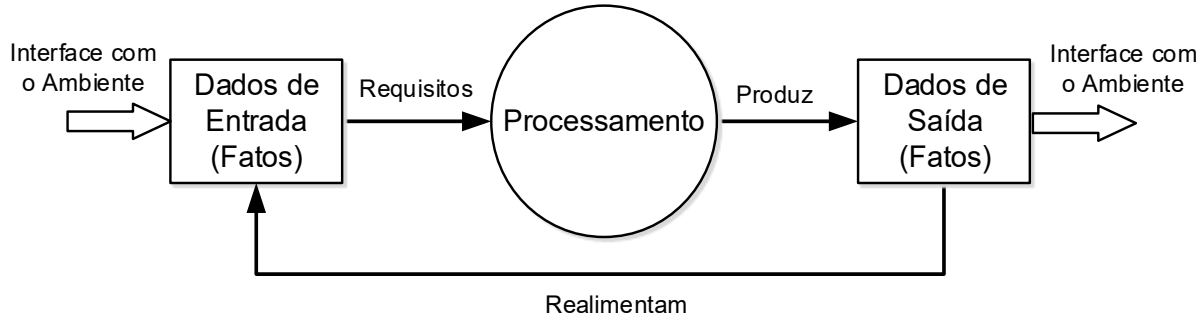
Figura 4 - Visão funcional do processamento de dados

Fonte: Autoria própria

De forma similar, todo processamento irá produzir um ou mais dados de saída, conforme ilustrado na Figura 4. Os dados de saída são uma decorrência do processamento a partir dos dados de entrada. Usando o mesmo exemplo de cálculo de consumo de energia elétrica, os dados de saída seriam as informações da fatura de energia incluindo o valor a ser pago, a identificação do documento fiscal e a data de vencimento. Pode-se entender que os dados de saída representam, também, fatos novos (ou seja, feitos ou ocorrências) produzidos pelo processamento.

Os fatos ou dados de saída produzidos podem, ainda, ser usados para realimentar o sistema, conforme ilustra a Figura 5. Esse modelo é análogo ao da Teoria Geral de Sistemas (TGS), introduzida por von Bertalanffy (1969) que inclui a realimentação e o ambiente. O ambiente é descrito na TGS como algo externo ao sistema, mas que influencia no seu funcionamento e é influenciado por ele. O ambiente inclui as pessoas, empresas, materiais, condições sociais e econômicas, entre outros elementos e entidades que podem se relacionar com o sistema. Na Figura 5 observam-se as interfaces adicionais entre o ambiente e os dados de entrada e de saída que representam as associações das quais provêm os dados ou que os consomem.

Figura 5 - Modelo da TGS aplicado ao processamento de dados



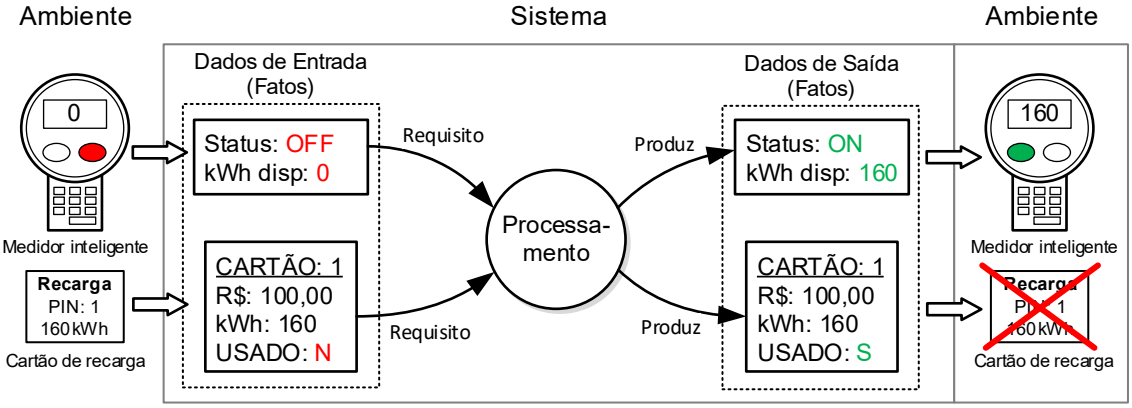
Fonte: Autoria própria

A realimentação introduz uma nova dinâmica ao processamento de dados. Além de o processamento ocorrer em razão dos dados de entrada, ele também se produz como decorrência das mudanças dos dados originárias da realimentação.

A dinâmica de realimentação traz consigo o conceito de “estado do sistema”, de forma que o processamento deve lidar com a atual situação do sistema e não mais com uma massa de dados predefinida. O “estado do sistema” é caracterizado por um conjunto de fatos presentes em determinado instante. A partir de um estado, o sistema pode evoluir para outros estados por meio do processamento ou de eventos externos a ele que modifiquem os fatos.

Como exemplo, a Figura 6 ilustra o processamento de uma recarga em um “medidor inteligente” de consumo elétrico. Os dados/fatos de entrada representam um estado inicial do sistema no qual o status do medidor é “desligado” (*OFF*) e a quantidade de kWh disponível é “0”. A partir deste estado, ocorre um evento externo em que o cliente insere um cartão de recarga que possui certo identificador (PIN), um valor de R$ 100,00, uma carga de 160kWh e um estado de uso. Estes dados recebidos constituem novos fatos para o sistema e, consequentemente, conduzem a um novo estado. Com o atendimento dos seus requisitos, o processamento modifica o status do medidor para “ligado” (*ON*) e a quantidade de kWh disponíveis para “160”, além do cartão de recarga ser marcado como usado. Esses novos fatos passam a representar o novo estado do sistema.

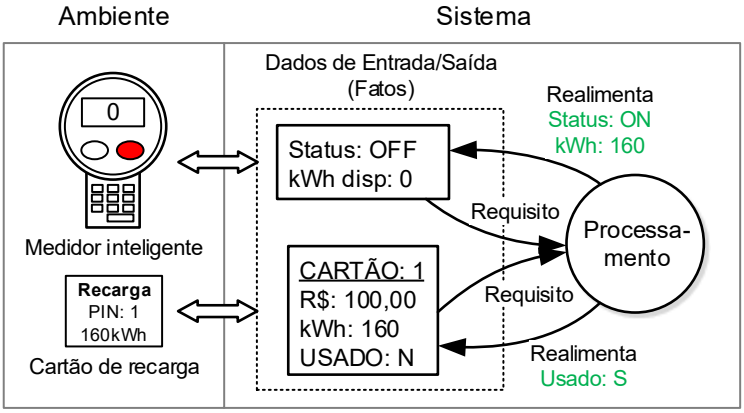
Figura 6 - Processamento de dados com interfaces com o Ambiente



Fonte: Autoria própria

Percebe-se, no exemplo apresentado pela Figura 6, que o novo estado do sistema se deu pela modificação de parte dos dados/fatos de entrada. Assim, essa figura pode ser reformulada unindo as representações dos dados de entrada, de saída e do ambiente, conforme Figura 7. Também, os resultados do processamento estão ilustrados na forma de realimentações sobre os dados/fatos do sistema.

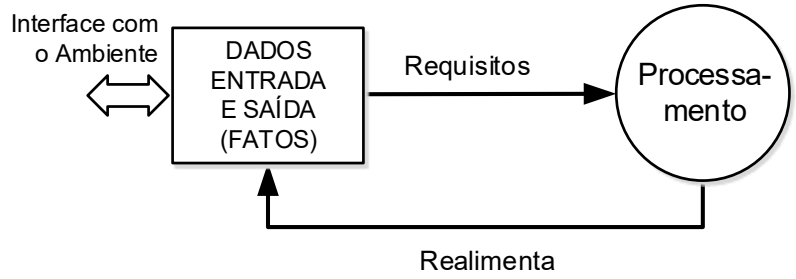
Figura 7 - Processamento de dados com representação de realimentação reformulada



Fonte: Autoria própria

A partir da visão apresentada na Figura 7, pode-se propor um modelo mais abstrato de processamento de dados com realimentação, conforme ilustrado na Figura 8.

Figura 8 - Modelo de processamento de dados com realimentação



Fonte: Autoria própria

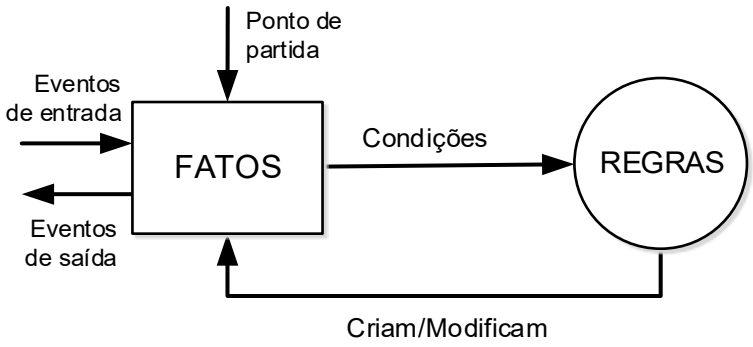
O modelo da Figura 8 pode ser usado como referência para a construção de sistemas de acordo com diferentes abordagens e tecnologias de programação, desde programação procedimental com laços de verificação dos fatos e execução de ações correspondentes, até modelos orientados a eventos mapeando mudanças nos fatos para determinar ações a serem executadas. Nas abordagens do paradigma declarativo, este modelo pode ser interpretado como os dados de entrada e saída sendo os fatos declarados e o processamento sendo as regras que provocam as ações ou geram novos fatos.

Assim, um processamento ou regra é executado quando os dados ou fatos associados (i.e., seus requisitos ou condições) estão presentes, produzindo mudanças nos dados ou fatos do sistema. No exemplo do “medidor inteligente” de consumo elétrico, o processamento de mudança do crédito de energia ocorre se o cartão de recarga estiver válido. Isto é equivalente a dizer que a regra de mudança de crédito de energia se aplica quando o cartão de recarga estiver válido.

Diante do exposto, é possível definir um modelo de processamento de dados compreendendo os conceitos de fatos e regras. Esse modelo é ilustrado na Figura 9 e denota que:

1. Os FATOS são condições para que as REGRAS sejam executadas.
2. As REGRAS podem criar ou modificar fatos.
3. Eventos de entrada (i.e., externos) podem criar ou modificar FATOS.
4. FATOS podem gerar eventos de saída (i.e., de interesse externo).
5. Há um ponto de partida que estabelece os fatos iniciais.

Figura 9 - Modelo de processamento de dados de um sistema computacional baseado em fatos e regras



Fonte: Autoria própria

Os fatos podem representar, ao mesmo tempo, dados de entrada e dados de saída, e são condições para que ocorram os processamentos. As regras, por sua vez, podem ser entendidas como processamentos que ocorrem sob certas condições e podem levar à criação de novos fatos ou à modificação de fatos existentes. Esses dois elementos, mais o conceito de notificações, representam os fundamentos do Paradigma Orientado a Notificações.

Programas construídos no Paradigma Imperativo empregam instruções que direcionam a sua execução pelo sistema operacional. Essas instruções, conhecidas por estruturas de controle, são divididas em: sequência, seleção e repetição. Elas determinam o fluxo de execução do software, uma instrução após a outra. O Paradigma Orientado a Notificações não requer a definição desses elementos de controle nesses termos. Em PON, a programação envolve a definição de Regras e Fatos, assim como no Paradigma Declarativo. O fluxo de execução se dá implicitamente pela ocorrência de encadeamentos e concorrências de regras, de acordo com as condições (i.e., estados) do sistema.

Usando-se o cenário de “medidor inteligente” de consumo elétrico, caso se deseje emitir um sinal sonoro e luminoso quando o indicador de quantidade de energia elétrica restante for menor do que 5kWh, cria-se uma regra para fazer isso. A regra poderia ser: “SE” indicador de kWh restante for menor do que 5, “ENTÃO” emitir avisos sonoro e luminoso. Nessa regra, o fato (i.e., kWh<5) é condição para que ela seja executada e a sua execução cria um novo fato (i.e., avisos sonoro e luminoso ativados).

Os fatos, como ilustrado anteriormente, são valores relacionados às entidades do sistema e podem sofrer alterações como, por exemplo: em

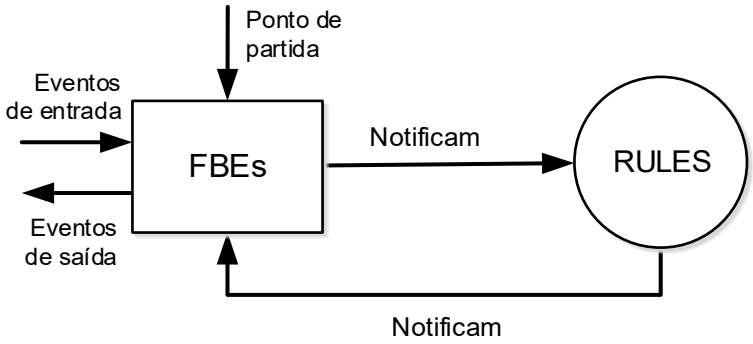
determinado momento o indicador de kWh restante de um cliente é 10 e em outro poderá ser 4. Em PON, esses fatos são representados por um elemento chamado *FBE* – do inglês *Fact Base Element*. Esses elementos permitem o agrupamento de dados relacionados a uma mesma entidade. Para tanto, cada *FBE* pode ser composto de um ou mais atributos (equivalente a uma variável), cada um armazenando determinado valor associado àquele *FBE*. Para o cenário do "medidor inteligente" de consumo elétrico, pode-se imaginar o *FBE Medidor* com os atributos "Código", "Status" e "kWhDisp", para registrar o código do aparelho, estado de funcionamento e quantidade de kWh disponíveis para consumo, respectivamente.

As Regras em PON são chamadas *Rules* e definem a lógica decisional de uma aplicação, ou seja, as Regras determinam as decisões a serem tomadas em razão das condições do sistema. Cada Regra descreve uma avaliação condicional sobre fatos do sistema representados pelos *FBEs*.

A avaliação dos fatos pelas regras é feita em PON de uma forma particular. No Paradigma Imperativo, é preciso criar fluxos para refazer as avaliações condicionais até que elas sejam satisfeitas, levando a algoritmos, por vezes, extensos, complexos e consumidores de processamento. No Paradigma Declarativo, um módulo de inferência analisa quais avaliações condicionais (i.e., regras) devem ser realizadas a cada mudança em algum fato do sistema, levando a um processo de busca (denominado *matching*) moroso. No PON, as avaliações condicionais (i.e., Regras) são realizadas por meio de "notificações" quando há a mudança em um fato do sistema. Uma "notificação" pode ser uma comunicação de um fato para uma regra, indicando uma mudança ocorrida naquele fato e levando à realização da avaliação condicional daquela Regra. Isso significa, que os *FBEs* são ativos e que, quando algum de seus fatos (relativos aos estados de seus atributos) tiver seu valor alterado, ele produz notificações às regras relacionadas. Por fim, quando a avaliação condicional de uma Regra for positiva, ela pode produzir mudanças em fatos do mesmo ou de outros *FBEs* (também por meio de notificações), podendo levar a novas notificações e ativações de regras.

Com esses conceitos, pode-se definir um modelo genérico do PON, no qual os fatos são representados pelos *FBEs*, as regras pelas *Rules* e o fluxo entre eles são conduzidos por meio de notificações (Figura 10).

Figura 10 - Modelo genérico do PON



Fonte: Autoria própria

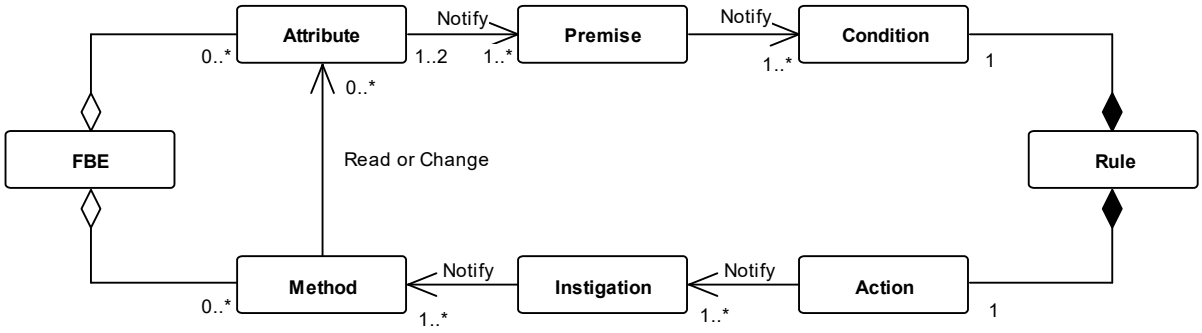
Esta seção apresentou uma breve introdução ao Paradigma Orientado a Notificações, relacionando-o com o processamento de dados usual, e abordou seus conceitos mais importantes que são os fatos (*FBE*), regras (*Rules*) e Notificações.

2.1.3 Fundamentos sobre o Paradigma Orientado a Notificações

Os componentes primários do PON são as Regra, *FBEs* e Notificações. Esta seção detalha os elementos da arquitetura do PON e apresenta sua dinâmica de execução. A Figura 11 apresenta o metamodelo do PON na forma de um diagrama de classes que ilustra os elementos da arquitetura e suas relações. A notação de diagrama de classes foi escolhida pois permite representar a tipologia dos elementos e as restrições de instanciação dadas por suas relações.

No desenho do metamodelo na Figura 11, percebem-se os dois elementos fundamentais da arquitetura (*FBE* e *Rule*) nos extremos esquerdo e direito do diagrama. Estes elementos (i.e., classes) podem originar um número indefinido de instâncias dentro de uma aplicação PON. Os demais elementos (i.e., classes) do metamodelo são instanciados de acordo com a lógica pretendida para a aplicação e seguindo as restrições impostas no metamodelo.

Figura 11 - Metamodelo do PON em UML



Fonte: Adaptado de Simão e Stadzisz (2009)

2.1.3.1 Elementos da arquitetura do PON

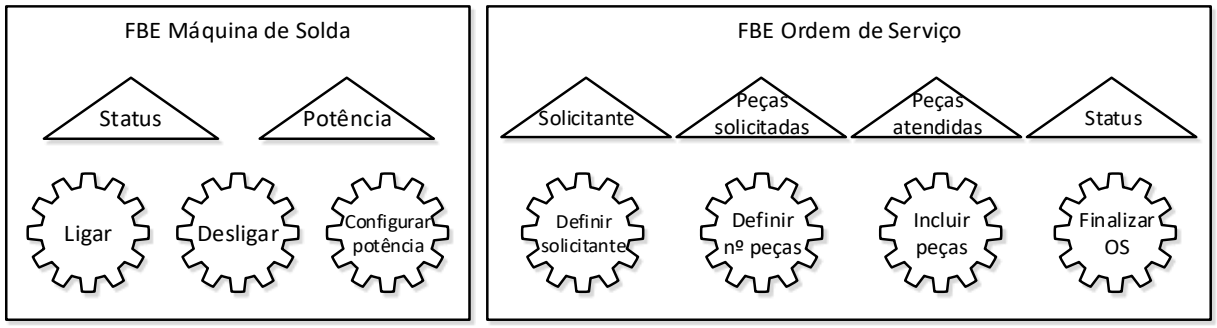
A classe ***FBE*** define um modelo para instanciar entidades que representam, cada uma, um conjunto coeso de atributos e métodos. Um *FBE* tipicamente está associado a uma entidade do mundo real ou entidade de caráter computacional relevante para os propósitos do sistema. Os atributos de um *FBE* são equivalentes a variáveis que armazenam valores. Cada atributo é representado por uma instância da classe ***Attribute*** agregada ao *FBE*. Assim, os atributos de um *FBE* contêm os fatos (e, por consequência, determinam o estado atual) relacionados com o *FBE* ao longo do seu ciclo de vida. Os métodos de um *FBE*, por sua vez, representam os serviços oferecidos por ele às Regra, ou seja, um conjunto de procedimentos (i.e., rotinas ou funções) do *FBE*. Cada método é representado por uma instância da classe ***Method*** agregada ao *FBE*.

Para ilustrar o uso de *FBEs* no PON serão demonstrados dois exemplos em um cenário de fabricação de peças de metal. O primeiro exemplo, ilustrado à esquerda na Figura 12, é de um *FBE* que representa uma máquina de solda. Essa máquina possui dois *Attributes*: *Status* e *Potência* (representados pelos dois triângulos). Os valores que o *Status* da máquina poderá assumir são “ligada” e “desligada” e, para o *Attribute Potência,* os valores possíveis são “baixa”, “média” e “alta”. Os *Methods* que esse *FBE* oferece são: *Ligar*, *Desligar* e *Configurar potência* (representados pelas engrenagens). Então, a partir dos métodos e atributos[3] que esse *FBE* possui, pode-se conhecer os estados que ele poderá assumir no decorrer

[3] Optou-se por padronizar, no decorrer do texto, o uso dos nomes dos elementos do PON em português. O termo será usado em inglês com a primeira letra maiúscula quando se referir a elementos de um modelo. Por exemplo, *Method* Ligar ou *Attribute* Status.

do seu ciclo de vida. A máquina de solda pode, por exemplo, estar “ligada e operando em potência média” ou “desligada e configurada para funcionar em potência alta”. Os *Methods* permitem o controle da operação da máquina de solda. O *Method Ligar* irá ativar a máquina de solda, consequentemente alterando o seu *Attribute Status* para “ligada”. O *Method Configurar potência* permite que a potência da máquina seja alterada para algum dos valores “baixa”, “média” ou “alta”. Deve-se perceber que, tipicamente, os métodos são os meios pelos quais os atributos são alterados.

Figura 12 - Representação de *FBEs* com seus atributos e métodos



Fonte: Autoria própria

Outro exemplo de *FBE* no mesmo cenário é uma *Ordem de Serviço* (OS) para confecção de peças. Os atributos e métodos desse *FBE* estão ilustrados na Figura 12, à direita. Os *Attribute*s são: *Solicitante*, *Peças solicitadas*, *Peças atendidas* e *Status*. O *Attribute Solicitante* registra o nome do solicitante na forma de uma cadeia de caracteres. Peças solicitadas e Peças atendidas registram o número inteiro de peças solicitadas na OS e atendidas, respectivamente. O *Status* aceita os valores “nova”, “em processamento” e “finalizada”. Os métodos desse *FBE* são: *Definir solicitante*, *Definir nº peças, Incluir peças* e *Finalizar OS*. Os dois primeiros métodos são usados, geralmente, na criação de uma OS, enquanto os outros dois, são usados à medida que as peças dessa OS são concluídas e para finalizar a OS, respectivamente.

A classe ***Rule*** define um modelo para instanciar entidades que representam, cada uma, uma regra lógico-causal em PON. A *Rule* é composta de uma instância da classe ***Condition*** e uma instância da classe ***Action*** (cf. Figura 11), que descrevem a condição lógica para a regra ser aprovada e a ação a ser realizada quando a regra for executada, respectivamente.

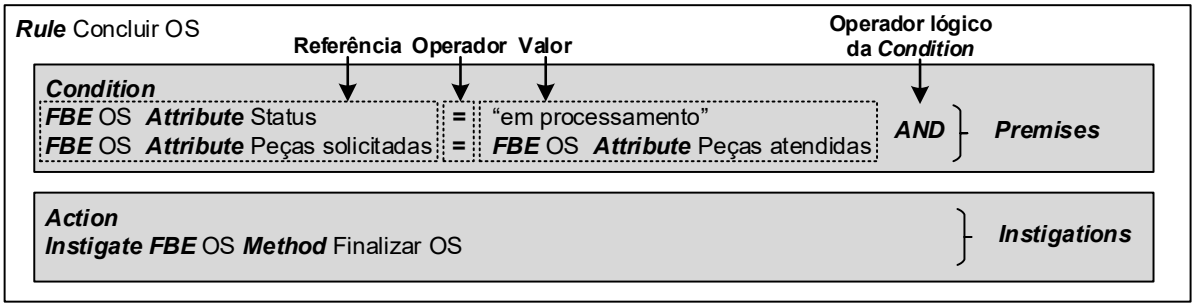
Um exemplo de *Rule* é a decisão sobre a conclusão de uma *Ordem de Serviço* (OS). A condição para que essa *Rule* seja executada é que o número de *Peças solicitadas* e *Peças atendidas* do *FBE OS* sejam iguais. A ação desencadeada pela *Rule*, por sua vez, é a finalização da OS realizada pelo *Method Finalizar OS*. Este *Method* deverá alterar o *Attribute Status* do mesmo *FBE* para "finalizada". A Figura 13 ilustra a composição dessa *Rule,* denominada *Concluir OS*.

Figura 13 - *Rule* que conclui uma Ordem de Serviço (OS)

Fonte: Autoria própria

Para apresentar os demais elementos do metamodelo do PON, será incluída uma nova exigência à *Rule* ilustrada na Figura 13. A nova exigência é que o *Status* do *FBE OS* seja "em processamento" para que a *Rule* execute. A *Rule* modificada é ilustrada na Figura 14[4].

Figura 14 - *Rule* que conclui a Ordem de Serviço (atualizada)



Fonte: Autoria própria

As classes ***Premise*** e ***Instigation*** definem modelos para instanciar entidades adicionais no PON. As *Premises* são entidades vinculadas às *Conditions* e as *Instigations* são vinculadas às *Actions*.

Cada *Premise* é composta de uma tripla envolvendo (i) uma referência a um *Attribute*, (ii) um operador lógico e (iii) um valor ou uma referência a um outro *Attribute* (cf. Figura 14). O papel de cada *Premise* é realizar a avaliação lógica (i.e.,

[4] A figura apresenta código no formato português estruturado para facilitar a interpretação, não sendo esse um formato de estruturação do código PON.

comparações) de um *Attribute* com um valor ou com outro *Attribute*. Os atributos referenciados em uma *Premise* correspondem a *Attributes* de *FBEs* da aplicação. A partir destas referências, sempre que o valor do *Attribute* for modificado, a *Premise* reavaliará o seu estado lógico.

A *Condition* de uma *Rule*, que determina quando a *Rule* poderá ser ativada, pode fazer a avaliação lógica sobre uma ou mais *Premises*. Quando mais de uma *Premise* constituem a avaliação da *Condition*, indica-se o tipo de avaliação: se uma conjunção (AND) ou se uma disjunção (OR). Como exemplo, a *Condition* ilustrada na Figura 14, faz a avaliação lógica por meio da conjunção de duas *Premises*, indicada pelo operador AND.

Como uma *Premise* é uma entidade instanciada, ela pode estar associada a diversas *Conditions.* Assim, a avaliação lógica contida em uma *Premise* é realizada uma única vez para todas as *Conditions* que consideram seu resultado. Elimina-se, desta forma, reavaliações lógicas redundantes entre regras.

As *Instigations*, por sua vez, são entidades utilizadas para compor conjuntos de chamadas a *Methods* de *FBEs*. As *Instigations* são referenciadas dentro da *Action* das *Rules* e definem as ações de uma *Rule* quando é ativada. *Instigations* funcionam como um *container* (i.e. bloco) de chamadas de *Methods*. Elas são reutilizadas entre *Rules* para indicar conjuntos idênticos de chamadas de *Methods* e evitar a redundância na especificação de *Rules* semelhantes. Seguindo o exemplo apresentado na Figura 14, quando a *Action* dessa *Rule* for executada, a *Instigation* irá executar o *Method Finalizar OS* do *FBE OS*. Se houvesse outras *Instigations*, o mesmo processo ocorreria com elas.

Cada elemento do metamodelo do PON é uma entidade computacional que tem papel bem definido na arquitetura do paradigma. A seguir esses elementos serão recapitulados.

- *FBEs*: elementos responsáveis por representar e organizar logicamente as entidades do mundo real ou computacionais envolvidas no sistema. Cada *FBE* é composto de um conjunto de *Methods* e *Attributes*.
- Métodos (*Methods*): são os serviços oferecidos (i.e., ou funções) pelos *FBEs*. Por meio deles ocorrem operações nos *FBEs* e, ademais, os *Attributes* do *FBE* são modificados.

- Atributos (*Attributes*): elementos responsáveis por armazenar os valores que representam os fatos a respeito de cada *FBE* e, por consequência, seu estado atual.
- Regras (*Rules*): elementos que definem a lógica decisional nas aplicações desenvolvidas no PON. *Rules* são a agregação de uma *Condition* e uma *Action*.
- Condições (*Conditions*): elementos que determinam quando uma *Rule* poderá ser ativada. São constituídas de uma ou mais *Premises* e usam operadores lógicos entre as *Premises* para definir a expressão lógica que ativa a sua *Rule*.
- Ações (*Actions*): elemento responsável pela sua execução das ações da *Rule*. Cada *Rule* possui somente uma *Action*, contudo poderá possuir diversas *Instigations* que, por sua vez, poderão executar diversos *Methods*.
- Premissas (*Premises*): elemento responsável por avaliar o valor de um *Attribute*. As *Premises* podem ser compartilhadas entre várias *Conditions*, evitando comparações redundantes.
- Instigações (*Instigations*): cada regra possui somente uma *Action*, a *Instigation* permite que as regras compartilhem conjuntos de *Methods* a serem invocados.

2.1.3.2 Mecanismo de Notificações do PON

As relações entre os elementos do metamodelo do PON definem como as aplicações são executadas no paradigma. As regras da aplicação são ativadas a partir de mudanças nos fatos, porém, diferentemente dos Sistemas Baseados em Regras (SBRs) que necessitam de um motor de inferência (ou módulo de inferência), o PON apresenta um mecanismo particular, baseado em notificações, que realiza a ativação das regras.

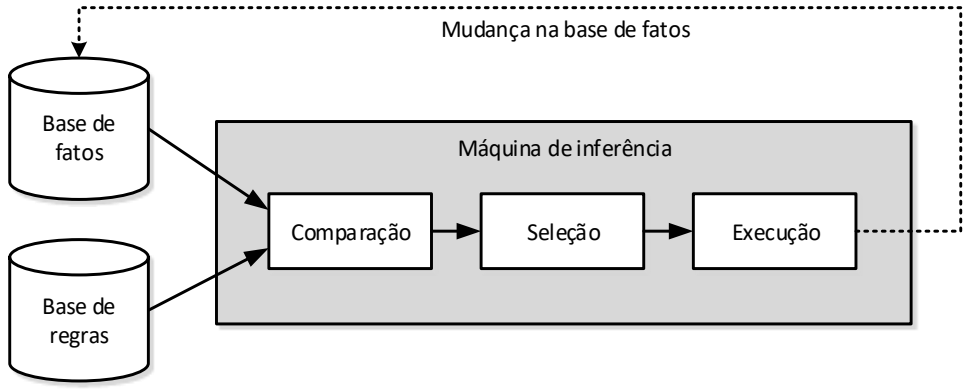
As arquiteturas de Sistemas Baseados em Regras possuem uma memória para armazenamento de fatos (Base de Fatos), uma memória para armazenamento de regras (Base de Regras) e um mecanismo, chamado de máquina de inferência, para inferir sobre essas duas memórias a fim de aprovar e executar as regras (FRIEDMAN-HILL, 2003). O mecanismo de inferência dos SBRs envolve três

passos, conforme ilustrado na Figura 15, para decidir quais ações deverão ser realizadas:

1) Comparação (*Matching*): nesse passo ocorre a análise das correspondências entre os fatos e as condições das regras para identificar quais regras estão aprovadas.
2) Seleção (*Selection*): como mais do que uma regra poderá estar aprovada após o passo anterior, nesse passo o mecanismo deverá decidir quais regras deverão ser executadas e em qual ordem.
3) Execução (*Execution*): nesse passo ocorre a execução das regras conforme seleção feita no passo anterior e, ao término, retorna-se para o passo 1.

O primeiro passo é um dos gargalos dos mecanismos de inferência, pois o aumento no número de fatos e regras faz com que esse processo de comparação se torne demorado (NALEPA *et al.*, 2011).

Figura 15 - Arquitetura interna de um SBR



Fonte: adaptado de Friedman-Hill (2003)

O PON faz uso de conceitos de SBRs, porém, no que diz respeito ao mecanismo de inferência, define um novo modo para conduzir a execução das aplicações que não faz uso de máquinas de inferência. No PON, a inferência sobre fatos e regras é conduzida por meio de notificações entre os elementos da aplicação.

As relações de notificação no metamodelo do PON representam associações entre instâncias das classes indicando que uma instância de um elemento pode notificar (ou seja, realizar uma comunicação a) uma instância de outro elemento, ou seja, os elementos sabem previamente quais elementos notificar

quando há alteração no seu estado. O Quadro 1 define o significado das notificações para os diferentes tipos de associações existentes.

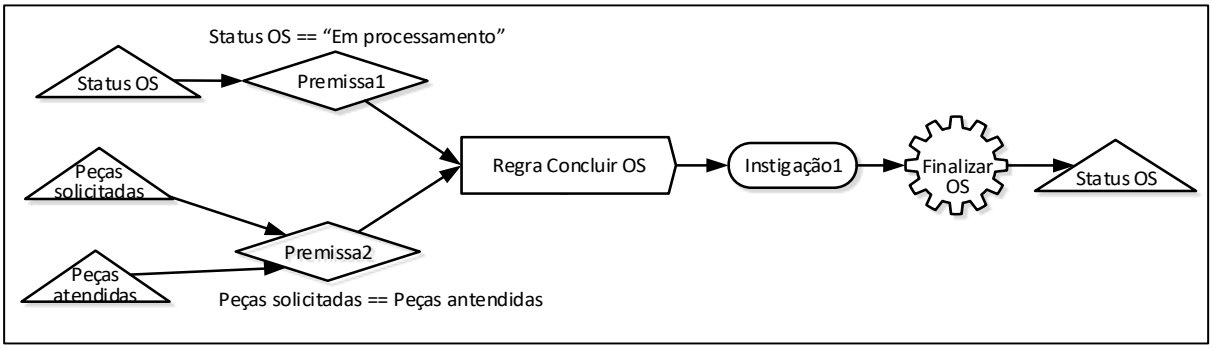
Quadro 1 - Tipos de notificações no PON

Tipo de Associação	Significado da Notificação
Attribute → Premise	Quando um atributo de um *FBE* tem seu valor alterado, ele envia uma mensagem (notificação) para uma ou mais premissas previamente conhecidas.
Premise → Condition	Quando uma premissa avaliar seu estado lógico e o resultado for uma avaliação positiva (i.e., verdadeiro), uma mensagem (notificação) será enviada para uma ou mais condições de regras previamente conhecidas.
Action → Instigation	Quando uma regra é aprovada e executada, sua *Action* irá enviar uma mensagem (notificação) a uma ou mais instigações predefinidas.
Instigation → Method	Quando uma instigação é notificada, ela irá enviar uma mensagem (notificação) a um ou mais métodos predefinidos.

Fonte: Autoria própria

Para exemplificar notificações em uma aplicação do PON, a Figura 16 ilustra as instâncias e associações para a *Rule Concluir OS* do exemplo da Figura 14. A *Rule* é uma agregação de uma *Condition* e uma *Action*. Ela é representada nessa figura por um pentágono irregular em que se considera a extremidade esquerda sendo a *Condition* e a extremidade direita sendo a *Action*. Na figura são apresentados, à esquerda da *Rule*, os *Attributes* (representados por triângulos) e *Premises* (representadas por losangos) envolvidos na avaliação da *Rule* e à direita são apresentados a *Instigation* (representada por um retângulo com bordas arrendondas), *Method* e *Attribute* que participam na execução da *Rule*.

Figura 16 - Instâncias e associações da *Rule Concluir OS*[5]



Fonte: Autoria própria

A Figura 16 ilustra três notificações de *Attributes* para *Premises*. O *Attribute Status OS* quando tiver seu valor alterado irá notificar esta alteração à *Premissa1*. Da mesma forma, os *Attributes Peças solicitadas* e *Peças atendidas* irão notificar a mesma *Premissa2* quando seus valores forem alterados. As *Premises* 1 e 2 avaliam seus estados lógicos quando forem notificadas e, caso essa avaliação seja verdadeira, irão notificar a *Condition* da *Rule Concluir OS*. Assim que a *Condition* da regra for satisfeita (i.e., assim que ela avaliar a sua expressão lógica como verdadeira, após receber a notificação das *Premises*), a regra será ativada e poderá executar a sua *Action*. A execução da *Action* da *Rule Concluir OS* irá notificar a *Instigação1*, que, por sua vez, irá notificar o *Method Finalizar OS*. Finalmente, este *Method* irá alterar o valor do *Attribute Status OS,* finalizando um ciclo de inferência no PON.

2.1.3.3 Fluxos de execução no PON

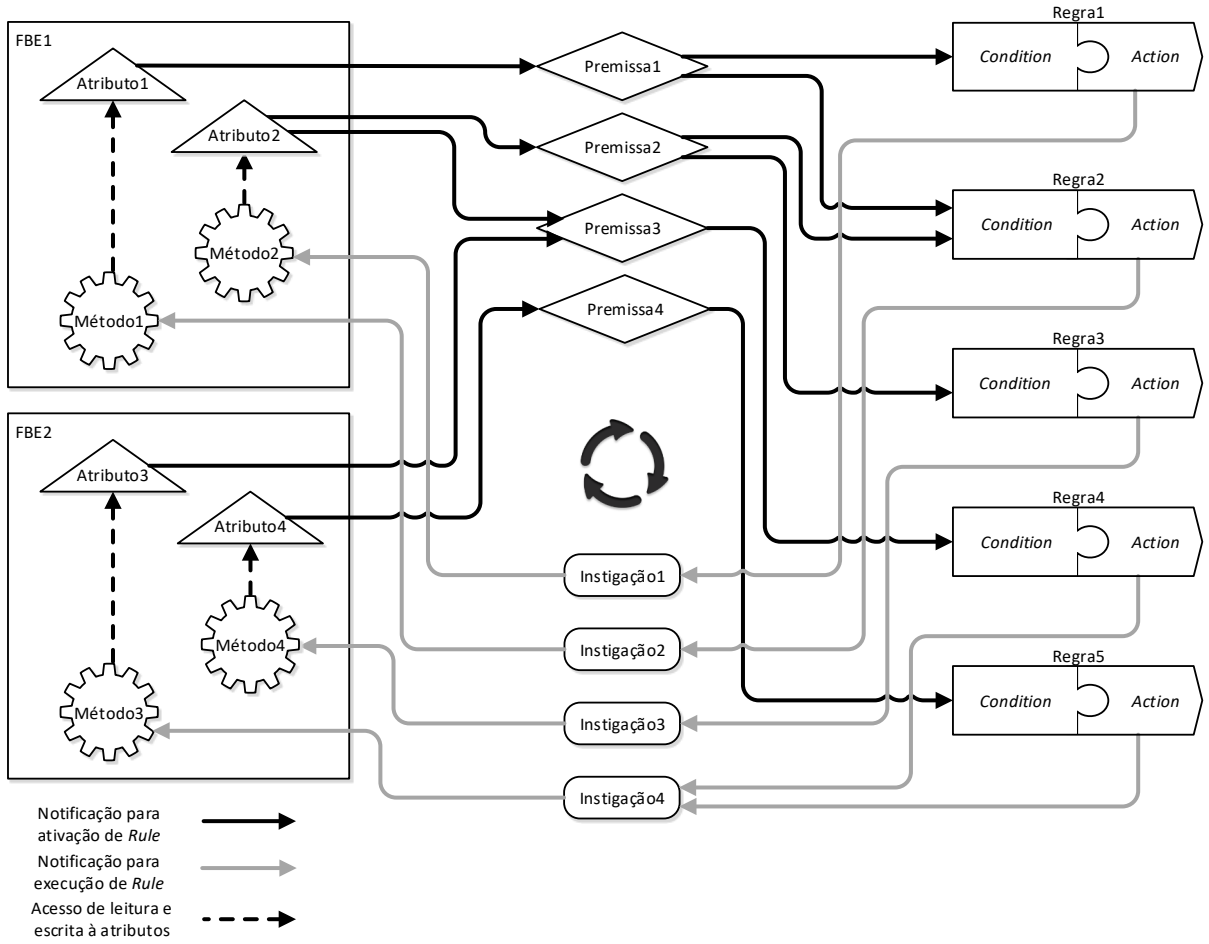
Um fluxo de execução pode ser entendido como um encadeamento sequencial ou concorrente de ativações de regras, compondo certa funcionalidade do software. Para exemplificar, a Figura 17 ilustra uma aplicação PON na qual foram instanciadas 5 *Rules*, 4 *Premises*, 4 *Instigations* e 2 *FBEs*, cada um com 2 *Attributes* e 2 *Methods*. As notificações entre os elementos dessa aplicação são representadas por setas direcionais. Um exemplo de fluxo de execução nesta ilustração inicia no *Atributo4* do *FBE2*. Quando o *Atributo4* tem seu valor alterado, ele notifica a *Premissa4*. A *Premise* é reavaliada como verdadeira e notifica a *Condition* da *Regra5*. Essa regra, cuja *Condition* não possui outras *Premises*, é executada. A

[5] Ilustração criada usando uma adaptação do modelo de objetos definido em (Kossoski, 2015).

Action dessa *Rule* notifica a *Instigação4* que irá notificar o *Método3* do *FBE2*. O *Método3*, por sua vez, irá modificar o *Atributo3* do mesmo *FBE*. A modificação no *Atributo3* dá sequência ao fluxo de execução, porém, como a *Premissa3*, por ele notificada, também depende do *Atributo2* do *FBE1*, este fluxo de execução aguarda a modificação no *Atributo2* para continuar.

Outro exemplo de fluxo de execução se inicia no *Atributo1* do *FBE1.* Quando esse *Attribute* tem seu valor alterado, ele notifica a *Premissa1*. A *Premise* é reavaliada como verdadeira e notifica as *Conditions* das *Regras 1 e 2*. A *Regra1*, cuja *Condition* não possui outras *Premises*, é executada. A *Action* dessa *Rule* notifica a *Instigação1* que irá notificar o *Método2* do *FBE1*. O *Método2*, por sua vez, irá alterar o *Atributo2* do mesmo *FBE*. A alteração no valor do *Atributo2* gera notificações para as *Premissas 2* e *3*. Neste caso, somente a reavaliação da *Premissa2* resulta verdadeira. Assim, as *Conditions* das *Regras 2* e *3* são notificadas. A Regra3, cuja *Condition* que não possui outras *Premises* é executada e a *Regra2*, cuja outra *Premise* da *Condition* havia sido notificada como verdadeira no passo anterior também é executada. As *Actions* dessas *Rules* notificam as Instigações 2 e 3, respectivamente, que notificam o *Método 1* e *4*. O *Método1* modifica o *Atributo1* e o *Método4* modifica o *Atributo4*, encerrando este fluxo de execução.

Figura 17 - Fluxos de execução no PON



Fonte: adaptado de Linhares (2015)

Do ponto de vista de desempenho, para o pior e irrealístico caso, a análise assintótica[6] da inferência no PON é representado por $O(n^3)$, apresentando-se mais eficiente que os algoritmos de inferência mais conhecidos[7]. Além disso, no caso médio[8] e realístico, a complexidade da inferência no PON torna-se linear $O(n)$ (BANASZEWSKI, 2009; RONSZCKA *et al.*, 2015; SIMÃO, 2005).

2.1.3.4 Resolução de conflitos e garantia de determinismo no PON

Um conflito é uma situação na qual duas, ou mais, atividades (ou entidades) dependem de um mesmo elemento (i.e., recurso) em um mesmo instante de tempo,

[6] A análise assintótica é um método usado para avaliar o desempenho de um algoritmo quando aplicado a uma grande massa de dados (EDMONDS, 2008).
[7] RETE (FORGY, 1982), TREAT (MIRANKER, 1987), LEAPS (MIRANKER *et al.*, 1990) e HAL (LEE; CHENG, 2002) são exemplos de algoritmos de inferência mais conhecidos.
[8] A análise assintótica determina o desempenho em cenários de massas de dados menores, médias e de pior caso, sendo que o caso médio o de maior ocorrência no algoritmo (EDMONDS, 2008).

sendo que este elemento deve ser usado exclusivamente por uma das atividades. Há, portanto, uma disputa ou conflito por um recurso. A falta de um tratamento adequado a essas situações pode acarretar em problemas com o determinismo da aplicação, ou seja, certas condições ora executam uma regra ora executam outra (SEGULJA; ABDELRAHMAN, 2014). Outro problema que poderá ocorrer, neste caso pela falta do tratamento do conflito, é a existência de regras mortas, ou seja, regras que nunca serão executadas.

Em PON, o determinismo da aplicação e o tratamento de conflitos podem ser garantidos com o uso de regras interbloqueadas (SIMÃO, 2005), ou seja, para cada conjunto de regras que disputem o acesso a um recurso, criam-se atributos para verificações adicionais. Esses atributos irão garantir que somente uma regra, que faça uso de certo recurso compartilhado, seja aprovada e, consecutivamente, executada por vez. Todavia, à medida que o número de *FBEs* e *Rules* aumenta, a criação de regras interbloqueadas torna-se trabalhosa e suscetível a erros (BANASZEWSKI, 2009).

No âmbito de SBRs, a resolução de conflitos consiste na ordenação das regras que foram aprovadas para, então, serem executadas naquela ordem. A ordenação segue uma estratégia predefinida à escolha do projetista (FRIEDMAN-HILL, 2003), evitando assim os conflitos existentes pela execução sequencial das regras.

Banaszewski (2009) propôs mecanismos similares aos encontrados nos SBRs para auxiliar no tratamento de conflitos no PON para ambientes monoprocessados. A proposta básica utiliza um mecanismo centralizado com uma entidade concentradora, que faz uso da estratégia predefinida para execução das regras aprovadas. As regras aprovadas, aguardando para execução, formam um conjunto de potencial conflito. A entidade concentradora usa uma das estratégias a seguir para definir a ordem de execução das regras: (i) BREADTH, na qual as regras são enfileiradas conforme ordem de aprovação em um escalonamento FIFO[9]; (ii) DEPTH, na qual as regras são enfileiradas conforme a ordem de aprovação por um escalonamento LIFO[10]; (iii) PRIORITY, na qual as regras são ordenadas conforme a

[9] FIFO (*First-In First-Out*) é uma estrutura de fila em que o primeiro elemento que entra é o primeiro que sai.
[10] LIFO (*Last-In First-Out*) é uma estrutura de fila em que o último elemento que entra na fila é o primeiro que sai.

sua prioridade. O projetista pode optar, ainda, por não usar o mecanismo de resolução de conflitos e essa é a estratégia padrão denominada NO_ONE. Neste caso, o projetista fica responsável por evitar os conflitos com o uso de regras interbloqueadas.

O uso do mecanismo centralizado e a escolha de uma das estratégias de escalonamento oferecem um meio para resolução de conflitos em ambientes monoprocessados. Para isso, o projetista precisa conhecer o comportamento de cada estratégia para associá-lo à lógica da aplicação sendo desenvolvida. Em casos mais simples, com um número reduzido de regras, o uso de prioridade poderá ser suficiente. O uso desse mecanismo de resolução de conflitos é um avanço em relação a regras interbloqueadas, porém o seu emprego não é trivial e pode necessitar ferramentas adicionais que simulem o comportamento de cada estratégia na fase de projeto do desenvolvimento de aplicações PON (BANASZEWSKI, 2009).

Ainda no âmbito de mecanismos de resolução de conflitos, Simão e Stadzisz (2011) propuseram outro mecanismo de garantia de determinismo e identificação e resolução de conflitos para implementações distribuídas ou sequenciais no PON. Nesse outro mecanismo são definidas notificações adicionais entre as classes *Attribute*, *Premise* e *Condition*. Para a garantia de determinismo, estas classes são especializadas e chamadas de *Deterministic-Attributes*, *Deterministic-Premises* e *Deterministic-Conditions*, respectivamente. O conceito chave deste mecanismo é a confirmação das notificações dos *Deterministic-Attributes* até as *Deterministic-Conditions.* Assim, é possível garantir que as premissas e condições afetadas pelas alterações nos *Deterministic-Attributes* tenham a oportunidade de avaliá-los.

Para a identificação e resolução de conflitos Simão e Stadzisz (2011) propuseram notificações adicionais entre as classes *Attribute*, *Premise* e *Condition* especializadas, chamando-as *Exclusive-Attribute*, *Exclusive-Premise* e *Exclusive-Condition*. O desenvolvedor usa essas classes para identificar as *Rules* potencialmente em conflito. O conflito é estabelecido no mecanismo quando duas, ou mais, *Exclusive-Conditions* de *Rules,* em estado verdadeiro, têm uma mesma *Exclusive-Premise* conectada. A *Exclusive-Premise* aguarda a confirmação das *Exclusive-Conditions* antes de tomar a decisão sobre qual *Rule* poderá executar. A decisão sobre qual *Rule* executar pode ser determinada pela própria *Exclusive-Premise,* quando a prioridade da *Rule* for suficiente ou pode ser delegada a uma

entidade chamada de *Conflict-Solver* (Resolvedor de conflitos), atualmente renomado como *Conflict-Advisor*.

No uso do *Conflict-Advisor*, ele recebe uma lista de *Rules* em conflito da *Exclusive-Premise* e decide, conforme sua estratégia, qual delas deverá ser aprovada. O mecanismo garante que somente uma *Rule* seja executada e as outras sejam reprovadas. Por fim, os autores afirmam que os mecanismos de garantia de determinismo e identificação e resolução de conflitos propostos são sinérgicos. Assim, as contra-notificações usadas nos mecanismos podem ser as mesmas, reduzindo o número de notificações necessárias para conseguir a garantia de determinismo e a resolução de conflito.

2.1.3.5 Principais contribuições do Paradigma Orientado a Notificações

As seções anteriores apresentaram os fundamentos do PON. Em resumo, as suas principais contribuições:

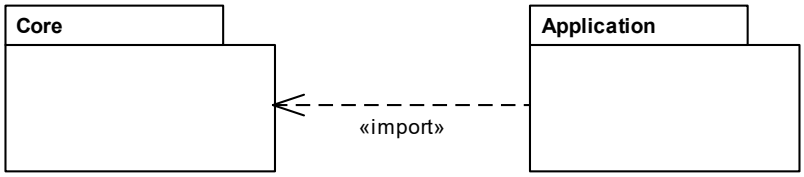
- Redução de avaliações causais: o PON é organizado em pequenas entidades reativas que se comunicam de forma preestabelecida por notificações (SIMÃO; STADZISZ, 2011), isto porque, toda notificação está prevista na estrutura do software. Ela ocorre porque a conexão entre as entidades foi construída (prevista) na programação. Assim, quando duas entidades A e B possuem uma dependência, a entidade dependente B é ativada (notificada) diretamente pela entidade A da qual depende, quando esta última identifica que seu estado foi modificado, sendo que ela faz isso, também, após ter sido ativada (LINHARES, 2015).
- Menor acoplamento na estrutura do software: as entidades que compõem o PON foram concebidas para serem autocontidas, isto é, coesas, reativas e desacopladas. Assim, o paradigma é suscetível a distribuição, ou execução paralela, desde que haja meios técnicos para propagar a notificação entre as entidades envolvidas na computação distribuída (ou paralela).
- Facilidade de programação advinda do Paradigma Declarativo em que se define a lógica do que deve acontecer e não como deve

acontecer, assim, os mecanismos internos que definem o fluxo lógico são frequentemente ocultados do projetista (KAISLER, 2005).

2.1.4 Estado de desenvolvimento do PON

O PON, como paradigma de programação de uso geral, teve sua primeira implementação por meio de uma versão prototipal na linguagem C++, evoluída a partir do *framework* de Controle Orientado a Notificações (SIMÃO, 2001, 2005). Baseado nessa versão, criou-se o primeiro *framework* PON, chamado de *Framework* PON C++ 1.0 no âmbito do trabalho de mestrado de Roni Fabio Banaszewski (BANASZEWSKI, 2009). Basicamente, o *Framework* PON C++ 1.0 fornece, por meio de dois pacotes base, apresentados na Figura 18, classes para instanciar os elementos do metamodelo do PON. O desenvolvedor estende a classe *Application* do pacote de mesmo nome, podendo assim fazer uso do *framework*.

Figura 18 - Estrutura do *Framework* PON C++ 1.0



Fonte: adaptado de Banaszewski (2009)

No *Framework* PON C++ 1.0 o desenvolvedor define cada elemento do PON. O Código 1 ilustra parte de um código que define a estrutura de um *FBE*. Nesse exemplo, a linha 01 mostra que o nome do *FBE* é *MaquinaDeSolda*. Na linha 02 é exibida a criação de um *Attribute* de nome *atStatus* do tipo inteiro e na linha 03 a definição de que existe um *Method* chamado *mtDesligar*. Nas linhas 06 a 08 o método é descrito e simplesmente atribui o valor “0” ao *Attribute atStatus*.

Código 1 - Exemplo de criação de *FBE* no *Framework* PON C++ 1.0

```
01 MaquinaDeSolda::MaquinaDeSolda(void) : FBE(){
02   atStatus = new Integer(this, 0);
03   mtDesligar = new MethodPointer<Gate>(this, &Gate::mtDesligar);
04 }
05 ...
06 void MaquinaDeSolda::mtDesligar() {
07   atStatus->setValue(0);
08 }
```

Fonte: Autoria própria

No caso das regras, o desenvolvedor faz referência aos *FBEs* criados. O Código 2 ilustra a definição de uma *Rule* no *Framework* PON C++ 1.0. A linha 01 desse código cria a *Rule* nomeando-a *rlDesligar*. Essa *Rule* é definida como "SINGLE", ou seja, possui somente uma *Premise*. Na linha 02, a *Premise* é informada, cuja verificação é se o *Attribute atBotaoDesliga* do *FBE maquina1* é "TRUE", e na linha 03 é definida a única *Instigation* da *Rule* que chama o *Method mtDesligar* do *FBE maquina1*. Na linha 04, o código encerra a criação da *Rule*.

Código 2 - Exemplo de criação de Rule no *Framework* PON C++ 1.0

```
01 Rule* rlDesligar = new Rule(Condition::SINGLE);
02   rlDesligar->addPremise(
                  maquina1->atBotaoDesliga,
                  Boolean::TRUE,
                  Premise::EQUAL);
03   rlDesligar->addInstigation(new Instigation(maquina1->mtDesligar);
04 rlDesligar->end();
```

Fonte: Autoria própria

Os exemplos apresentados servem somente para ilustrar como o desenvolvedor cria uma aplicação instanciando os objetos a partir do *Framework* PON C++ 1.0. Por se tratar de um *framework* em C++, o desenvolvedor pode construir processamentos particulares (como a interação de entrada e saída com o usuário) dentro do corpo dos métodos do *FBEs*. Exemplos completos e mais elaborados de códigos utilizando o *Framework* PON C++ 1.0 podem ser encontrados na dissertação (BANASZEWSKI, 2009) e nos trabalhos subsequentes (LINHARES, 2015; RONSZCKA, 2012; VALENÇA *et al.*, 2011; WIECHETECK, 2011).

A primeira versão do *Framework* PON C++ 1.0 representou uma base de desenvolvimento que permitiu a avaliação das características desse novo

paradigma. Ronszcka (2012) e Valença (2012) evoluíram o *Framework* PON C++ 1.0 em diferentes aspectos em seus trabalhos de mestrado.

Ronszcka (2012) identificou e implementou padrões de projeto para a nova versão do *framework*. Os padrões definidos pela “Gangue dos Quatro” (Gang of Four - GoF)[11] foram implementados da seguinte maneira: *Iterator*[12] para realizar a iteração nas coleções de endereços notificáveis nas entidades PON; *Abstract Factory*[13] na criação de entidades do PON; *Singleton*[14] para centralizar a criação de entidades a partir de uma única instância da “fábrica PON”; *Command*[15] para delegação da execução do fluxo de execução de forma minimamente acoplada; e, por fim, *Observer*[16], responsável pela realização de notificações entre as entidades PON. Isto permitiu discutir a questão de que o PON não se constitui em uma simples aplicação do *Observer*, mas sim a extrapolação deste padrão para realização do cálculo lógico-causal.

Ronszcka (2012) também contribuiu para o novo *framework* PON, nomeadamente *Framework* PON C++ 2.0, com a implementação de conceitos, ou padrões em PON, que melhoram aplicações desenvolvidas no paradigma, reduzindo o número de notificações. Um deles é o conceito de regras dependentes, no qual um conjunto de regras depende de uma regra mestre e não somente de suas próprias premissas. Com isso, o desenvolvedor pode agrupar regras que compartilhem as mesmas premissas. Outro conceito implementado foi o de “atributos impertinentes”. Um atributo impertinente é aquele que modifica seu estado constantemente, gerando um grande número de notificações que não aprovam regras em um dado contexto. O desenvolvedor pode marcar atributos como “impertinentes” para o conjunto de premissas afetadas pela impertinência. Assim, o atributo não irá notificar

---

[11] GoF é um termo cunhado para denotar Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides, autores do livro *Design Patterns* (Gamma *et al.*, 1995) que definiu padrões de projeto na Engenharia de *Software*.

[12] *Iterator* é um padrão da GoF que define um objeto para que programador possa examinar um container, particularmente listas.

[13] *Abstract Factory* é um padrão da GoF que permite a criação de famílias de objetos relacionados ou dependentes por meio de uma única interface e sem que a classe concreta seja especificada.

[14] *Singleton* é um padrão da GoF que garante a existência de apenas uma instância de uma classe, mantendo um ponto global de acesso ao seu objeto.

[15] *Command* é um padrão da GoF que permite encapsular uma solicitação como um objeto, desta forma permitindo que clientes parametrizem diferentes solicitações, enfileirem ou façam o registro (*log*) de solicitações e suportem operações que podem ser desfeitas.

[16] *Observer* é um padrão da GoF que define uma dependência um-para-muitos entre objetos de modo que quando um objeto muda o estado, todos seus dependentes são notificados e atualizados automaticamente.

essas premissas. Quando as condições da regra tiverem as outras premissas verdadeiras, o atributo impertinente é habilitado a notificar.

Outra contribuição apresentada por Ronszcka (2012) foi a definição de Métodos PON para operações matemáticas. Anteriormente, o desenvolvedor precisava recorrer a métodos do Paradigma Orientado a Objetos para realizar operações entre os atributos do PON. Na nova versão do *framework* PON, o *Framework* PON C++ 2.0, a definição desses métodos matemáticos permite que essas operações sejam realizadas puramente em PON. Ronszcka (2012) implementou, ainda, melhorias no *Framework* PON C++ 2.0 para permitir a depuração de aplicações, dada as particularidades no fluxo de execução de uma aplicação PON. Por fim, contribuiu na definição e uso de pseudônimos (i.e. instruções *define* do C/C++) para simplificar a instanciação de objetos do *framework* PON. O Código 3 ilustra a criação de uma mesma *Premise* sem e com o uso dos pseudônimos, nas linhas 01 a 05 e linha 07, respectivamente.

Código 3 - Criação de Premissa sem e com o uso de pseudônimos

```
01 Premise* prHasNoLives =
02   SingletonFactory::createPremise(
03     game->score->atNumLives, 0, Premise::EQUAL
04   );
05 prHasNoLives ->setName("prHasNoLives");
06
07 PREMISE(prHasNoLives, game->score->atNumLives, 0, Premise::EQUAL);
```

Fonte: adaptado de Ronszcka (2012)

As referências entre elementos da arquitetura do PON usadas para notificações eram representadas na forma de listas. Na primeira versão do *framework* PON , i.e. *Framework* PON C++ 1.0, usaram-se estruturas de lista e pilha da biblioteca *Standard Template Library* (STL) que, apesar de serem padronizadas, possui certa ineficiência computacional, por ter sido criada para uso genérico. Valença (2012) criou estruturas de dados personalizadas para o *Framework* PON C++ 2.0 que se mostraram mais eficientes.

Particularmente Valença (2012), mas também Ronszcka (2012), refatoraram o *framework* PON, seguindo os conceitos apresentados em Beck (2000) e Fowler (2004). Essa refatoração criou uma nova versão do *framework* PON que passou a

ser chamada de “*Framework* Otimizado” ou, conforme já salientado, *Framework* PON C++ 2.0.

Valença (2012) implementou, também, melhorias pontuais no *Framework* PON C++ 2.0, parte delas para permitir ao desenvolvedor maior controle sobre o ciclo de notificações. Uma dessas melhorias foi a alteração no mecanismo que modifica o valor de um atributo. Assim, o desenvolvedor pode definir, por meio do parâmetro NO_NOTIFY, que determinado atributo não notifique as suas premissas. Esta funcionalidade é útil em casos nos quais a alteração do estado de um atributo não implica diretamente a aprovação ou desaprovação de regras. Outro parâmetro possível é o RENOTIFY, cujo uso força a notificação das premissas, mesmo que o valor do atributo não tenha sido modificado. Esse recurso permite que uma regra seja reavaliada e, eventualmente, reativada repetidamente.

Ainda, a ferramenta *Wizard* CON criada por Lucca e Simão (2008) foi aprimorada e generalizada para o PON, sendo chamada com o nome de *Wizard* PON (VALENÇA, 2012). Sua principal contribuição é permitir a geração de códigos para o *Framework* PON C++ 2.0, a partir de uma interface gráfica. O trabalho de Valença (2012) apresenta, ainda, um processo de desenvolvimento de aplicações PON usando o *Wizard* PON. Esse processo diz respeito à fase de implementação e ocorre após a aplicação do método DON (WIECHETECK, 2011), no qual estão concentradas as fases de requisitos e projeto de aplicações para o PON. O método DON é apresentado na seção 2.1.5.

Outro aspecto em evolução nas pesquisas do PON é a plataforma computacional de execução. Os *frameworks* PON existentes, baseados em C++, Java, C#, compilam aplicações para execução em computadores de arquitetura *von Neumann*[17] (ou seja, computadores tradicionais como os PCs). Assim, em razão da sequencialização imposta pela arquitetura von Neumann, mesmo que o cálculo assintótico para o PON apresente uma baixa complexidade computacional, certas aplicações PON na prática não alcançam esse resultado (LINHARES *et al.*, 2011).

Ferreira, Ronszcka e outros (FERREIRA *et al.*, 2013) iniciaram o desenvolvimento de uma linguagem e compilador específicos para o PON em uma

[17] Arquitetura von Neumann é caracteriza pela possibilidade de uma máquina digital armazenar seus programas no mesmo espaço de memória que os dados, podendo assim manipular tais programas. A divisão da arquitetura em CPU e memória criou o que é chamado de gargalo de von Neumann pois a comunicação entre esses dois elementos é mais lenta do que eles podem processar/armazenar (Backus, 1978).

disciplina do Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial (CPGEI). Esses esforços foram evoluídos com as pesquisas de Ferreira (2015) para criar uma versão do compilador para a linguagem do PON (LingPON), a luz das pesquisa de Ronszcka (2019). Antes desses esforços era possível programar para o PON em computadores usuais (i.e., computadores baseados na arquitetura de von Neumann) somente por meio de *frameworks* sobre outras linguagens, assim, o desempenho esperado do PON, sobretudo se considerado o seu cálculo assintótico, era sacrificado na compilação de plataformas não otimizadas ao PON.

O compilador desenvolvido por Ferreira (2015) tem como entrada um programa desenvolvido em LingPON e pode gerar como saída, mediante prévia configuração, código para o *framework* PON, código em C++ ou código em C. Os códigos deverão ser compilados por um compilador tradicional dessas linguagens para se criar o arquivo executável. Os testes comparativos realizados por Ferreira (2015) mostraram que o código C apresentou melhor desempenho. O autor atribui esse resultado pelo fato do C possuir menos estruturas para dar suporte à orientação a objetos do C++. Ainda sobre o trabalho de Ferreira (2015), a suposta facilidade de programação no PON não podia ser explorada, pois o uso de *frameworks* sobre outras linguagens obriga o desenvolvedor a usar a sintaxe para o compilador da linguagem sobre o qual o *framework* PON foi desenvolvido. Para ilustrar a facilidade de programação com o uso do LingPON, os códigos 4 e 5 ilustram a declaração de um FBE no *Framework* PON C++ 1.0 e em LingPON.

Código 4 - Exemplo de código para o *Framework* PON C++ 1.0

```
#ifndef Archer_H_
#define Archer_H_
#include "framework/utils/SingleInclude.h"
class Archer : public FBE {
public:
          Archer();
          ~Archer();
          //Atributes
          Boolean * atHasFired;
          Integer * atCount;
          Method * mtFire1; //methods
          Method * mtFire2;
          Method * mtFire3;
          Method * mtFire4;
};
#endif /* Archer_H_ */

#include "Archer.h"
Archer::Archer(void) {
          BOOLEAN(this,atHasFired,false);
          INTEGER(this,atCount,0);
          METHOD(this,mtFire1,atHasFired,true,0);
          METHOD_OPERATION(this,mtFire2,atCount,
                              new Addition(atCount,1),0);
          METHOD_OPERATION(this,mtFire3,atCount,
                              new Addition(atCount,atCount),0);
          METHOD(this,mtFire4,,,0);
}
Archer::~Archer(void) {
}
```

Fonte: (Ferreira, 2015)

Código 5 - Exemplo de código em LingPON

```
fbe Archer
    attributes
          boolean atHasFired false
          integer atCount 0
    end_attributes
    methods
          method mtFire1(atHasFired = true)
          method mtFire2(atCount = atCount + 1)
          method mtFire3(atCount = atCount + atCount)
          method mtFire4()  begin_method //código C/C++
                              end_method
    end_methods
end_fbe
```

Fonte: (Ferreira, 2015)

Na recente pesquisa sobre o PON foram desenvolvidas diversas implementações em software e hardware com o objetivo de evoluir o paradigma e verificar suas propriedades conceituais. Contudo, Ronszcka (2019) identificou diversos problemas decorrentes da falta de padronização nessas implementações e propôs o Método de Compilação para o PON (MCPON). Nesse método, foi concebido um Grafo PON que é um elemento balizador para criação padronizada de materializações PON. Assim, segundo o estudo, é possível garantir as características do paradigma e a compatibilidade entre as diversas materializações criadas.

Alguns trabalhos de pesquisa (JASINSKI, 2012; PETERS, 2012; WITT *et al.*, 2011) atuaram na criação de *hardware*, baseado em FPGA[18], para implementar parte, ou toda, a cadeia de notificações do PON desvinculada da arquitetura *von Neumann*.

Witt, Linhares e outros (WITT *et al.,* 2011) propuseram um conjunto de circuitos combinacionais que implementam (FPGA) os elementos do metamodelo de notificações em hardware. O desempenho nesta abordagem tende a ser muito superior ao desempenho das implementações em software que dependem de buscas em estruturas de dados (LINHARES, 2015).

Ademais e particularmente, as aplicações podem efetivamente fazer uso de execução paralela. Porém, a solução apresentou restrições quanto à resolução de conflitos, definição de prioridades, suporte a instigações em série (limitado a instigações em paralelo) e os métodos têm funções limitadas (somente comparação de atributos com constantes). Não obstante, Jasinski (2012) propôs um formato de descrição para aplicações PON em alto nível, em XML, que pode ser traduzido e sintetizado para executar em um *hardware* baseado em FPGA, a partir do trabalho de Witt, Linhares e outros (WITT *et al.*, 2011).

Isto posto, esta tecnologia chamada de PON-HD prototipal se demonstrou com efetivo potencial a ser explorado subsequentemente. Por outro lado, a reconfiguração do *hardware,* que deve ser feita sempre que houver uma modificação na lógica da aplicação PON, é mais trabalhosa do que seria uma alteração no software da aplicação PON. Além disso, as aplicações são restritas ao número de

[18] FPGA (*Field Programmable Gate Array*), ou *hardware* configurável, é um circuito integrado projetado para ser configurado por um consumidor ou projetista após a fabricação.

elementos que o FPGA suporta e deve-se considerar, ainda, o alto custo desta tecnologia[19].

No intuito de amenizar, um tanto, idiossincrasias do PON-HD prototipal, Peters (2012), baseado nos trabalhos de Jasinski (2012) e Witt, Linhares e outros (WITT *et al.*, 2011), desenvolveu um coprocessador PON (CoPON) que realiza parte do ciclo de inferência do paradigma, atuando especificamente nos *Attributes*, *Premises*, *Conditions* e *Rules*. A execução dos métodos é responsabilidade do processador ao qual o coprocessador está conectado, via *Framework* PON C++ 1.0. Os resultados apresentados nesse trabalho mostram um ganho de desempenho entre 23 e 45 vezes, se comparado às aplicações desenvolvidas puramente no *Framework* PON C++ 1.0. Entretanto, o CoPON apresenta limitações no que diz respeito à escalabilidade e flexibilidade.

Mais recentemente, Kerschbaumer (2018) desenvolveu uma nova implementação para o PON em *hardware*, chamada Tecnologia PON-HD 1.0, inspirada no PON-HD prototipal mas distinta, na qual todos os elementos do paradigma foram modelados usando lógica reconfigurável, na linguagem VHDL. Ainda, a partir de uma linguagem de alto nível chamada LingPON HD 1.0, também desenvolvida no trabalho a luz do MCPON, é possível criar soluções que são transformadas para VHDL e, na sequência podem ser executadas em plataformas de *Field Programmable Gate Arrays* (FPGA). O objetivo do trabalho foi aproveitar a facilidade de programação no PON em alto nível, orientado a regras, com o potencial de execução paralela implícita da tecnologia FPGA. Mesmo com este objetivo alcançado, ainda a questão limitação de uma aplicação no FPGA persistiu.

Linhares, Simão e Stadzisz (2015) propuseram uma arquitetura de *hardware* específica para o PON, chamada de *Notification-Oriented Computer Architecture* (NOCA). Esta arquitetura inclui um conjunto de microprocessadores organizados em classes, cada uma com um propósito específico para tratar a cadeia de notificações nas aplicações PON. A arquitetura é programável e a aplicação PON é executada de forma concorrente entre os microprocessadores. Assim, a NOCA representa uma plataforma de execução mais próxima do ideal para o Paradigma Orientado a

[19] O trabalho de doutorado de Ricardo Kerschbaumer estudou e evoluíu o PON em hardware e parte desde esforços foram publicados no Congresso Brasileiro de Inteligência Computacional (Kerschbaumer *et al.*, 2015).

Notificações, ou seja, que oferece um ambiente concorrente, reprogramável e orientado a notificações.

Os trabalhos de pesquisa sobre a NOCA avançam em busca de maior escalabilidade e de ambientes de simulação visando a validação dos circuitos de implementação da arquitetura. Em relação a simuladores, Pordeus (2017) implementou o NOCASim em seu trabalho de mestrado. O atual custo elevado das plataformas de FPGA (com muitas unidades de processamento) são impeditivos para sua popularização e impediam, até então, que se verificassem o porquê de algumas desvantagens apresentadas pela NOCA, em especial a baixa escala de paralelização. Assim, o simulador criado no trabalho de pesquisa de Pordeus pôde demonstrar por meio de simulações que à medida em que o número de unidades de processamento aumenta, a NOCA também apresenta melhores desempenhos (LINHARES *et al.*, 2020).

Outrossim, Schütz e outros (SCHÜTZ, 2019; SCHÜTZ *et al.*, 2018) investigaram o potencial baixo acoplamento e execução paralela do PON para aplicação em Redes Neurais Artificiais (RNAs). Os autores apresentam uma implementação para RNA em PON, chamada NeuroPON, que permite, de forma transparente a partir de programação em alto nível, a geração paralela de código para plataformas *multicore*. Com a pesquisa os autores obtiveram resultados demonstrando que a NeuroPON apresenta melhores níveis de desacoplamento e uso explícito de elementos lógico-causais, que são, respectivamente, úteis para distribuição, compreensão e aprimoramento de aplicações em RNAs, quando comparadas com implementações equivalentes em outras plataformas.

No âmbito da Engenharia de Software, Wiecheteck e outros (SIMÃO; WIECHETECK; STADZISZ, 2015) (WIECHETECK; STADZISZ; SIMÃO, 2011) (WIECHETECK, 2011) propuseram o primeiro método de desenvolvimento de software para o PON, que será detalhado na próxima seção. Batista (2013) investigou a aplicabilidade da Teoria de Projeto Axiomático (PA) no Paradigma Orientado a Notificações (PON) e ao Paradigma Orientado a Regras (POR). O resultado da pesquisa foi a definição de um método, chamado PA-PON-POR, que quando usado auxilia “no processo de criação de Regras e de sistemas PON-POR com alguma garantia de qualidade” (BATISTA, 2013, pg. 7).

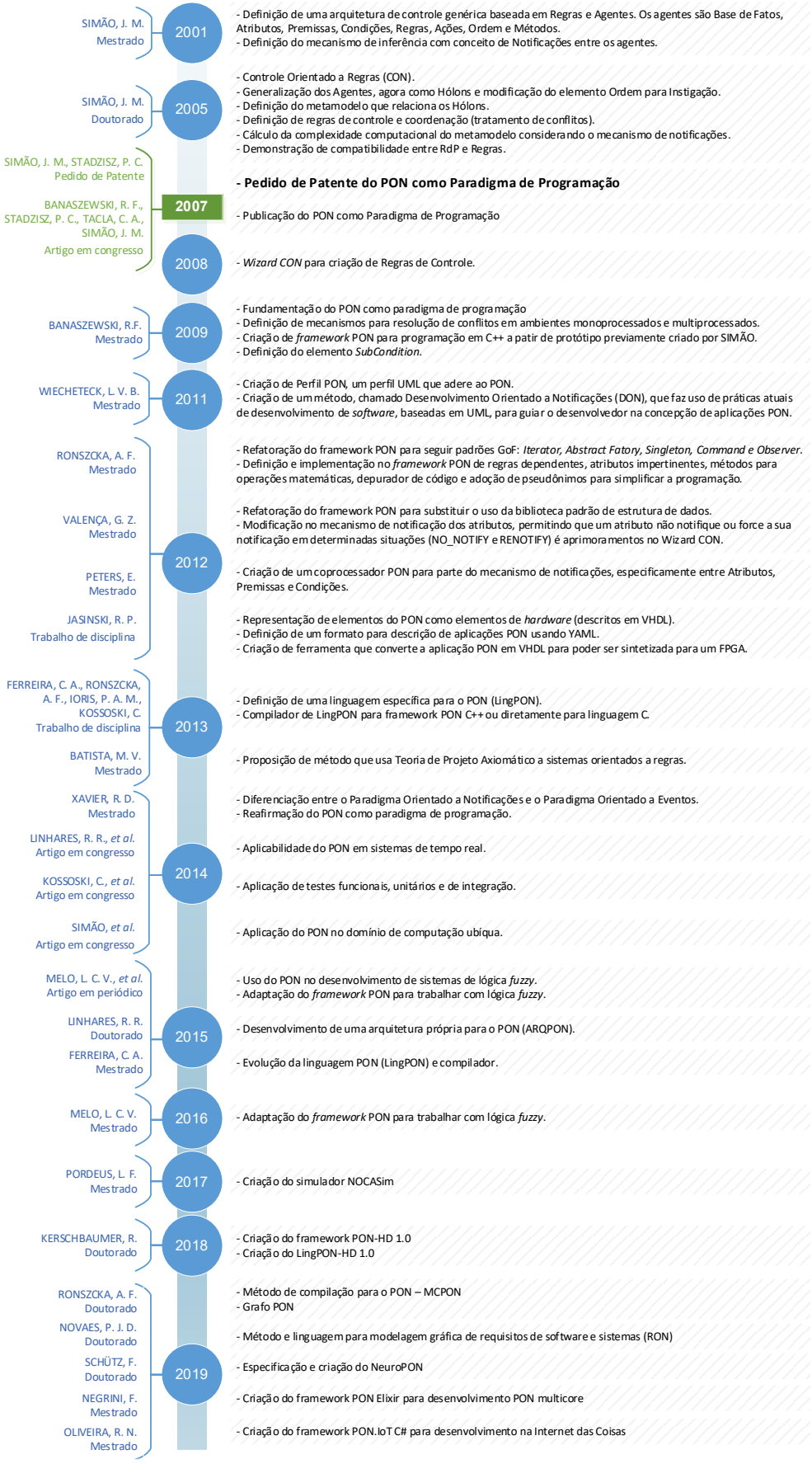
Outros estudos no PON, como o de Melo, Simão e Fabro (2015) estenderam o *framework* PON para permitir o desenvolvimento de sistemas usando lógica

fuzzy[20]. Kossoski, Simão e Stadzisz (2014) (KOSSOSKI, 2015) introduziram a temática de testes funcionais no PON e, para isso, definiram um novo diagrama de fluxo de notificações entre os elementos do PON. Negrini (2019) criou o *Framework* PON Elixir para execução de software PON em processadores *multicore*. Oliveira (2019) criou o *framework* PON.IoT C#, que permite a execução do PON em microcomputadores Raspberry Pi para aplicações na Internet da Coisas (IoT – *Internet of Things*).

A Figura 19 apresenta uma ilustração da evolução do Paradigma Orientado a Notificações em uma linha do tempo. Essa figura pretende ilustrar em que ponto do tempo cada contribuição no estado da arte e da técnica do PON ocorreu por meio de publicações.

[20] Lógica fuzzy, ou lógica difusa é uma extensão da lógica booleana que admite valores lógicos intermediários entre o FALSO (0) e o VERDADEIRO (1); por exemplo, o valor médio 'TALVEZ' (0,5). Isto significa que um valor lógico difuso é um valor qualquer no intervalo de valores entre 0 e 1.

Figura 19 - Linha do tempo do Paradigma Orientado a Notificações



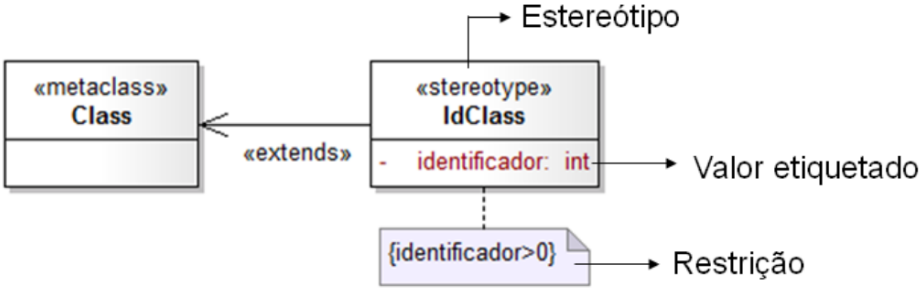
Fonte: Autoria própria

2.1.5 Desenvolvimento Orientado a Notificações (DON)

O Desenvolvimento Orientado a Notificações (DON) foi a primeira iniciativa do grupo de pesquisa em PON para definir um método de Engenharia de Software que pudesse guiar o desenvolvimento de aplicações usando o paradigma proposto. Desenvolvido no trabalho de mestrado de Luciana Vilas Boas Wiecheteck (WIECHETECK, 2011), o DON envolveu (i) estender diagramas da UML para representar os conceitos do PON e (ii) definir um método que fizesse uso dessa extensão em uma sequência de passos conduzindo o desenvolvedor na criação de aplicações PON. Inicialmente, criou-se um perfil UML, denominado *NOP Profile* (Perfil PON), que adequa a UML ao PON. Este perfil fundamentou-se nos conceitos do paradigma e, principalmente, nos modelos de classes empregados na implementação do *framework* PON 1.0.

Um perfil UML, como usado no DON, permite que o metamodelo UML seja refinado para um domínio específico de aplicações. Assim, é possível determinar uma nova sintaxe e semântica aos elementos existentes na UML por meio de estereótipos, valores etiquetados e restrições, conforme ilustra a Figura 20. Os detalhes do perfil PON estão disponíveis na dissertação de Wiecheteck (2011) e resumidos no trabalho de Wiecheteck, Stadzisz e Simão (2011).

Figura 20 - Mecanismos de extensão em um perfil UML

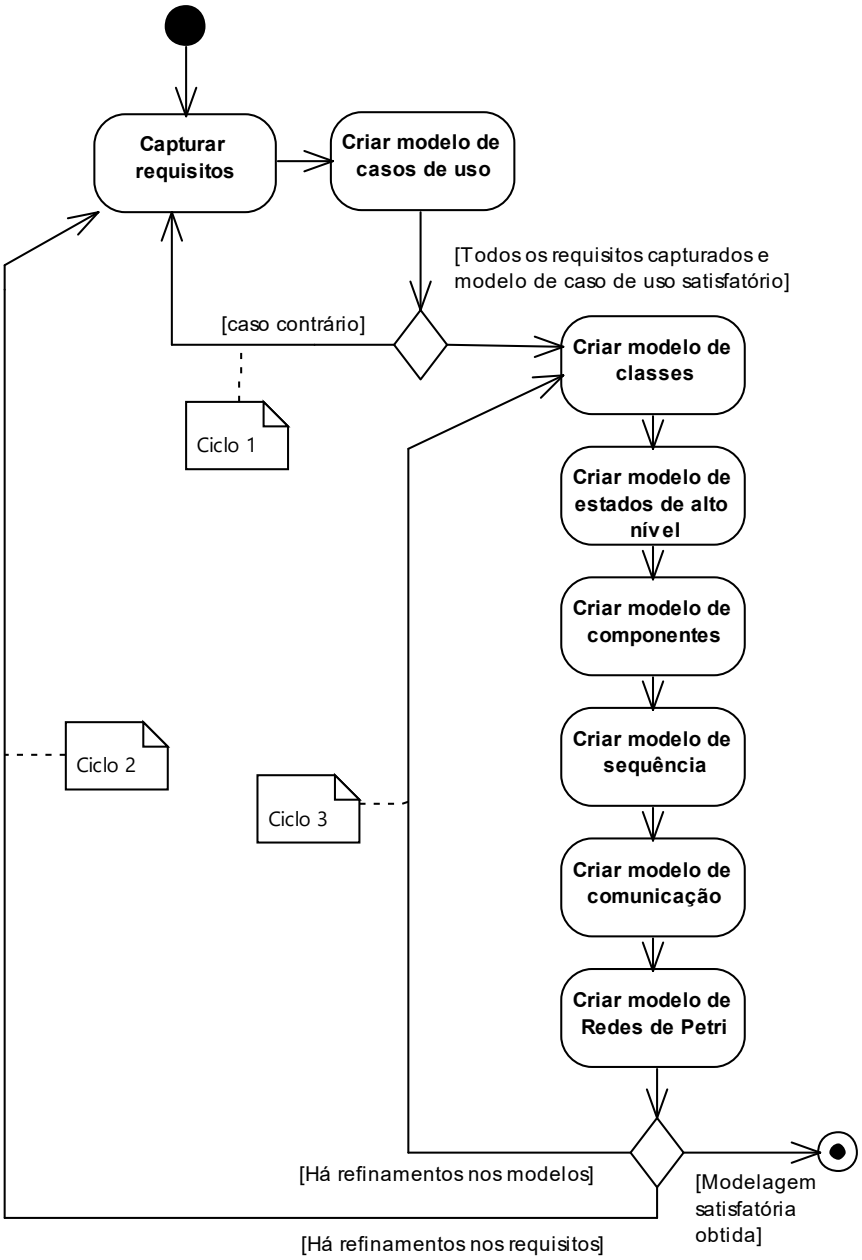


Fonte: (WIECHETECK; STADZISZ; SIMÃO, 2011)

O método denominado DON abrange atividades das fases de Requisitos e Análise e Projeto para sistemas baseados no PON. Todavia, os autores afirmam que o método é aderente aos processos de software tradicionais e deve ser usado em conjunto com eles. O DON é organizado em oito passos, cujos dois primeiros são usados na captura de requisitos do sistema e os outros seis são focados na criação de modelos para representar o sistema. Os modelos desenvolvidos no método são: modelo de classes, modelo de estados de alto nível, modelo de componentes,

modelo de sequência, modelo de comunicação e modelo de Redes de Petri (WIECHETECK, 2011). O diagrama de atividades da Figura 21 ilustra os oito passos do método DON.

Figura 21 - Atividades do método DON



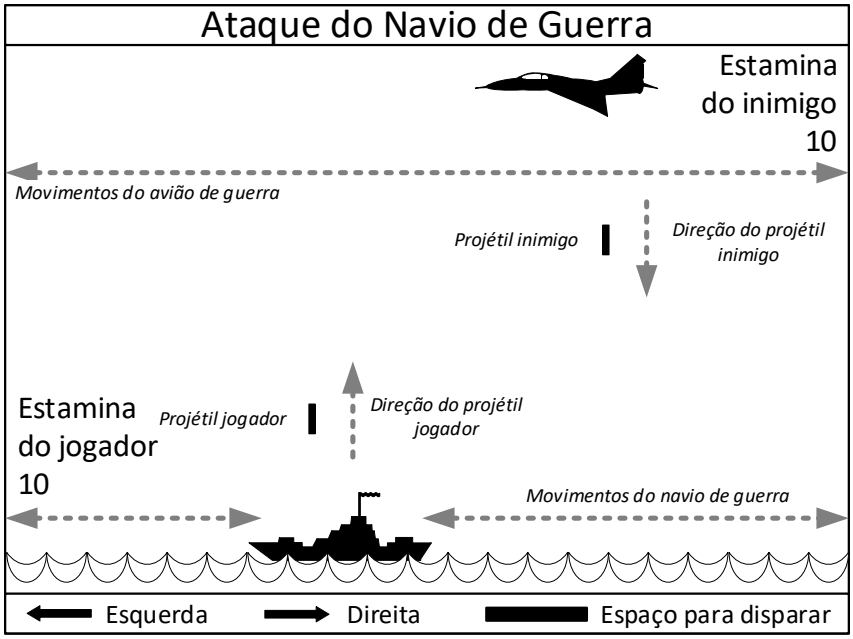
Fonte: Autoria própria

O método é dividido em três ciclos básicos. O primeiro compreende as atividades de levantamento de requisitos do software e é finalizado quando todos os requisitos foram capturados e tem-se um modelo de casos de uso satisfatório. O segundo ciclo cria as primeiras versões dos diagramas e retorna aos requisitos enquanto for necessário fazer refinamentos. Quando não houver mais refinamentos, o terceiro ciclo se concentra em finalizar os modelos até que os requisitos do

software sejam atendidos. A codificação poderá ocorrer em ciclos intermediários, se o processo de software usado for iterativo e incremental, ou poderá iniciar ao final da aplicação do DON, caso seja baseado no modelo sequencial.

Os passos do método serão explicados a seguir e, como exemplo, será usada a modelagem de um jogo hipotético, chamado “Ataque do navio de guerra”, em que o jogador controla um navio de guerra atirando contra um avião de guerra inimigo, cuja Figura 22 ilustra o esboço.

Figura 22 - Esboço do jogo “Ataque do navio de guerra”.



Fonte: (MENDONÇA *et al.*, 2015)

2.1.5.1 Capturar requisitos e Criar modelo de casos de uso

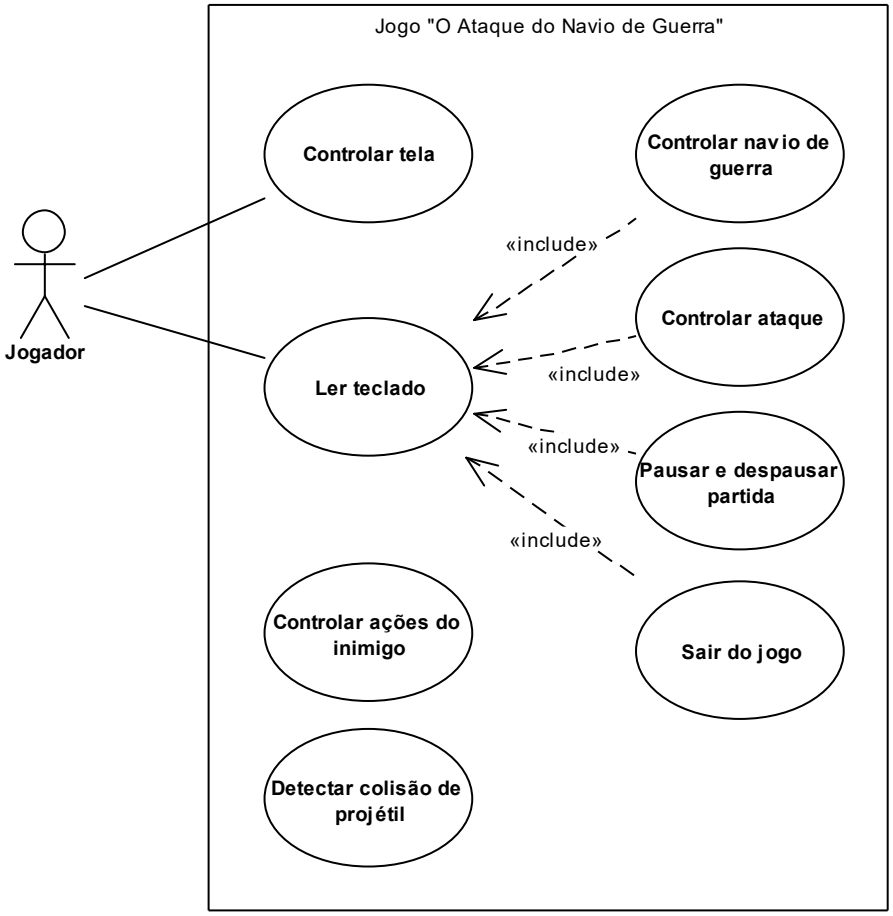
As duas primeiras atividades do método DON não diferem de um processo usual de desenvolvimento de software. Nessas atividades, desenvolvidas inicialmente no ciclo 1, são levantados os requisitos para o software, criando uma lista de requisitos e o modelo de casos de uso, seguindo o diagrama usual da UML. O Quadro 2 ilustra o artefato criado na atividade *Capturar requisitos* e a Figura 23 ilustra o modelo de casos de uso. Deve existir ainda, para o modelo de casos de uso, a descrição de cada caso de uso, bem como qual requisito ele atende (Quadro 3).

Quadro 2 - Exemplo de artefato criado no passo *Capturar requisitos*

#	Requisitos funcionais
RF1	O sistema deve mover o navio de guerra para a esquerda quando a seta para a esquerda for pressionada.
RF2	O sistema deve mover o navio de guerra para a direita quando a seta para a direita for pressionada.
RF3	O sistema deve fazer o navio de guerra atirar quando o botão de disparo for pressionado.
RF4	O sistema deve mover o inimigo (avião de guerra) para a esquerda e depois para a direita.
RF5	O sistema deve fazer o inimigo atirar um projétil a cada intervalo de dois segundos.
RF6	O sistema deve diminuir a estamina do inimigo quando ele for atingido pelo projétil do jogador.
RF7	O sistema deve diminuir a estamina do jogador quando ele for atingido pelo projétil do inimigo.
RF8	O sistema deve explodir o navio de guerra quando a estamina do jogador acabar e mostrar a mensagem “Você perdeu!”, também deverá mostrar uma opção para voltar ao menu inicial.
RF9	O sistema deve explodir o avião de guerra quando a estamina do inimigo acabar e mostrar a mensagem “Você ganhou!”, também deverá mostrar uma opção para voltar ao menu inicial.
RF10	O sistema deve permitir que o jogador pause o jogo e volte a jogar.
RF11	O sistema deve sair do jogo quando o botão de sair for pressionado.
RF12	O sistema deve mostrar continuamente o navio de guerra.
RF13	O sistema deve mostrar continuamente o avião de guerra inimigo.
RF14	O sistema deve mostrar a quantidade de estamina do jogador.
RF15	O sistema deve mostrar a quantidade de estamina do avião de guerra.
RF16	O sistema deve apresentar um menu principal no início do jogo e quando encerrar uma partida.
#	**Requisitos não funcionais**
RNF1	O menu principal deverá conter duas opções: *Iniciar um novo jogo* e *Sair.* (ref. RF16)
RNF2	O navio de guerra deverá ser mostrado na parte inferior da tela. (ref. RF12)
RNF3	O avião de guerra deverá ser mostrado na parte superior da tela. (ref. RF13)
RNF4	O sistema deve limitar a movimentação do navio de guerra somente na parte visível do jogo. (ref. RF1) (ref. RF2)
RNF5	A tela do sistema deve ter o tamanho de 800x600 pixels.

Fonte: Autoria própria

Figura 23 - Modelo de casos de uso



Fonte: Autoria própria

Quadro 3 - Descrição dos casos de uso de relação com os requisitos do jogo

Caso de uso: descrição	**RF e RNF relacionado**
Controlar tela: atualiza a tela para mostrar os objetos em suas posições.	RF12, RF13, FR14, RF15, RNF2, RNF3, RNF4, RNF5
Ler teclado: os comandos do jogador são capturados e interpretados neste caso de uso.	RF1, RF2, RF3, RF10, RF11
Controlar navio de guerra: define o controle do jogador sobre o navio de Guerra, incluindo as movimentações para a esquerda e para a direita.	RF1, RF2
Controle de ataque: define o ataque do jogador contra o avião de guerra inimigo.	RF3
Pausar e despausar partida: responsável por pausar e despausar uma partida.	RF10
Sair do jogo: responsável por sair do jogo.	RF11
Controlar ações do inimigo: responsável por controlar os movimentos do inimigo para a esquerda e para a direita e de controlar os tiros do inimigo.	RF4, RF5
Detectar colisão de projétil: responsável por detectar colisão de projétil contra navio ou avião de guerra e diminuir a estamina de quem foi atingido.	RF6, RF7

Fonte: Autoria própria

2.1.5.2 Criar modelo de classes

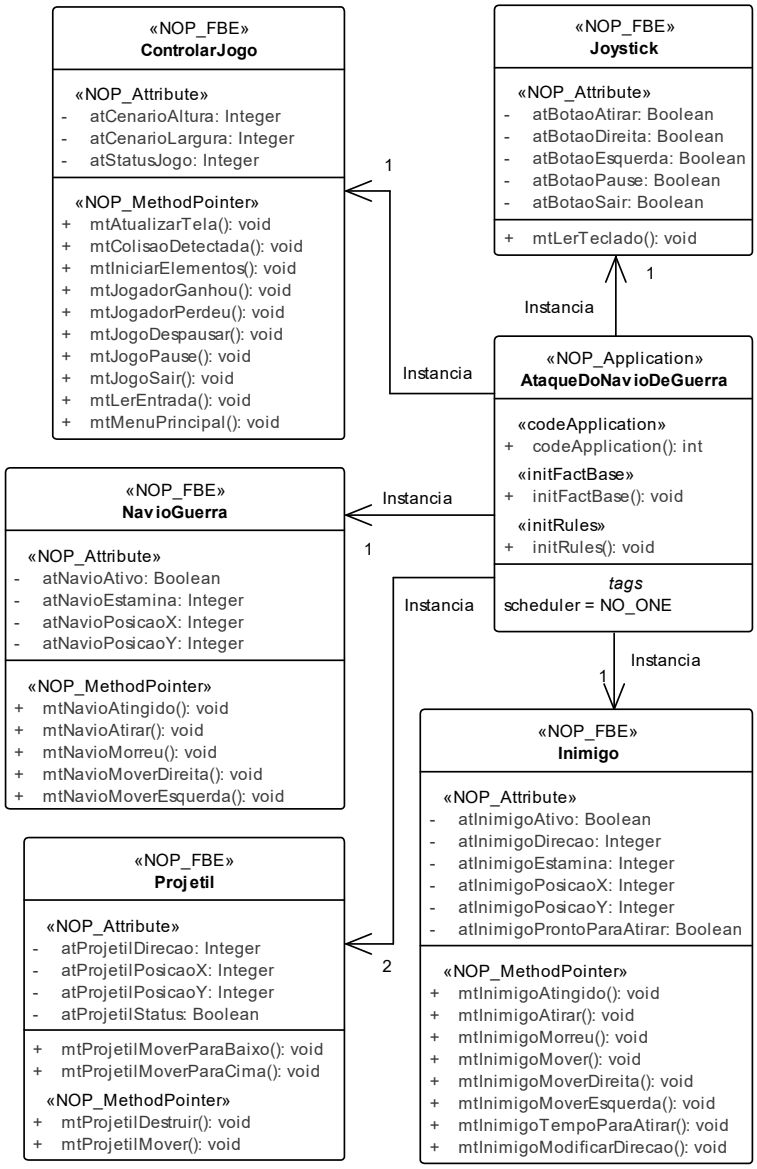
O modelo de classes é criado no ciclo 2 e é refinado nesse mesmo ciclo e no ciclo 3 do DON. O primeiro uso do perfil PON se faz neste modelo de classes. O modelo de classes é o usual da UML, porém o uso de estereótipos auxilia a identificar os elementos do metamodelo do PON no modelo de classes da aplicação. Os *Attributes* e *Methods* do *framework* PON, instanciados para a aplicação serão descritos como atributos e métodos das classes *FBE*. A Figura 24 ilustra o modelo de classes criado para o exemplo do jogo.

A primeira classe criada nesse modelo representa a aplicação PON (Figura 24, elemento estereotipado com <<*NOP_Application*>>). Essa classe é responsável por instanciar os objetos *FBEs* definidos pelas outras classes. As classes *FBEs* são criadas pelo projetista (Figura 24, elementos estereotipados com <<*NOP_FBE*>>) e incluem os *Attributes* e *Methods* do *FBE*, conforme indicado pelos estereótipos <<*NOP_Attribute*>> e <<*NOP_MethodPointer*>>.

O *FBE Joystick* é responsável por capturar as ações do jogador e registrá-las em atributos que produzirão notificações às regras de processamento do jogo. Os *FBEs NavioGuerra*, *Inimigo* e *Projetil* constituem os objetos do jogo que são visíveis ao usuário. O *FBE ControlarJogo* atualiza os elementos na tela e é responsável por mecanismos adicionais de controle do jogo, como a apresentação do menu, pausa e saída do jogo.

Os *FBEs* e seus atributos fornecem uma visão geral dos elementos do jogo. Os métodos mostram as ações que são executadas pelos *FBEs*. Esses elementos são identificados à medida que as iterações de refinamento do DON ocorrem. Por exemplo, o caso de uso “*Pausar e despausar partida”* estabelece que o jogo pode estar “pausado” ou “jogando”. Desta forma, o *FBE ControlarJogo* deverá ter um atributo para representar esses estados. Um elemento se movimentando na tela necessita atributos para indicar a sua posição, e assim por diante.

Figura 24 - Modelo de classes do jogo



Fonte: Autoria própria

2.1.5.3 Modelo de estados de alto nível

No passo de construção do “modelo de estados de alto nível” do método DON, a lógica de mais alto nível do sistema é definida. Em PON, o encadeamento lógico da aplicação é definido pelas regras que a compõem. O modelo de estados auxilia a identificação das regras necessárias.

O modelo de estados de alto nível criado neste passo será construído na forma de um diagrama de atividades ou um diagrama de estados, ambos da UML. O primeiro passo é criar um diagrama para cada classe de *FBE* identificado. Cada

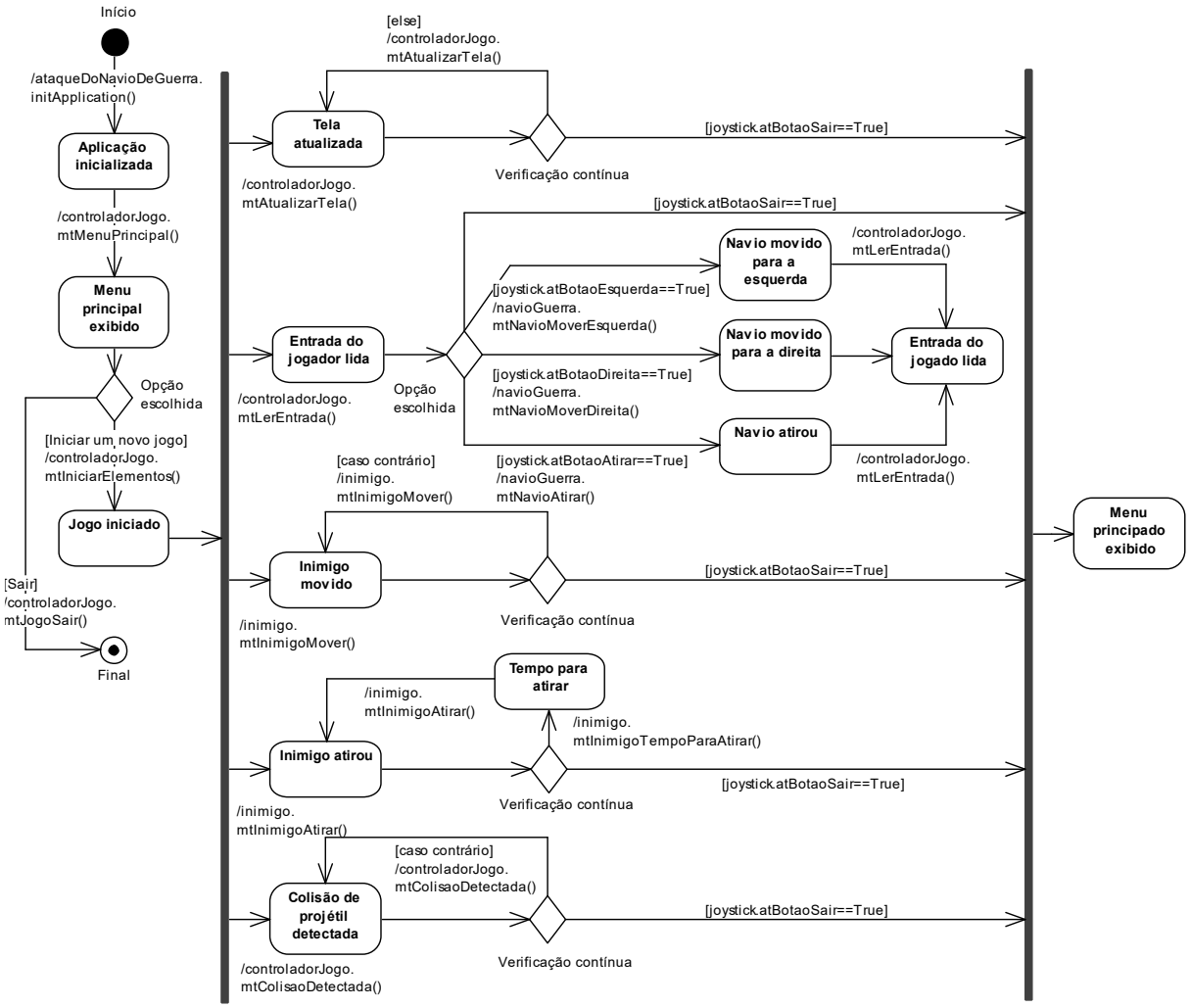
diagrama irá mostrar as mudanças de estados ou fluxo de atividades no âmbito de cada classe. Essas mudanças ou fluxos podem ser vistas como regras do sistema. Por exemplo, quando o *Method mtInimigoMover* do *FBE Inimigo* for executado, ele irá alterar o atributo que indica a posição da aeronave do inimigo no jogo. Isto corresponderá a uma ação de alguma regra da aplicação.

A próxima etapa é agrupar os diagramas, resultando no modelo de estados de alto nível, que facilita a visualização da interação e relação entre os estados dos modelos das classes, além de fornecer uma visão geral do sistema. A Figura 25 ilustra o modelo de estados de alto nível após algumas iterações do DON. Esse modelo pode ser melhorado com a inclusão de mais detalhes, porém este nível é apropriado para ilustrar alguns conceitos. Nesse modelo, após o passo de inicialização, o software permite iniciar uma nova partida do jogo ou ser fechado. Caso inicie uma nova partida, cinco atividades ocorrerão simultaneamente. Uma delas mostra os elementos do jogo na tela e leva o jogo ao estado de *Tela atualizada*. Esse estado é atualizado constantemente enquanto o botão de sair não for pressionado. Outra atividade executada é a leitura da entrada do jogador, que leva ao estado de *Entrada do jogador lida*. Após esse estado, o software irá executar a operação solicitada pelo usuário e retornar à leitura de entrada do jogador. As outras três atividades são similares à primeira.

O modelo de estados de alto nível, permite ainda observar as interrelações entre objetos nos diferentes fluxos da aplicação. Por exemplo, mudar a posição do Navio de guerra depende dos objetos das classes *Joystick*, *NavioGuerra* e *ControlarJogo*. A classe *Joystick* age capturando a entrada do jogador (*joystick.mtLerEntrada*), a classe *NavioGuerra* age modificando a posição do Navio de guerra (*navioGuerra.mtNavioMoverEsquerda*) e a classe *ControlarJogo* age atualizando a tela para mostrar o Navio de guerra em sua nova posição (*controladorJogo.mtAtualizarTela*).

O modelo criado neste passo mostra a sequência de estados envolvida na execução dos casos de uso. O caso de uso *Controlar ataque*, por exemplo, pode ser visto seguindo os estados: *Aplicação inicializada*, *Menu principal exibido*, *Jogo iniciado*, *Entrada do jogador lida* e *Navio atirou*.

Figura 25 - Modelo de estados de alto nível



Fonte: Autoria própria

O modelo de estados de alto nível auxilia na descoberta de regras do sistema. Porém, como no processo de identificação de classes no Paradigma Orientado a Objetos, a descoberta de regras é também uma atividade de síntese realizada pelo projetista. Então, são usados os elementos *<<NOP_FBE>>* do modelo de classes e o modelo de estados de alto nível como instrumentos de ajuda.

Uma vez que uma regra é identificada, deve-se responder algumas questões para auxiliar no processo:

1) Qual o objetivo da Regra?
2) O que precisa acontecer para a Regra ser executada?
3) O que acontece se a Regra for executada?

No exemplo considerado, o processo de análise dos modelos de classes e de estados de alto nível identificou 19 regras. O Quadro 4 ilustra essas regras com as respostas para as três questões apresentadas.

Quadro 4 - Regras identificadas para o exemplo estudado

#	Qual o objetivo da Regra?	O que precisa acontecer para a Regra ser executada?	O que acontece se a Regra for executada?
1	Mostrar os objetos em suas posições	O status do jogo deve ser JOGANDO.	Os objetos são posicionados na tela nas posições indicadas em seus atributos.
2	Mover avião de guerra	O status do jogo deve ser JOGANDO, o inimigo deve estar vivo e a posição dele deve estar na área visível do jogo.	A posição do inimigo deve ser movida para a direita ou para a esquerda mantendo a direção até o final da área visível.
3	Fazer o avião de guerra atirar	O status do jogo deve ser JOGANDO, a estamina do inimigo deve ser maior do que zero, não deve haver projétil do inimigo na área visível e a posição do inimigo deve estar na área visível do jogo.	O projétil do inimigo deve ser direcionado para baixo a partir da posição do avião de guerra.
4	Modificar direção de voo do avião de guerra	O status do jogo deve ser JOGANDO, a estamina do inimigo deve ser maior do que zero e a posição do inimigo deve estar fora da área visível do jogo.	A direção do inimigo deve ser invertida.
5	Mover projétil do inimigo	O status do jogo deve ser JOGANDO, o status do projétil do inimigo deve ser ativo.	O projétil do inimigo deve ser movido para baixo.
6	Detectar colisão de projétil contra o avião de guerra	O status do jogo deve ser JOGANDO, a estamina do inimigo deve ser maior do que zero, o projétil do jogador deve estar na mesma posição do avião de guerra.	A estamina do inimigo deve ser diminuída.
7	Detectar colisão de projétil contra o navio de guerra	O status do jogo deve ser JOGANDO, a estamina do jogador deve ser maior do que zero, o projétil do inimigo deve estar na mesma posição do navio de guerra.	A estamina do jogador deve ser diminuída.
8	Mover navio de guerra para a esquerda	O status do jogo deve ser JOGANDO, a estamina do jogador deve ser maior do que zero, o botão esquerda deve ter sido pressionado e a posição do navio de guerra não pode ser no canto esquerdo da tela do jogo.	A posição do navio de guerra deve ser movida para a esquerda.
9	Mover navio de guerra para a direita	O status do jogo deve ser JOGANDO, a estamina do jogador deve ser maior do que zero, o botão direita deve ter sido pressionado e a posição do navio de guerra não pode ser no canto direito da tela do jogo.	A posição do navio de guerra deve ser movida para a direita.
10	Fazer navio de guerra atirar	O status do jogo deve ser JOGANDO, a estamina do jogador deve ser maior do que zero, o botão de atirar deve ter sido pressionado e o último tiro não pode ter ocorrido há menos de um segundo.	Um projétil deve ser criado e direcionado para cima.
11	Pausar jogo	O status do jogo deve ser JOGANDO e o botão pause deve ter sido pressionado.	O status do jogo deve ser modificado para PAUSADO.
12	Despausar jogo	O status do jogo deve ser PAUSADO e o botão pause deve ter sido pressionado.	O status do jogo deve ser modificado para JOGANDO.
13	Sair do jogo	O jogador deve ter pressionado o botão sair.	O status do jogo deve ser modificado para PARADO.

#	Qual o objetivo da Regra?	O que precisa acontecer para a Regra ser executada?	O que acontece se a Regra for executada?
14	Jogador ganha	O status do jogo deve ser JOGANDO e a estamina do inimigo deve ser zero.	Uma mensagem deve ser mostrada ao jogador e o status do jogo deve ser modificado para PARADO.
15	Jogador perde	O status do jogo deve ser JOGANDO e a estamina do jogador deve ser zero.	Uma mensagem deve ser mostrada ao jogador e o status do jogo deve ser modificado para PARADO.
16	Iniciar um novo jogo	O status do jogo deve ser PARADO.	Uma nova partida do jogo inicia.
17	Mover projétil do jogador	O status do jogo deve ser JOGANDO, o status do projétil do jogador deve ser ativo.	O projétil do jogador deve ser movido para cima.
18	Detectar colisão do projétil do inimigo contra a parede	O status do jogo deve ser JOGANDO, o status do projétil do inimigo deve ser ativo e a posição do projétil do inimigo deve ser igual à posição da parede.	O status do projétil do inimigo deve ser modificado para desativado, não sendo mais exibido na área visível.
19	Detectar colisão do projétil do jogador contra a parede	O status do jogo deve ser JOGANDO, o status do projétil do jogador deve ser ativo e a posição do projétil do jogador deve ser igual à posição da parede.	O status do projétil do jogador deve ser modificado para desativado, não sendo mais exibido na área visível.

Fonte: Autoria própria

2.1.5.4 Criar modelo de componentes

O diagrama de componentes é usado de forma particular no DON para modelar estaticamente as regras do software. As regras identificadas no passo anterior são declaradas neste passo, determinando seus nomes, premissas e instigações, estes dois últimos aplicando o perfil PON.

A criação do modelo de componentes é realizada em três etapas: (i) definir as regras, (ii) definir premissas e instigações e (iii) relacionar as regras aos *FBEs*.

2.1.5.4.1 Definir as regras

As regras identificadas anteriormente, recebem nomes, conforme exemplo exibido na Quadro 5.

Quadro 5 - Nomeação das regras do exemplo estudado

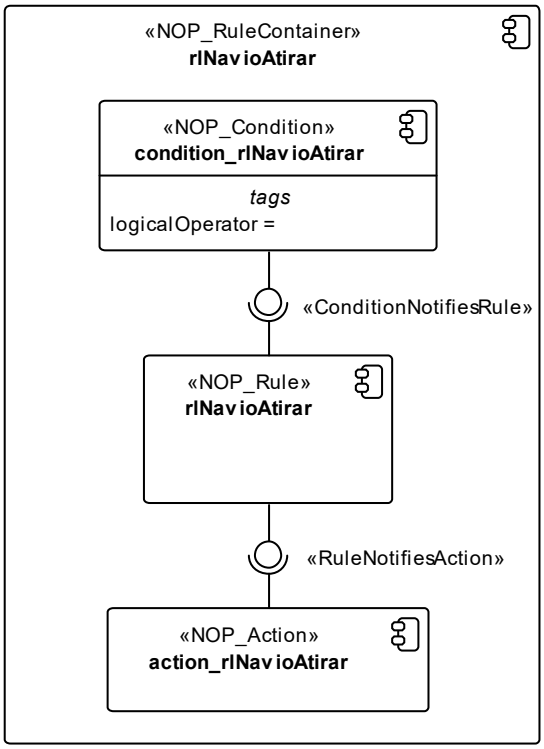
Regra	Nome
1	rlAtualizarPosicaoObjetos
2	rlInimigoMover
3	rlInimigoAtirar
4	rlInimigoModificarDirecao
5	rlInimigoMoverProjetil

Regra	Nome
6	rlDetectarColisaoContraInimigo
7	rlDetectarColisaoContraJogador
8	rlNavioMoverEsquerda
9	rlNavioMoverDireita
10	rlNavioAtirar
11	rlJogoPausar
12	rlJogoDespausar
13	rlJogoParar
14	rlJogadorGanha
15	rlJogadorPerde
16	rlIniciarNovoJogo
17	rlJogadorMoverProjetil
18	rlInimigoDetectarColisaoProjetilParede
19	rlJogadorDetectarColisaoProjetilParede

Fonte: Autoria própria

Para cada regra, devem ser criados, usando o perfil PON, três componentes estereotipados com *<<NOP_Condition>>, <<NOP_Rule>> e <<NOP_Action>>.* Os três componentes são, então, relacionados usando os estereótipos *<<ConditionNotifiesRule>> e <<RuleNotifiesAction>>*. Além disso, um componente estereotipado com *<<NOP_Rule>>* pode ser criado para conter os outros três, representando, assim, a regra como um todo. Para melhorar a semântica do elemento contentor, a criação de um novo estereótipo *<<NOP_RuleContainer>>* foi proposto em (MENDONÇA *et al.*, 2015), conforme ilustra a Figura 26. Esses estereótipos definidos no perfil PON, incluindo o novo estereótipo para o contêiner de regra, determinam a semântica e as restrições entre os elementos. Além disso, o perfil PON inclui, automaticamente, a *tag logicalOperator* no elemento *<<NOP_Condition>>*, que será definida posteriormente.

Figura 26 - Diagrama de componentes para a *Rule rlNavioAtirar*.



Fonte: Autoria própria

2.1.5.4.2 Definir premissas e instigações

As premissas e instigações são identificadas por meio da análise das regras estabelecidas anteriormente, dos atributos e métodos no modelo de classes e do comportamento expresso no modelo de estados de alto nível. O Quadro 6 exibe o resultado dessa análise para o exemplo estudado.

Quadro 6 - Premissas e instigações das regras do exemplo estudado

Regras / Premissas	Instigações
Regra 1 – rlAtualizarPosicaoObjetos	
controlarJogo.atStatusJogo==Jogando	controlarJogo.mtAtualizarTela()
Regra 2 – rlInimigoMover	
controlarJogo.atStatusJogo==Jogando & inimigo.atInimigoAtivo==True & inimigo.atInimigoPosicaoX>0 & inimigo.atInimigoPosicaoX<800	inimigo.mtInimigoMover()
Regra 3 – rlInimigoAtirar	
controlarJogo.atStatusJogo==Jogando & inimigo.atInimigoAtivo==True & inimigoProjetil.atProjetilStatus==Inativo &	inimigo.mtInimigoAtirar()

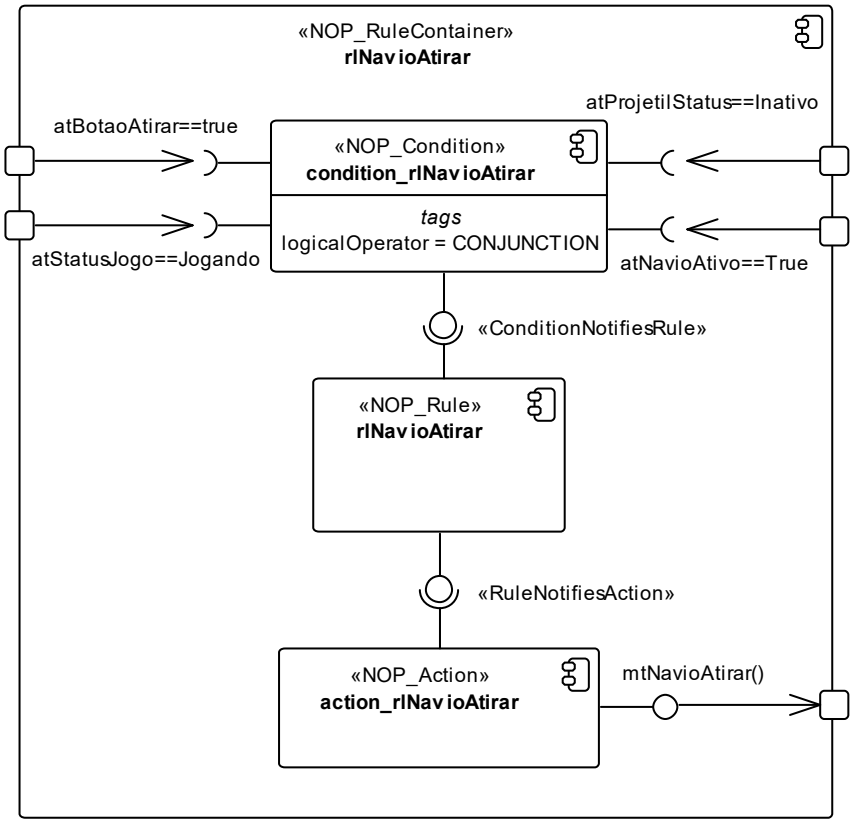
Regras / Premissas	Instigações
inimigo.atInimigoPosicaoX>0 & inimigo.atInimigoPosicaoX<800	
Regra 4 – rlInimigoModificarDirecao	
controlarJogo.atStatusJogo==Jogando & inimigo.atInimigoAtivo==True & ((inimigo.atInimigoPosicaoX==1 & inimigo.atInimigoDirecao==1) | (inimigo.atInimigoPosicaoX==800 & inimigo.atInimigoDirecao==2))	inimigo.mtInimigoModificarDirecao()
Regra 5 – rlInimigoMoverProjetil	
controlarJogo.atStatusJogo==Jogando & inimigoProjetil.atProjetilStatus==Ativo	inimigoProjetil.mtProjetilMover()
Regra 6 – rlDetectarColisaoContraInimigo	
controlarJogo.atStatusJogo==Jogando & inimigo.atInimigoPosicaoY==jogadorProjetil.atProjetilPosicaoY & inimigo.atInimigoPosicaoX==jogadorProjetil.atProjetilPosicaoX	inimigo.mtInimigoAtingido()
Regra 7 – rlDetectarColisaoContraJogador	
controlarJogo.atStatusJogo==Jogando & navioGuerra.atNavioPosicaoY==inimigoProjetil.atProjetilPosicaoY & navioGuerra.atNavioPosicaoX==inimigoProjetil.atProjetilPosicaoX	navioGuerra.mtNavioAtingido()
Regra 8 – rlNavioMoverEsquerda	
controlarJogo.atStatusJogo==Jogando & navioGuerra.atNavioAtivo==True & joystick.atBotaoEsquerda==True & navioGuerra.atNavioPosicaoX>1	navioGuerra.mtNavioMoverEsquerda()
Regra 9 – rlNavioMoverDireita	
controlarJogo.atStatusJogo==Jogando & navioGuerra.atNavioAtivo==True & joystick.atBotaoDireita==True & navioGuerra.atNavioPosicaoX<800	navioGuerra.mtNavioMoverDireita()
Regra 10 - rlNavioAtirar	
controlarJogo.atStatusJogo==Jogando & navioGuerra.atNavioAtivo==True & joystick.atBotaoAtirar==True & jogadorProjetil.atProjetilStatus==Inativo	navioGuerra.mtNavioAtirar()
Regra 11 – rlJogoPausar	
controlarJogo.atStatusJogo==Jogando & joystick.atBotaoPause==True	controlarJogo.mtJogoPause()
Regra 12 – rlJogoDespausar	
controlarJogo.atStatusJogo==Pausado & joystick.atBotaoPause==True	controlarJogo.mtJogoDespausar()
Regra 13 – rlJogoParar	
(controlarJogo.atStatusJogo==Pausado | controlarJogo.atStatusJogo==Jogando) & joystick.atBotaoSair==True	controlarJogo.mtJogoSair()
Regra 14 – rlJogadorGanha	
controlarJogo.atStatusJogo==Jogando &	controlarJogo.mtJogadorGanhou()

Regras / Premissas	Instigações
inimigo.atInimigoEstamina==0	
Regra 15 – rlJogadorPerde	
controlarJogo.atStatusJogo==Jogando & navioGuerra.atNavioEstamina==0	controlarJogo.mtJogadorPerdeu()
Regra 16 – rlIniciarNovoJogo	
controlarJogo.atStatusJogo==Stopped	controlarJogo.mtIniciarElementos()
Regra 17 – rlJogadorMoverProjetil	
controlarJogo.atStatusJogo==Jogando & jogadorProjetil.atProjetilStatus==Ativo	jogadorProjetil.mtProjetilMover()
Regra 18 – rlInimigoDetectarColisaoProjetilParede	
controlarJogo.atStatusJogo==Jogando & inimigoProjetil.atProjetilStatus==Ativo & inimigoProjetil.atProjetilPosicaoY==0	inimigoProjetil.mtProjetilDestruir()
Regra 19 – rlJogadorDetectarColisaoProjetilParede	
controlarJogo.atStatusJogo==Jogando & jogadorProjetil.atProjetilStatus==Ativo & jogadorProjetil.atProjetilPosicaoY==600	jogadorProjetil.mtProjetilDestruir()

Fonte: Autoria própria

Após a identificação das premissas e instigações, esses elementos podem ser representados nos diagramas de componentes. Em um nível maior de abstração, as premissas e instigações podem ser apresentadas como interfaces de componentes. Em um nível mais baixo de abstração, eles podem ser apresentados como componentes. Neste trabalho optou-se por usar níveis maiores de abstração, pois o diagrama torna-se mais conciso. A Figura 27 ilustra a *Rule rlNavioAtirar* após a inclusão das interfaces para premissas e instigações.

Figura 27 - Diagrama de componentes para a regra *rlNavioAtirar*



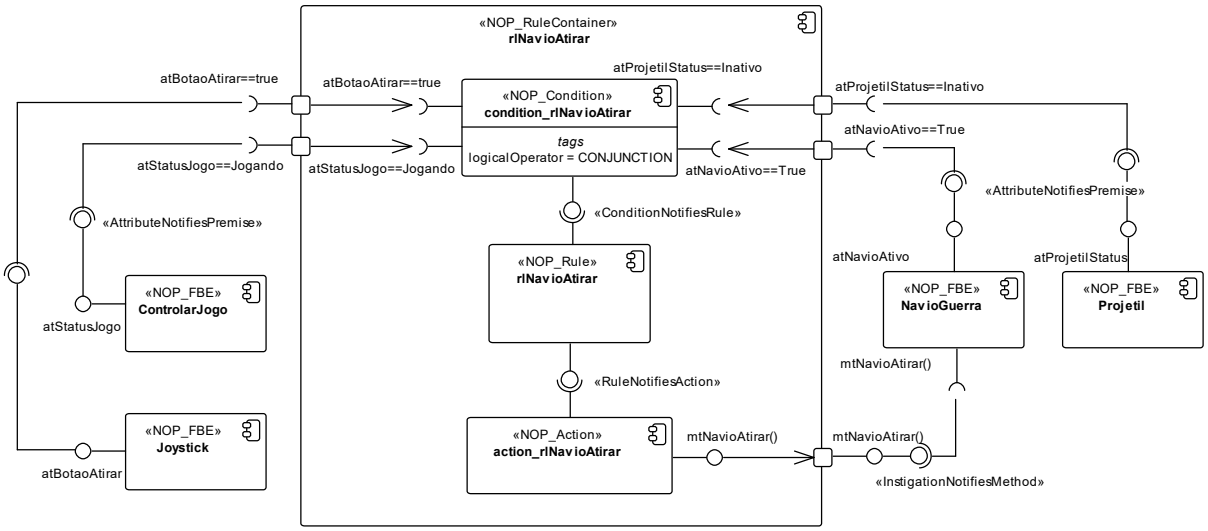
Fonte: Autoria própria

Outra atividade dessa etapa é a definição do operador lógico para a regra. O operador “CONJUNCTION” foi usado para a *Rule rlNavioAtirar* para indicar que todas as premissas são obrigatórias. Outros operadores disponíveis são “DISJUNCTION”, usado quando somente uma das premissas é suficiente para ativar a regra, e “SINGLE” quando há somente uma premissa.

2.1.5.4.3 Relacionar as regras aos FBEs

Nesta última etapa da criação do modelo de componentes, as regras são conectadas aos *FBEs*. Para isso, identificam-se os *FBEs* com atributos que notificam as premissas de cada regra e aqueles cujas instigações, partindo da regra, chamam seus métodos. A Figura 28 mostra o diagrama de componentes da *Rule rlNavioAtirar* após a inclusão das relações com os respectivos *FBEs*.

Figura 28 - Diagrama de componentes da regra *rlNavioAtirar* após etapa 3



Fonte: Autoria própria

Os atributos dos *FBEs* (e.g. *atStatusJogo* do *FBE ControlarJogo*) são representados nesse diagrama por interfaces fornecidas pelos componentes. Os métodos são apresentados como interfaces requisitadas pois, diferente dos atributos, esses elementos são requisitados por outros componentes.

Os diagramas de componentes de cada regra são, então, agrupados em um único diagrama para formar o modelo de componentes do software. Esse modelo oferece uma visão global das ligações entre as regras e os outros elementos do metamodelo do PON.

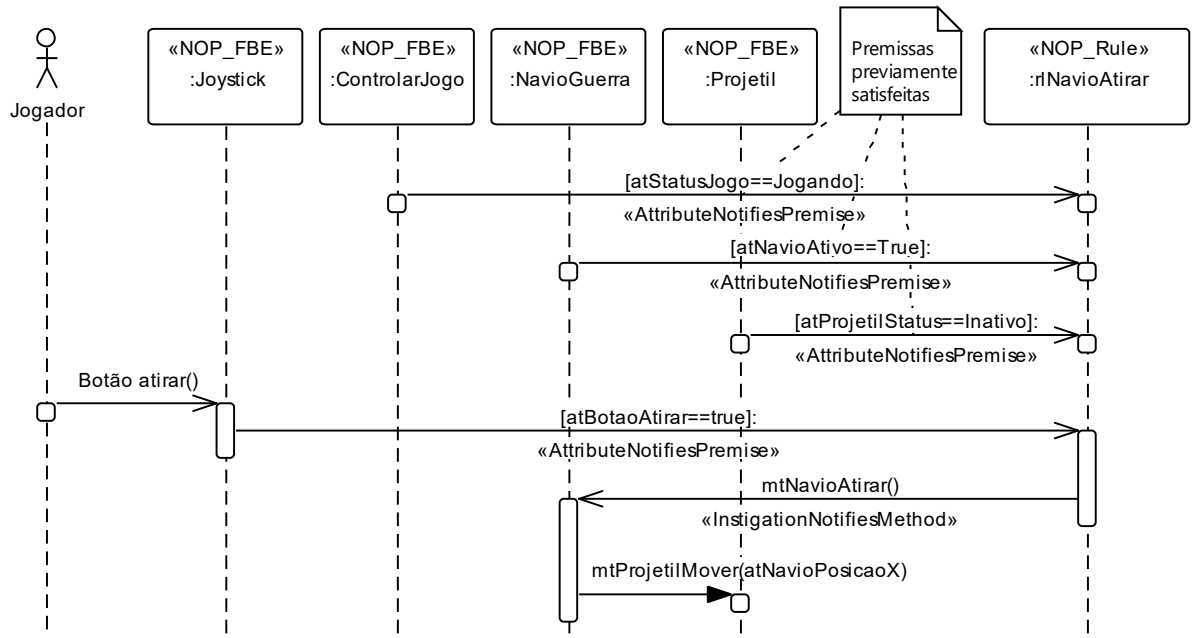
2.1.5.5 Criar modelo de sequência e de comunicação

Os modelos de sequência e de comunicação são usados para descrever as interações entre os objetos da aplicação em PON. O nível de abstração dos diagramas nesses modelos, assim como a sua extensão, pode variar de acordo com a importância do que se pretende modelar e decisão do projetista.

A seguir são apresentados dois diagramas de sequência e seus respectivos diagramas de comunicação para a *Rule rlNavioAtirar* em diferentes níveis de abstração. Esses diagramas demonstram a interação entre os *FBEs Joystick*, *NavioGuerra*, *Projetil* e *ControlarJogo* com a regra em questão. Apresenta-se, também, um diagrama de sequência que ilustra as *Rules rlNavioAtirar* e *rlDetectarColisaoContraInimigo* sendo executadas e os respectivos *FBEs* envolvidos.

O primeiro diagrama de sequência, exibido na Figura 29, pode ser considerado de alto nível de abstração. Nele, somente *FBEs* e regras estão presentes. Os outros elementos do metamodelo do PON permanecem subentendidos. O diagrama mostra a *Rule rlNavioAtirar* com três premissas previamente confirmadas por mensagens do diagrama. Essas premissas indicam que o status do jogo é *Jogando*, a estamina do jogador é maior do que zero e o status do projétil é inativo. Quando uma última premissa é confirmada pela ação do jogador que pressionou o botão de atirar, a regra é ativada e pode ser executada. A execução desta regra faz com que o *Method mtNavioAtirar* seja executado, alterando o valor do status do projétil para ativo e executando o *Method mtProjetilMover*.

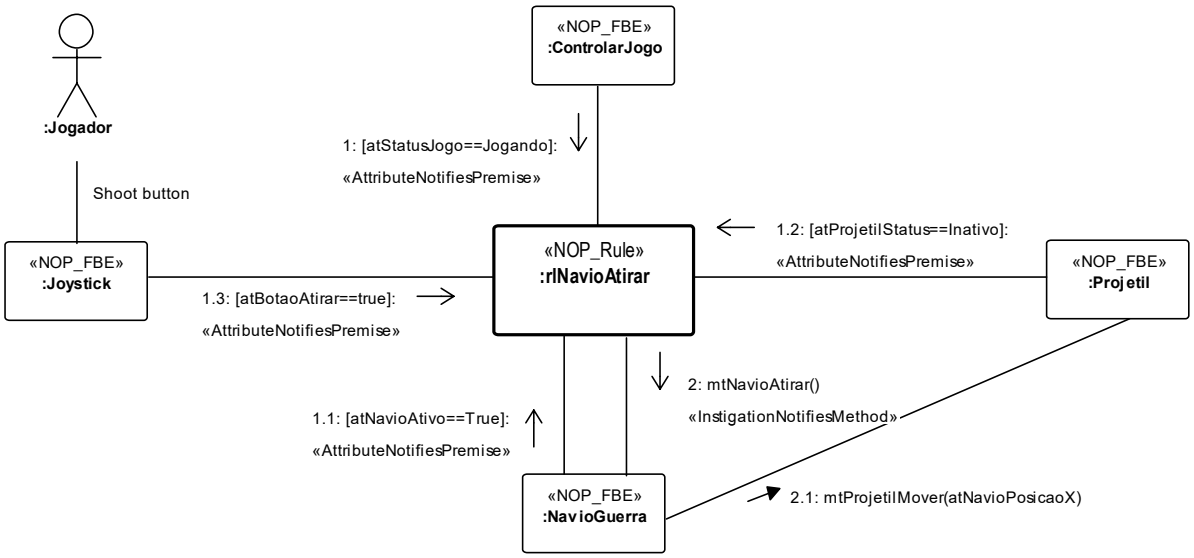
Figura 29 - Diagrama de sequência de alto nível para a *Rule rlNavioAtirar*



Fonte: Autoria própria

O diagrama de comunicação equivalente ao diagrama de sequência da Figura 29 é apresentado na Figura 30. Os diagramas de comunicação mostram informações similares às apresentadas pelos diagramas de sequência, porém com foco maior na relação entre os elementos participantes.

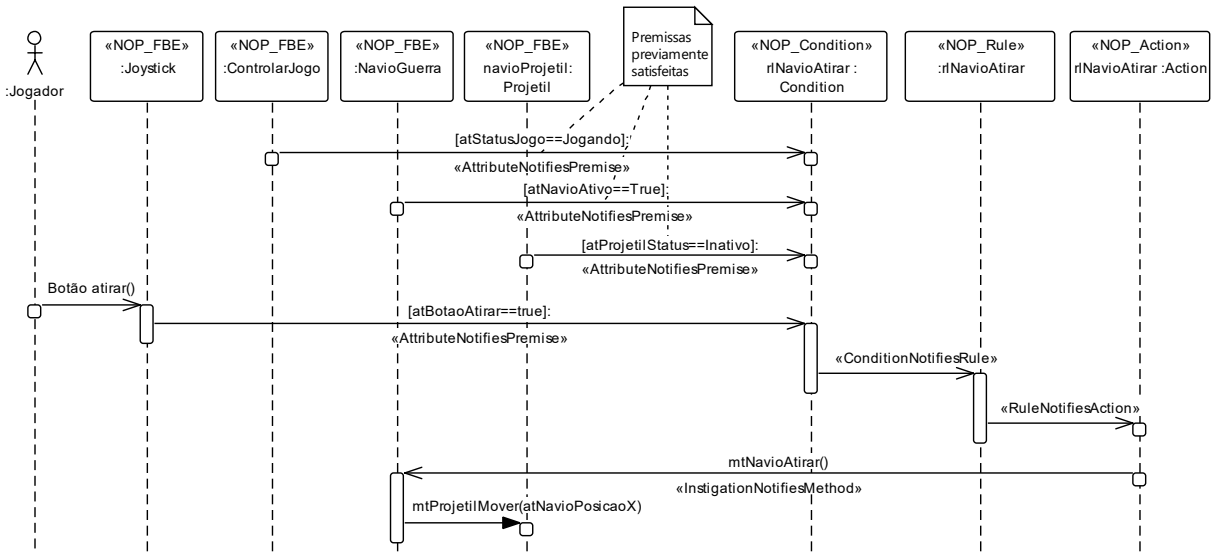
Figura 30 - Diagrama de comunicação equivalente ao diagrama de sequência apresentado na Figura 29



Fonte: Autoria própria

O próximo diagrama de sequência (Figura 31) representa a mesma execução de regra, mas possui um nível maior de detalhes, incluindo os elementos *Condição* e *Ação* da regra.

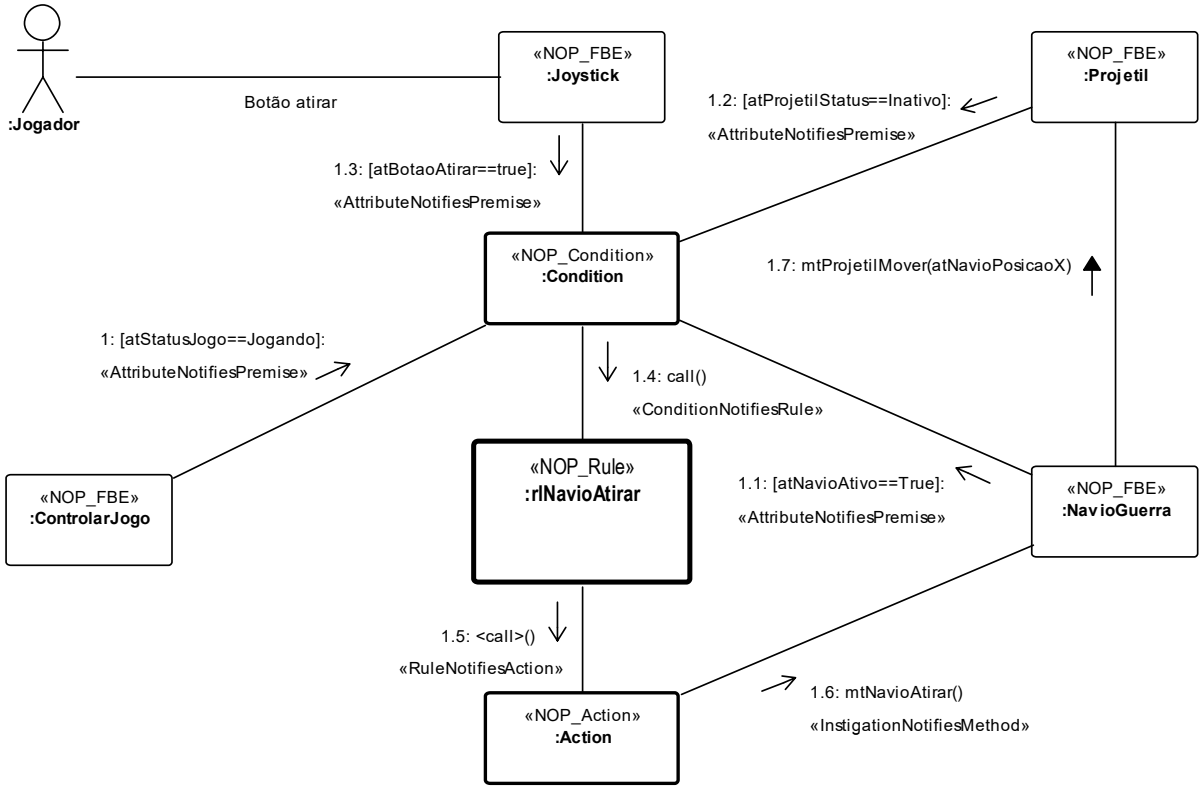
Figura 31 - Diagrama de sequência para a *Rule rlNavioAtirar* incluindo elementos do tipo *FBE*, Regra, Condição e Ação.



Fonte: Autoria própria

A Figura 32 apresenta o diagrama de comunicação correspondente ao diagrama de sequência da Figura 31.

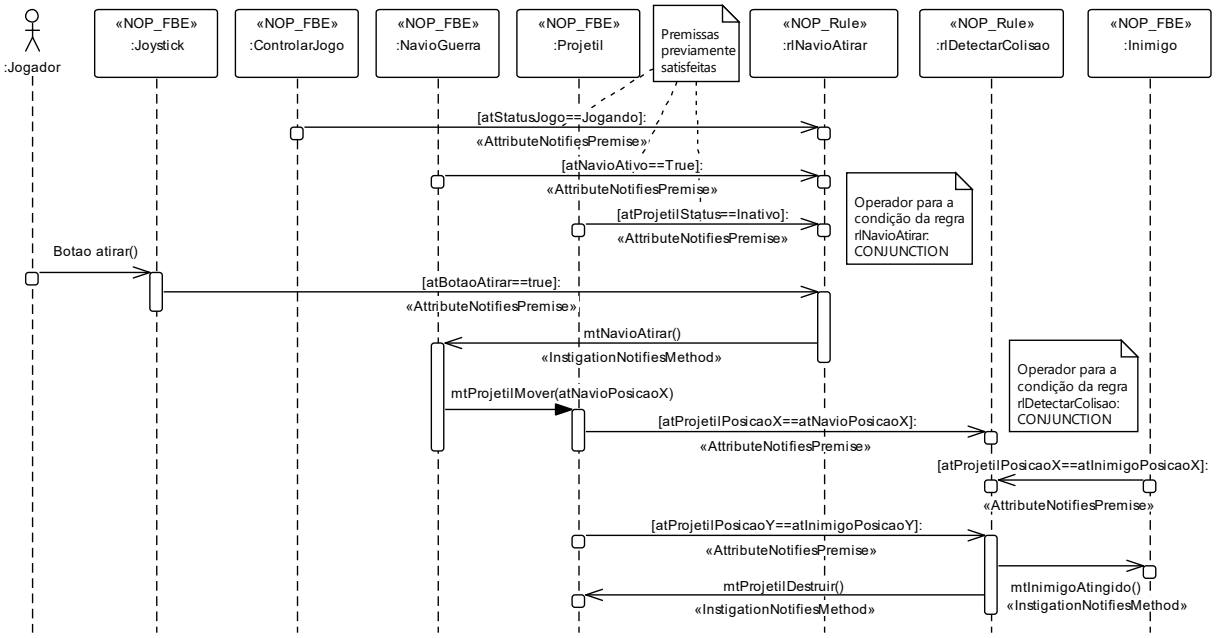
Figura 32 - Diagrama de comunicação equivalente ao diagrama de sequência apresentado na Figura 31.



Fonte: Autoria própria

Por fim, o diagrama de sequência da Figura 33 apresenta duas regras sendo executadas em ordem, ou seja, uma regra desencadeia a execução da outra. As primeiras mensagens do diagrama de sequência são equivalentes ao diagrama mostrado na Figura 29. A *Rule rlDetectarColisao* é ativada quando o projétil do jogador está na mesma posição que o avião de guerra inimigo. Para demonstrar essa sequência no diagrama, os *FBEs Projetil* e *Inimigo* notificam a *Rule rlDetectarColisao* sobre suas posições no eixo X. Na sequência, o *FBE Projetil* reporta à *Rule rlDetectarColisao* sua posição Y. Se o teste das premissas for verdadeiro, isto é, se o avião de guerra estiver na mesma posição que o projétil, a *Rule rlDetectarColisao* é ativada e os *Method*s *mtInimigoAtingido()* do *FBE Inimigo* e *mtProjetilDestruir()* do *FBE Projetil* são executados.

Figura 33 - Diagrama de sequência para apresentar Regras sendo executadas em ordem.



Fonte: Autoria própria

Os diagramas criados nesse passo são importantes para entender e comparar a ordem de execução das regras, especialmente se mais de uma regra estiver no diagrama. O conjunto de diagramas compõe os modelos de sequência e comunicação do DON. Além disso, o uso do Perfil PON nesses modelos contribui para a execução correta da lógica do PON enquanto o processo de modelagem avança.

2.1.5.6 Criar modelo de Redes de Petri

O modelo de Redes de Petri é usado para apresentar a visão dinâmica do software PON. Na UML esta visão é, usualmente, criada com diagramas de estados ou de atividades para cada classe ou grupo de classes. Porém, esta abordagem torna-se impraticável quando o número de estados do software é muito grande. Na realidade, poucos projetos modelam toda a dinâmica usando a UML (DÖLL, 2002).

As Redes de Petri (RdPs) permitem a modelagem de concorrência, sincronização e compartilhamento de recursos em sistemas (CARDOSO; VALETTE, 1997). A compatibilidade entre as RdPs e o predecessor do PON foi discutida no trabalho de Simão (2005). Naquele trabalho, em uma RdP Lugar/Transição, as Regras do PON são interpretadas como Transições da RdP, as Condições e Ações são interpretadas como arcos de entrada e saída da Transição que representa a

Regra, e os estados dos Atributos são representados por Lugares da RdP. Posteriormente, Linhares *et al.* (2011) apresentam um modelo de RdP colorida para a modelagem de um sistema telefônico em PON.

O DON sugere a criação de uma RdP para cada caso de uso, que são integradas em um único modelo de Redes de Petri, posteriormente. O modelo de RdP é obtido a partir do mapeamento do modelo de componentes. O DON sugere duas alternativas de mapeamento, ambas na forma de RdPs Lugar/Transição, sendo uma detalhada e outra resumida.

As etapas de mapeamento para a rede detalhada são:

1) Os componentes estereotipados <<*NOP_Condition*>>, <<*NOP_Rule*>> e <<*NOP_Action*>> são mapeados para Transições da RdP.
2) Os componentes ou interfaces estereotipados <<*NOP_Premise*>> e <<*NOP_Instigation*>> são mapeados em Lugares da RdP. Esta representação demonstra o estado dos Atributos, uma vez que as Premissas e Instigações testam e modificam estes estados.
3) Os relacionamentos de notificação são mapeados em Lugares da RdP.

Embora não descrito no método, está subentendido que os arcos da RdP surgem em decorrência dos fluxos de notificação na aplicação PON.

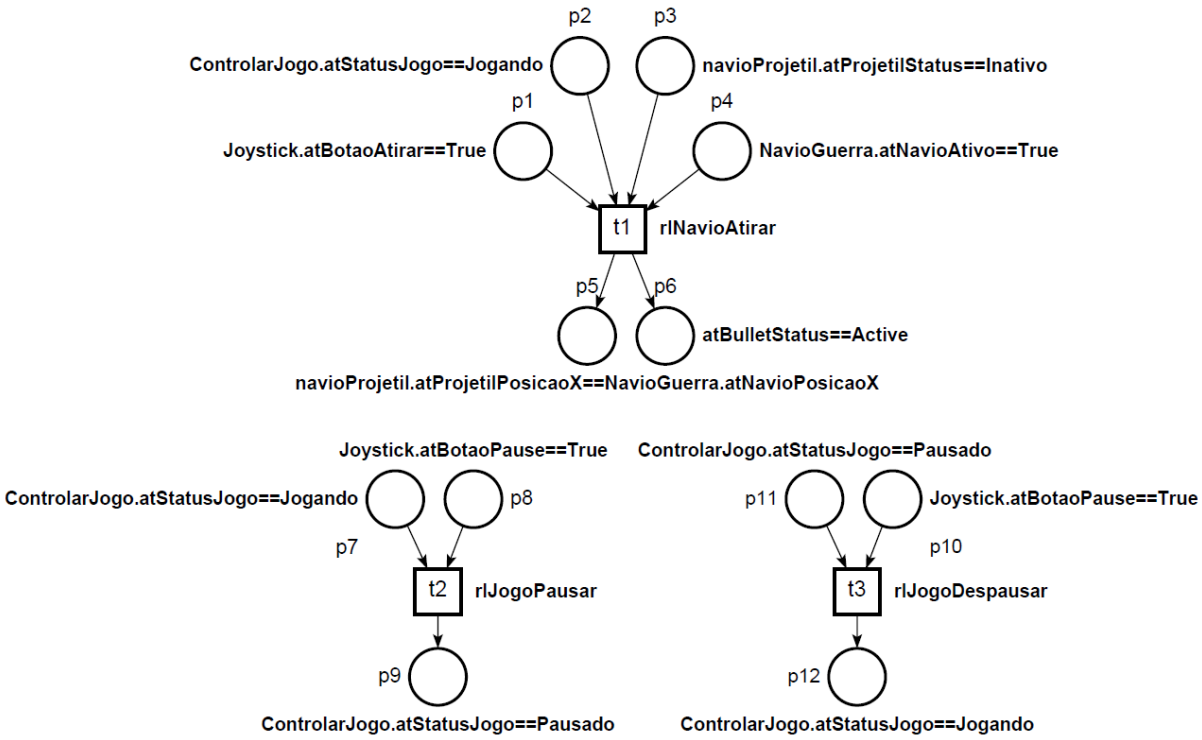
As etapas do processo de mapeamento resumido são:

1) Os componentes estereotipados <<*NOP_Rule*>> são mapeados em Transições na RdP.
2) Os Atributos e seus estados pertencentes aos componentes <<*NOP_Premise*>> e <<*NOP_Instigation*>> são mapeados em Lugares na RdP.
3) Os componentes estereotipados <<*NOP_Premise*>> ou interfaces requeridas das regras são mapeados em arcos chegando nas transições que representam os elementos <<*NOP_Rule*>>.
4) Os componentes estereotipados <<*NOP_Instigation*>> ou interface fornecidas são mapeados em arcos que saem das transições que representam os elementos <<*NOP_Rule*>>.

A Figura 34 mostra um exemplo de três regras mapeadas no modelo de RdP Lugar/Transição resumida. A *Rule rlNavioAtirar*, mostrada no diagrama de

componentes da Figura 28, é ilustrada na parte superior (i.e., transição t1) da Figura 34. Esta é a única regra que compõe o caso de uso *Controlar ataque*. Outras duas *Rules* também são mostradas na Figura 34: *rlJogoPausar* e *rlJogoDespausar*, ambas pertencentes ao caso de uso *Pausar e despausar partida*.

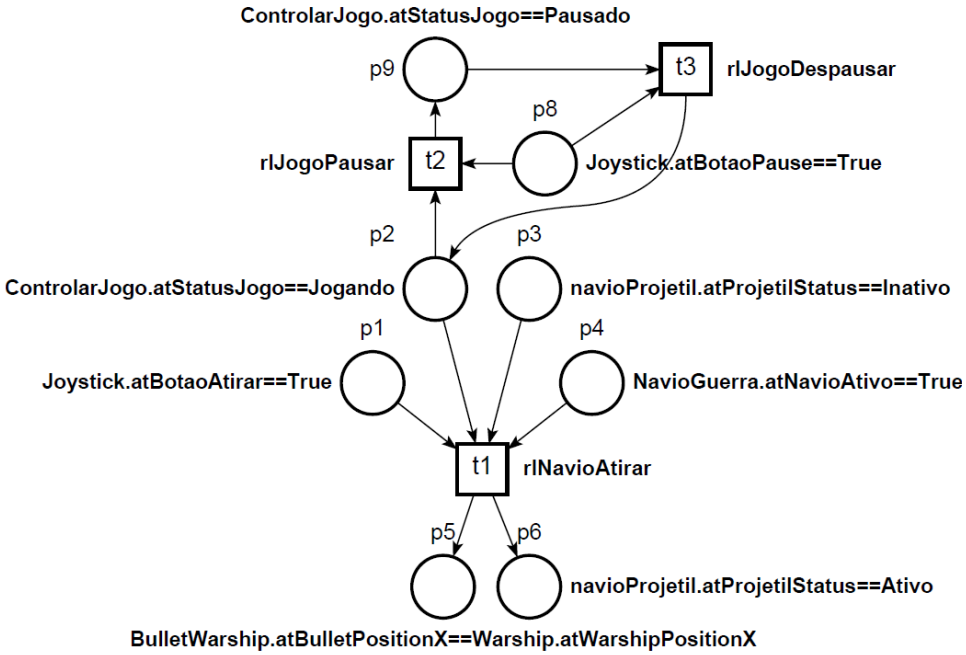
Figura 34 - Redes de Petri mapeadas do modelo de componentes. *Rules*: *rlNavioAtirar*, *rlJogoPausar* e *rlJogoDespausar*



Fonte: Autoria própria

Os próximos passos são: (i) integrar os elementos da RdP por caso de uso; e (ii) integrar as RdP por caso de uso em um diagrama único para formar o Modelo de Redes de Petri. Na Figura 34, os lugares p8 e p10 têm o mesmo significado, assim como os lugares p7 e p12 e os lugares p9 e p11. Esses lugares são fundidos para criar o diagrama de RdP do caso de uso *Pausar e despausar partida*. A Figura 35 mostra o modelo de RdP, em construção, cujos casos de uso *Controlar ataque* e *Pausar e despausar partida* foram integrados.

Figura 35 - Diagrama de RdP após fusão dos casos de uso *Controlar ataque* e *Pausar* e *despausar partida*



Fonte: Autoria própria

O modelo de Redes de Petri permite a análise e simulação da solução proposta. A simulação é especialmente útil para verificar se o comportamento da proposta atende aos casos de uso e aos requisitos do software.

2.1.5.7 Discussões sobre o DON

O Desenvolvimento Orientado a Notificações (DON) foi concebido para modelar software seguindo o Paradigma Orientado a Notificações (PON). Os esforços se concentraram em estender o uso de técnicas convencionais para se adequarem ao PON. O método DON abrange atividades relacionadas aos requisitos, análise e projeto de softwares em PON, sendo aderente aos processos tradicionais de desenvolvimento de software.

O método DON é iterativo, permitindo a melhoria dos modelos à medida que os ciclos ocorrem. Na execução do método para este trabalho, por exemplo, diversos atributos e métodos foram identificados somente durante a criação do modelo de componentes. Portanto, os ciclos 2 e 3 do método permitiram que os modelos anteriores fossem atualizados.

A criação de um perfil específico para o PON permitiu que a semântica dos conceitos do PON fossem representadas nos diagramas da UML. O modelo de Redes de Petri pôde simular o comportamento do software e permitir que ele fosse comparado aos requisitos previamente estabelecidos.

O uso do método DON supera algumas limitações existentes na modelagem de softwares PON e fornece uma abordagem abrangente para desenvolvimento de softwares baseados no PON. Por outro lado, o uso extensivo da UML, sobretudo na quantidade de modelos, acaba por onerar o processo de desenvolvimento com artefatos muitas vezes repetitivos, que se sobrepõem. Por exemplo, os Modelos de Sequência e Comunicação são artefatos equivalentes, diferenciando-se pela qualidade do Diagrama de Comunicação em expressar melhor as relações entre os elementos (i.e., objetos). Ainda, esses mesmos modelos são de certa forma equivalentes ao Modelo de Componentes que é usado para modelar as Regras e suas relações com os Fatos e as outras entidades do metamodelo do PON, com a diferença de que eles apresentam certos casos da aplicação sendo modelada enquanto o Modelo de Componentes apresenta as relações gerais entre as entidades da aplicação. Estes artefatos repetitivos seriam úteis se houvesse automatismos, por exemplo, se a partir deles fosse realizada a geração de código para o PON.

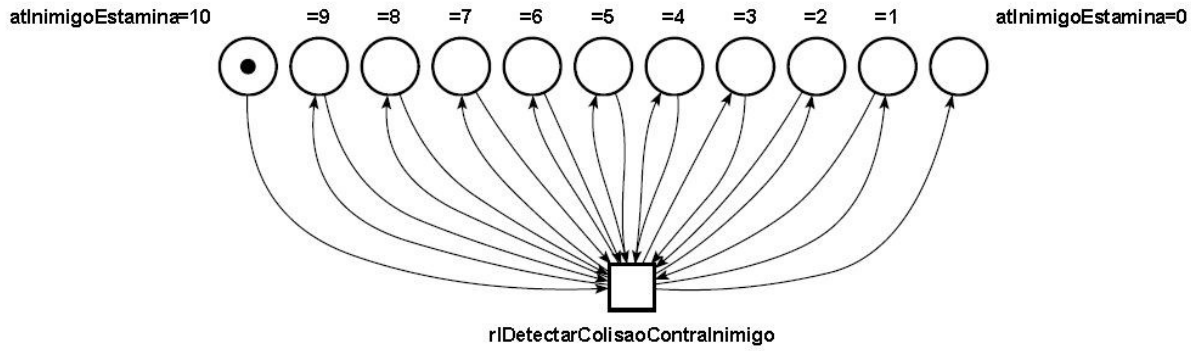
Outro aspecto a respeito do DON é a sua fundamentação no *Framework* PON C++ 1.0. Assim, o perfil PON considera características daquela plataforma. Por exemplo, na versão 1.0 do *framework* uma aplicação PON pode ter somente uma estratégia de escalonamento de regras, enquanto na versão 2.0 do *framework* a estratégia pode ser definida em tempo de execução. Além disso, diversos outros *frameworks* para o PON foram criados desde então, cada um abordando diferentes particularidades. Apesar disso, os diagramas do DON que modelam os elementos fundamentais do paradigma se mantiveram funcionais, sendo usados até hoje nos trabalhos de tese e dissertação para modelagem de sistemas PON desenvolvidos pelo grupo pesquisa. De fato, os modelos UML especializados para o DON permitem a efetiva modelagem das características do paradigma.

Entretanto, uma lacuna deixada pelo DON, supostamente por usar a UML, é a dificuldade na descoberta das regras do PON. Isso acontece, em parte, porque a UML não foi desenvolvida para este paradigma, carecendo de diagramas apropriados. Assim, o processo de identificação de regras torna-se um intenso

processo de síntese e requer muito esforço do desenvolvedor. Nota-se a falta de “algo” (i.e., métodos, ferramentas, artefatos etc) que auxilie o desenvolvedor na descoberta dos principais elementos que compõem o PON e fazem parte de suas primitivas, como as Regras, os Fatos e as Notificações.

Outro aspecto identificado foi que as técnicas de mapeamento de diagramas de componentes para Redes de Petri não foram suficientes para criar uma rede simulável. Por exemplo, quando se tem um atributo do tipo inteiro cujo valor é modificado por uma Regra. Um exemplo concreto é a Regra 6 *rlDetectarColisaoContraInimigo*, que deverá diminuir a estamina do inimigo quando o mesmo for atingido por um projétil. Seguindo o mapeamento definido por Wiecheteck (2011) chega-se à RdP, não simulável, ilustrada na Figura 36.

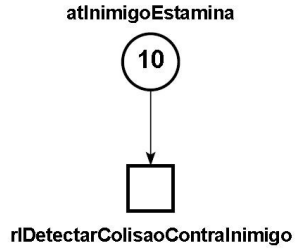
Figura 36 - Diagrama RdP para *Rule rlDetectarColisaoContraInimigo* (exemplo 1)



Fonte: Autoria própria

Uma solução para a RdP apresentada na Figura 36 seria o desmembramento da *Rule rlDetectarColisaoContraInimigo* em outras 9 Regras, cada uma para modificar o estado do *Attribute atInimigoEstamina* para o próximo valor. De outra forma, o projetista poderia recorrer aos exemplos que constam no trabalho de Simão (2005) e usar somente um Lugar na RdP para o atributo e representar a quantidade de estamina com marcadores. O diagrama ficaria como ilustrado na Figura 37, porém teria problema ao integrar com a *Rule rlJogadorGanha* cuja *Condition* inclui a *Premise atInimigoEstamina==0*.

Figura 37 - Diagrama RdP para *Rule rlDetectarColisaoContraInimigo* (exemplo 2)



Fonte: Autoria própria

A discussão sobre o mapeamento para RdPs não é exaustiva, mas fica evidenciado que a técnica e este tipo de Rede de Petri não são um método consolidado para modelagem de software no PON. Assim, o projetista é obrigado a criar elementos adicionais, ou técnicas diferentes do que o método sugere, para que a rede possa ser simulada ou ter suas propriedades verificadas.

Este estudo de aplicação do DON representou uma das primeiras atividades realizadas pelo autor desta tese durante seu doutoramento e resultou na publicação de um artigo em conferência (MENDONÇA *et al.*, 2015). O objetivo foi conhecer o método proposto pela equipe de pesquisa sobre o PON como fundamentação para as proposições desenvolvidas a seguir na pesquisa desenvolvida e descrita neste documento.

Em suma, o potencial do DON é poder expressar precisamente os elementos fundamentais do Paradigma Orientado a Notificações, ou seja, as Regras, os *FBEs* e as Notificações. Isto advém do uso da UML e de um perfil UML, denominado Perfil PON, criado especificamente para o referido paradigma. Por outro lado, percebe-se no DON a falta de mecanismos que auxiliem o projetista na identificação de tais elementos fundamentais a partir da especificação de um problema. Este trabalho de tese busca preencher essa lacuna oferecendo uma metodologia focada em aspectos de concepção de software para o PON. Assim, busca-se auxiliar o projetista no processo de síntese de um problema para uma solução em termos do PON. A metodologia proposta neste trabalho é complementar ao DON e sugere-se o eventual uso em conjunto.

2.2 ABORDAGENS PARA DESENVOLVIMENTO DE SOFTWARE ORIENTADO A REGRAS

No decorrer da pesquisa desta tese, realizou-se um Mapeamento Sistemático da Literatura (MSL) pelo autor deste trabalho, em conjunto com seus orientadores, para identificar o estado da arte no que diz respeito a modelagem de Sistemas Baseados em Regras (SBRs). O MSL difere da Revisão Sistemática da Literatura (RSL) no foco, ou seja, o MSL é uma investigação cuja questão de pesquisa é mais abrangente e, consequentemente menos profunda. Porém, mantém as mesmas etapas e critérios de uma RSL (DERMEVAL; COELHO; BITTENCOURT, 2020).

O Paradigma Orientado a Notificações (PON) se apropria de uma das características dos SBRs, que é a concepção de software na forma declarativa com uso de regras, o que permitiria dizer que PON e SBR faria parte de um conjunto maior chamado de Paradigma Orientado a Regras (POR) (BATISTA, 2013; SIMÃO; STADZISZ; WIECHETECK, 2015). Assim, o MSL apresentado a seguir buscou identificar na literatura recente quais estratégias são usadas na academia para modelagem dessa classe de sistemas.

A pergunta de pesquisa que guiou o MSL foi: “Quais técnicas, métodos, ferramentas, linguagens, representações, especificações e instrumentos são usados para modelagem de software nos quais regras são os elementos lógicos de modelagem?”. A partir desta questão, chegou-se à seguinte *string* de busca (em inglês): “("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[21], Science Direct[22], Engineering Village[23] e IEEE Xplore[24]. Foram considerados somente trabalhos escritos em inglês que tinham sido publicados em revistas, anais de eventos e padrões de especificação nos últimos dez anos de quando o MSL foi realizado, i.e., 2006-2016.

[21] https://www.scopus.com/
[22] https://www.sciencedirect.com/
[23] https://www.engineeringvillage.com/
[24] https://ieeexplore.ieee.org/

A busca pelos estudos sobre abordagens de engenharia de software para modelagem de regras retornou 824 artigos, dos quais 127 estavam no banco de dados Scopus, 76 no Science Direct, 418 na Engineering Village, 129 no IEEE Xplore e 4 foram incluídos manualmente pois são padrões e especificações de conhecimento prévio dos autores do mapeamento sistemático. Na fase de seleção, foram identificados 519 trabalhos, após remoção dos duplicados. Em seguida, a triagem foi realizada em duas etapas. Durante a primeira etapa, foram excluídos 305 estudos, após a revisão do título, resumo e palavras-chave de cada trabalho. Na segunda etapa, os trabalhos em texto completo dos 122 artigos restantes foram obtidos e 71 deles foram excluídos. Um total de 51 trabalhos foi selecionado no mapeamento sistemático de literatura.

Os estudos selecionados estão distribuídos uniformemente, com uma média de 6 publicações por ano, entre 2006 e 2013, tendo diminuído nos últimos anos do período buscado (i.e., 2014 e 2016). Similarmente, os estudos estão distribuídos com 49% em jornais e em eventos, sendo os 2% restantes publicações de especificações. O MSL classificou os estudos selecionados segundo uma ou mais abordagens de ES para sistemas de regras em (i) linguagens, representações e especificações (32 estudos), (ii) *frameworks*, métodos e metodologias (20 estudos) e (iii) técnicas, instrumentos e ferramentas (36 estudos).

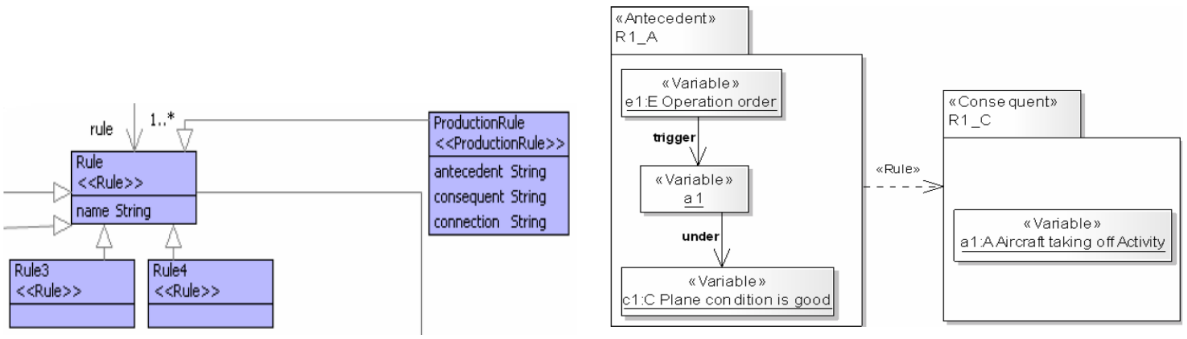
Os resultados do MSL evidenciaram que processos de software em que regras são os elementos de concepção não possuem um padrão *de facto*, tal qual há a UML para a OO. Percebeu-se, pelo mapeamento, dificuldades próprias da modelagem de sistemas que se orientam a regras. Ainda assim, a maior parte dos trabalhos faz uso de processos tradicionais de software com alguma adaptação para sistemas de regras. O Quadro 7 apresenta as abordagens mais frequentes encontradas nos trabalhos selecionados e destaca-se o número de ocorrências sem o uso de padrões, ou seja, de forma *ad-hoc*.

Quadro 7 - Abordagens mais frequentes na modelagem de regras

Linguagem	Sublinguagem	Estudos	Sub-total	Total
Ad-hoc		(CORRADINI *et al.*, 2015), (IASSINOVSKI; ARTIBA; FAGNART, 2008a), (BOUKHEBOUZE *et al.*, 2009), (DANIEL; MATERA; POZZI, 2006), (PRANEVICIUS; BUDNIKAS, 2008), (EITER; FEIER; FINK, 2012), (ERICSSON *et al.*, 2008), (AGARAM; LIU, 2012), (HUANG; HUANG; ZHANG, 2010), (AGÜERO *et al.*, 2012), (LEE; SU; LAM, 2006), (SILVA; CHNITI, 2013), (KROGSTIE, 2008), (IASSINOVSKI; ARTIBA; FAGNART, 2008b).		**14**
XML	RuleML	(BASSILIADES; ANTONIOU; VLAHAVAS, 2006)	1	**8**
	R2ML	(LUKICHEV; GIURCA; WAGNER, 2007), (DIACONESCU, 2007), (RIBARIĆ *et al.*, 2008), (GIURCA; WAGNER, 2007).	4	
	Ad-hoc	(FRITZEN; MAY; SCHENK, 2008), (ZHANG; XU; SHOU, 2011), (WANG; WANG; LI, 2012).	3	
UML	Profile	(ABDULLAH *et al.*, 2007a), (ZHI-XUE *et al.*, 2012), (ABDULLAH *et al.*, 2007b).	3	**8**
	Ad-hoc extension	(OLMEDO-AGUIRRE; LA ROSA; MORALES-LUNA, 2008)	1	
	URML	(LUKICHEV; GIURCA; WAGNER, 2007), (RIBARIĆ *et al.*, 2008), (GIURCA; WAGNER, 2007), (LUKICHEV; WAGNER, 2006).	4	
Petri Nets		(ZAMANI *et al.*, 2010), (LI; MEDINA; CHAPA, 2007), (LE; HE, 2008), (HU; LU; ZHAO, 2013), (ZHOU ; GAO, 2010), (GONÇALVES, 2012), (ZHANG; XU; HELO, 2013), (BABA-HAMED, 2006).		**8**
PRR		(PRAT; AKOKA; COMYN-WATTIAU, 2012), (PRAT; AKOKA; COMYN-WATTIAU, 2011), (PRAT; WATTIAU; AKOKA, 2010), (MARTÍNEZ-FERNÁNDEZ; MARTÍNEZ; GONZÁLEZ-CRISTÓBAL, 2009), (PRAT; COMYN-WATTIAU; AKOKA, 2011).		**5**
CML		(CANADAS; PALMA; TÚNEZ, 2009), (CAÑADAS; PALMA; TÚNEZ, 2011).		**2**
Finite state machine		(MOULIN, 2006)		**1**

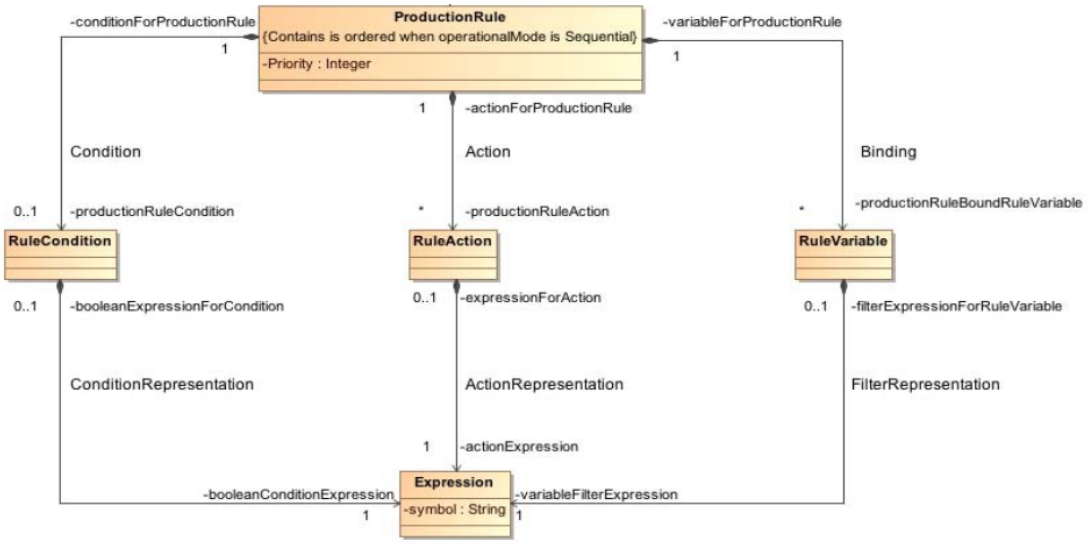
Fonte: Autoria própria

Além disso, a variedade de notações pode ser observada mesmo nas abordagens baseadas em uma mesma linguagem. Por exemplo, Abdullah *et al.* (2007a) usam classes UML para descrever regras de produção nas quais o antecedente e consequente da regra são atributos (veja a Figura 38a). Zhi-xue et al. (2012), por outro lado, usam relacionamentos de dependência para descrever as regras (ver Figura 38b). No entanto, o mais comum é usar classes para modelar a regra, o antecedente e o consequente (ver Figura 38c) (OBJECT MANAGEMENT GROUP, 2009; PRAT; COMYN-WATTIAU; AKOKA, 2011).



(a) (b)

Figura 38 - Diferentes abordagens para modelagem de regras usando UML



(c)

Fonte: (a) (ABDULLAH *et al.*, 2007a), (b) (ZHI-XUE *et al.*, 2012) e (c) (OMG, 2009; PRAT; COMYN-WATTIAU; AKOKA, 2011)

Por sua vez, os estudos que aplicam as Redes de Petri (RdP) como notação para modelagem de regras usam, em geral, os mesmos princípios entre si. Neles, as regras são modeladas como transições e os lugares de entrada e de saída representam seus antecedentes e consequentes, respectivamente.

Por fim, o Mapeamento Sistemático da Literatura permitiu aos autores uma visão abrangente sobre as técnicas, métodos, ferramentas, linguagens, representações, especificações e instrumentos para modelar regras. Assim, a partir dos estudos selecionados no MSL, pôde-se conjecturar algumas direções de pesquisa que resume as principais contribuições:

- Padronização: o desenvolvimento de software orientado a objetos geralmente adota a UML como notação padrão. Por outro lado, regras declarativas e tecnologias emergentes relacionadas parecem não ter uma notação padrão de fato (como apontado no Quadro 7). Como as abordagens

para SBRs e orientadas a objetos estão em diferentes paradigmas, elas devem envolver notações especializadas e distintas. Adicionalmente, considera-se que são necessárias mais investigações sobre modelagem de regras, a fim de identificar se é possível estabelecer um consenso entre as pesquisas sobre uma notação comum neste campo de estudo.

- Processos de desenvolvimento: poucos estudos explicitam todo o processo de desenvolvimento de software construído na forma de regras. Frequentemente, não está claro como mapear requisitos em regras ou como modelar a arquitetura de um sistema baseado em regras, por exemplo. A partir dos estudos selecionados, percebe-se a falta de processos de desenvolvimento para sistemas baseados em regras.
- Redes de Petri: esta revisão encontrou um número significativo de abordagens usando RDPs para modelagem de regras (conforme Quadro 7). A maioria delas concorda em como modelar as regras. No entanto, poucos estudos apresentam técnicas para analisar as propriedades de modelos de regras, como mencionado também por Ericsson *et al.* (2008) e Huang, Huang e Zhang (2010), enquanto a análise de propriedades é conhecida como uma importante vantagem das Redes de Petri. O desenvolvimento de técnicas para analisar as propriedades dos modelos de regras declarativas parece ser uma contribuição promissora neste campo de pesquisa.
- Concepção: os estudos revisados revelaram diferentes formas de modelar regras. No entanto, a maioria dos estudos se concentrou em propor notações novas ou adaptadas. Poucos estudos apresentaram contribuições para ajudar os desenvolvedores no projeto (no sentido de conceber a solução) de um sistema baseado em regras. Novas técnicas de *design* para sistemas baseados em regras são necessárias para garantir uma melhor qualidade das soluções desenvolvidas. A qualidade pode ser medida, por exemplo, como o grau de interdependência funcional e a complexidade associada (BATISTA, 2013; SUH, 1990).
- Comparativos: os estudos revisados não apresentaram estudos comparativos entre abordagens referentes a regras. Krogstie (2008) usou uma estrutura formal chamada SEQUAL para avaliar especificamente sua abordagem à modelagem de regras. Novos estudos podem ser desenvolvidos aplicando métodos, como o SEQUAL, para comparar e

avaliar técnicas, métodos, ferramentas, linguagens, representações, especificações e instrumentos para modelar as regras.

2.3 RECOMENDAÇÕES PARA CRIAÇÃO DE LINGUAGENS DE MODELAGEM

Tradicionalmente na engenharia de software, usam-se modelos computacionais para mapear ou criar representações de entidades do mundo real. O processo que envolve a criação desses modelos é subjetivo, ou seja, diferentes desenvolvedores podem criar soluções distintas para o mesmo problema (HAWRYSZKIEWYCZ, 1991 apud SCHUETTE; ROTTHOWE, 1998), ainda que usando uma única linguagem de modelagem. Assim, esforços vêm sendo empregados para reduzir esse subjetivismo e melhorar o processo de modelagem.

Schuette e Rotthowe (1998) propuseram um conjunto de diretrizes para modelagem, chamado *Guidelines of Modeling* (GoM), com princípios de projeto para melhoria na qualidade de modelos criados. O foco do estudo de Schuette e Rotthowe (1998) é na qualidade dos modelos e os princípios por eles concebidos têm sido referência na criação de linguagens de modelagem para domínios específicos, a exemplo de (JANIESCH *et al.*, 2019) (MALINOVA, 2016) (REINHARTZ-BERGER; STURM, 2008) (EVERMANN, 2003). Os seis princípios definidos no GoM são (SCHUETTE; ROTTHOWE, 1998):

1. **Princípio de adequação de construção**: um modelo é concebido para representar entidades do mundo real. Assim, este princípio de adequação de modelos avalia quanto ele é correto em relação à realidade. Como um modelo é a visão do projetista, sua adequação pode ser avaliada em consenso com as pessoas interessadas (e.g. projetistas, usuários) ou, caso o problema esteja bem definido, apoiar-se em documentação existente.
2. **Princípio de adequação de linguagem**: este princípio é identificado de duas formas. A primeira, chamada de conformidade de linguagem, está ligada à seleção de técnicas e linguagens de modelagem que possam representar a realidade em que se encontra o problema. Por exemplo, para o projeto de software orientado a objetos, a UML parece mais apropriada do que os diagramas de fluxos de dados (DFD). A segunda forma, chamada de correção da linguagem, está ligada ao uso correto da sintaxe da linguagem.

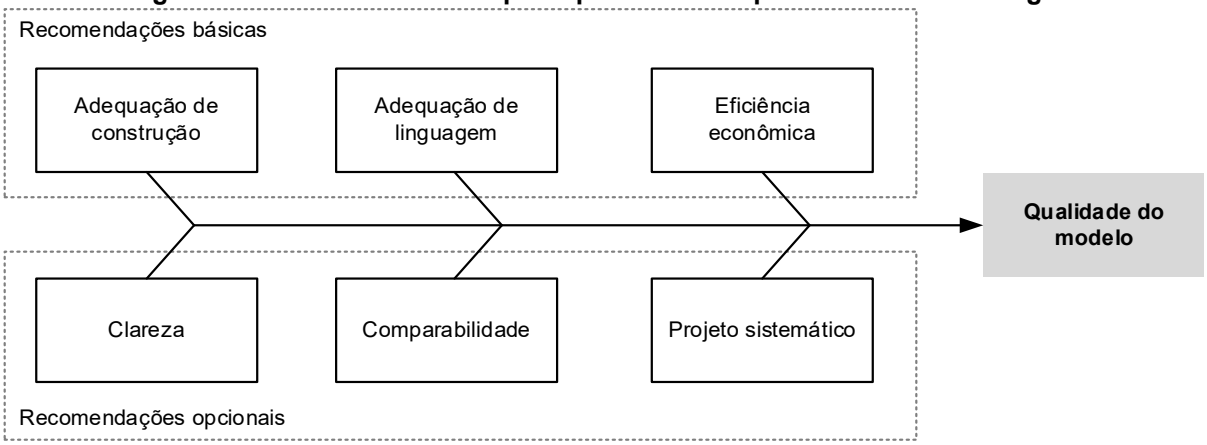
Nela, o metamodelo da linguagem tem papel fundamental. Além disso, ter uma ferramenta de modelagem é importante para garantir que os modelos estejam corretos em relação à sua sintaxe.

3. **Princípio de eficiência econômica**: a modelagem é uma etapa, dentre outras, do processo de software. Assim, este princípio afere os benefícios dessa etapa em relação aos custos envolvidos. Segundo Schuette e Rotthowe, o custo da linguagem está relacionado a sua complexidade de uso. Eles afirmam que linguagens mais flexíveis permitem modelagens mais rápidas, porém, caso não possuam o poder semântico necessário, podem requerer mais esforços em momentos posteriores. Segundo eles, o mesmo se aplica ao grau de formalização da linguagem, ou seja, menor grau de formalização, maior facilidade de criação dos modelos, porém serão necessárias mais informações para especificação do sistema, posteriormente.
4. **Princípio de clareza**: corresponde ao poder do modelo em explicitar o sistema de forma clara. Os autores do GoM atribuem a este princípio os aspectos de organização hierárquica dos modelos, organização dos elementos da linguagem e filtros de informações. A organização hierárquica dos modelos permite aos projetistas a criação de visões em diferentes níveis de abstração, facilitando a compreensão do sistema. A organização dos elementos da linguagem diz respeito a questões gráficas, ou seja, como os elementos da linguagem são apresentados. Neste aspecto, são consideradas características como simetria, alinhamento e distribuição dos elementos do modelo, por exemplo. No aspecto de filtro de informações, os autores do GoM destacam a possibilidade do usuário escolher o nível de informações que deseja visualizar.
5. **Princípio de projeto sistemático:** este princípio diz respeito à capacidade do modelo em apresentar de forma consistente as diferentes visões da modelagem. Os autores do GoM destacam, em especial, a necessidade desta consistência entre as visões estruturais e comportamentais. Além disso, destacam a importância de usar os mesmos elementos nas duas visões e atribuem ao metamodelo essa responsabilidade.
6. **Princípio de comparabilidade**: neste último princípio, o foco é na comparação semântica entre dois modelos, isto é, o conteúdo de dois

modelos deve ser comparado em relação à sua correspondência e semelhança. Os autores do GoM apontam a convenção de nomes e *layouts* pré-configurados como características que melhoram a qualidade deste princípio nos modelos.

A partir dos seis princípios apresentados, o GoM se fundamenta como uma arquitetura de referência para qualidade no processo de modelagem, chamado de *GoM-Architecture*. Dentre os seis princípios da *GoM-Architecture*, três são considerados básicos, ou seja, são pré-condição da arquitetura para alcançar a qualidade nos modelos e três são considerados opcionais (ROSEMANN; SEDERA; SEDERA, 2001). A Figura 39 apresenta o desenho da arquitetura. Assim, a partir dela, e seus princípios, é possível formular recomendações práticas para processos e linguagens de modelagem existentes (SCHUETTE; ROTTHOWE, 1998).

Figura 39 - GoM-Architecture para qualidade nos processos de modelagem



Fonte: traduzido e adaptado de (ROSEMANN; SEDERA; SEDERA, 2001)

A linguagem de modelagem pode ser considerada a principal ferramenta no processo de modelagem. Assim, para que o projetista consiga alcançar os seis princípios de qualidade do GoM, a linguagem tem papel fundamental. Entendendo que o foco principal da arquitetura GoM é a qualidade do modelo e não na linguagem de modelagem, abordagens têm sido desenvolvidas com foco no desenvolvimento de linguagens de modelagem, em especial em linguagens de modelagem para domínios específicos (DSML – do inglês *Domain-Specific Modeling Language*). Uma dessas abordagens é o trabalho de Ulrich Frank (FRANK, 2011) (FRANK, 2013), que propõe diretrizes e um método para apoiar o desenvolvimento de DSML.

Para Frank (2013), o desenvolvimento de uma DSML envolve a criação do metamodelo da linguagem e do processo de modelagem em si. Assim, seu trabalho parte de requisitos mais gerais para DSML e requisitos específicos para a linguagem de metamodelagem. No que se refere aos requisitos gerais, são cinco (FRANK, 2013):

Requisito P1: Os conceitos de uma linguagem de modelagem devem corresponder aos conceitos que os usuários em potencial estão familiarizados. Quanto mais os usuários estiverem familiarizados com os conceitos de uma DSML e sua representação, mais fácil será para eles entenderem e usarem-na adequadamente.

Requisito P2: uma linguagem de modelagem deve fornecer conceitos específicos de domínio, desde que sua semântica seja invariável dentro do escopo de aplicação da linguagem. Isto porque, somente é possível satisfazer todos os usuários em potencial se a semântica de um conceito de modelagem for invariável dentro do intervalo de uso pretendido.

Requisito P3: os conceitos de uma linguagem devem permitir a modelagem em um nível de detalhe suficiente para todas as aplicações previsíveis. Para cobrir outras aplicações possíveis, ela deve fornecer mecanismos de extensão.

Requisito P4: uma linguagem de modelagem deve fornecer conceitos que permitam distinguir claramente diferentes níveis de abstração dentro de um modelo.

Requisito P5: deve haver um mapeamento claro dos conceitos da linguagem com o que se pretende representar. Em um caso ideal, todas as informações que se deseja representar podem ser extraídas do modelo. Deixar para que os potenciais usuários mapeiem os conceitos poderá contribuir para aumentar os riscos e os custos.

Os requisitos gerais para DSMLs apresentados, P1 a P5, focam em boas características que uma linguagem deve ter para ser bem sucedida. Ainda, a criação dessa linguagem deverá usar uma linguagem de metamodelagem. Assim sendo, Frank (2013) define oito requisitos para seleção dela, sendo eles:

Requisito MM1: a linguagem de metamodelagem deve ser suportada por um ambiente de software. O desenvolvimento de uma linguagem de modelagem requer grande esforço. Portanto, ter suporte de um ambiente de metamodelagem pode ser crucial para a economia do desenvolvimento de uma DSML. Um ambiente de

metamodelagem pode suportar a geração de código a partir de metamodelos e permitir a instanciação de metamodelos diretamente para os modelos.

Requisito MM2: os conceitos da linguagem usados em diferentes níveis de abstração, como M2 ou M1, devem ser claramente separados. Se uma linguagem de metamodelagem não permitir distinguir diferentes níveis de abstração necessários para especificar uma linguagem de modelagem, isso não apenas causará confusão, mas também exigirá um esforço substancial para que restrições e ferramentas adicionais resolvam a ambiguidade.

Requisito MM3: a notação gráfica de uma linguagem de metamodelagem deve corresponder a notações gráficas predominantes, por exemplo, de dados ou linguagens de modelagem de objetos. Ao mesmo tempo, a notação deve incluir elementos que permitam distinguir com facilidade um metamodelo de um modelo em nível de instâncias. Muitos projetistas de linguagens estarão familiarizados com o paradigma de modelagem entidade relacionamento. No entanto, representar metamodelos na mesma notação que os modelos no nível do objeto contribuirá para causar confusão entre os níveis de abstração mencionados no requisito anterior.

Requisito MM4: os conceitos de uma linguagem de metamodelagem devem ser complementados por uma linguagem para especificar restrições.

Requisito MM5: os conceitos oferecidos por uma linguagem de metamodelagem devem permitir um mapeamento claro dos conceitos utilizados para o desenvolvimento de software. O uso eficiente de uma linguagem de metamodelagem requer uma ferramenta de metamodelagem correspondente, cujo desenvolvimento é facilitado por um respectivo mapeamento.

Requisito MM6: uma linguagem de metamodelagem deve permitir distinguir entre diferentes níveis de abstrações. Isso inclui, em especial, a distinção entre características de tipos e instâncias correspondentes. Isto porque, embora uma linguagem de modelagem geralmente seja focada na descrição de conceitos (e.g., tipos ou classes) em vez de instâncias específicas, às vezes é necessário expressar características que se aplicam a todas as instâncias de um tipo.

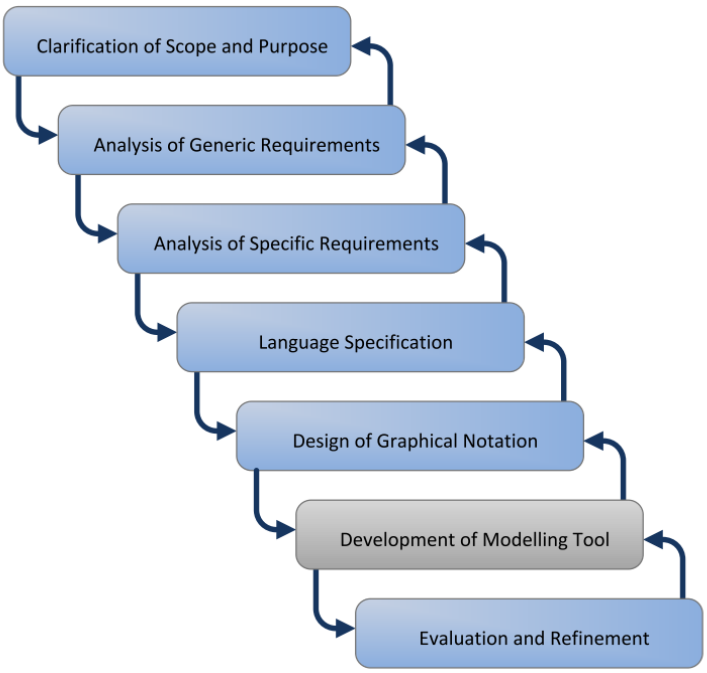
Requisito MM7: uma linguagem de metamodelagem deve fornecer conceitos que permitam representar instâncias. Em casos raros, porém importantes, pode ser necessário modelar instâncias. Somente se uma linguagem de metamodelagem oferecer um conceito correspondente, esse aspecto poderá ser especificado para

uma DSML. Por exemplo: Um DSML para modelar sistemas logísticos pode exigir a representação de cidades específicas.

Requisito MM8: uma linguagem de metamodelagem deve levar em conta a disseminação e padronização. Quanto maior a disseminação da linguagem, mais ela será atrativa em termos econômicos de escala. Na verdade, a padronização contribui para a proteção do investimento.

Os esforços empreendidos por Ulrich Frank (FRANK, 2011) (FRANK, 2013), delinearam requisitos importantes para o desenvolvimento de DSMLs. A partir deles, o autor propôs um método para criação de DSMLs que é ilustrado na Figura 40. O método propõe um ciclo iterativo e a figura apresenta as fases e a sua sequência de execução.

Figura 40 - Fases do método de criação de DSMLs



Fonte: (FRANK, 2013)

A primeira fase, Esclarecimento do Escopo e Objetivo (*Clarification of Scope and Purpose*), tem como meta descrever os principais objetivos do projeto, seu escopo e orçamento. A fase de Análise de Requisitos Genéricos (*Analysis of Generic Requirements*) diz respeito a criação ou compilação de uma lista dos requisitos de qualidade desejáveis para a DSML. O conjunto de requisitos P1 a P5, apresentado anteriormente é um exemplo. Na sequência, a fase de Análise de Requisitos Específicos (*Analysis of Specific Requirements*) é elencada pelos autores do método como sendo uma das mais importantes. Nela, o objetivo é criar uma lista

de requisitos específicos para a DSML e deve envolver os potenciais interessados na linguagem. A partir de experiências prévias com desenvolvimento de DSMLs, o autor sugere o refinamento dos requisitos específicos a partir da apresentação de modelos preliminares da DSML aos interessados na linguagem (FRANK, 2013).

A fase de Especificação da Linguagem *(Language Specification*) corresponde à formalização da sintaxe e da semântica de uma DSML. Nesta fase ocorre a criação do metamodelo da linguagem e nela muitas decisões de projeto são definidas. Nesta fase, o autor do método de construção de DSMLs sugere iniciar com a criação de um glossário de termos-chave do domínio da linguagem que será criada. Nem todos os termos serão conceitos na linguagem, por isso, é importante que a semântica de um conceito da linguagem seja invariável em todo o domínio e no tempo (Requisito P2) para que a DSML seja útil em diversos contextos (FRANK, 2013).

A fase de Projeto da Notação Gráfica (*Design of Graphical Notation*) é considerada desafiadora pelo autor do método pois, em geral, os desenvolvedores de DSML concentram maiores esforços na especificação da linguagem. Assim, propõe-se algumas diretrizes para o projeto da notação gráfica.

Diretriz 1: Defina categorias semânticas de conceitos. Por exemplo, na modelagem de processos, uma categoria-chave pode incluir processos e outra categoria pode incluir eventos.

Diretriz 2: Crie símbolos genéricos por categoria, assim, os conceitos abrangidos por ela podem ser representados por variações do respectivo símbolo genérico. Esta diretriz deve promover a percepção e interpretação apropriadas, portanto, a compreensibilidade de uma notação gráfica.

Diretriz 3: Quanto maior a diferença semântica entre dois conceitos, maior deve ser a diferença gráfica dos símbolos correspondentes.

Diretriz 4: Dê preferência aos ícones às formas. Embora duas formas geométricas possam ser claramente diferentes, por exemplo, um círculo e um retângulo, elas carecem de uma referência ao conceito representado. Incluir uma referência perceptual em um símbolo contribuirá para uma compreensão mais intuitiva de uma notação (Requisito P1).

Diretriz 5: Combine forma (incluindo ícones), cor e texto de forma eficaz. Estudos em Psicologia Cognitiva sugerem que a forma é um instrumento eficaz para realizar a discriminação visual (MOODY, 2009 apud FRANK, 2013).

Diretriz 6: Evite “sobrecarga de símbolos” para evitar ambiguidade e confusão.

Diretriz 7: Evite símbolos redundantes para facilitar a interpretação do modelo.

Diretriz 8: Represente características semânticas monotônicas de um conceito por meio de composições de símbolos. Isto porque, combinar elementos gráficos contribui para uma construção mais sistemática de uma notação gráfica que está alinhada com a semântica dos conceitos relacionados, além de evitar sobrecarregar os usuários com enormes paletas de símbolos.

Diretriz 9: Uma notação gráfica deve incluir símbolos que permitam reduzir a complexidade do diagrama. Um exemplo seria um símbolo usado para representar processos agregados. Reduzir a complexidade visual de um diagrama pode contribuir substancialmente para um melhor entendimento e, portanto, para aumentar a produtividade. O resultado da fase de Projeto da Notação Gráfica é a documentação de notação gráfica, ilustrada por meio de exemplos do diagrama (FRANK, 2013).

A fase de Avaliação e Refinamento (*Evaluation and Refinement*) visa garantir um determinado nível de qualidade, avaliando e possivelmente revisando a DSML. Primeiro, recomenda-se a avaliação em relação aos requisitos genéricos (P1 a P5) e específicos da fase Análise de Requisitos Específicos. Para alguns requisitos, será fácil decidir se eles são cumpridos ou não. Para outros nem tanto, por exemplo, “Os conceitos da linguagem devem ser fáceis de compreender por diferentes grupos de usuários.”. O autor do método recomenda criar cenários de uso para a análise de requisitos. Cada cenário pode servir para analisar se, e como, os requisitos correspondentes são atendidos pela DSML. Para tanto, uma avaliação discursiva é de fundamental importância. Deve-se incluir os potenciais usuários da linguagem e especialistas do domínio. O resultado da fase de Avaliação e Refinamento é o relatório de avaliação e a DSML revisada e aprovada (FRANK, 2013).

A proposta de método e diretrizes para criação de DSMLs de Frank (2013) é fundamentada em sua experiência no desenvolvimento de diversas linguagens. Mesmo assim, o autor destaca que são necessárias mais pesquisas para consolidação deste e outros métodos para criação de DSMLs e sugere que o método seja usado como referência, adaptando-o às necessidades de cada projeto.

2.4 SISTEMAS HOLÔNICOS

Ao se observar qualquer forma de organização social que tenha certo grau de coerência e estabilidade, ver-se-á que ela é **hierarquicamente ordenada**. O mesmo se dá com a estrutura dos organismos vivos e dos seus processos de funcionamento (KOESTLER, 1969). Na área da computação, os computadores (i.e., o hardware) são organizados em uma hierarquia que define a sua arquitetura (STALLINGS, 2010). No contexto de softwares, também a hierarquia entre os seus módulos define sua arquitetura (MEDVIDOVIC; TAYLOR, 2010).

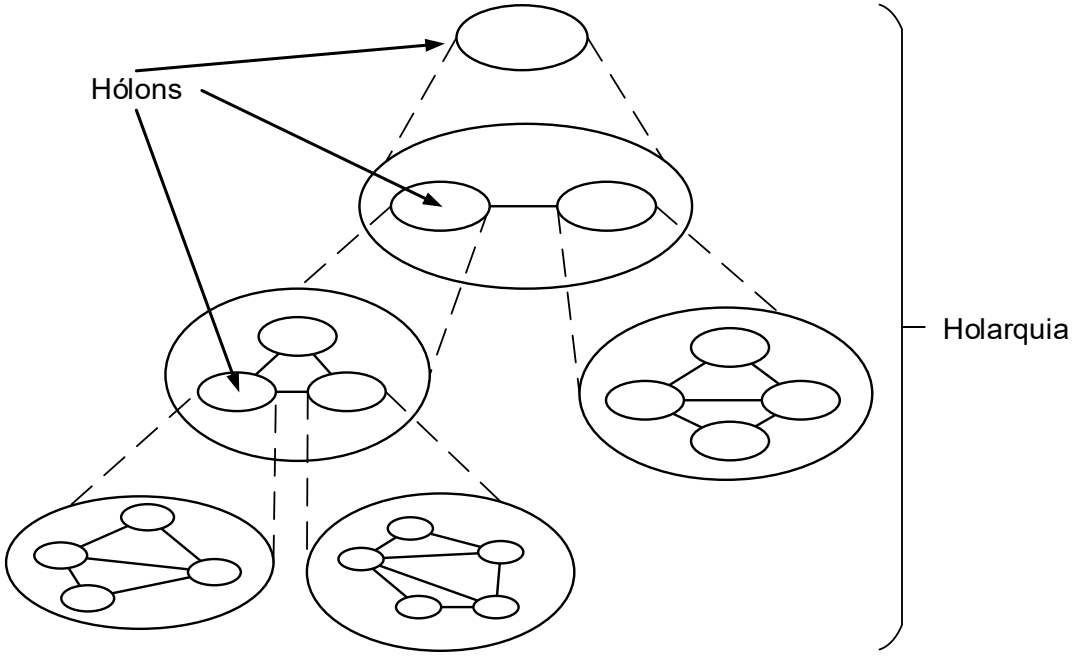
Koestler (1969) desenvolveu uma teoria que lançou um olhar diferenciado às hierarquias dos sistemas. Segundo ele, as primeiras características universais de uma hierarquia são a relatividade e a ambiguidade entre os termos “parte” e “todo”, quando aplicados a qualquer membro da hierarquia. Uma “parte”, como geralmente a palavra é usada, significa algo fragmentário e incompleto, que não teria nenhuma existência por si mesmo. Por outro lado, um “todo” é algo completo em si mesmo, que dispensa qualquer explicação adicional. Mas “todos” e “partes” nesse sentido absoluto simplesmente não existem. O que é encontrado são estruturas intermediárias em diversos níveis e em uma ordem ascendente de complexidade: subtodos que revelam, de acordo com a maneira pela qual são observadas, algumas das características comumente atribuídas aos “todos” e algumas das características comumente atribuídas às “partes”. Por exemplo, os fonemas, as palavras, as frases são “todos” de pleno direito, mas “partes” de uma unidade maior; e assim são as células, os tecidos, os órgãos, bem como as famílias, os clãs, as tribos. Os membros da hierarquia têm dois lados que apontam para direções opostas, um voltado para o lado dos níveis subordinados que é a do “todo” e outro voltado para o lado a quem é subordinado, sendo este o lado da “parte”.

O conceito desenvolvido por Koestler (1969), no qual os membros da hierarquia são “parte” e “todo” ao mesmo tempo, fez com que ele cunhasse o termo hólon (do grego *holos* = todo, com o sufixo *on* que, como em próton ou nêutron, sugere uma partícula ou parte) para denominar esses elementos.

A hierarquia de hólons, denominada por Koestler (1969) **holarquia** (cf. Figura 41), faz com que os níveis intermediários nos sistemas holônicos assumam certa autonomia. Esta autonomia permite que membros de níveis intermediários tomem algumas decisões sem a necessidade de consultar os seus níveis superiores

e, neste caso, podem se comunicar com outros hólons do seu nível hierárquico para atingir objetivos em comum.

Figura 41 - Organização de hólons: holarquia



Fonte: adaptado de (NEGERI; BAKEN; POPOV, 2013)

Os hólons possuem outras características em comum, além da sua dualidade de “todo” e “parte” (WALLACE, 2008):

- Autonomia: cada hólon pode realizar suas próprias atividades sem o controle direto de hólons superiores; no entanto, ainda forma uma parte de, e contribui para o funcionamento geral de um sistema maior.
- Naturalmente distribuído: isto decorre da característica de autonomia dos hólons de realizar suas próprias atividades.
- Simplicidade: cada hólon tem uma tarefa simples e singular para executar, concentrando-se exclusivamente nessa tarefa. O sistema realiza tarefas de maior escala por meio da combinação, cooperação ou competição entre hólons.
- Interatividade: embora hólons funcionem de forma autônoma, sua interação com outros hólons pode produzir fluxos complexos de informações, a fim de atingir os objetivos de cada hólon. Portanto, um hólon deve processar e responder aos dados vindos de outras fontes, bem como fornecer a outros hólons informações solicitadas.
- Hierarquia: como os hólons interagem, a soma de suas ações torna-se maior do que a ação do hólon individual. Todo sistema holônico tem uma finalidade

e está organizado hierarquicamente em relação a sua função e ao controle de seus processos internos.

A teoria de Koestler tem sido usada em diferentes frentes e nas áreas de engenharia e computação. Algumas abordagens são em modelagem organizacional (ZHANG; LI, 2008), gerenciamento da manufatura (VAN BRUSSEL *et al.*, 1998), redes de sensores sem fio (YE *et al.*, 2008), gerenciamento de semáforos de trânsito (MOGHADAM; MOZAYANI, 2011) e sistemas multi-agentes (BOGGINO, 2005). A teoria de hólons é particularmente interessante para sistemas complexos e sistemas distribuídos (WALLACE, 2008) (VALCKENAERS; BRUSSEL; HOLVOET, 2008). Algumas das razões para isso são (WALLACE, 2008):

- Escalabilidade: cada hólon tem propriedades de autonomia, assim pode funcionar com pouco ou nenhum conhecimento de outros hólons. Deste modo, dependendo do sistema, pode-se adicionar hólons sem afetar a operação dos hólons existentes e mantendo a hierarquia, na qual hólons mais abstratos gerenciam hólons mais detalhados.
- Robustez: assim como se pode adicionar hólons, pode-se removê-los sem afetar o funcionamento de outros hólons ou do sistema como um todo.
- Simplicidade de controle: à medida que cada hólon, geralmente, tem uma tarefa simples e singular a realizar, necessitará de um mecanismo simples de controle, que pode ser compreendido mais facilmente quando comparado com um sistema de controle centralizado.

Uma característica importante dos sistemas holônicos é a de que tais sistemas possuem certo grau de coerência e estabilidade. Koestler (1969) apresenta exemplos de sistemas e organizações sociais e biológicas que só existem devido as suas características holônicas. Trabalhos nas áreas de engenharia e computação (BOGGINO, 2005; NEGERI; BAKEN; POPOV, 2013; SIMÃO, 2005; VALCKENAERS; BRUSSEL; HOLVOET, 2008) que se apoiam na teoria dos sistemas holônicos também se caracterízam por serem coerentes e estáveis. Porém, a teoria de sistemas holônicos não aborda como estes sistemas são concebidos.

Na área de engenharia de software existem diversas técnicas e métodos usados na concepção de aplicações. Contudo, a concepção de software é um processo de síntese que envolve esforços do projetista para chegar a uma boa

solução (SUH, 1990). Para auxiliar na concepção de softwares, os métodos mais comuns são iterativos, ou seja, a cada ciclo, o projeto é testado e verificado de acordo com seus objetivos. Os métodos de engenharia atuais, como o *Axiomatic Design* (BATISTA, 2013; PIMENTEL, 2007; SUH, 1997), permitem verificar se uma dada solução é boa ou não. Assim, um desafio é propor métodos ou técnicas que auxiliem os projetistas na concepção de sistemas, em especial os holônicos por terem as características apresentadas.

Outrossim, nas origens do Paradigma Orientado a Notificações (PON), nomeadamente o atualmente chamado Controle Orientada a Notificações (CON) e outrora chamada de Metamodelo de Controle Holônico, estão os Sistemas de Manufatura Holônicos (SIMÃO, 2005; SIMAO; STADZISZ, 2009). Assim, há também indícios históricos de que a concepção de aplicações PON deve considerar a teoria de sistemas holônicos. Adicionalmente, a visão holônica é hierárquica e fractal, permitindo que a concepção possa ocorrer por meio de etapas de detalhamento sucessivas, sendo compatível com as práticas mais comuns de desenvolvimento de software.

2.5 DISCUSSÕES SOBRE O CAPÍTULO

Este capítulo apresentou o embasamento teórico necessário para a compreensão do trabalho desenvolvido. Inicialmente, o Paradigma Orientado a Notificações (PON) foi apresentado, de uma forma diferente das que têm sido usadas, relacionando-o com os conceitos tradicionais de processamento de dados. Na sequência, os elementos da arquitetura do PON foram apresentados, bem como sua dinâmica de operação por meio de notificações.

No âmbito da Engenharia de Software, algumas técnicas e ferramentas haviam sido usadas para o PON, nomeadamente o Desenvolvimento Orientado a Notificações (DON), sendo que o DON foi a primeira iniciativa para definir um método para o desenvolvimento de software usando o PON. O DON baseia-se nos processos de software mais comuns e especializa a UML, por meio de um perfil, para adequar o seu uso ao PON. Como resultado, obteve-se um método iterativo, focado na fase de projeto de software, que conduz o desenvolvedor na elaboração de softwares baseados em PON. Por outro lado, a análise realizada com UML não

beneficia a descoberta das regras no PON, em parte porque a UML não foi desenvolvida para este paradigma. Assim, o processo de identificação de regras torna-se um intenso processo de síntese e requer muito esforço do desenvolvedor. Mesmo assim, o DON continua sendo uma opção válida para especificação de projetos PON uma vez que os aspectos de concepção de software PON podem ser abordados de outras formas, como a exemplo do que esta pesquisa de tese propõe.

O PON relaciona-se com Sistemas Baseados em Regras (SBRs), sendo ambos pertencentes ao agora chamado Paradigma Orientado a Regras (POR), cujos processos de Engenharia de Software não tiveram avanços marcantes. Não há na literatura muitas opções de processos de software para SBRs e afins, como foi evidenciado no Mapeamento Sistemático da Literatura apresentado neste capítulo. Tradicionalmente, são usadas tabelas de decisão e árvores de decisão como ferramentas de modelagem nesse tipo de sistema. Na literatura, encontram-se, também, técnicas de agrupamento de regras como forma de segmentar problemas e criar componentes de regras reusáveis. Apesar de não ter aparecido no mapeamento sistemático da literatura, o projeto mais recente encontrado é chamado HeKatE (NALEPA; LIGĘZA, 2010) que definiu um processo de software voltado a SBRs.

O HeKatE faz uso de dois diagramas, criados por seus autores, que auxiliam no projeto conceitual e lógico para identificar e definir as regras do sistema. Os diagramas do modelo lógico, chamado de XTT2, aparentam ser especialmente úteis, pois, além das regras, definem o fluxo de execução do software modelado. Por outro lado, o XTT2 objetiva a modelagem de todas as situações (i.e., estados) possíveis do software sendo projetado. Neste caso, chega-se à conclusão de que é impraticável (ou ao menos muito difícil) modelar todos os estados de sistemas mais complexos, assim como no POO com o uso de máquina de estados. Também, o diagrama conceitual, chamado de ARD+, modela atributos e os relacionamentos, porém de maneira muito simplificada e com pouca legibilidade (NALEPA; LIGĘZA, 2010).

Conceber ou escolher uma linguagem de modelagem corretamente pode ser fator determinante no sucesso de um projeto de software. Apresentou-se, neste capítulo, o *Guidelines of Modeling* (GoM) que define recomendações para criação de modelos de software com qualidade. Além disso, este trabalho de pesquisa propõem uma linguagem para um domínio específico (i.e. o PON), assim, o capítulo

apresenta, também, o trabalho de Frank (2011, 2013) que propõe recomendações/diretrizes para criação de linguagens de domínios específicos (DSML), bem como para a escolha de sua linguagem de metamodelagem, além de um método para criação da DSML. As recomendações do GoM e o trabalho de Frank são importantes contribuições no tema desta pesquisa de doutorado.

Por fim, este capítulo apresentou as características, vantagens e usos recentes da teoria de Sistemas Holônicos. A caracterização de um sistema como holônico lhe confere coerência e estabilidade. A bibliografia aponta que os sistemas holônicos são uma boa abordagem para desenvolvimento de sistemas complexos. O próprio PON tem em suas origens conceitos dos Sistemas Holônicos, pois surgiu embrionariamente como uma solução de controle para Sistemas de Manufatura Holônicos. Estudos adicionais serão necessários para compatibilizar a teoria de Sistemas Holônicos com as propostas de concepção de aplicações PON recomendadas neste trabalho.

3 METODOLOGIA DE PROJETO DE SOFTWARE ORIENTADO A NOTIFICAÇÕES (NOM)

Este capítulo apresenta a proposta da Metodologia de Projeto de Software Orientado a Notificações, denominada NOM. Para isso, o capítulo foi subdividido em 10 seções. A primeira seção apresenta questões metodológicas de desenvolvimento da própria NOM. A segunda seção de delimita a metodologia em relação ao Processo de Software, descrevendo seus dados de entrada e de saída. Na sequência, a terceira seção apresenta uma visão geral da metodologia em que seus macroprocessos são ilustrados. A quarta seção apresenta a especificação do diagrama criado para a NOM, ou seja, seu metamodelo, denominado Diagrama de Fluxo Holônico (DFH). A quinta seção ilustra as formas de decomposição dos elementos do diagrama. Na sexta seção é apresentado o uso do diagrama na concepção de aplicações para o PON. A sétima seção introduz técnicas de verificação e validação dos modelos da NOM. A oitava seção trata sobre geração de código semiautomatizada. A nona seção discute os dados de saída da NOM e como eles podem ser mapeados para modelos lógicos dos *frameworks* PON existentes. Por fim, a última seção apresenta discussões e considerações sobre a metodologia NOM.

3.1 MATERIAIS E MÉTODOS

O desenvolvimento da Metodologia de Projeto de Software Orientado a Notificações (NOM) ocorreu durante a pesquisa de doutorado apresentada neste documento de tese. Assim, as etapas do desenvolvimento da NOM são, em grande parte, as etapas do método da pesquisa em si. Desta forma, é possível destacar como as etapas de desenvolvimento de linguagens para domínios específicos (DSML) sugerido por Frank (2013) foram usadas, com adaptações, na criação da NOM.

As três primeiras fases do método de Frank, i.e., Esclarecimento do Escopo e Objetivo, Análise de Requisitos Genéricos e Análise de Requisitos Específicos, correspondem à fase de Estudo Exploratório do método da pesquisa (cf. Figura 1), na qual as atividades realizadas foram: definir tema de pesquisa, realizar revisão

inicial da literatura, definir o objetivo da pesquisa e definir os procedimentos técnicos da pesquisa.

As etapas de Especificação da Linguagem e Projeto da Notação Gráfica correspondem à fase de Desenvolvimento no método da pesquisa, cujas atividades, assim como sugere Frank (2013), criaram uma versão inicial da linguagem para avaliação de especialistas e possíveis usuários e posteriores refinamentos. A etapa de desenvolvimento de ferramenta de modelagem proposta por Frank (2013) não estava no escopo da pesquisa, por isso não foi executada.

A etapa de Avaliação e Refinamento proposta por Frank (2013) corresponde à fase de mesmo nome no método da pesquisa. Nela, a avaliação da NOM foi realizada com especialistas em PON e possíveis usuários da linguagem (pesquisadores do PON) em um grupo focal que será apresentado no capítulo de avaliação da tese. Nesta fase, a metodologia foi refinada, à luz das observações da avaliação, e sua formalização foi finalizada.

No que diz respeito a materiais, optou-se pela formalização matemática tanto para definição do metamodelo da linguagem proposta na NOM, quanto para a definição de suas restrições. Isto porque, não foi foco da pesquisa a criação de uma ferramenta de modelagem. Assim, a especificação matemática foi realizada de forma a garantir que uma ferramenta possa ser implementada, independentemente da plataforma que seja escolhida. Os aspectos diagramáticos da linguagem foram elaborados usando a ferramenta Visio da Microsoft. O processo de verificação e validação da modelagem apresentado neste capítulo sugere o uso de diagramas de estados, assim, optou-se pelo uso da ferramenta YAKINDU[25] que permite modelagem e simulação de diagramas de estados.

3.2 ESCOPO DA METODOLOGIA NOM

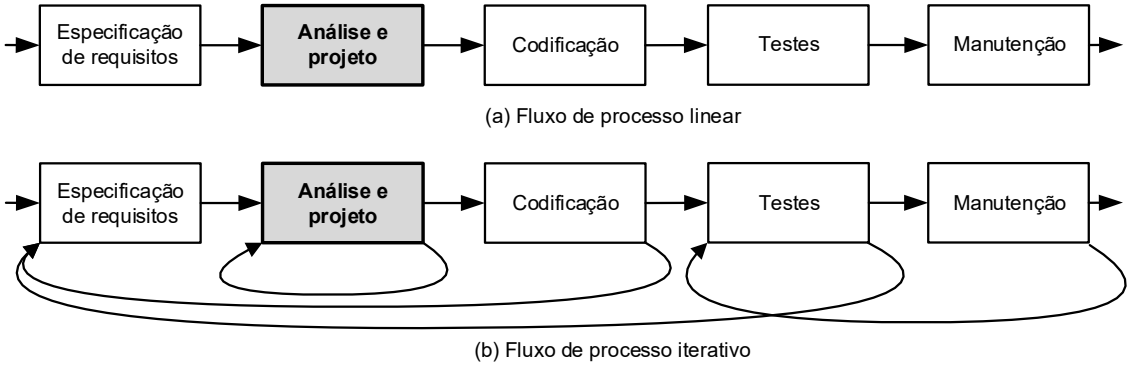
No contexto da Engenharia de Software, processos[26] determinam encadeamentos de atividades, objetos, transformações e eventos, que incorporam estratégias para a criação e evolução de software (SCACCHI, 2002). Desde que o

[25] https://www.itemis.com/en/yakindu/state-machine/
[26] Na literatura processos de desenvolvimento de software são chamados, também, de processos de *software*.

termo Engenharia de Software foi cunhado, na conferência da NATO *Science Committee* em 1968, diversos processos de software foram criados.

Independentemente do fluxo de execução do processo de desenvolvimento de software, seja ele linear ou iterativo, conforme exemplos na Figura 42, o conjunto de atividades, geralmente, envolve: o levantamento de requisitos, a análise e projeto, a codificação, os testes e a manutenção do software (OKTABA; GONZÁLEZ, 1998).

Figura 42 - Localização da NOM em processos de software



Fonte: adaptado de (PRESSMAN, 2011, pág. 54)

A metodologia NOM foi desenvolvida para ser parte de processos de desenvolvimento de software, especificamente como um subconjunto de atividades de “Análise e projeto”, conforme destaque na Figura 42. Esta atividade concebe uma solução de software, na forma de modelos, para atender às exigências do cliente. Além disso, os modelos são uma solução de software passíveis de implementação em uma dada linguagem (OKTABA; GONZÁLEZ, 1998). Os modelos concebidos na NOM são direcionados ao Paradigma Orientado a Notificações.

A atividade de especificação de requisitos, que precede o uso da NOM, deve descrever as exigências para o software desejado, especialmente em termos de funcionalidades. A UML oferece um diagrama também empregado na especificação de requisitos, denominado diagrama de casos de uso (OMG, 2013). Este diagrama descreve os usos pretendidos do software para cumprir os requisitos especificados. A NOM toma como entrada o diagrama de casos de uso e documentos complementares (e.g., descrição de fluxos, identificação de cenários e dicionário) que se denomina Modelo de Casos de Uso.

Com o uso da NOM, o projetista é apoiado na criação de modelos de um software para que possa, posteriormente, ser implementado no PON. Desta forma, a saída da metodologia NOM é um conjunto de diagramas ou modelos de software

PON. Apesar da NOM não ter como foco *à priori* a implementação do software, será visto na sequência deste trabalho que, a partir dos modelos mais detalhados, é possível a geração semiautomatizada de código utilizando a LingPON (FERREIRA, 2015).

3.3 DESCRIÇÃO DA METODOLOGIA NOM

A metodologia NOM aplica o modelo de fluxo iterativo e incremental como praticamente a totalidade dos processos modernos de engenharia de software. Adicionalmente, a metodologia NOM apresenta uma abordagem baseada na teoria dos sistemas holônicos para atuar de maneira integrada nas dimensões estruturais e comportamentais de concepção de software PON. Assim, um projeto de software PON, usando a NOM, se desenvolve por meio de evoluções ou iterações sucessivas a partir de modelos mais abstratos, ou conceituais, para os modelos lógicos do paradigma. Inicia-se a partir de artefatos mais abstratos, do ponto de vista do PON, e os refinamentos evoluem ao encontro das primitivas do paradigma.

O modelo de casos de uso do software PON a ser projetado representa os dados de entrada para a NOM. A partir desse dado de entrada, o projetista desenvolve os modelos que representam o software PON (conforme ilustrado na Figura 43). Os diagramas de casos de uso são considerados artefatos de alto nível e descrevem as diferentes utilizações de um software e que proporcionam os resultados mensuráveis de valor para os atores (WINTERS, 2005).

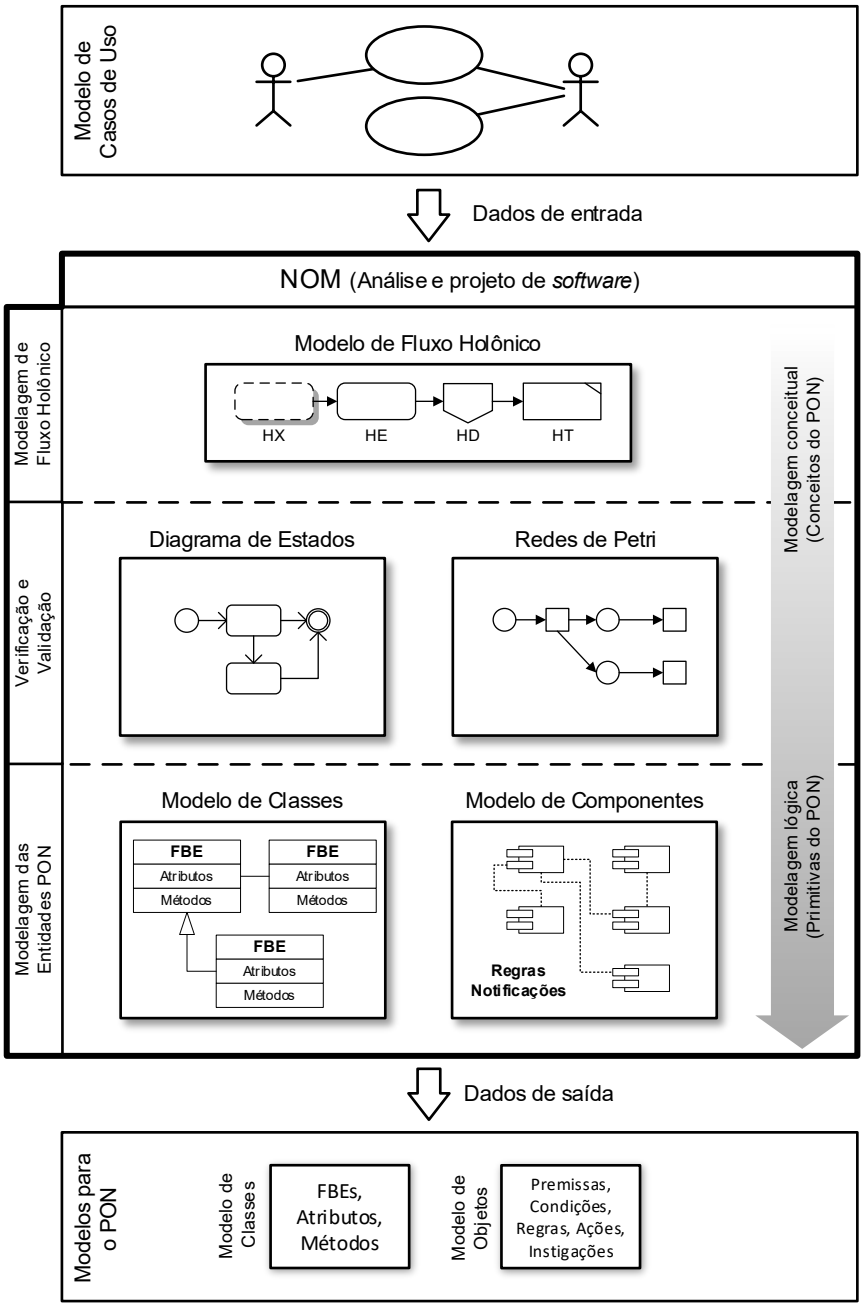
Anteriormente, no Desenvolvimento Orientado a Notificações (DON), o projetista criava o Modelo de Classes para descrever os *FBEs*, Atributos e Métodos da aplicação no PON. A concepção desses elementos requer esforço e experiência do desenvolvedor no PON, assim como a identificação de classes no Paradigma Orientado a Objetos é, também, um desafio para projetistas daquele paradigma (LIU *et al.*, 2003) (MORE; PHALNIKAR, 2012). Entendendo que a concepção desses elementos para o PON representa uma dificuldade na modelagem de software, esta tese propõe uma abordagem por meio de um modelo descrito na forma de um Diagrama de Fluxo Holônico (DFH) que será apresentado na subseção 3.4.

Em suma, o modelo proposto nesta tese, denominado Modelo de Fluxo Holônico (MFH), visa oferecer ao projetista de aplicações PON um mecanismo de

concepção orientado aos princípios do Paradigma Orientado a Notificações. Desta forma, a partir do MFH, os modelos lógicos podem ser concebidos. Assim, os modelos criados anteriormente no DON são incorporados na fase de modelagem lógica da NOM. Portanto, o modelo de classes pode ser usado na NOM para apresentar a estrutura dos *FBEs*, bem como seus Atributos e Métodos. O modelo de componentes pode ser usado para representar a estrutura das Regras, suas condições e premissas, bem como suas ações e instigações. A NOM tem a etapa de "concepção do software" como diferença fundamental do DON, na qual o processo de modelagem por meio do Modelo de Fluxo Holônico auxilia a fazer emergir as entidades do PON. Por fim, para a verificação e validação dos modelos conceituais e lógicos, podem ser usadas Redes de Petri ou Diagramas de Estados.

A Figura 43 ilustra os modelos criados e escolhidos para a NOM na sequência em que aparecem no processo de modelagem de software. O modelo de caso de uso é o dado de entrada da NOM, assim, ele é apresentado no início da figura. O projetista escolhe um caso de uso para iniciar a primeira atividade do processo, que é a Modelagem de Fluxo Holônico. Na sequência, as atividades de verificação e validação auxiliam o projetista no processo de verificação dos modelos concebidos e sua validação em relação ao dado de entrada.

Figura 43 - Visão geral da NOM



Fonte: Autoria própria

A NOM aplica um modelo de fluxo iterativo e incremental, assim, o projetista não precisa desenvolver toda a modelagem de uma atividade para seguir para a próxima. Ao contrário disso, o projetista é encorajado a seguir para as próximas atividades, por exemplo, quando identificar que os artefatos criados no modelo de fluxo holônico podem ser verificados e validados, retornando à modelagem para novas iterações.

A modelagem das entidades PON, última atividade da NOM, depende da plataforma alvo para qual a solução está sendo desenvolvida. Isto porque, há

variação entre os *frameworks* de implementação PON. Assim, a Figura 43 apresenta, como sugestão, o Modelo de Classes e de Componente do DON, cujos modelos foram criados *à priori* para o *Framework* PON C++ 1.0, muito embora possam naturalmente ser reaproveitados para outras materializações (SIMÃO; STADZISZ; WIECHETECK, 2015). A saída da metodologia NOM, apresentada ao final da Figura 43, é o conjunto de artefatos suficiente para codificação da solução concebida em uma plataforma alvo do PON. Neste conjunto estão os diagramas que compõem o Modelo de Fluxo Holônico, devidamente validados e verificados, e os modelos lógicos das entidades PON.

3.4 ESPECIFICAÇÃO DO MODELO DE FLUXO HOLÔNICO - MFH

O desenvolvimento de softwares complexos, frequentemente, emprega uma abordagem por etapas sucessivas de criação de modelos em diferentes níveis de abstração (SUH, 1990). Inicia-se com modelos em níveis mais abstratos com menos detalhes e outros níveis são concebidos a partir deles, com maiores níveis de detalhamento.

Para lidar com a complexidade no desenvolvimento de software, a metodologia proposta neste trabalho de tese se baseia na abordagem holônica, apresentada na seção 2.4. Abordagem essa, fundamentada em conceitos de hierarquia flexível ou holística, permitindo que o seu uso possa ocorrer por meio de etapas de detalhamento sucessivas. Além disso, os sistemas holônicos em si são flexíveis e se aplicam a sistemas biológicos e naturais (WALLACE, 2008), assim como a sistemas complexos e artificiais (VALCKENAERS; BRUSSEL; HOLVOET, 2008) porque o conceito de hólon é generalista.

A generalidade da abordagem holônica permitiu o seu uso no modelo proposto nesta subseção. Adicionalmente, a sua flexibilidade ofereceu facilidade de adequação aos propósitos e primitivas de modelagem do PON, isto é, pôde-se conceber elementos de modelagem inspirados nas primitivas do paradigma em questão. Assim, fazendo uso da abordagem holônica, propôs-se uma técnica de modelagem, denominada **Modelagem de Fluxo Holônico** (MFH), que aplica uma notação chamada **Diagrama de Fluxo Holônico** (DFH). O objetivo da MFH é apoiar o projetista no desenvolvimento de uma solução de software PON a partir da

especificação dos requisitos do software. Com o uso desta técnica de modelagem proposta, o projetista pode identificar o conjunto de regras, *FBEs* e notificações que constitui o software PON sendo modelado.

A técnica de MFH proposta, 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. Isto é, são criados elementos dentro de outros elementos para descrever a proposta de solução, mantendo as características de autonomia, distribuição, simplicidade e interatividade da abordagem holônica. Desta forma, a MFH adota uma organização hierárquica para o desenvolvimento do modelo em que cada iteração torna o modelo mais detalhado e completo.

Os elementos usados na MFH foram inspirados nos três elementos fundamentais do PON, i.e., **regras**, ***FBEs*** e **notificações**. O objetivo é direcionar esforços de modelagem dos projetistas na lógica do próprio paradigma. As **regras** no PON são elementos lógicos criados para definir as **decisões** do software, ou seja, dadas determinadas condições a regra executa determinadas **transações**. Essas transações são definidas nos elementos ***FBE*** do PON. Resumidamente, as regras e os *FBEs* definem as decisões e transações que um software desenvolvido no PON possui. As **notificações**, juntamente com os demais elementos da arquitetura (c.f. seção 2.1.3.1), criam o elo entre as decisões e transações em um software desenvolvido no PON.

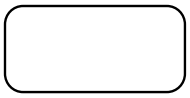
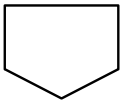
Entendendo que os conceitos de decisão e de transação podem ser vistos como fundamentos que norteiam os elementos do PON, a MFH aplica um princípio de detalhamento dirigido a eles. Este princípio determina que uma decisão ou uma transação deve ser decomposta (ou seja, é detalhada) em um novo conjunto de decisões e transações mais refinadas. Assim, uma decisão de alto nível é detalhada na forma de decisões e transações de menor granularidade que ao final produzem a decisão de mais alto nível. Da mesma forma, uma transação de alto nível é detalhada na forma de decisões e transações de menor granularidade que ao final 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 envolvidas nas decisões ou transações. Desta forma,

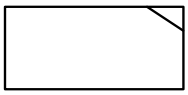
a MFH propõe a construção gradativa de um modelo de fluxo lógico do software PON por meio de uma abordagem holônica, hierárquica e iterativa.

A MFH foi concebida para permitir que o desenvolvimento de softwares PON seja conduzido orientado aos conceitos fundamentais do PON que são as regras, entidades (*FBEs*) e notificações. Ela é, também, baseada em casos de uso, ou seja, o projetista irá desenvolver a solução de projeto por caso de uso, detalhando o fluxo de cada caso de uso separadamente. Isto, entretanto, não impede que entidades e elementos de decisão ou de transação possam ser reutilizados entre diferentes fluxos de casos de uso.

O Diagrama de Fluxo Holônico (DFH) emprega cinco tipos de hólons e dois tipos de relações, conforme segue: Hólon de Entidade Externa (HX), Hólon de Entidade (HE), Hólon Decisional (HD), Hólon Transacional (HT), Relação de Notificação e Relação de Chamada. Cada tipo de hólon e relação possui uma semântica própria e símbolo, descritos no Quadro 8. Os símbolos dos hólons devem ser apresentados com um rótulo textual que identifique o seu objetivo. As relações de notificação e de chamada podem ter um rótulo no entorno da sua apresentação para explicitar o fluxo que por elas passa.

Quadro 8 - Primitivas do Modelo de Fluxo Holônico

Primitiva	**Símbolo**	**Semântica**	
		Descrição	**Dados de entrada e saída**
Hólon de Entidade Externa (HX)		Representa uma entidade externa 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 é equivalente ao “ator” nos diagramas de casos de uso.	Entrada: Nula, dado ou conjunto de dados a ser criado ou modificado na entidade externa (HX). Saída: Nula, dado ou conjunto de dados a ser usado em uma decisão ou transação subsequente.
Hólon de Entidade (HE)		Representa 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.	Entrada: Dado ou conjunto de dados que determinam a criação ou modificação de entidades. Saída: Dado ou conjunto de dados a ser usado em uma decisão ou transação subsequente.
Hólon Decisional (HD)		Representa decisão e pode direcionar o fluxo de execução no software. O projetista pode usar este tipo de hólon para definir tomadas de decisão baseadas em estados ou fatos do software.	Entrada: Dado ou conjunto de dados necessários para tomada de decisão. Saída: Confirmações de sucesso na tomada de decisão ou direcionamentos

Primitiva	Símbolo	Semântica	
		Descrição	**Dados de entrada e saída**
			de fluxo.
Hólon Transacional (HT)		Representa entidade que realiza 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.	Entrada: Ativador, dado ou conjunto de dados a serem transformados, modificados, armazenados. Saída: Nula, confirmação de ação pretendida, dado ou conjunto de dados criados ou alterados em Entidades Externas (HX).
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.	Entrada: Dado ou conjunto de dados que determinam a modificação de entidades. Saída: Dado ou conjunto de dados a ser usado em uma decisão ou transação subsequente.
Relação de notificação		Representa uma relação de fluxo dirigida entre dois hólons por meio de notificação. O projetista deve utilizar esta relação para indicar o encadeamento lógico dos papéis exercidos pelos hólons.	
Relação de chamada		Representa, também, uma relação de fluxo dirigida entre dois hólons. Porém, devem ser usadas pelo projetista quando desejar explicitar que se trata de uma chamada e não de uma notificação. Similarmente à anterior, o projetista deve utilizar esta relação para indicar o encadeamento lógico dos papéis exercidos pelos hólons.	

Fonte: Autoria própria

Os hólons no DFH são interligados por arcos direcionais que definem o sentido 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 Diagrama de Fluxo Holônico (DFH) é formalmente definido por:

(a) Definição. Uma 8-tupla $\mathrm{H} = (\mathrm{HX}, \mathrm{HE}, \mathrm{HD}, \mathrm{HT}, \mathrm{HF}; \mathrm{RN}, \mathrm{RC}, \mathrm{I})$ é chamada de DFH se e somente se:

(i) $\mathrm{HX}, \mathrm{HE}, \mathrm{HD}, \mathrm{HT}$ 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 são chamados Hólons Decisionais, os elementos de HT são chamados Hólons Transacionais e os elementos de HF são chamados de Hólons FBE.

(ii) Sejam $H_1 = \mathrm{HX} \cup \mathrm{HE} \cup \mathrm{HD} \cup \mathrm{HT} \cup \mathrm{HF}$ e $H_2 = (\mathrm{HX} \times \mathrm{HX}) \cup (\mathrm{HX} \times \mathrm{HE}) \cup (\mathrm{HE} \times \mathrm{HX}) \cup (\mathrm{HX} \times \mathrm{HD}) \cup (\mathrm{HD} \times \mathrm{HX}) \cup (\mathrm{HT} \times \mathrm{HX}) \cup (\mathrm{HX} \times \mathrm{HT}) \cup (\mathrm{HF} \times \mathrm{HX}) \cup (\mathrm{HX} \times \mathrm{HF})$

(1) $RN = \{((x,y) \in H_1 \times H_1 : x,y \notin H_2), z : z \in \mathbb{Z}\}$ ou seja, RN é uma terna ordenada, dada pelo produto cartesiano dos conjuntos definidos em H_1 excluindo as relações em H_2 e um número inteiro. Nos casos em que x (i.e., o hólon) possuir mais do que uma relação de notificação de saída, o número inteiro z determina a ordem de execução, podendo usar números repetidos para indicar fluxos paralelos. RN é denominada relação de fluxo holônico por **notificação** de H.

(2) $RC = \{((x,y) \in H_1 \times H_1 : x,y \in H_2), z : z \in \mathbb{Z}\}$, ou seja, RC é uma terna ordenada, dada pelo produto cartesiano dos seguintes conjuntos definidos em H_1, cujas relações pertençam ao conjunto definido em H_2 e um número inteiro. Nos casos em que x (i.e., o hólon) possuir mais do que uma relação de chamada de saída, o número inteiro z determina a ordem de execução, podendo usar números repetidos para indicar fluxos paralelos. RC é denominada relação de fluxo holônico por **chamada** de H.

(3) Sejam RN e RC fluxos de notificação e de chamada de um DFH e $\mathrm{H}_3 = \mathrm{RN} \cup \mathrm{RC}$.

Dois elementos $p, q \in H_3$ são ditos fluxos paralelos de saída de um hólon se e somente se $p((a,b),c) \in H_3 \wedge q((x,y),z) \in H_3 : a = x \wedge c = z$.

Dois elementos $\mathrm{p}, \mathrm{q} \in \mathrm{H}_3$ são ditos fluxos sequenciais de saída de um hólon se e somente se $p((a,b),c) \in H_3 \wedge q((x,y),z) \in H_3 : a = x \wedge c \neq z$.

(iii) I é o conjunto de hólons marcados como iniciais. Hólons são definidos como iniciais no DFH para que sua notação gráfica seja modificada no intuito de melhorar a interpretação dos diagramas. Todos os tipos de hólons podem ser definidos como iniciais e, por padrão, terão a espessura de suas bordas mais grossa que os demais elementos do diagrama.

(iv) O hólon HX não pode ser decomposto. O hólon HX não pode ser decomposto pois representa entidades externas, por exemplo um teclado ou um sistema de software externo.

(v) HE, HD e HT podem ser decompostos, usando a mesma definição formal para DFHs, exceto a inclusão de HX e HF em sua decomposição, isto é, um Hólon Detalhado é definido pela 6-tupla $H_{Detalhado} = (\mathrm{HE}, \mathrm{HD}, \mathrm{HT}; \mathrm{RN}, \mathrm{RC}, \mathrm{I})$.

(vi) HF podem ser decompostos, usando a mesma definição formal para DFHs, exceto a inclusão de HD, HX e HF em sua decomposição, isto é, um HF Detalhado é definido pela 5-tupla $HF_{Detalhado} = (\mathrm{HE}, \mathrm{HT}; \mathrm{RN}, \mathrm{RC}, \mathrm{I})$.

(vii) Ao $HF_{Detalhado}$ aplica-se a restrição de que seus hólons internos não podem ser decompostos.

(viii) Hólons com decomposição consideram todas as relações de chamada e notificação pré-existentes no hólon que originou a decomposição.

Graficamente, os elementos $\mathrm{HX}, \mathrm{HE}, \mathrm{HD}, \mathrm{HT}$ e HF são representados conforme a coluna símbolos do Quadro 8. 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, ambos ilustrados no Quadro 8. Hólons iniciais são diferenciados na notação gráfica pelo seu contorno de maior espessura que dos demais elementos. Os hólons que possuem decomposições são diferenciados graficamente dos demais por um sinal de mais no seu canto superior direito e por uma cor de preenchimento, cujo padrão será cinza.

A Figura 44 ilustra um exemplo hipotético de DFH apresentando o diagrama nas formas declarativa e gráfica.

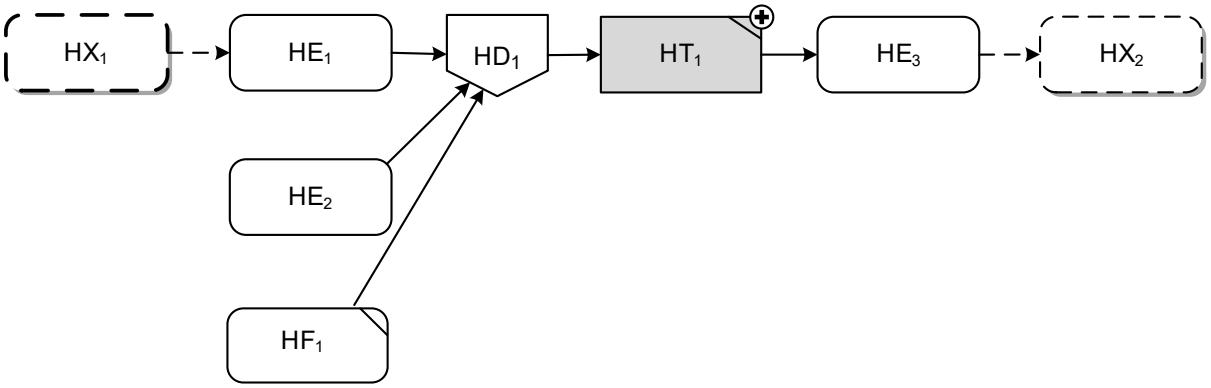
Figura 44 - Representação da notação declarativa e gráfica de um DFH. Os rótulos dos hólons não são apresentados por se tratar de uma ilustração técnica.

Notação declarativa do DFH H

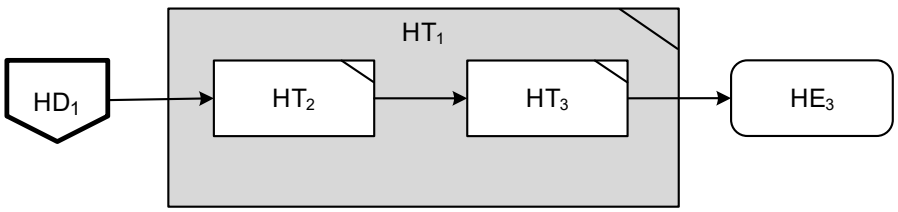
```
H = (HX, HE, HD, HT, HF; RN, RC, I)
HX = { HX1, HX2 }
HE = { HE1, HE2, HE3 }
HF = { HF1 }
HD = { HD1 }
HT = { HT1 }
RN = { ((HE1,HD1),0), ((HE2,HD1),0), ((HF1,HD1),0), ((HF1,HE3),0), ((HD1,HT1),0),
       ((HT1,HE3),0) }
RC = { ((HX1,HE1),0), ((HE3,HX2),0) }
I  = { HX1 }

HT1 = (HE1, HD1, HT1, RN1, RC1, I1)
HE1 = {  }
HF1 = {  }
HD1 = {  }
HT1 = { HT2, HT3 }
RN1 = { ((HD1,HT2),0), ((HT2,HT3),0), ((HT3,HE3),0) }
RC1 = {  }
I1 = { HD1 }
```

Notação gráfica do DFH *H*



Notação gráfica de detalhamento de HT_1



Fonte: Autoria própria

É importante ressaltar que não se dispõe de uma ferramenta de modelagem própria para o DFH. Assim, os diagramas desenvolvidos 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 a decomposição dos elementos segundo a teoria dos sistemas holônicos, tampouco permite a definição de metamodelos para validação dos

modelos. O desenvolvimento de uma ferramenta própria para modelagem de software PON usando a NOM que comporte as etapas de verificação e validação está no planejamento do grupo de pesquisa. Atualmente essas etapas são realizadas com o auxílio de ferramentas adicionais, conforme será apresentado em seção subsequente deste trabalho.

3.5 DECOMPOSIÇÃO DE HÓLONS NO MODELO DE FLUXO HOLÔNICO

A decomposição de hólons é a forma como os níveis de abstração são tratados na modelagem de fluxo holônico. Inicia-se com hólons para representar os conceitos mais abstratos do software PON e se avança, decompondo os hólons, para representar níveis lógicos e próprios do Paradigma Orientado a Notificações. Não há restrições quanto ao número de níveis que o projetista poderá criar na MFH.

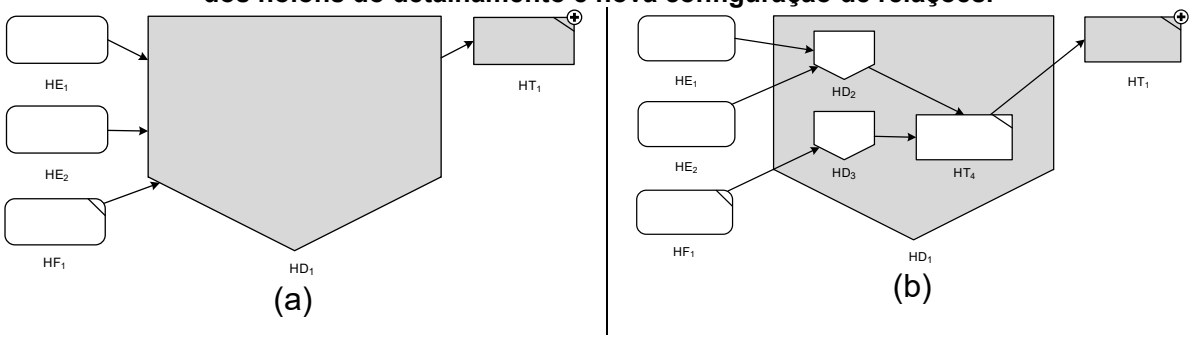
A dinâmica usada na decomposição de hólons é um mecanismo de *zoom-in* e *zoom-out*. Nesse mecanismo, o *zoom-in* cria, ou exibe, um novo nível de decomposição para um dado hólon. Neste novo nível, o hólon em questão é exibido como um *container* para novos hólons. Além disso, também são exibidos os hólons que possuem relação direta com o hólon detalhado ou com seus hólons internos. O *zoom-out* realiza a ação contrária, indo ao nível superior, portanto mais abstrato.

O mecanismo de *zoom-in* e *zoom-out* leva em consideração que dos cinco tipos de hólons do DFH, o HX não pode ser decomposto. Optou-se por não permitir a decomposição do Hólon Entidade Externa (HX), pois entende-se que essas entidades se relacionam com o fluxo, porém por serem externas a ele, não cabe decomposição. Exemplos dessas entidades são impressoras, teclados, monitores, outros sistemas e usuários. O hólon FBE (HF) pode ser decomposto, porém seus hólons internos não podem ser decompostos. Este hólon é recomendado nos casos de reuso de modelos preexistentes ou quando o projetista julgar que a identificação prematura de um elemento FBE do PON pode auxiliar na modelagem de fluxo holônico.

A decomposição de um hólon inicia com a escolha do item que se deseja detalhar. Para exemplificar, será decomposto o HD_1 apresentado no DFH hipotético da Figura 44. Na figura, o HD_1 possui relações de notificação com os hólons HE_1, HE_2, HF_1 e HT_1. Graficamente, o hólon escolhido para decomposição será

apresentado conforme ilustrado na Figura 45 (a). Na sequência, o projetista deverá criar os hólons que compõem o HD_1 de forma a considerar todas as relações que ele possui. Se alguma das relações com o hólon detalhado não for mapeada para algum hólon da decomposição, há um erro de edição do detalhamento. Neste caso hipotético, são 3 relações de notificação partindo dos hólons HE_1, HE_2, HF_1 (lado esquerdo da Figura 45 (a)) e uma relação de notificação para o HT_1 (lado direito da Figura 45 (a)). A Figura 45 (b), por sua vez, apresenta o HD_1 detalhado com a inclusão de novos hólons (2 HDs e 1 HT) e a nova configuração de relações com os hólons previamente relacionados com o HD_1.

Figura 45 - Representação de notação para decomposição do HD_1 antes e depois da inclusão dos hólons do detalhamento e nova configuração de relações.



Fonte: Autoria própria

A decomposição pode continuar com o projetista escolhendo um novo hólon para detalhar. Inclusive, se a abordagem adotada for em profundidade, o projetista pode optar por decompor os hólons recém criados na decomposição anterior.

3.5.1 Decomposição de Hólon Decisional (HD)

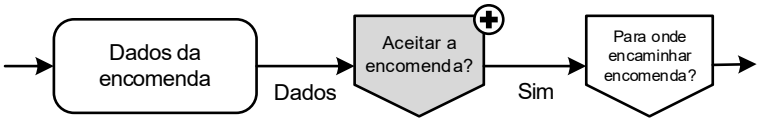
O Hólon Decisional é um elemento chave no processo de modelagem de fluxo holônico. Ele é usado pelo projetista para modelar decisões e pode direcionar os fluxos no modelo. No processo da MFH, o projetista cria HDs para representar decisões que, inicialmente, estão em um nível mais abstrato do modelo e realiza a decomposição desses hólons, se necessário. A cada novo nível de decomposição, ou seja, a cada novo hólon detalhado, o modelo se aproxima dos elementos lógicos do PON. Os HDs de níveis mais abstratos podem ser decompostos em HDs de menor nível ou em um novo conjunto de hólons que atuam naquela decisão.

Nas próximas seções serão apresentados exemplos completos da decomposição de HDs na modelagem de fluxo holônico a partir de um caso de uso. Nesta seção, para simplificar a explicação, o exemplo será ilustrado de forma mais direta, sem apresentar o caso de uso, nem o diagrama de fluxo holônico completo. Assim, tome-se como exemplo a atividade de receber encomendas realizada pelo setor de recepção de uma empresa. Uma das decisões a serem tomadas nesse setor ao atender o entregador da encomenda é a de aceitar ou não a entrega.

Desta forma, a partir dos dados da encomenda, o setor de recepção deverá decidir pelo aceite da encomenda e depois decidir sobre para onde a encomenda deve ser encaminhada. Para aceitar a encomenda, é preciso verificar se o destinatário é um funcionário ou um departamento da empresa e se há autorização de recebimento de encomendas para eles. Nesta descrição simplificada é possível perceber que a decisão de aceitar uma encomenda envolve outras etapas. Porém, em um primeiro nível do modelo, não é recomendado a criação de todos os detalhes.

A Figura 46 apresenta um diagrama parcial, de uma primeira iteração, em que, a partir da atividade descrita anteriormente, os dados da encomenda são modelados como um HE. A decisão de aceitar ou não a encomenda é modelada como um HD chamado “Aceitar a encomenda?” e, caso a encomenda seja aceita, a próxima etapa é modelada pelo HD “Para onde encaminhar encomenda?”.

Figura 46 - DFH parcial da recepção de encomendas do setor de recepção de uma empresa

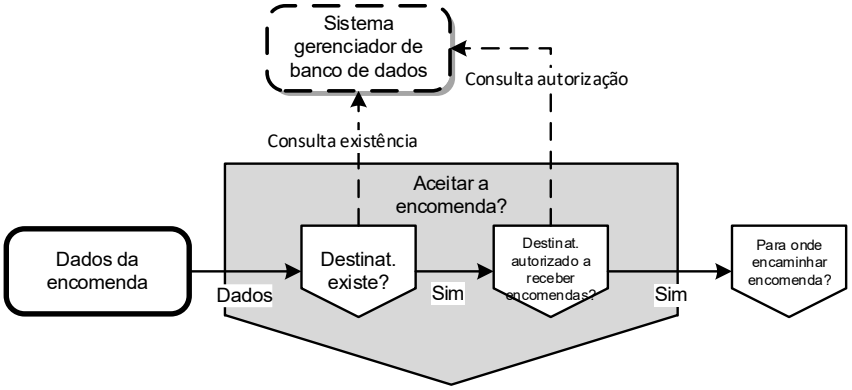


Fonte: Autoria própria

Na sequência, ao escolher o HD “Aceitar a encomenda?” para decomposição, criou-se outro HD para verificar se o destinatário existe e é um funcionário ou um departamento da empresa (Figura 47). A opção pelo Hólon Decisional se deu pelo fato dessa verificação ter que decidir pelo aceite ou não da encomenda, baseado no retorno dessa verificação. Assim, o HD “Destinat. existe?” consulta no HX “Sistema gerenciador de banco de dados”, a partir dos dados do HE “Dados da encomenda” se o destinatário existe. Pertinente perceber que a relação com o HX não existia no nível anterior. De fato, ela foi identificada neste novo nível e deverá ser atualizada no nível superior. Na continuação da iteração, mostrada na

Figura 47, criou-se o HD “Destinat. autorizado a receber encomendas?” para modelar esta atividade que foi descrita como condição para o aceite da encomenda. De forma análoga ao exemplo anterior, optou-se pelo HD.

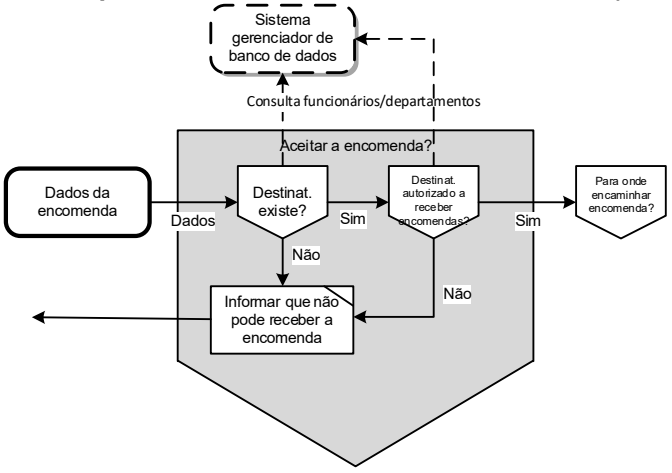
Figura 47 - Decomposição do HD “Aceitar a encomenda?”



Fonte: Autoria própria

Pode-se notar, no exemplo apresentado, a falta dos fluxos para quando alguma decisão é desfavorável. Optou-se pela modelagem dos fluxos de sucesso na primeira iteração. Na continuação do processo de MFH os demais fluxos e hólons vão sendo incorporados. Isto pode ocorrer em uma nova iteração na decomposição do HD “Aceitar a encomenda?”, criando-se fluxos e hólons no mesmo nível ou iniciando um novo nível de decomposição para um hólon existente. A Figura 48 apresenta a primeira situação em que se criou o HT “Informar que não pode receber encomenda” no mesmo nível. Assim, os HDs internos passaram a possuir fluxo para este novo HT para lidar com os casos em que o destinatário não existe na empresa ou se existe, não tem autorização para receber encomendas.

Figura 48 - Decomposição do HD “Aceitar a encomenda?” (nova iteração)



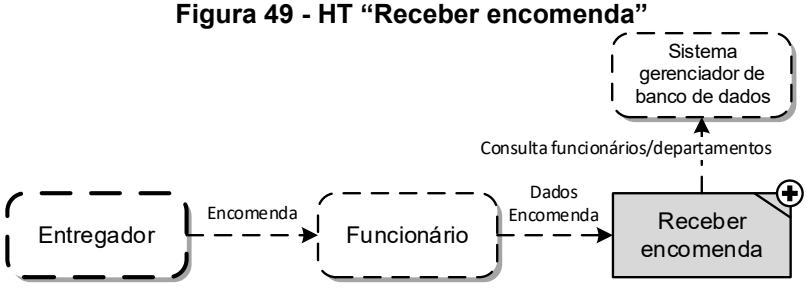
Fonte: Autoria própria

3.5.2 Decomposição de Hólon Transacional (HT)

O HT representa uma entidade para modelagem de ações ou um conjunto delas. O projetista usa este tipo de hólon quando identifica que ações devem ser realizadas no contexto de um DFH. Nos primeiros níveis de modelagem, o HT representa, em geral, um conjunto ações e decisões. Seguindo o exemplo de receber encomenda, apresentada na seção anterior, o conjunto de decisões e ações que envolvem a atividade de receber encomenda pode ser modelada como um HT “Receber encomenda”.

A Figura 49 apresenta o HT “Receber encomenda” e os HXs criados para este modelo. Na parte superior da figura é apresentado o HX “Sistema gerenciador de banco de dados”, pois sabe-se que será necessário realizar consultas a ele a respeito do funcionário ou departamento destinatário da encomenda. O HX “Funcionário” é a entidade que representa a interface com o HT “Receber encomenda” e o HX “Entregador” representa a entidade que iniciou a transação de receber encomenda. Exemplos de uso de HT em níveis mais lógicos, isto é, mais próprios do PON, são a modificação de valores em um HE ou a relação com HXs que modelam dispositivos de entrada e saída de dados (i.e., teclado, monitor, impressora).

Figura 49 - HT “Receber encomenda”

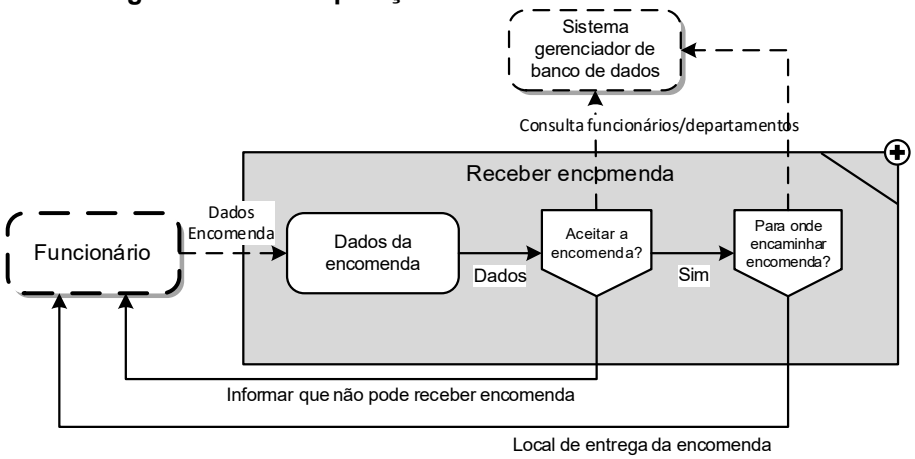


Fonte: Autoria própria

Na sequência, selecionou-se o HT “Receber encomenda” para realizar a decomposição. Na Figura 49 o HT aparece em cinza, com o sinal de mais no canto superior direito, para indicar que ele é um hólon detalhado. Na decomposição desse hólon, conforme ilustra a Figura 50, os HXs que possuíam relação com ele, continuam. A atividade de receber a encomenda envolve o seu aceite e o seu encaminhamento para o destinatário. Assim, a figura apresenta, à direita, os HDs que representam essas decisões. O HX “Funcionário” é responsável por criar a

informação do HE “Dados da encomenda”, que é usada na sequência do fluxo. Perceba que o exemplo apresentado nesta seção é de um nível anterior ao apresentado na Figura 47.

Figura 50 - Decomposição do HT “Receber encomenda”

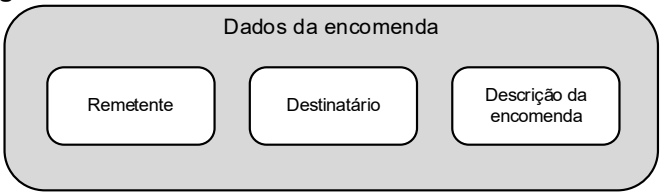


Fonte: Autoria própria

3.5.3 Decomposição de Hólon Entidade (HE)

O HE é usado no DFH para representar os dados usados nos fluxos do software sendo modelado. Em geral, o projetista irá usar este hólon sem realizar decomposições, porém em alguns casos pode se aplicar. Na seção anterior, por exemplo, foi criado o HE “Dados da encomenda” que representa um conjunto de dados. O projetista pode decompor esse HE caso entenda que isso pode facilitar a interpretação do modelo. Por exemplo, o HE pode ser decomposto em novos HEs. A Figura 51 apresenta um detalhamento possível para o HE “Dados da encomenda” em que são criados hólons HE que o compõe. Os HEs são “Remetente”, “Destinatário” e “Descrição da encomenda”.

Figura 51 - Detalhamento do HE “Dados da encomenda”



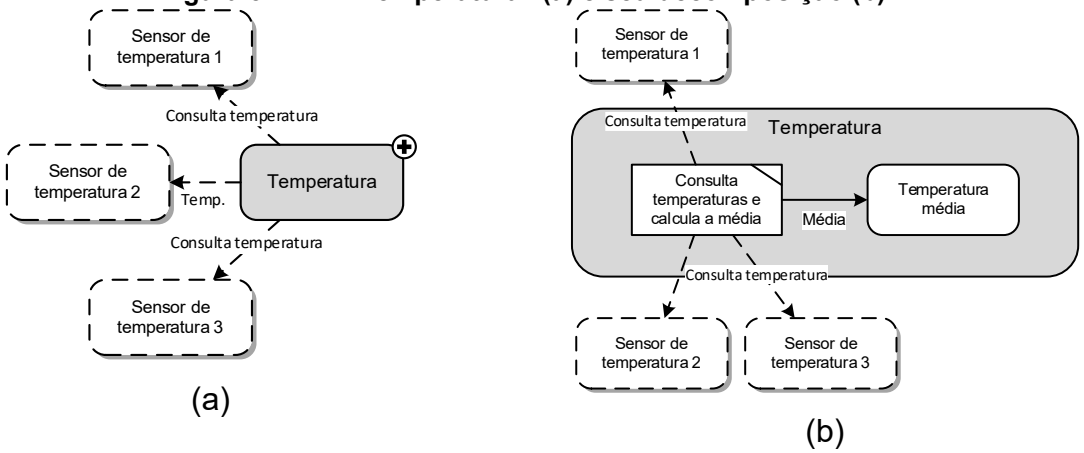
Fonte: Autoria própria

Não é restrito o uso de HEs para decompor um HE, ou seja, pode-se usar HFs, HTs e HDs. Para exemplificar, outra possibilidade para decomposição de HEs

é a criação de um conjunto de hólons que explicitam como alguma informação, ou conjunto de informação, é criado ou modificado em um determinado HE. Por exemplo, na modelagem de um sistema de automação residencial no qual há leituras em sensores de temperatura. O HE que representa a temperatura poderá ser detalhado para ilustrar como a temperatura é obtida.

No exemplo da Figura 52 (a), o HE "Temperatura" é apresentado com relações a HXs sensores de temperatura. Ao analisar este nível, é possível identificar que o HE recupera as temperaturas de diversos sensores. A decomposição deste HE é apresentada na Figura 52 (b) em que foram criados um HT e um HE. O primeiro hólon é responsável por consultar a temperatura dos sensores e calcular a média e no segundo hólon a temperatura média é registrada.

Figura 52 - HE "Temperatura" (a) e seu decomposição (b)



Fonte: Autoria própria

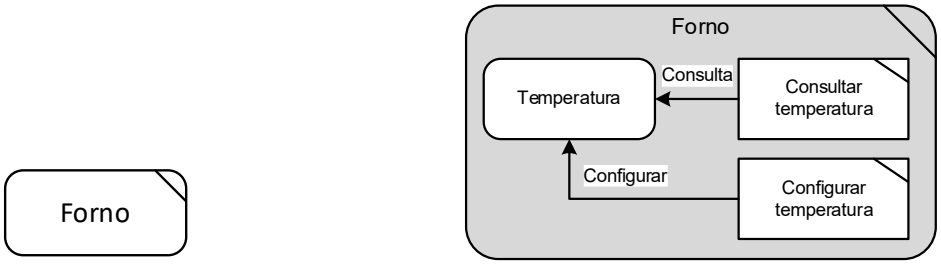
3.5.4 Decomposição de Hólon FBE (HF)

O hólon FBE (HF) representa uma entidade lógica do PON, o elemento FBE. Assim, buscando preservar a semântica deste hólon quando ocorre a sua decomposição, duas restrições se aplicam. A primeira, é a restrição de uso exclusivo de hólons dos tipos entidade e transacional (HE e HT) no HF detalhado. O elemento FBE no PON é composto de Atributos e Métodos e os hólons HE e HT do DFH foram inspirados neles. Assim, a restrição de uso exclusivo desses hólons na decomposição do HF visa preservar a semântica do hólon em relação a sua representação lógica no PON.

A

Figura 53 ilustra um exemplo de HF e sua decomposição. À esquerda da figura é apresentado o HF “Forno”. O HF detalhado é apresentado à direita da figura e é composto do HE “Temperatura” e dos HTs “Consultar temperatura” e “Configurar temperatura”, ambos com relação de notificação com o HE.

Figura 53 - HF “Forno” (a) e sua decomposição (b)



Fonte: Autoria própria

3.6 EXEMPLO DE MODELAGEM USANDO O MODELO DE FLUXO HOLÔNICO

Esta seção apresenta a etapa de modelagem de fluxo holônico, que faz uso do diagrama descrito na seção anterior. Optou-se por apresentar 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 exemplo escolhido, 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 no Quadro 9.

Quadro 9 - Fluxo principal e um fluxo de exceção do Caso de Uso “Sacar Dinheiro”

Fluxo principal
1. O funcionário do banco inicia o ATM. 2. O ATM mostra a sua tela inicial. 3. O cliente verifica visualmente se o ATM está operando normalmente. 4. O cliente insere o seu cartão no ATM. 5. O ATM efetua a validação do cartão. 6. O ATM solicita a senha do cliente. 7. O cliente digita sua senha. 8. O ATM cria uma sessão para o cliente. 9. O ATM mostra as opções para o cliente. 10. O cliente escolhe a opção “Sacar dinheiro”. 11. O ATM solicita o valor desejado. 12. O cliente digita o valor desejado. 13. O ATM verifica se o cliente tem saldo suficiente. 14. O ATM verifica se possui saldo suficiente na máquina. 15. O ATM dispensa o dinheiro solicitado. 16. O ATM mostra a opção de imprimir recibo da transação. 17. O cliente escolhe não imprimir o recibo da transação. 18. O ATM mostra mensagem final e fecha a sessão do cliente. 19. Retorna ao passo 2
Fluxo de exceção: Cartão bloqueado
1. O funcionário do banco inicia o ATM. 2. O ATM mostra a sua tela inicial. 3. O cliente verifica visualmente se o ATM está operando normalmente. 4. O cliente insere o seu cartão no ATM. 5. O ATM efetua a validação do cartão. **Exceção: [Cartão bloqueado]** 6. O ATM registra a tentativa de uso de cartão bloqueado. 7. O ATM exibe uma mensagem de tentativa com cartão bloqueado. 8. Retorna ao passo 2.

Fonte: Autoria própria

No Modelo de Fluxo Holônico (MFH), são criados Diagramas de Fluxo Holônico (DFHs) 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, especificadas em seção anterior. 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 decomposições em cada um deles ou criar elementos mais detalhados e posteriormente agrupá-los em elementos mais

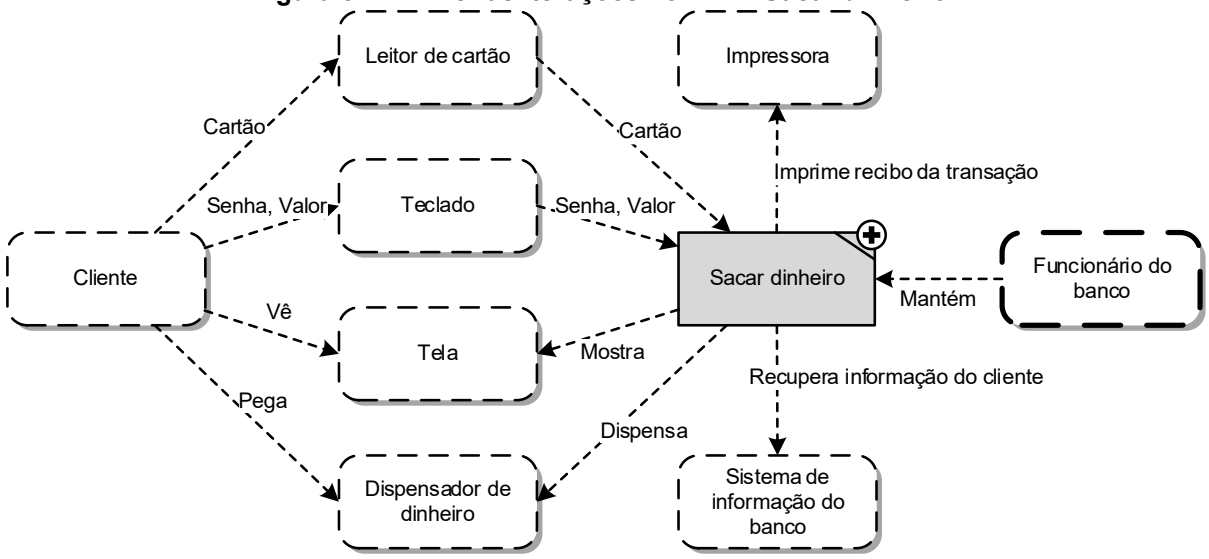
abstratos. Na sequência, optou-se por apresentar uma abordagem mista. Inicia-se com uma modelagem em largura, criando os elementos principais do caso de uso, e na sequência a abordagem em profundidade é usada, na qual os hólons vão sendo decompostos.

Nos Modelos de Caso de Uso, cada caso de uso tem a função de descrever uma utilização do sistema que gera um resultado de valor para o usuário. Assim, o projetista precisa compreender o caso de uso antes de iniciar a criação de um DFH. Neste passo, a atividade do projetista é analisar a documentação do caso de uso e identificar hólons e suas relações, que sejam compatíveis tanto com a descrição do caso de uso, quanto com a semântica do hólon. Na abordagem em profundidade, cada nova iteração no DFH cria visões mais detalhadas do modelo e os hólons inspirados no PON direcionam os esforços para a lógica do paradigma.

O projetista pode iniciar a modelagem do DFH de diversas formas. Por exemplo, na primeira iteração em uma abordagem em profundidade, o projetista pode criar um hólon para representar todo o caso de uso sendo modelado e decompô-lo nas próximas iterações. Neste caso, é indicado que o hólon escolhido para representar o caso de uso seja do tipo HT, HD ou HE, pois estes hólons não possuem restrições de decomposição. Outro exemplo, usando a abordagem em largura, é o projetista criar um conjunto de hólons e suas relações para representar o caso de uso. A modelagem apresentada nesta seção usa a primeira abordagem.

Assim, 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. Por exemplo, o caso de uso descreve no passo 1 que o funcionário do banco inicia o ATM, assim, criou-se o HX “Funcionário do banco” e uma relação dele com o HT “Sacar dinheiro” rotulada “Mantém”. Este mesmo HX, “Funcionário do banco”, é apresentado como um hólon inicial do diagrama (hólon apresentado com borda mais grossa), direcionando a atenção do leitor para ele. Outro exemplo é a relação do HT “Sacar dinheiro” com o HX “Leitor de Cartão” que deverá gerar a indicação de início de transação bancária. O restante da representação visual dessa primeira iteração no DFH é ilustrado na Figura 54.

Figura 54 - Primeiras iterações no DFH “Sacar dinheiro”

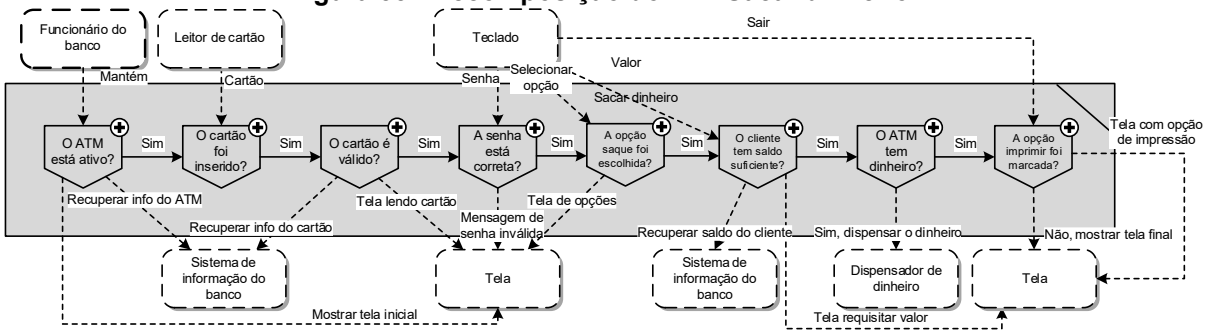


Fonte: Autoria própria

O HT “Sacar dinheiro” está preenchido em cinza e com o sinal de mais no canto superior direito, pois ele será decomposto na sequência. Percebe-se, na Figura 54, que todas as relações apresentadas são do tipo chamada. Isto porque, todas as relações contidas nesta primeira iteração são de, ou para, um HX. As relações de chamada do HT “Sacar dinheiro” com outros HXs serão detalhadas nas próximas iterações da modelagem. As relações entre HXs são de cunho ilustrativo e servem de apoio na interpretação do modelo em relação ao caso de uso.

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 nesta seção), optou-se pela criação das decisões que determinam primeiro o fluxo de sucesso do caso de uso, ou seja, o fluxo principal. Após a análise do caso de uso “Sacar dinheiro” e algumas iterações, chegou-se à decomposição do HT “Sacar dinheiro” apresentado na Figura 55.

Figura 55 - Decomposição do HT “Sacar dinheiro”



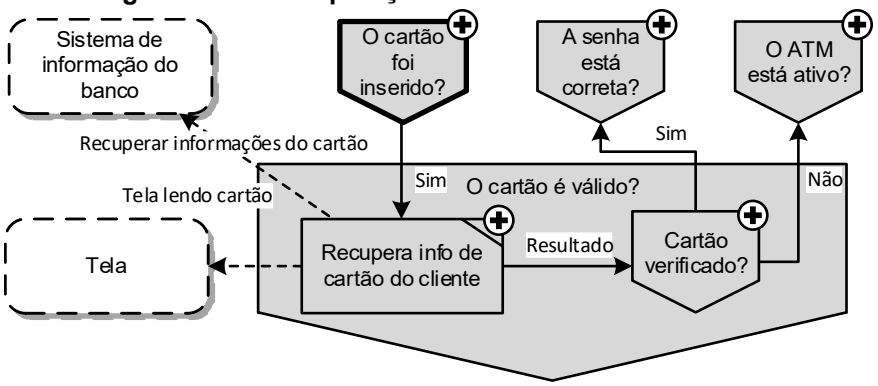
Fonte: Autoria própria

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 decomposto para representar o fluxo alternativo e, por esta razão, estão em cinza com o símbolo “+”. No interior do HD descreve-se textualmente o que ele representa, de preferência na forma de 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 Figura 55 foram criados a partir dos passos 3, 4, 5, 6, 10, 13, 14 e 17 do fluxo no Quadro 9, respectivamente. Na Figura 55 o HT “Sacar dinheiro” é apresentado em cinza, contendo os hólons que fazem parte da sua decomposição. Neste novo nível de detalhamento, os hólons que tinham relação com o HT, passam a ter relação com os hólons criados no detalhamento. Conforme a definição (a)(vi) do DFH, ao decompor um hólon, ou seja, criar um hólon detalhado, as relações que existem com o hólon que originou a decomposição deverão ser associadas a novos hólons no detalhamento, bem como novas relações podem ser criadas para representar os fluxos decompostos internos do hólon detalhado. É possível visualizar esta situação comparando a Figura 54 com a Figura 55. Por exemplo, o HX “Leitor cartão” passa a se relacionar com o HD “O cartão foi inserido?”.

Na sequência do processo, inicia-se a decomposição dos hólons (se necessário) para refletir a especificação de caso de uso dada. O fluxo de exceção apresentado no caso de uso do Quadro 9 reflete uma ocorrência em que se tenta acesso com um cartão inválido. Os passos 1 a 5 são os mesmos do fluxo principal e o HD “O cartão é válido?” representa a verificação do cartão apresentado no passo 5. Assim, detalhou-se o HD “O cartão é válido?” (Figura 56) para continuar refletindo o sucesso na validação do cartão e atuar no fluxo de exceção.

Figura 56 - Decomposição do HD "O cartão é válido?"

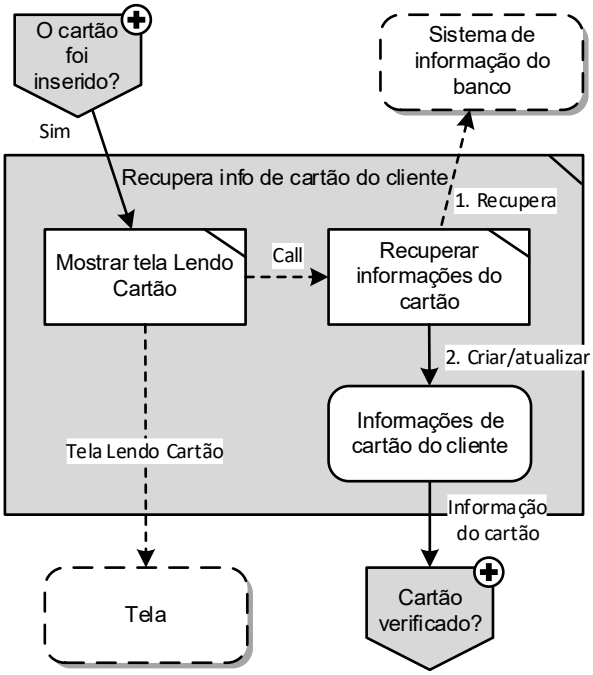


Fonte: Autoria própria

Na decomposição do HD "O cartão é válido?", os 2 HXs e 2 HDs que já se relacionavam com ele se mantém no diagrama do hólon 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 8 do fluxo de exceção descrito no Quadro 9. O fluxo que inicia este hólon detalhado vem do HD "O cartão foi inserido?", por esse motivo este último hólon foi definido como hólon inicial da decomposição. Esse fluxo inicial 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, conforme definições (a)(2)(ii) e (a)(2)(iii) do DFH e podem ser exibidas nos rótulos das relações. No caso desse DFH, a omissão significa que ambos poderiam executar simultaneamente. A decomposição do HT "Recupera info de cartão de cliente" é apresentado na Figura 57.

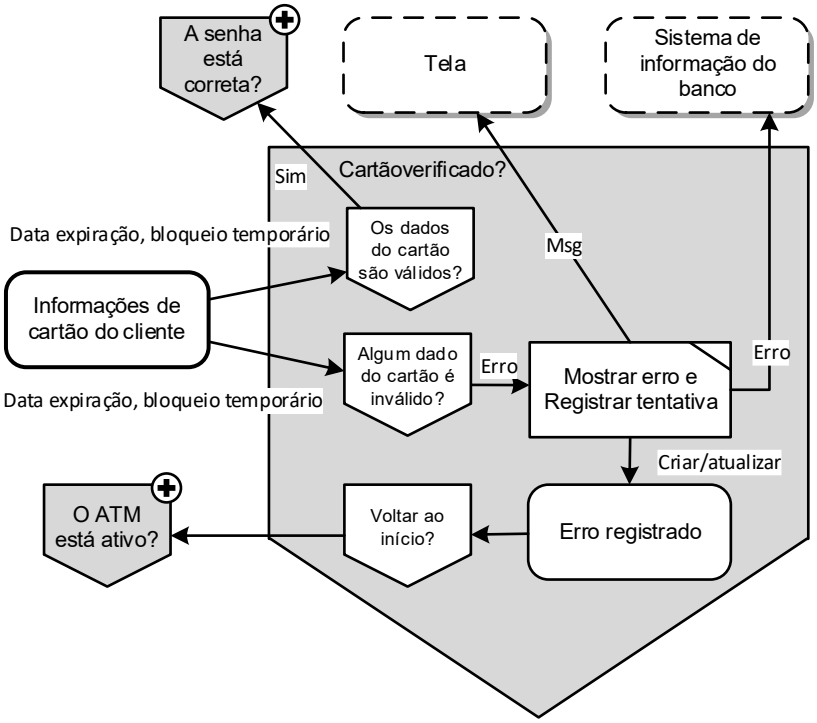
Figura 57 - Decomposição do HT “Recupera info de cartão do cliente”



Fonte: Autoria própria

A decomposição continua com o HT “Recupera info de cartão do cliente”, ilustrado na Figura 57, que 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 Figura 58. 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. Na decomposição 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.

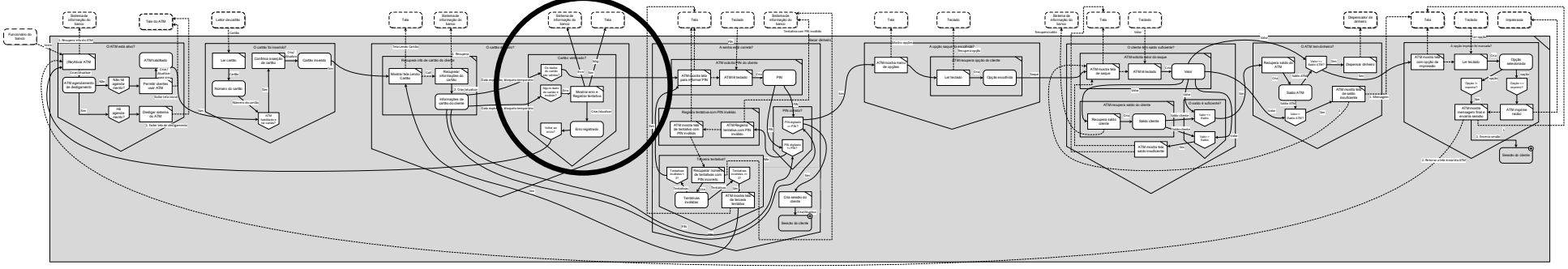
Figura 58 - Decomposição do HD “Cartão verificado?”



Fonte: Autoria própria

A decomposição dos hólons 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 uma visão planificada. A Figura 59 apresenta a visão planificada do DFH criada 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, esta visão permite que o projetista tenha uma referência visual da quantidade de notificações e relações entre os elementos concebidos.

Figura 59 - Visão planificada do DFH “Sacar dinheiro”, destaque no HD “Cartão Verificado?”



Fonte: Autoria própria

A visão planificada, apresentado na Figura 59, é uma visão do DFH criada a partir da união das visões dos hólons detalhados. O objetivo desta forma de apresentação é oferecer uma visão geral da dimensão que o modelo possui. Adicionalmente, em uma futura implementação de software de modelagem para o DFH, a visão planificada pode facilitar a navegação e a interpretação dos modelos criados. Basicamente, a visão planificada de um DFH é definida pelo conjunto de elementos que não possuem decomposição, pelo conjunto de relações de chamada e notificação (excluindo-se aquelas relacionadas a elementos com decomposição) e pelo hólon inicial do DFH. Graficamente, a visão planificada é ilustrada na Figura 59 e apresenta os hólons sem decomposição (em branco) dentro dos hólons de níveis superiores (em cinza). Os hólons em cinza atuam como elementos agrupadores para facilitar a interpretação do modelo.

3.7 VERIFICAÇÃO E VALIDAÇÃO DOS MODELOS

O processo de concepção de software usando o DFH, a partir dos casos de uso, permite ao projetista conceber uma solução PON com elementos em um fluxo que dão origem às primitivas do paradigma (*FBEs*, regras e notificações). Isto porque, os hólons criados para o MFH são baseados nos princípios do paradigma. Assim, espera-se que nos níveis mais detalhados dos modelos, as relações de notificação e chamada entre os elementos do diagrama sejam compatíveis com os padrões de notificações do PON. Por exemplo, no PON as transações são executadas em decorrência das decisões, então manter a compatibilidade significa que no nível mais detalhado não é permitida a relação de notificação partindo de um HE para um HT. Assim, para verificar a compatibilidade de um DFH com a lógica do PON, foram mapeadas as regras apresentadas a seguir.

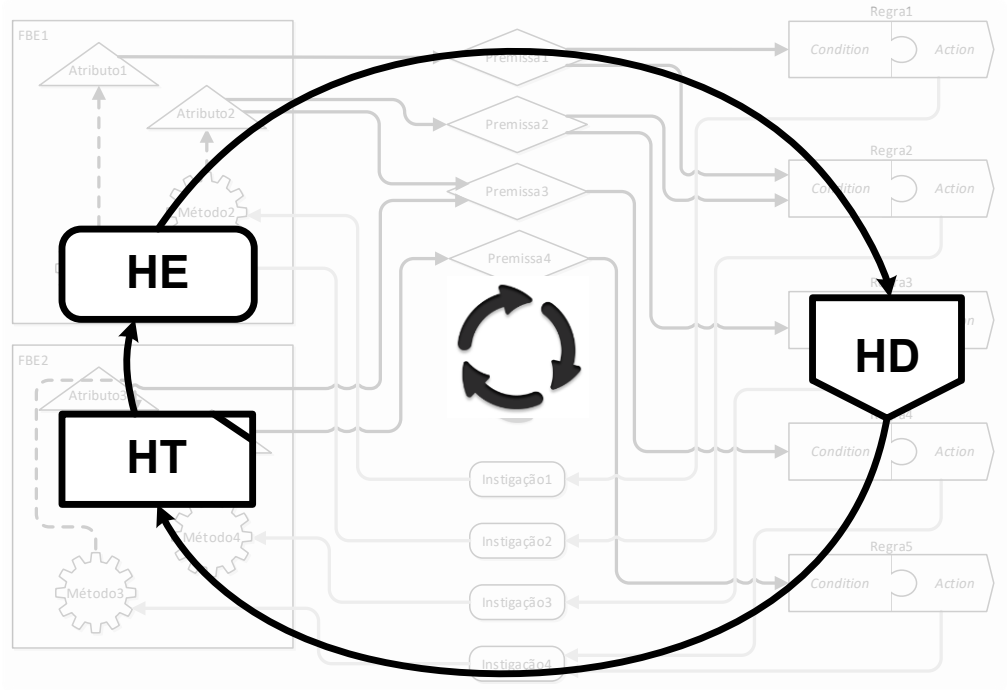
Seja o DFH $\mathrm{H} = (\mathrm{HX}, \mathrm{HE}, \mathrm{HD}, \mathrm{HT}, \mathrm{HF}; \mathrm{RN}, \mathrm{RC}, \mathrm{I})$ e considerando-se somente a visão planificada, ou seja, excluindo-se os hólons que possuem decomposições. O modelo será compatível com o PON se e somente se:

(i) $\forall\big((x,y),z\big) \in RN \Longrightarrow (x,y) \in (HE \times HD) \cup (HD \times HT) \cup (HT \times HE)$

(ii) $\forall\big((x,y),z\big) \in RC \Longrightarrow (x,y) \in (HX \times HX) \cup (HX \times HT) \cup (HT \times HX)$

A regra (i) cria uma correspondência do DFH com o fluxo de notificações do PON (cf. Figura 17). Ou seja, ela compatibiliza as relações de notificação entre os hólons com as notificações do Paradigma Orientado a Notificações. É importante ressaltar, conforme mencionado antes, que os hólons do MFH foram baseados nas primitivas do paradigma (i.e., *FBEs*, regras e notificações). A Figura 60 ilustra a sobreposição dos hólons nas primitivas do PON para evidenciar quais primitivas inspiraram a criação de quais hólons. Na figura sobreposta, os elementos FBE estão à esquerda e as regras à direita. Na figura que sobrepõe, os Hólons Entidade (HE) e transacional (HT) localizam-se sobre elementos FBE e o Hólon Decisional (HD) posiciona-se sobre elementos regras do PON.

Figura 60 - Sobreposição do fluxo de notificações do PON por hólons e relações de notificação da NOM



Fonte: Autoria própria

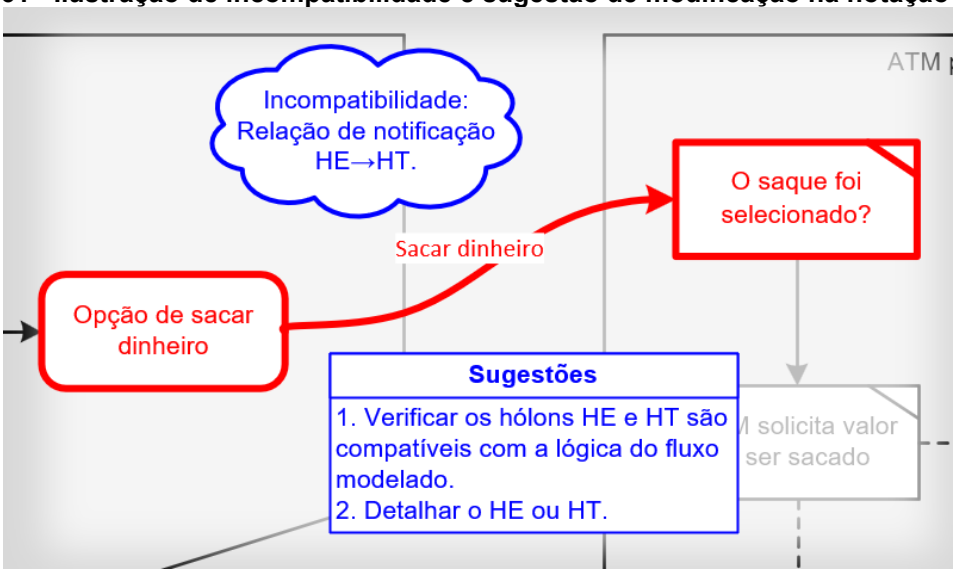
A partir das definições de compatibilidade da visão planificada com o PON, algoritmos poderiam identificar as incompatibilidades e, inclusive, sugerir modificações para que o modelo possa ser decomposto, ou corrigido, para possuir lógica compatível com o Paradigma Orientado a Notificações. O Algoritmo 1 apresenta parte do código (em português estruturado) que faz a verificação sobre as relações de notificação partindo de Hólons Entidade (HE). A Figura 61 exibe como pode ser exibida a incompatibilidade na notação gráfica identificada pelo algoritmo de verificação.

Algoritmo 1 - Verificação de relações de notificação partindo de HE na visão planificada

```
1  Para cada relação de notificação RN partindo de um HE
2    Se a relação RN não for com um HD
3      Erro!
```

Fonte: Autoria própria

Figura 61 - Ilustração de incompatibilidade e sugestão de modificação na notação gráfica



Fonte: Autoria própria

Em relação ao processo de validação dos modelos criados com a NOM, o modelo deve ser compatível com o dado de entrada, ou seja, com os casos de uso. O Modelo de Fluxo Holônico (MFH) apresenta certa similaridade com os casos de uso no que diz respeito ao artefato de descrição de fluxos de forma textual (i.e., Quadro 9). Desta forma, é possível comparar a descrição de fluxos na forma textual de um caso de uso com o respectivo DFH criado para ele. Neste caso, a visão planificada se torna uma importante ferramenta. Para ilustrar, toma-se como exemplo parte da visão planificada do caso de uso “Sacar Dinheiro”, que destaca o fluxo no DFH para os passos do “Fluxo de exceção: Cartão bloqueado” (Figura 62) e os passos do fluxo de exceção apresentados no Quadro 9, que define:

1. O funcionário do banco inicia o ATM.
2. O ATM mostra a sua tela inicial.
3. O cliente verifica visualmente se o ATM está operando normalmente.
4. O cliente insere o seu cartão no ATM.
5. O ATM efetua a validação do cartão.

Exceção: [Cartão bloqueado]

6. O ATM registra a tentativa de uso de cartão bloqueado.

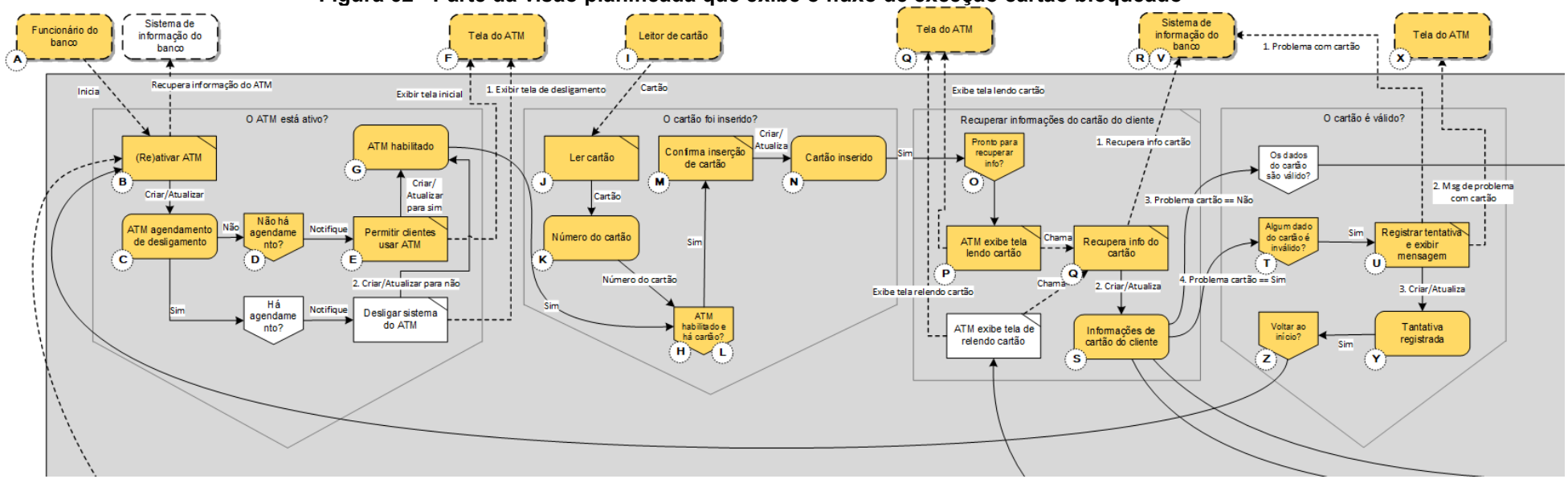
7. O ATM exibe uma mensagem de tentativa com cartão bloqueado.

8. Retorna ao passo 2.

Para facilitar a localização dos elementos da explicação a seguir e que estão na Figura 62, os hólons foram marcados com letras maiúsculas de A-Z. Os passos 1 à 5 refletem passos do fluxo principal do caso de uso. Os passos 1 e 2 descrevem as operações de ligamento e habilitação do ATM e são descritos no DFH pelos fluxos entre as letras A e H em que o funcionário inicia o ATM e o sistema de informação do banco habilita o seu funcionamento. Após, o ATM está habilitado para o uso e estará exibindo a sua tela inicial.

Os passos 3 e 4 do caso de uso descrevem as ações do cliente inserindo o cartão na máquina. Esta ação irá desencadear o fluxo no DFH para iniciar a validação do cartão inserido. Esta validação é representada no DFH das letras I até Q em que o ATM faz o processo de leitura dos dados do cartão inserido. Na letra Q, o HT “Recupera info do cartão” gera a exceção do caso de uso ao recuperar informações do cartão no HX “Sistema de informação do banco” (letra R). O HE “Informações de cartão de cliente” (letra S) é atualizado com informações de que o cartão possui restrições. Assim, o fluxo segue para as letras T até Z em que se registra a tentativa com cartão inválido, se avisa o usuário na tela do ATM sobre a restrição e o fluxo retorna para a tela inicial do ATM, encerrando o caso de uso. Com o uso da visão planificada, o fluxo principal e os outros fluxos do caso de uso podem ser validados para o DFH concebido.

Figura 62 - Parte da visão planificada que exibe o fluxo de exceção cartão bloqueado

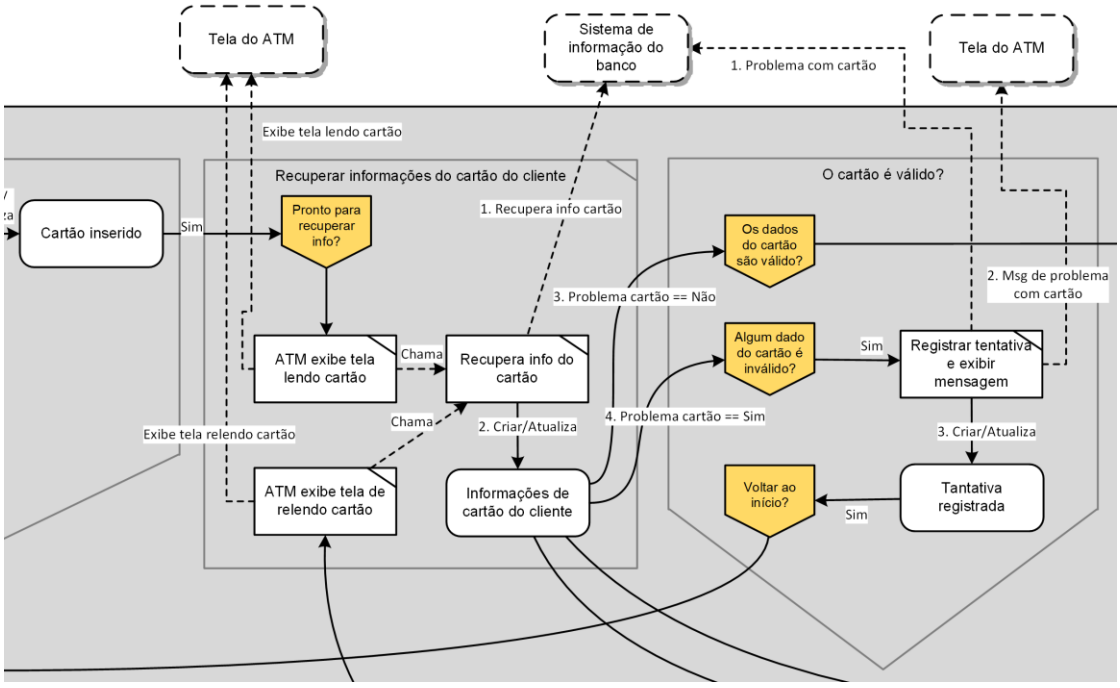


Fonte: Autoria própria

Uma alternativa para validação dos casos de uso no Modelo de Fluxo Holônico é por meio do diagrama de estados. Neste caso, pode-se comparar as sequências de estados com a sequências de fluxos principal e alternativos, ou de exceção, dos casos de uso. Para isso, é preciso identificar quais estados o MFH apresenta, como se dá a transição entre os estados e validá-los junto aos casos de uso. A lógica por regras do PON coloca os elementos decisionais (HDs) em evidência. Isto porque esses elementos definem os caminhos dos fluxos de processamento. Então, pode-se inferir que a partir dos HDs ocorrem as trocas de estado. Basicamente, os estados que serão validados com os casos de uso estão antes e depois dos elementos decisionais e o próprio elemento decisional é o ponto de transição entre eles.

Para criar o diagrama de estados de uma visão planificada, usa-se os HDs como pontos de trocas de estados, ou seja, os HDs serão as transições no novo diagrama. Os demais hólons e relações que antecedem ou sucedem ao HD são agrupados ao estado antecedente ou sucessor, respectivamente. Para ilustrar, toma-se como exemplo parte da visão planificada do caso de uso “Sacar dinheiro” e parte do diagrama de estados criado a partir dele, apresentados na Figura 63 e na Figura 64, respectivamente.

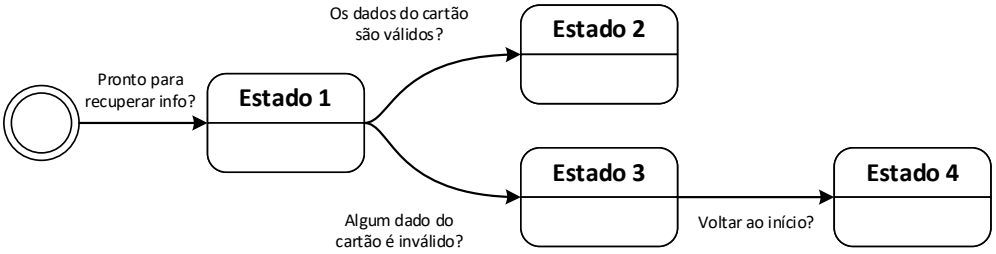
Figura 63 - Parte da visão planificada escolhida para exemplificar o mapeamento do MFH para o diagrama de estados



Fonte: Autoria própria

A Figura 63 apresenta os quatro HDs destacados. A partir do estado inicial, cria-se a transição “Pronto para recuperar info?” para um novo estado “Estado 1”. Na Figura 64 o estado inicial representa a situação antes da ocorrência do HD “Pronto para recuperar info?” e o estado “Estado 1” representa a situação após a ocorrência desse HD. Na sequência, são criadas as transições dos próximos HDs “Os dados do cartão são válidos?” e “Algum dado do cartão é inválido?” e os respectivos estados 2 e 3 que são as consequências dessas decisões. A Figura 63 ainda apresenta o HD “Voltar ao início?” cuja transição criada leva ao novo estado 4. O processo descrito deve ocorrer para toda a visão planificada e ao seu final apresentará o diagrama de estados similar ao que é apresentado na Figura 65.

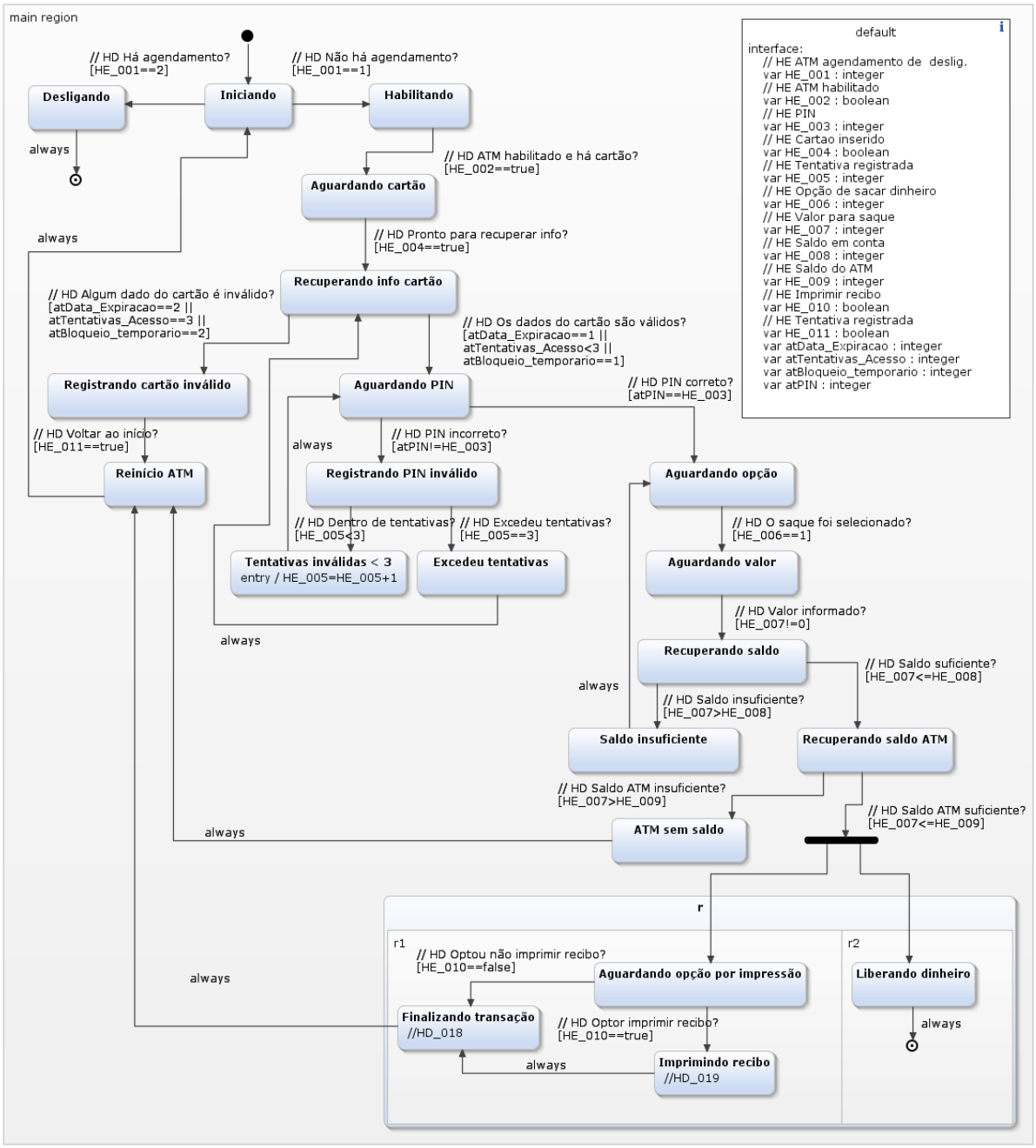
Figura 64 - Parte do diagrama de estados criado a partir da parte da visão planificada apresentado na Figura 63.



Fonte: Autoria própria

Com o diagrama de estados apresentado na Figura 65, é possível realizar a validação do MFH junto ao caso de uso que o originou. Para ilustrar esse processo, toma-se o mesmo exemplo apresentado anteriormente do “Fluxo de exceção: Cartão bloqueado”, cujos passos são apresentados no Quadro 9. O primeiro passo do cliente é precedido no MFH pela habilitação do terminal ATM, sendo compreendido pelos estados “Iniciando”, “Habilitando” e chegando ao estado “Aguardando cartão”, que antecede o passo 4 do caso de uso. Nesse passo, em que o cliente insere o cartão no ATM, o estado é alterado para “Recuperando info cartão”. Na sequência do fluxo (passo 5), o ATM faz a validação do cartão, alcançando a exceção, o que gera um novo estado “Registrando cartão inválido” (passos 6 e 7) e por fim leva ao estado “Reinício ATM” (passo 8) que retorna ao estado inicial do diagrama de estados.

Figura 65 - Diagrama de estados criado a partir da visão planificada para o caso de uso “Sacar dinheiro”



Fonte: Autoria própria

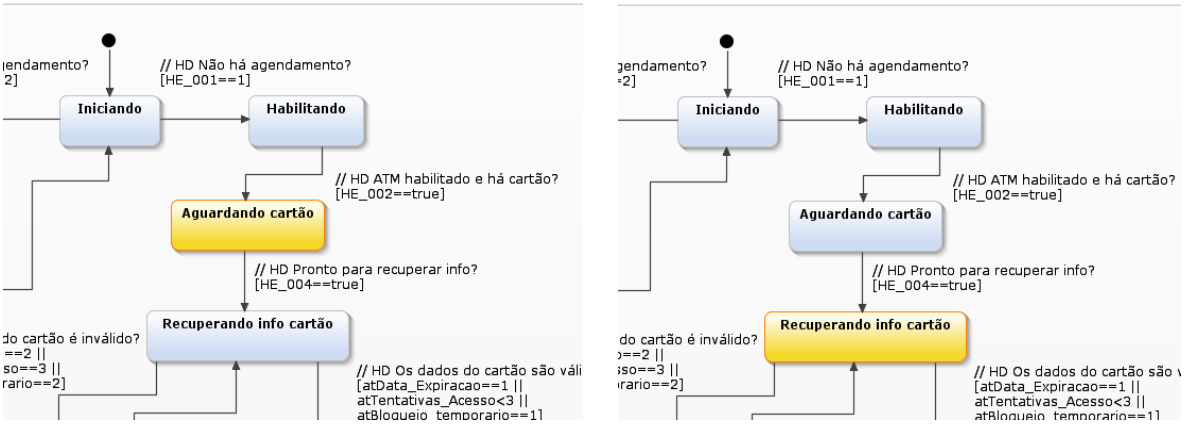
O uso de ferramentas que possam simular as transições do diagrama de estados ou criar cenários de testes pode auxiliar na validação dos casos de uso, tornando o processo menos suscetível a erros. Por exemplo, o diagrama ilustrado na Figura 65 foi criado usando o aplicativo YAKINDU[27], que possui as funcionalidades citadas, além da modelagem do próprio diagrama.

[27] https://www.itemis.com/en/yakindu/state-machine/

Para simular ou criar cenários de testes no diagrama de estados é necessário definir condições para as transições. Neste caso, como as transições entre estados representam elementos decisionais, definiu-se nelas, como condição de guarda, a condição do próprio elemento decisional. Assim, no momento da simulação ou teste, o aplicativo YAKINDU irá considerar essas condições para que as transições sejam ativadas.

A Figura 66 (lado esquerdo) ilustra o momento da simulação no aplicativo em que o estado “Aguardando cartão” é alcançado, sendo este, o estado que antecede o passo 4 do “Fluxo de exceção: Cartão bloqueado”. Na sequência, a ação do cliente de inserir o cartão no ATM (passo 4) gera um novo estado. Isto é simulado no aplicativo alterando o valor de HE_004 para “true”, que corresponde à condição de guarda da transição “HD Pronto para recuperar info?”. Assim, o estado em destaque na simulação passa a ser o “Recuperando info cartão” Figura 66 (lado direito).

Figura 66 - Simulação de estados usando aplicativo YAKINDU



Fonte: Autoria própria

A validação dos casos de uso também pode ser realizada com a criação de testes unitários no aplicativo YAKINDU. A partir do diagrama de estados, o projetista pode criar testes para cada fluxo do caso de uso. O Algoritmo 2 exemplifica o teste criado para o fluxo de exceção: Cartão bloqueado.

Basicamente, para cada teste cria-se um método do tipo `operation` e definem-se os parâmetros do teste, ou seja, os valores que serão verificados como condições de guarda e, partindo do estado inicial, define-se a sequência esperada de estados para o fluxo do caso de uso que se esteja testando. No exemplo do Algoritmo 2 foram definidos cinco parâmetros, sendo eles `HE_001 = 1` indica que não há agendamento de desligamento do ATM, `HE_002 = true` indica que o ATM está

habilitado ao uso, `HE_004 = true` indica que o cartão foi inserido no leitor de cartão, `atBloqueio_temporario = 2` indica a identificação de cartão bloqueado e `HE_011 = true` indica que a tentativa de acesso com cartão bloqueado foi registrada. Na sequência, usa-se os comandos `proceed 1 cycle` para proceder um ciclo na máquina de estados e `assert active` para verificar se o estado corresponde ao estado esperado no fluxo do caso de uso sendo testado.

Ao executar o teste, o aplicativo apresenta um relatório confirmando os resultados. Nos casos em que o teste não for bem sucedido é apresentado em que estado o erro ocorreu. Por exemplo, se o teste apresentado no Algoritmo 2 tiver o parâmetro modificado para `HE_004 = false`, indicando que o cartão não foi inserido, a execução do teste no aplicativo apresentará o seguinte erro: "*Assertion failed: assert active (^default.main.Recuperando_info_cartao) : Line 25. Expected: Recuperando info cartao, actual: [Aguardando cartao]*", indicando que o estado esperado era "Recuperando info cartão", porém estava no estado "Aguardando cartão".

Algoritmo 2 - Código para teste unitário do fluxo de exceção: Cartão bloqueado

```
testclass WithdrawByHD for statechart ^default {

	// Teste do fluxo de exceção: Cartão bloqueado
	@Test
	operation testCartaoBloqueado() {
		// Definição dos parâmetros do teste
		HE_001 = 1
		HE_002 = true
		HE_004 = true
		HE_011 = true
		// Atributo com valor indica cartão bloqueado
		atBloqueio_temporario = 2
		// Iniciar o teste
		enter
		// Primeiros itens de inicialização do ATM (opcionais)
		// pois não fazer parte do fluxo de exceção: Cartão bloqueado
		assert active (^default.main.Iniciando)
		proceed 1 cycle
		assert active (^default.main.Habilitando)
		proceed 1 cycle
		// Passo 1 "O cliente verifica se o ATM está operando normalmente."
		assert active (^default.main.Aguardando_cartao)
		// Passo 2 "O cliente insere o seu cartão no ATM."
		proceed 1 cycle
		assert active (^default.main.Recuperando_info_cartao)
		// Passo 3 "O ATM efetua a validação do cartão."
		proceed 1 cycle
		assert active (^default.main.Registrando_cartao_invalido)
		// Passos 4, 5 e 6 do fluxo de exceção: Cartão bloqueado
		proceed 1 cycle
		assert active (^default.main.Reinicio_ATM)
```

```
            exit
        }

}
```

Fonte: Autoria própria

3.8 GERAÇÃO SEMIAUTOMATIZADA DE CÓDIGO

Dentre os objetivos desta pesquisa não consta a geração de código para uma plataforma de execução do PON. Porém, no decorrer do desenvolvimento do trabalho, percebeu-se que a partir dos modelos DFH detalhados, verificados e validados, pode-se mapear a solução para a linguagem de programação do PON, LingPON[28], da chamada Tecnologia LingPON (i.e., linguagens, compilação unificada e geradores de códigos) que permite gerar código inclusive para os *frameworks* do PON (FERREIRA, 2015; RONSZCKA, 2019) como camada intermediária.

Isto posto, optou-se por apresentar este mapeamento de DFH para LingPON como resultado dos esforços de pesquisa de desenvolvimento desta tese. A geração de código para o PON considera a visão planificada (ex. Figura 59). Em especial, o processo de verificação e compatibilização com o PON, definido na seção 3.7, garante que os elementos dessa visão, que possui o nível mais detalhado, estejam em uma lógica compatível com os princípios do Paradigma Orientado a Notificações. Assim, a primeira etapa é o mapeamento dos elementos da visão planificada para elementos do PON, conforme definições a seguir:

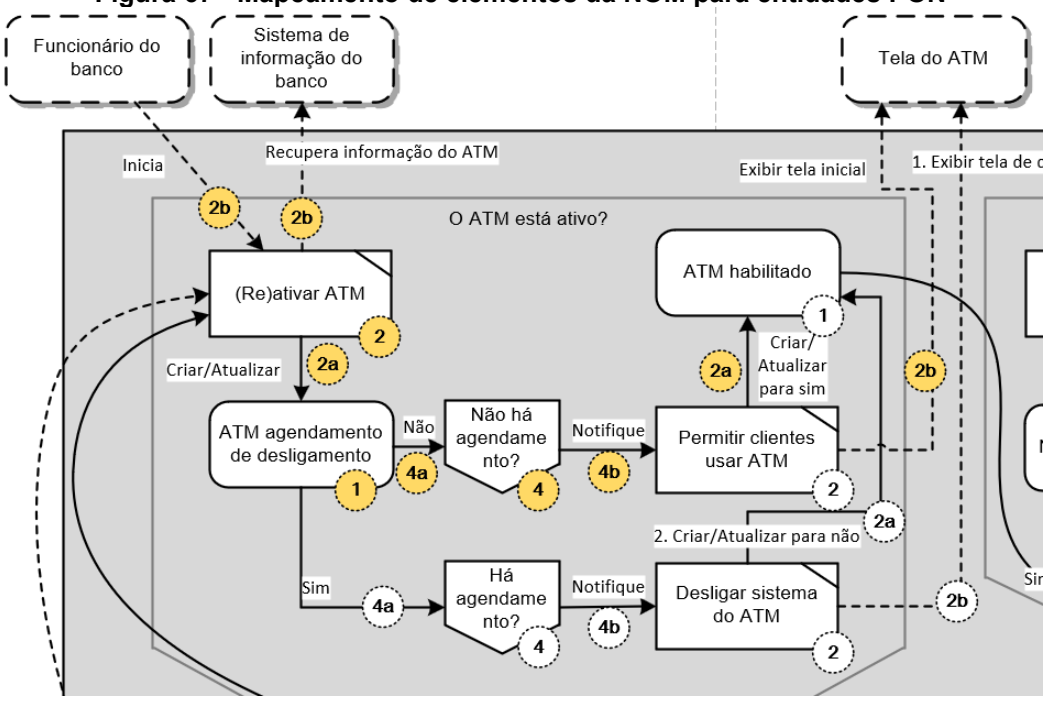
1) Para cada Hólon Entidade (HE) da visão planificada, define-se um FBE com um atributo no PON.
2) Para cada Hólon Transacional (HT) da visão planificada, define-se um FBE no PON.
    a. Para cada Relação de Notificação (RN) do HT com HE, cria-se no FBE métodos PON correspondentes às modificações possíveis.
    b. Para cada Relação de Chamada (RC) do HT com HX ou HT, cria-se no FBE um método para implementar a lógica da chamada com o respectivo HX ou HT.

[28] A tecnologia LingPON possui a versão 1.0 e a 2.0, ambas criadas a luz de um método para criação de linguagens e compiladores para o PON chamado de MCPON (Ronszcka, 2019). No âmbito deste presente trabalho, foi considerado a LingPON disponível na Tecnologia LingPON 1.0, muito embora seja também naturalmente factível considerar a LingPON da Tecnologia LingPON 2.0.

3) Para cada Hólon FBE (HF) da visão planificada, define-se um FBE no PON com atributos e métodos definidos no HF.
4) Para cada Hólon Decisional (HD) da visão planificada, define-se uma Regra no PON.
    a. As Relações de Notificação (RN) que chegam no HD definem as *Premises* e a *Condition* da Regra no PON.
    b. As Relações de Notificações de saída do HD para o HT definem a *Action* e as *Instigations* da Regra no PON, sendo criadas *instigations* para cada relação de chamada ou notificação existente do HT.

Para exemplificar, a Figura 67 ilustra parte da visão planificada da NOM para o caso de uso “Sacar dinheiro”, com os itens de mapeamentos acima mencionados e destacando aqueles cujo código será criado em LingPON.

Figura 67 - Mapeamento de elementos da NOM para entidades PON



Fonte: Autoria própria

Na LingPON utilizada, cria-se, primeiro, o código para definição dos elementos FBE, cf. (FERREIRA, 2015). Assim, no exemplo do Algoritmo 3, as linhas 1 a 5 ilustram a criação do FBE para o HE “ATM agendamento de desligamento” e as linhas 7 a 16 ilustram a criação do FBE para o HT “(Re)ativar ATM”. Neste último FBE, foram criados 2 métodos PON, o primeiro que altera o valor do HE para 1 e o segundo que altera o valor para 2, correspondendo ao não agendamento de

desligamento do ATM e ao agendamento de desligamento, respectivamente. Outros dois métodos foram criados no FBE para as relações de chamada (RC) com os HX “Funcionário do banco” e “Sistema de informação do banco”.

Após a definição de todos os FBEs na LingPON, faz-se a instanciação desses elementos. O código foi suprimido no algoritmo por questões de simplificação, mas foi criada uma instância para cada elemento definido com o mesmo nome, alterando a primeira letra pela sua equivalente em minúsculo (ex: HE_001 foi instanciado como hE_001).

A partir dos FBEs instanciados, definem-se as regras. As linhas 18 a 28 ilustram a regra criada a partir do HD “Não há agendamento?”, na qual a condição é não haver agendamento de desligamento, representada pelo *Attribute* `atHE_001` no estado `1` (linha 21 do algoritmo). A Relação de Notificação deste HD com o HT “Permitir clientes usar ATM” define a *Action* e as *Instigations* da *Rule* (linhas 25 e 26 do algoritmo). Os mesmos passos devem ser realizados para todos os elementos da visão planificada, produzindo o código LingPON para o caso de uso como resultado.

Algoritmo 3 - Código criado a partir do mapeamento da visão planificada do caso de uso ilustrado na Figura 67

```
01 fbe HE_001
02     attributes
03            integer atHEn_001 0
04     end_attributes
05 end_fbe
06
07 fbe HT_001
08     methods
09            method mtHT_001_1(atHE_001 = 1)
10            method mtHT_001_2(atHE_001 = 2)
11            method mtHT_001_3()
12                   begin_method end_method
13            method mtHT_001_4()
14                   begin_method end_method
15     end_methods
16 end_fbe
17 [...]
18 rule HD_001
19   condition
20      subcondition scHD_001
21         premise prHD_001 hE_001.atHE_001 == 1
22      end_subcondition
23   end_condition
24   action
25      instigation in1HD_001 hT_002.mtHT_002_1();
26      instigation in2HD_001 hT_002.mtHT_002_2();
27   end_action
28 end_rule
```

Fonte: Autoria própria

É importante frisar que, no momento, não há uma ferramenta computacional que faça o mapeamento da visão planificada para o código LingPON. O Algoritmo 3 foi criado a partir da interpretação das etapas de mapeamento introduzidas nesta seção. Porém, como as definições descrevem os passos de criação do código, a criação de uma ferramenta poderá ocorrer com base nelas.

Adicionalmente, define-se este processo como geração de código semiautomatizada pois alguns códigos precisam ser completados pelo projetista ou desenvolvedor e esta é a segunda etapa do processo. A característica de verbosidade de concepção da NOM implica que na geração semiautomatizada de código seja necessário a interação na definição do tipo dos atributos. Por exemplo, o HE “ATM agendamento de desligamento” tem como saída os fluxos “sim” e “não”.

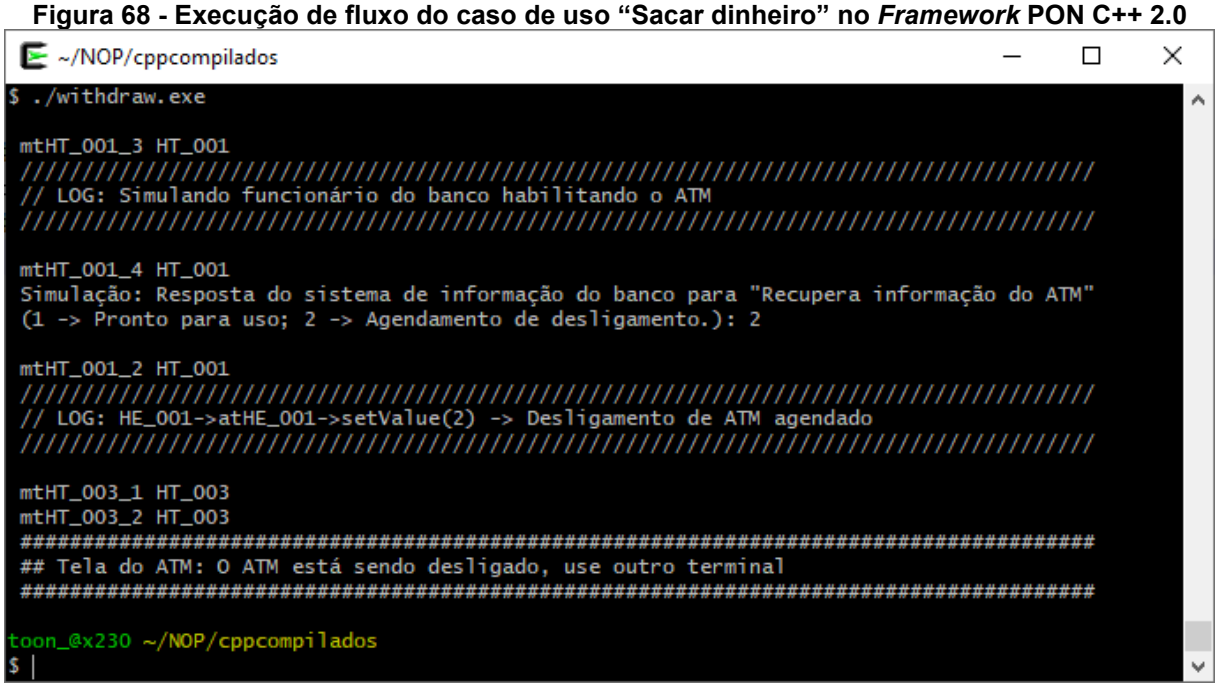
No código LingPON, o atributo criado para representar o HE é do tipo inteiro (linha 3 do algoritmo), tendo valor inicial 0 e o valor 1 representando o “não” e o valor 2 representando o “sim”. Outro ponto de interação necessário é nos métodos dos FBEs. Isto porque, são códigos que dependem da plataforma na qual o código será executado. Em geral, são interfaces com os dispositivos de entrada e saída de dados e devem ser implementados entre as instruções `begin_method` e `end_method` (cf. linhas 12 e 14). Neste exemplo em questão, os códigos a serem implementados se referem à inicialização do ATM à recuperação de informações do sistema de informação do banco, ambos envolvendo entidades externas (HX). Em tempo, faz-se a ressalva de que as evoluções na própria LingPON e na Tecnologia LingPON como um todo tendem a contribuir na resolução destas questões, cf. (RONSZCKA, 2019).

Conforme mencionado antes, na Tecnologia LingPON, a LingPON é usada para a geração de código via um processo de compilação unificado, em cada versão da tecnologia, que permite geração de código para alvos distintos. Assim, deve ser definido previamente qual a plataforma alvo de execução PON será usada quando realmente for gerar código executável. Na dita Tecnologia LingPON (1.0 ou 2.0 cf. o caso), há compilador da LingPON para as seguintes plataformas (RONSZCKA, 2019): (a) *Framework* PON C++ 1.0; (b) *Framework* PON C++ 2.0; (c) *Framework* PON C++ 3.0 adaptado (Multithread & PON IP); (d) *Framework* PON Java; (e) *Framework* PON C#; (f) C++ Namespace com notificações *monothread*; (g) Assembly NOCA; e (h) *Framework* PON Erlang/Elixir. Isto dito, conforme o caso, há a necessidade de complementação do código. Isto ocorreria, sobretudo, nos Métodos que venham a ser interface com alguma plataforma visada. Na sequência,

com o código em LingPON, o desenvolvedor poderá realizar os processos de compilação e execução do código, sendo esta a terceira, e última, etapa do processo de geração de código semiautomatizada antes da execução na plataforma alvo.

No âmbito deste trabalho, a plataforma escolhida para teste foi o *Framework* PON C++ 2.0, considerada na Tecnologia LingPON 1.0 (esta usada neste presente trabalho) e também Tecnologia LingPON 2.0. A Figura 68 ilustra a execução do código na plataforma alvo que simula o agendamento de desligamento do ATM. Para fins de demonstração e verificação de funcionamento, foram incluídos códigos adicionais para imprimir na tela quais métodos são executados. Nos casos de interação com entidades externas, por exemplo o sistema de informação do banco, foi usado um recurso de simulação, no qual o usuário digita no teclado qual seria o retorno de tal entidade externa (cf. `(1 -> Pronto para uso; 2 -> Agendamento de desligamento.): 2` ilustrado na Figura 68).

Figura 68 - Execução de fluxo do caso de uso "Sacar dinheiro" no *Framework* PON C++ 2.0



Fonte: Autoria própria

O processo de geração semiautomatizada de código e sua execução em uma plataforma do PON pode facilitar o desenvolvimento do software. Isto porque, possíveis problemas não identificados nas etapas de verificação e validação anteriores poderão ser testados diretamente em uma plataforma alvo do paradigma. Adicionalmente, a geração de código semiautomatizada a partir de um modelo da

NOM é um indicativo de sucesso da metodologia em direcionar os esforços do projetista ou desenvolvedor na criação de soluções de software para o PON.

3.9 DISCUSSÕES SOBRE A CODIFICAÇÃO NO PON

Uma das diretrizes da Metodologia de Projeto de Software Orientado a Notificações (NOM) é "... produzir artefatos suficientes para que o software possa ser implementado no PON". Assim, esta seção discute os principais aspectos de cumprimento desta diretriz, levando em consideração que alguns deles já foram abordados em seções anteriores. Em especial, destaca-se a apresentação da visão planificada e seu processo de verificação e validação. Essa visão é, de fato, o principal artefato a ser considerado para a criação do código para o PON. Isto porque, apesar de apresentar elementos conceituais, o MFH contém as descrições dos diferentes fluxos de execução do software e suas primitivas se equivalem àquelas do metamodelo do PON.

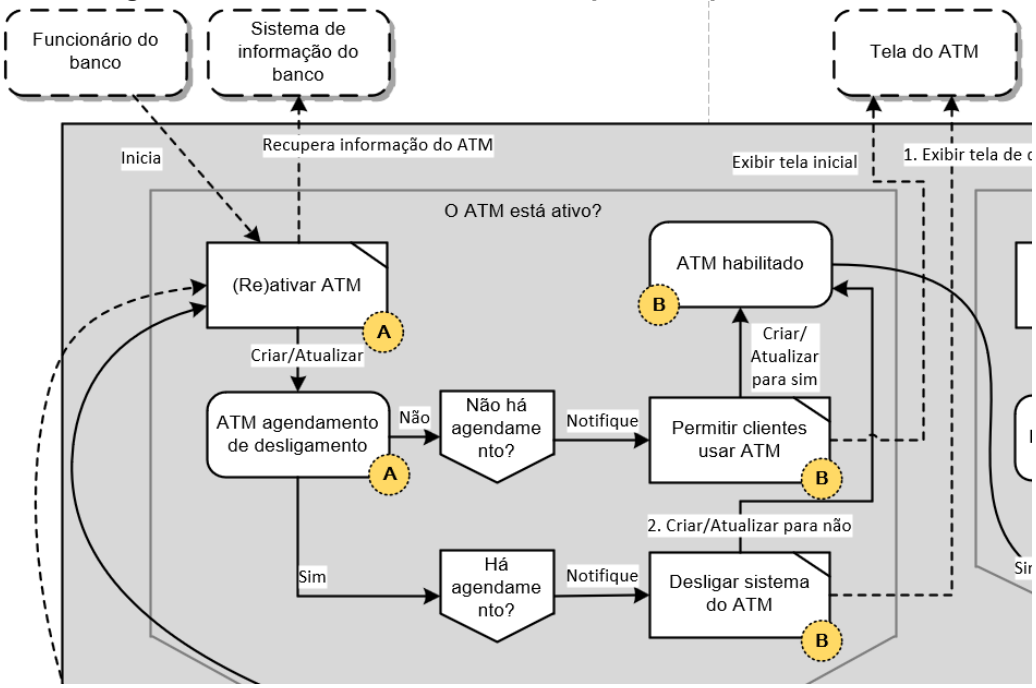
A seção anterior introduziu uma sequência de passos ilustrando como transformar elementos do MFH em elementos do PON. Os Hólons Entidade (HE) e Hólons Transacionais (HT) são transformados em FBEs. Porém, a conotação semântica do elemento FBE no PON vai além de agrupar métodos e atributos. O FBE é criado no PON como uma representação de uma entidade real ou computacional, conjugando seus métodos e atributos de forma coesa. Desta forma, os passos indicados na seção anterior, apesar de não serem incorretos, são insuficientes para lidar com esta característica particular do paradigma.

No processo de concepção da NOM, o desenvolver tem a seu dispor o elemento Hólon FBE (HF). Este elemento lógico do PON pode ser usado quando for identificado prematuramente a sua necessidade ou por reuso de modelos de outros projetos. De outra forma, a identificação de FBEs é tarefa do projetista ou desenvolvedor a partir da sua experiência com o Paradigma Orientado a Notificações e mesmo com o Paradigma Orientado a Objetos conforme entidades (ou objetos) reais ou abstratos que ele quer tratar a luz das entidades que realmente existem ou que se está abstraindo. Neste último caso, a visão planificada também é ponto de partida para auxiliar na identificação dos FBEs. As Relações de Notificação (RC) entre HT e HE podem indicar que os elementos fazem parte de um mesmo

FBE. Assim, uma primeira análise é a identificação dos conjuntos de HTs relacionados com HEs.

Por exemplo, a Figura 69 ilustra a parte inicial da visão planificada para o caso de uso “Sacar dinheiro” em que a marcação “A” identifica um HT com relação a um HE. A marcação “B”, nessa mesma figura, identifica dois HTs com relação a um mesmo HE. Na sequência, o projetista avalia se os fluxos nos conjuntos ensejam a criação de um FBE. No conjunto “A” da Figura 69, o HT e o HE se referem a uma operação interna do ATM. Assim, é possível presumir que os dois elementos pertençam a um mesmo FBE. Para o conjunto “B” da mesma figura, o raciocínio é similar, inclusive, como também se trata de uma operação interna do ATM, os conjuntos “A” e “B” podem fazer parte de um mesmo FBE.

Figura 69 - Identificação de elementos para composição de FBEs

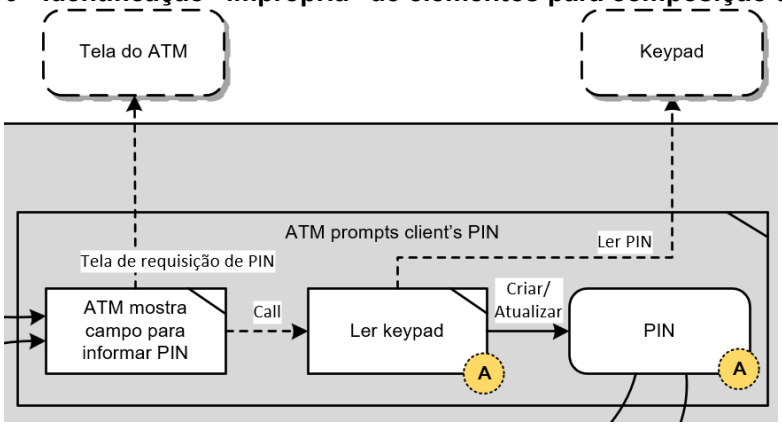


Fonte: Autoria própria

O papel do projetista na etapa de descoberta de FBEs é importante, pois, mesmo com a facilidade de identificar da visão planificada as relações de notificação entre os elementos HT e HE, nem sempre a relação significa que os elementos pertencem a um mesmo FBE. Para exemplificar, a Figura 70 ilustra outra parte do mesmo caso de uso “Sacar dinheiro”. Na figura, o conjunto identificado “A” pode não fazer parte de um mesmo FBE. Isto porque, a interpretação pode ser que o HT “Ler keypad” faz parte de uma entidade computacional que representa o *keypad*

enquanto o HE “PIN” faz parte de uma entidade computacional que representa a sessão do cliente.

Figura 70 - Identificação “imprópria” de elementos para composição de FBEs



Fonte: Autoria própria

Por sua vez, os elementos HD na visão planificada representam regras no Paradigma Orientado a Notificações. Assim, o projetista poderá fazer a transformação seguindo as orientações que constam no item 4 da primeira etapa da geração de código semiautomatizada apresentada na seção anterior.

Uma vez que os FBEs, as Regras e os elementos que os compõem foram identificados a partir do MFH, as Notificações também terão sido. Isto porque, o processo de compatibilização da visão planificada do MFH com os conceitos do PON, garante que as relações de notificação em um MFH sejam compatíveis com as notificações do próprio PON.

A codificação para o PON, em si, depende de cada projeto e da plataforma alvo escolhida. O método Desenvolvimento Orientado a Notificações (DON), por exemplo, apresenta uma série de modelos UML que representam os elementos lógicos do PON para o *Framework* PON C++ 1.0, mas que pode ser usado para outros *frameworks* do PON concebidos sobre linguagens do paradigma imperativo. Assim, após identificar os FBEs, Regras e outros elementos no processo da NOM o projetista pode, inclusive, optar por especificar esses elementos usando os modelos de Classes, Estados de alto nível, Componentes, Sequência, Comunicação e Redes de Petri, todos do DON.

Ainda, os esforços empreendidos no grupo de pesquisa PON sobre a linguagem para o paradigma (LingPON) (FERREIRA, 2015) e, consequentemente, a tese que versa sobre o método para compilação PON (MCPON) (RONSZCKA, 2019)

criaram ambientes propícios para materialização do software concebido com auxílio da NOM em código PON. Parte deste cenário foi abordado na seção anterior, no qual, a partir de um modelo da NOM ilustrou-se a geração semiautomatizada de código para o *Framework* PON C++ 2.0. Evidencia-se, desta forma, a viabilidade de, a partir dos modelos da NOM, associados ao DON ou não, criar-se códigos na linguagem do paradigma e, consequentemente, executá-los.

Ao bem da verdade, o uso da LingPON oferece uma interface entre o processo de concepção do software para o paradigma e sua implementação, ou seja, define o elo entre os elementos conceituais e os lógicos. Adicionalmente, a dinâmica do processo de compilação da LingPON oferece flexibilidade como um todo, isto porque a partir da linguagem é possível optar pela plataforma alvo do PON em que se deseja criar e executar o software. Assim, define-se o desacoplamento entre o processo de concepção da NOM e sua implementação, sendo possível um mesmo projeto ser compilado para diferentes plataformas alvo do PON.

3.10 DISCUSSÕES SOBRE O CAPÍTULO

O PON apresenta conceitos inovadores em relação a outros paradigmas de programação que, por serem novos, não foram ainda devidamente explorados no contexto dos processos de Engenharia de Software. A metodologia apresentada neste capítulo complementa os esforços do grupo de pesquisa no PON e adiciona um elo no que pode ser interpretado como um arquétipo ou *framework* conceitual de desenvolvimento de software para o paradigma, no qual, a partir dos requisitos do software, a NOM contribui na concepção das aplicações PON e fornece um modelo do software interpretado à luz dos conceitos que regem este paradigma. A continuação natural deste *framework* conceitual é a especificação detalhada dos elementos lógicos e a escrita do código fonte para compilação em alguma plataforma alvo, sendo estas etapas previstas no tocante aos esforços futuros em DON, DOR, Tecnologia LingPON e MCPON, entre outros.

No que diz respeito a este *framework* conceitual em si, a NOM cobre, particularmente, um aspecto menos explorado nas pesquisas sobre o PON até então, que é o processo de concepção de software. Por isso, a NOM se enquadra na fase de análise e projeto de software PON, ou seja, na modelagem em PON,

tendo como entrada a especificação de requisitos no formato de modelos de casos de uso. O processo definido na NOM busca auxiliar na concepção de modelos de software para o PON e seu principal artefato é o Modelo de Fluxo Holônico (MFH).

A metodologia de concepção de software para o PON, desenvolvido neste trabalho, inspirou-se nos princípios dos sistemas holônicos, que nortearam a criação do próprio paradigma. Assim, o projetista que usa a NOM possui a seu favor uma metodologia de concepção de software que transforma as necessidades do software, ou seja, os seus requisitos já tratados na forma de casos de uso, em uma solução alinhada aos princípios do PON. Ainda, o uso da abordagem holônica oferece um arcabouço conceitual sólido para estruturação da metodologia, proporcionando flexibilidade na definição dos elementos da linguagem e organização estrutural para lidar com os modelos mais simples e mais complexos, ao mesmo tempo em que o desenho da arquitetura do software é definido.

O Diagrama de Fluxo Holônico (DFH) foi a ferramenta criada na NOM para materializar o processo de concepção de software PON usando a abordagem holônica. Assim, o diagrama apresenta elementos inspirados no paradigma para dar suporte ao projetista no processo de transformação dos requisitos em uma solução que possa ser implementada no PON. A abordagem holônica do DFH e o processo iterativo e incremental da NOM oferecem uma sequência de modelagem na qual os modelos do software PON evoluem de modelos mais simples aos mais detalhados para representar o software de forma que ele possa ser implementado no paradigma.

As etapas de verificação e validação dos modelos criados na NOM buscam garantir que os requisitos do software sejam cumpridos e que os modelos, em seu nível mais detalhado, sejam compatíveis com o metamodelo do PON. Desta forma, a NOM garante que seja possível detalhar o modelo de software usando técnicas já desenvolvidas pelo grupo de pesquisas no PON, como o DON, ou, em alguns casos, sua codificação direta para uma plataforma de programação pertinente ao paradigma.

4 AVALIAÇÃO DA METODOLOGIA PROPOSTA

A avaliação da Metodologia de Projeto de Software Orientado a Notificações (NOM) proposta nesta tese está dividida em três etapas. Primeiro, elaborou-se um estudo de caso no qual a metodologia foi aplicada para avaliar sua utilização na concepção de software. Na sequência, um grupo focal com especialistas e pesquisadores do Paradigma Orientado a Notificações foi planejado e aplicado. Por fim, a NOM foi avaliada em relação às recomendações para criação de linguagens de modelagem.

4.1 MATERIAIS E MÉTODOS

O estudo de caso como forma de avaliação tem sua significância associada à complexidade dos seus exemplos (SHAW, 2003). Assim, optou-se por apresentar neste trabalho a modelagem completa das operações em um caixa eletrônico (ATM). Isto porque, ao mesmo tempo em que esta aplicação apresenta certa complexidade de operações, o 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 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 uma sequência de instruções. A operação escolhida foi a de “Sacar dinheiro”, cujo início de modelagem foi apresentado em capítulo anterior.

O estudo de caso do caixa eletrônico foi usado como cenário para o Grupo Focal exploratório e confirmatório com especialistas e pesquisadores 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. Os procedimentos para o Grupo Focal seguem a proposta apresentada por Dresch, Lacerda e Antunes (2014). No dia 11/12/2017, participaram do grupo focal oito especialistas e pesquisadores em PON em uma sessão que durou três horas. Como resultado, houve um incremento na NOM, cujas melhorias já constam na apresentação da metodologia em capítulo anterior. Outras

considerações da avaliação pelos especialistas estão detalhadas em seção posterior.

Por fim, a avaliação da NOM em relação às recomendações para criação de linguagens de modelagem inicia com uma verificação das seis diretrizes para modelagem do *Guidelines of Modeling* (GoM) propostas por Schuette e Rotthowe (1998). Na sequência, considerando que a NOM propõe uma linguagem de propósito específico, avaliam-se os cinco requisitos de qualidade para este tipo de linguagem, definidos por Ulrich Frank (FRANK, 2011) (FRANK, 2013). Baseando-se na mesma referência, avalia-se a linguagem de metamodelagem escolhida na NOM segundo oito requisitos e, por fim, compara-se as etapas do método proposto por (FRANK, 2013) com as escolhas da metodologia proposta nesta tese.

4.2 ESTUDO DE CASO

O capítulo “Metodologia de Projeto de Software Orientado a Notificações (NOM)” apresentou um preâmbulo com um “Exemplo de modelagem usando o Modelo de Fluxo Holônico” para o caso de uso “Sacar dinheiro”. No exemplo, apresentou-se o início da modelagem do fluxo principal e a modelagem de um fluxo de exceção, conforme “Quadro 9 - Fluxo principal e um fluxo de exceção do Caso de Uso “Sacar Dinheiro”. No mesmo capítulo, apresentou-se uma visão geral do caso de uso “Sacar dinheiro” (cf. Figura 59). Esta seção completa o caso de uso como um estudo de caso para a NOM. O fluxo principal e os fluxos de exceção são apresentados do Quadro 10 até o 18. Assim como na abordagem do capítulo anterior, a modelagem seguirá em profundidade, decompondo os hólons identificados no DFH.

Quadro 10 - Fluxo principal do Caso de Uso Sacar Dinheiro

Fluxo principal
1. O funcionário do banco inicia o ATM. 2. O ATM mostra a sua tela inicial. 3. O cliente verifica visualmente se o ATM está operando normalmente. 4. O cliente insere o seu cartão no ATM. 5. O ATM efetua a validação do cartão. 6. O ATM solicita a senha do cliente. 7. O cliente digita sua senha. 8. O ATM cria uma sessão para o cliente.

9. O ATM mostra as opções para o cliente. 10. O cliente escolhe a opção “Sacar dinheiro”. 11. O ATM solicita o valor desejado. 12. O cliente digita o valor desejado. 13. O ATM verifica se o cliente tem saldo suficiente. 14. O ATM verifica se possui saldo suficiente na máquina. 15. O ATM dispensa o dinheiro solicitado. 16. O ATM mostra a opção de imprimir recibo da transação. 17. O cliente escolhe não imprimir o recibo da transação. 18. O ATM mostra mensagem final e fecha a sessão do cliente. 19. Retorna ao passo 2

Fonte: Autoria própria

Quadro 11 - Fluxo de exceção “ATM não parece estar funcionando normalmente” do Caso de Uso Sacar Dinheiro

Fluxo de exceção: ATM não parece estar funcionando normalmente
1. O funcionário do banco inicia o ATM. 2. O ATM mostra a sua tela inicial. 3. O cliente verifica visualmente se o ATM está operando normalmente. **Exceção: [ATM não parece estar funcionando normalmente].** 4. O cliente dirige-se a outro terminal de ATM.

Fonte: Autoria própria

Quadro 12 - Fluxo de exceção “Cartão bloqueado” do Caso de Uso Sacar Dinheiro

Fluxo de exceção: Cartão bloqueado
1. O funcionário do banco inicia o ATM. 2. O ATM mostra a sua tela inicial. 3. O cliente verifica visualmente se o ATM está operando normalmente. 4. O cliente insere o seu cartão no ATM. 5. O ATM efetua a validação do cartão. **Exceção: [Cartão bloqueado].** 6. O ATM registra a tentativa de uso de cartão bloqueado. 7. O ATM exibe uma mensagem de tentativa com cartão bloqueado. 8. Retorna ao passo 2.

Fonte: Autoria própria

Quadro 13 - Fluxo de exceção “PIN incorreto” do Caso de Uso Sacar Dinheiro

Fluxo de exceção: PIN incorreto
1. O funcionário do banco inicia o ATM. 2. O ATM mostra a sua tela inicial. 3. O cliente verifica visualmente se o ATM está operando normalmente. 4. O cliente insere o seu cartão no ATM. 5. O ATM efetua a validação do cartão. 6. O ATM solicita a senha do cliente. 7. O cliente digita sua senha. **Exceção: [PIN incorreto].** 8. O ATM registra a tentativa com senha incorreta. 9. O ATM exibe uma mensagem de tentativa com senha incorreta.

10. Retorna ao passo 6.

Fonte: Autoria própria

Quadro 14 - Fluxo de exceção "Número de tentativas com PIN incorreto excedido" do Caso de Uso Sacar Dinheiro

Fluxo de exceção: Número de tentativas com PIN incorreto excedido
1. O funcionário do banco inicia o ATM. 2. O ATM mostra a sua tela inicial. 3. O cliente verifica visualmente se o ATM está operando normalmente. 4. O cliente insere o seu cartão no ATM. 5. O ATM efetua a validação do cartão. 6. O ATM solicita a senha do cliente. 7. O cliente digita sua senha. **Exceção: [Número de tentativas com PIN incorreto excedido].** 8. O ATM registra a tentativa com senha incorreta. 9. O ATM exibe uma mensagem de tentativa com senha incorreta. 10. O ATM registra terceira tentativa com senha incorreta. 11. O ATM exibe uma mensagem de terceira tentativa com senha incorreta. 12. Retorna ao passo 5.

Fonte: Autoria própria

Quadro 15 - Fluxo de exceção "Cliente com saldo insuficiente" do Caso de Uso Sacar Dinheiro

Fluxo de exceção: Cliente com saldo insuficiente
1. O funcionário do banco inicia o ATM. 2. O ATM mostra a sua tela inicial. 3. O cliente verifica visualmente se o ATM está operando normalmente. 4. O cliente insere o seu cartão no ATM. 5. O ATM efetua a validação do cartão. 6. O ATM solicita a senha do cliente. 7. O cliente digita sua senha. 8. O ATM cria uma sessão para o cliente. 9. O ATM mostra as opções para o cliente. 10. O cliente escolhe a opção "Sacar dinheiro". 11. O ATM solicita o valor desejado. 12. O cliente digita o valor desejado. 13. O ATM verifica se o cliente tem saldo suficiente. **Exceção: [Cliente com saldo insuficiente]** 14. O ATM mostra mensagem de saldo insuficiente. 15. Retorna ao passo 11.

Fonte: Autoria própria

Quadro 16 - Fluxo de exceção "ATM com saldo insuficiente" do Caso de Uso Sacar Dinheiro

Fluxo de exceção: ATM com saldo insuficiente
1. O funcionário do banco inicia o ATM. 2. O ATM mostra a sua tela inicial. 3. O cliente verifica visualmente se o ATM está operando normalmente. 4. O cliente insere o seu cartão no ATM. 5. O ATM efetua a validação do cartão.

6. O ATM solicita a senha do cliente. 7. O cliente digita sua senha. 8. O ATM cria uma sessão para o cliente. 9. O ATM mostra as opções para o cliente. 10. O cliente escolhe a opção “Sacar dinheiro”. 11. O ATM solicita o valor desejado. 12. O cliente digita o valor desejado. 13. O ATM verifica se o cliente tem saldo suficiente. 14. O ATM verifica se possui saldo suficiente na máquina. **Exceção: [ATM com saldo insuficiente]** 15. O ATM mostra mensagem de saldo insuficiente nesta máquina. 16. Retorna ao passo 11.

Fonte: Autoria própria

Quadro 17 - Fluxo de exceção “Imprimir recibo de transação” do Caso de Uso Sacar Dinheiro

Fluxo de exceção: Imprimir recibo de transação
1. O funcionário do banco inicia o ATM. 2. O ATM mostra a sua tela inicial. 3. O cliente verifica visualmente se o ATM está operando normalmente. 4. O cliente insere o seu cartão no ATM. 5. O ATM efetua a validação do cartão. 6. O ATM solicita a senha do cliente. 7. O cliente digita sua senha. 8. O ATM cria uma sessão para o cliente. 9. O ATM mostra as opções para o cliente. 10. O cliente escolhe a opção “Sacar dinheiro”. 11. O ATM solicita o valor desejado. 12. O cliente digita o valor desejado. 13. O ATM verifica se o cliente tem saldo suficiente. 14. O ATM verifica se possui saldo suficiente na máquina. 15. O ATM dispensa o dinheiro solicitado. 16. O ATM mostra a opção de imprimir recibo da transação. **Exceção: [Imprimir recibo de transação]** 17. O ATM imprime recibo de transação 18. O ATM mostra mensagem final e fecha a sessão do cliente. 19. Retorna ao passo 2

Fonte: Autoria própria

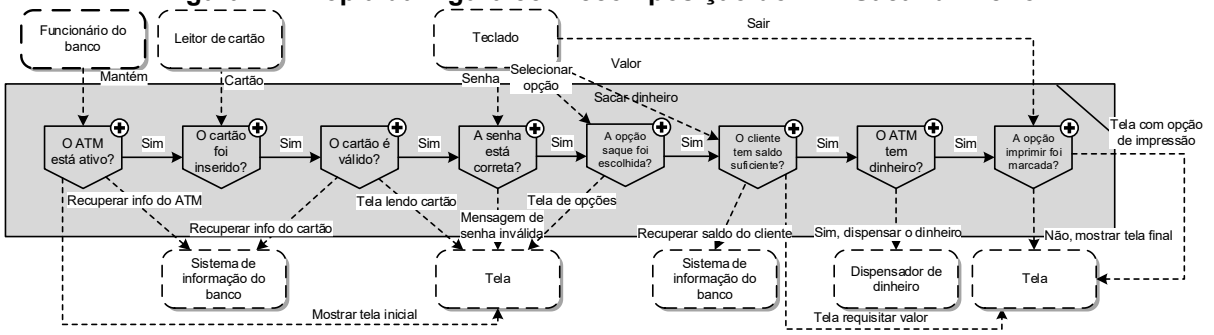
Quadro 18 - Fluxo de exceção “Desligamento programado do ATM” do Caso de Uso Sacar Dinheiro

Fluxo de exceção: Desligamento programado do ATM
1. O funcionário do banco inicia o ATM. **Exceção: [Desligamento programado do ATM]** 2. O ATM mostra a tela desligamento. 3. O ATM realiza o seu desligamento.

Fonte: Autoria própria

O início da modelagem do fluxo principal apresentou os principais passos na Figura 55, que é reapresentada a seguir como Figura 71 para facilitar a leitura deste capítulo. Porém, no capítulo anterior modelou-se por completo somente o fluxo de exceção “Cartão bloqueado” apresentado no Quadro 12, cujo Diagrama de Fluxo Holônico (DFH) pode ser revisto da Figura 56 até a Figura 58. Assim, retoma-se, nesta seção, a modelagem do fluxo principal do caso de uso “Sacar dinheiro” (cf. Quadro 10).

Figura 71 - Cópia da Figura 55: Decomposição do HT “Sacar dinheiro”



Fonte: Autoria própria

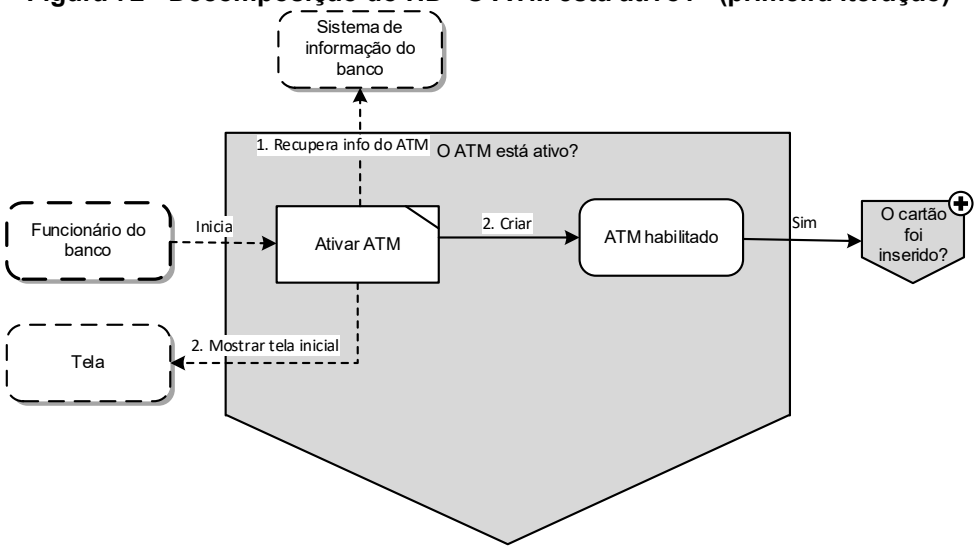
A Figura 71 apresenta a primeira iteração do fluxo principal do caso de uso “Sacar dinheiro” em que se identificou 8 Hólons Decisionais (HDs). Retomando o processo e a lógica de modelagem iniciados no capítulo anterior, em que se sabe que o primeiro HD, chamado “O ATM está ativo?”, foi criado pela interpretação do passo “3. O cliente verifica visualmente se o ATM está operando normalmente.”.

Além disso, sabe-se que o primeiro HD entrega o fluxo para o próximo HD “O cartão foi inserido?”. Desta forma, o projetista deverá detalhar as atividades do HD “O ATM está ativo?” de forma que o fluxo continue ao hólon sucessor. Ao analisar o Quadro 10, percebe-se que os passos 1, 2 e 3 podem ser interpretados como passos que compõem o HD “O ATM está ativo?”. Os passos definem “O funcionário do banco inicia o ATM.”, “O ATM mostra a sua tela inicial.” e “O cliente verifica visualmente se o ATM está operando normalmente.”.

A Figura 72 apresenta a primeira iteração que decompõe o HD em questão. O funcionário do banco foi modelado como um Hólon Entidade Externa (HX) e, assim, para que ele possa ativar o ATM criou-se um Hólon Transacional (HT) “Ativar ATM”. Este HT depende da recuperação de informações do Sistema de Informação do Banco, representado no DFH pelo HX de mesmo nome e mostra a tela inicial do ATM quando o mesmo é habilitado a funcionar. Percebe-se, na figura, que o recurso

de ordem para as relações de notificação e de chamada foram usados. O modelo indica que o HT “Ativar ATM” irá, primeiro, “1. Recuperar info do ATM” e na sequência irá, paralelamente, “2. Criar” o HE “ATM habilitado” e “2. Mostrar tela inicial” no HX “Tela”. O Hólon Entidade (HE) “ATM habilitado” foi criado para ser responsável por entregar a informação da habilitação do ATM e provocar o fluxo para o hólon sucessor (i.e. HD “O cartão foi inserido?”).

Figura 72 - Decomposição do HD “O ATM está ativo?” (primeira iteração)



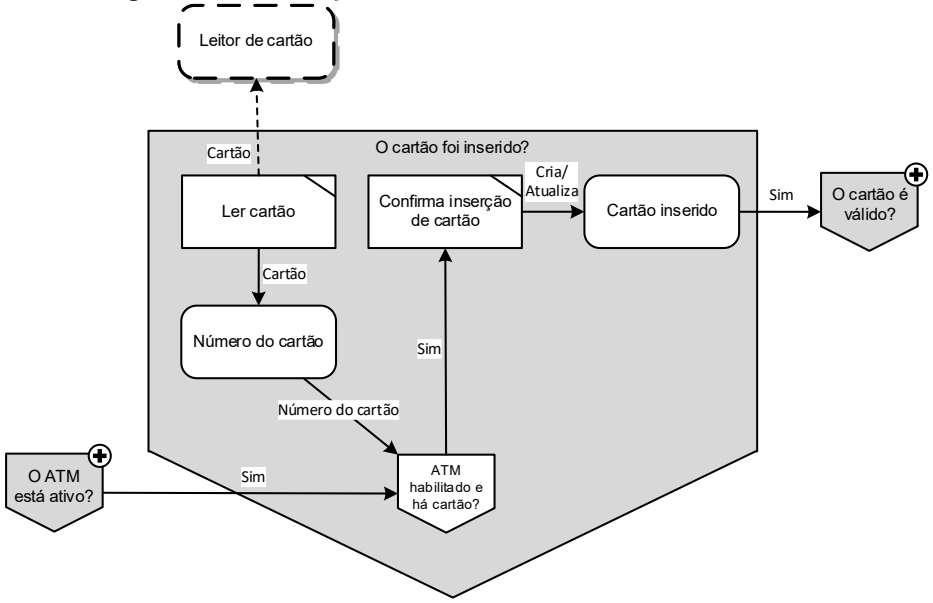
Fonte: Autoria própria

A partir desta primeira iteração, pode-se avaliar algumas escolhas. Somente um HT é suficiente para modelar os passos deste HD? No exemplo apresentado o HT é responsável por recuperar informações do ATM no sistema de informação do banco, mostrar a tela inicial do ATM e dar sequência ao fluxo do DFH. Além disso, a criação do HE “ATM habilitado” é obrigatória para dar continuidade ao fluxo ou somente uma relação de notificação do HT “Ativar ATM” para o HD “O cartão foi inserido?” seria suficiente? Trata-se das iterações iniciais no modelo, então um projetista que tenha maior familiaridade com a linguagem e com o Paradigma Orientado a Notificações (PON) poderá criar modelos mais detalhados.

Porém, como será visto na sequência do processo de modelagem, a validação e verificação dos modelos irá requerer novas iterações para tornar o modelo mais completo. O fato é que o projetista poderá ampliar o número de hólons neste nível se desejar, ou mesmo decompor o próprio HT “Ativar ATM” para melhor especificar as atividades sob sua responsabilidade. A criação do HE “ATM habilitado” ocorreu pela experiência do projetista com PON e que sabe da necessidade deste tipo de hólon para habilitar o próximo.

A decomposição do HD “O ATM está ativo?” habilita o funcionando do ATM e o mantém em um estado com a tela inicial aguardando que algum cliente insira o cartão na máquina. O próximo HD da Figura 71 é “O cartão foi inserido?”, criado a partir do passo “4. O cliente insere o seu cartão no ATM.”. Após algumas iterações para esse HD, chegou-se na solução apresentada na Figura 73.

Figura 73 - Decomposição do HD “O cartão foi inserido?”



Fonte: Autoria própria

Para que o HD fique em estado de espera até que um cartão seja inserido na máquina, em sua decomposição criou-se um HD, chamado “ATM habilitado e há cartão?”, para decidir pela continuação do fluxo quando sua condição for satisfeita.

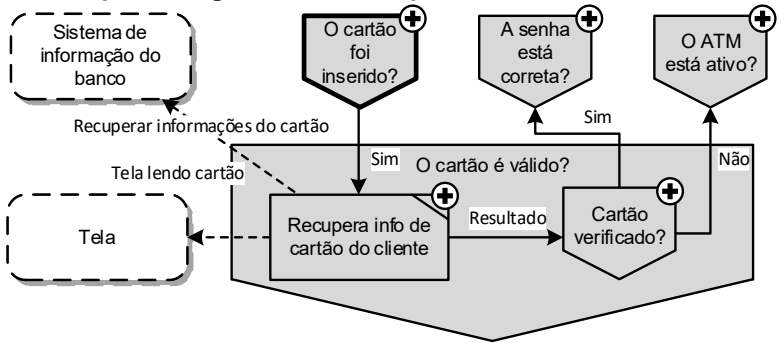
Para este HD, a habilitação do ATM vem de fluxo anterior e para modelar o fluxo em que o HD recebe a informação de que um cartão foi inserido, criou-se um HT com relação de chamada com o HX “Leitor de cartão” com a responsabilidade de criar um HE “Número do cartão” quando receber essa informação da entidade externa. Na sequência, quando o HD “ATM habilitado e há cartão” decide pela continuidade, o fluxo é direcionado para o HT “Confirma inserção de cartão” que, por sua vez, cria o HE “Cartão inserido” para dar continuidade ao fluxo holônico para o hólon sucessor.

Novamente, as escolhas nesta iteração podem ser avaliadas. Especialmente no que diz respeito ao número de hólons que compõem a solução. Nas primeiras iterações desta decomposição havia somente o HT “Ler cartão” e o HE “Cartão inserido”. Porém, à medida em que o caso de uso foi sendo interpretado, percebeu-se que havia decisões a serem tomadas e, por isso, criou-se o Hólon Decisional. O

projetista poderia optar, porém, por ter decomposto o HD “ATM habilitado e há cartão?” incluindo os próximos dois hólons do fluxo (i.e. HT “Confirma inserção de cartão” e HE “Cartão inserido”).

O próximo HD do fluxo é “O cartão é válido?”, criado a partir do passo “5. O ATM efetua a validação do cartão.”. Porém, esse HD já foi apresentado em capítulo anterior. Assim, apresenta-se somente o primeiro nível da decomposição deste HD para discussões da sequência de fluxo. Basicamente, o HD “O cartão é válido?” é notificado quando um cartão foi inserido e faz a verificação de sua validade, direcionando o fluxo para a verificação de senha do cliente, caso o cartão seja válido ou retornando para o início do fluxo, caso o cartão seja inválido.

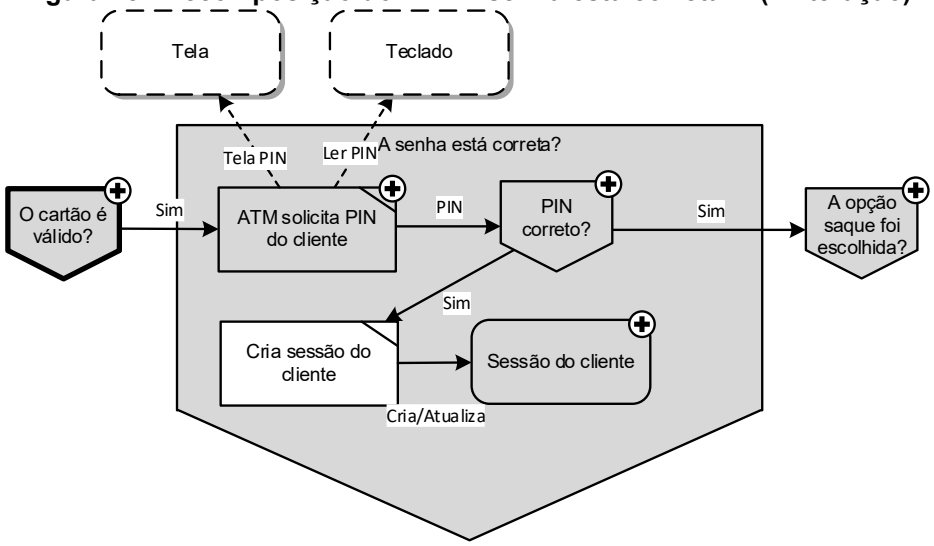
Figura 74 - Cópia da Figura 56: Decomposição do HD “O cartão é válido?”



Fonte: Autoria própria

Na sequência do fluxo principal do caso de uso “Sacar dinheiro”, apresentam-se os passos 6 a 10 que correspondem a “O ATM solicita a senha do cliente.”, “O cliente digita sua senha.”, “O ATM cria uma sessão para o cliente.” E “O ATM mostra as opções para o cliente.”. Para o passo 7, o HD “A senha está correta?” havia sido criado entendendo que neste ponto haveria uma decisão a tomar, dependendo de o cliente digitar a senha correta ou não. A decomposição deste hólon inicia com o passo 6 em que é solicitado que o cliente digite a sua senha (cf. Figura 75). Assim, o HT “ATM solicita PIN do cliente” mostra na tela o campo para senha e aguarda digitação do cliente no teclado. A senha digitada deve ser verificada e, assim, criou-se o HD “PIN correto?”. Caso o PIN seja correto o ATM cria a sessão do cliente, representada pelo HE “Sessão do cliente” e o fluxo continua para o próximo hólon do fluxo principal.

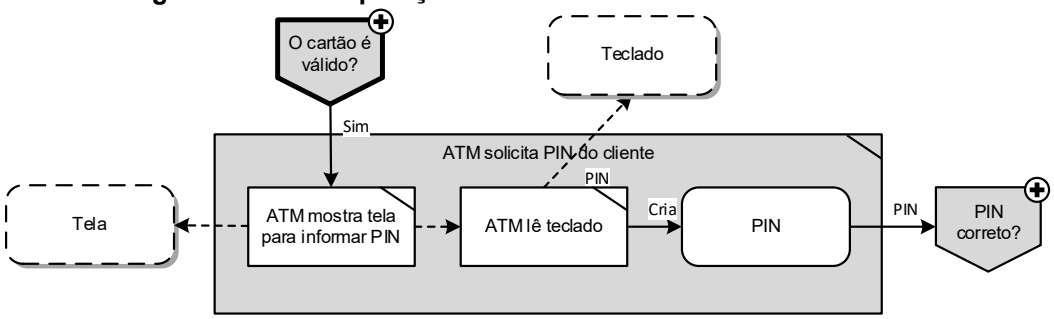
Figura 75 - Decomposição do HD "A senha está correta?" (1ª iteração)



Fonte: Autoria própria

A Figura 76 ilustra a decomposição do HT "ATM solicita PIN do cliente". Este hólon é relativamente simples, mas optou-se pela decomposição para manter uma boa interpretação do nível anterior. Basicamente, como pode ser observado na Figura 76 o hólon mostra a tela para que o cliente digite a sua senha e recupera do teclado a senha digitada, criando uma entidade para a senha e a passando para o próximo hólon do fluxo.

Figura 76 - Decomposição do HT "ATM solicita PIN do cliente"

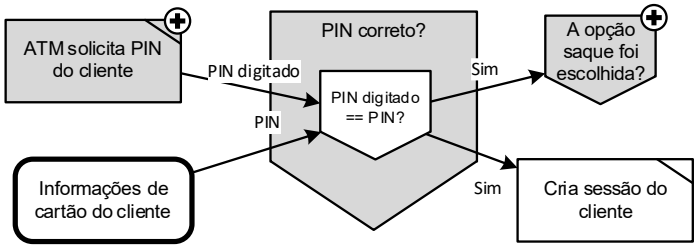


Fonte: Autoria própria

Na sequência, o hólon HD "PIN correto?", exibido na Figura 75, é decomposto na Figura 77. Este hólon recebe o PIN correto do cliente a partir do HE "Informações de cartão do cliente" criado em passo anterior e o PIN digitado no teclado do ATM. Assim, o HD "PIN digitado == PIN?" verifica que são iguais, direcionando o fluxo para os próximos hólons do fluxo que são o HD "A opção saque foi escolhida?", que já existia no fluxo principal, assim como para o HT "Cria sessão do cliente". O HD "PIN correto?" pode parecer muito simples, porém, esta é a

primeira iteração e diz respeito somente ao fluxo principal. Ainda há os hólons para o fluxo de exceção de senha incorreta.

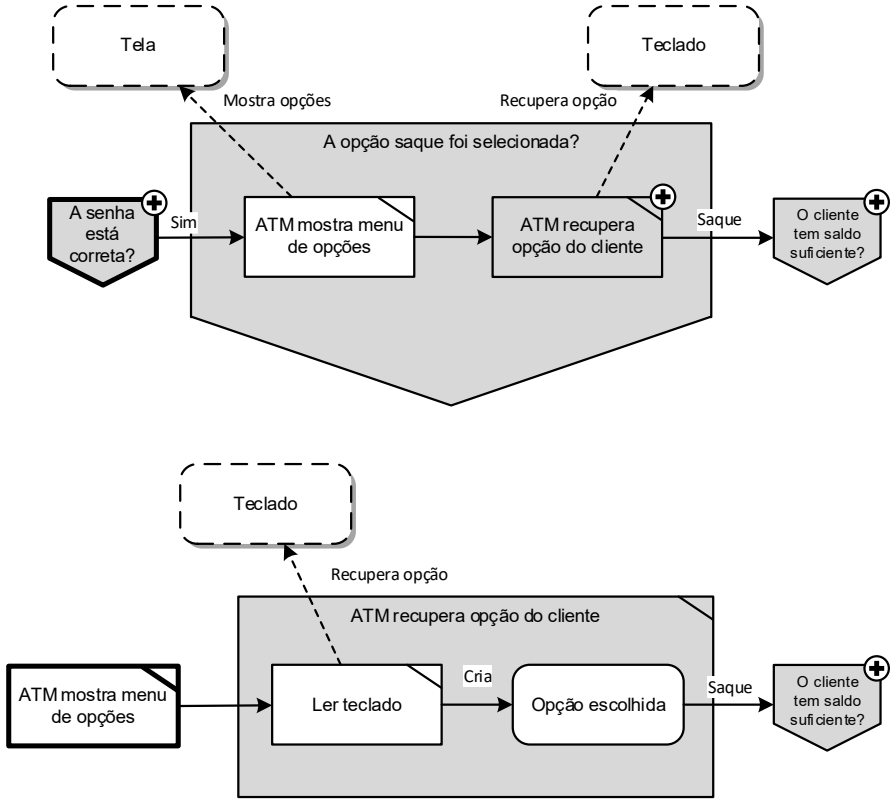
Figura 77 - Decomposição do HD “PIN correto?” (1ª iteração)



Fonte: Autoria própria

O próximo passo do fluxo principal do caso de uso diz respeito ao cliente selecionar a opção saque (passo 10 do Quadro 10). Este caso de uso modela somente “Sacar dinheiro”, porém, criou-se o HD “A opção saque foi escolhido?” entendendo-se que há outras opções em um sistema de caixa eletrônico. Assim, a Figura 78 ilustra a decomposição do HD “A opção saque foi selecionada?” e de seu HT “ATM recupera opção do cliente”. O HT “ATM mostra menu de opções” e recupera a opção do cliente lendo o teclado e dando sequência ao próximo hólon do fluxo HD “O cliente tem saldo suficiente?”.

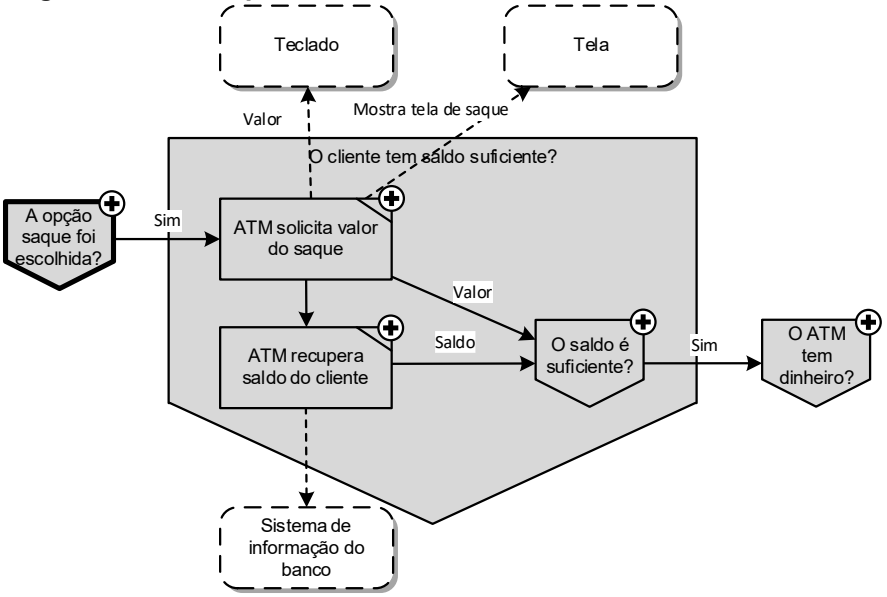
Figura 78 - Decomposição do HD "A opção saque foi selecionada?" e HT "ATM recupera opção do cliente"



Fonte: Autoria própria

Após a opção pelo saque, o HD "O cliente tem saldo suficiente?" é responsável pelos seguintes passos do caso de uso: "11. O ATM solicita o valor desejado", "12. O cliente digita o valor desejado" e "13. O ATM verifica se o cliente tem saldo suficiente". A Figura 79 mostra o HD detalhado, que possui em seu início o HT "ATM solicita valor do saque". Este HT mostra a tela de saque e solicita que o cliente informe o valor para o saque. Na sequência, o fluxo segue para o HD "O saldo é suficiente?" e o HT "ATM recupera saldo do cliente". O HD depende do saldo do cliente, então, conforme ilustra a figura, também recebe fluxo do HT. Assim, de posse do valor de saque solicitado e do saldo do cliente, o HD "O saldo é suficiente?" e direciona o fluxo para o próximo hólon do fluxo principal HD "O ATM tem dinheiro?" se o cliente tem saldo suficiente.

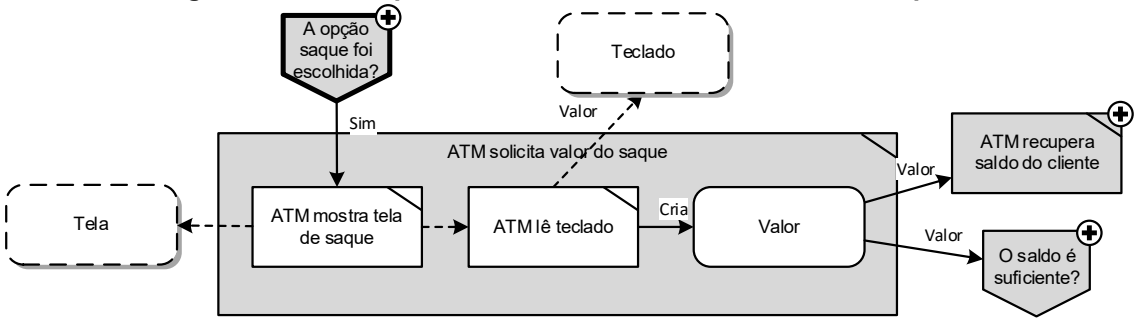
Figura 79 - Decomposição do HD "O cliente tem saldo suficiente?"



Fonte: Autoria própria

A Figura 80 mostra a decomposição do HT "ATM solicita valor do saque". Este HT é similar ao apresentado na Figura 76 - Decomposição do HT "ATM solicita PIN do cliente", com a diferença que no fluxo de saída o valor é direcionado para dois hólons ao invés de um.

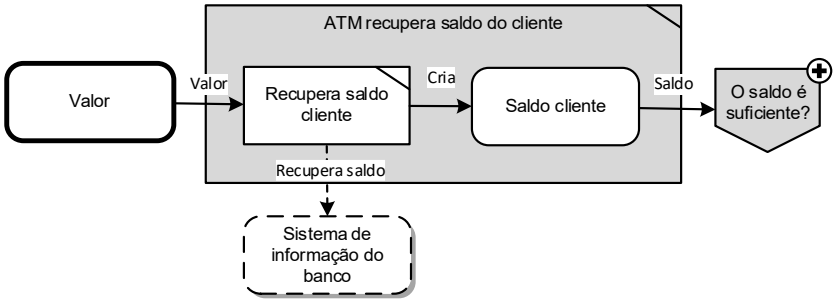
Figura 80 - Decomposição do HT "ATM solicita valor do saque"



Fonte: Autoria própria

A Figura 81 mostra a decomposição do HT "ATM recupera saldo do cliente" em que o HE "Saldo cliente" é criado a partir da consulta ao HX "Sistema de informação do banco" pelo HT "Recupera saldo cliente".

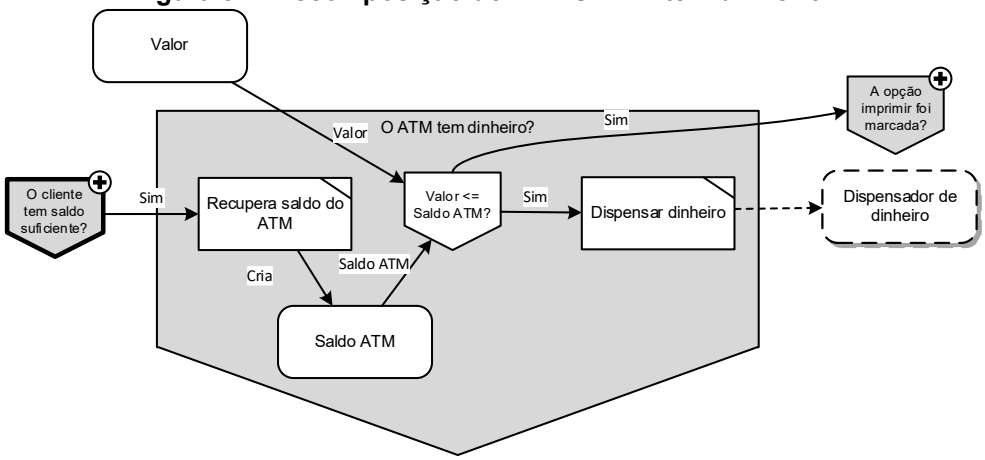
Figura 81 - Decomposição do HT "ATM recupera saldo do cliente"



Fonte: Autoria própria

Na sequência do fluxo principal, em que o saldo do cliente é suficiente, o fluxo é direcionado para o HD "O ATM tem dinheiro?" (cf. Figura 79). Neste HD estão concentrados os passos "14. O ATM verifica se possui saldo suficiente na máquina" e "15. O ATM dispensa o dinheiro solicitado". Assim, o HD detalhado inicia com a recuperação do saldo do ATM pelo HT "Recupera saldo do ATM", conforme Figura 82. Este HT define o HE "Saldo ATM" e o fluxo é direcionado para o HD "Valor <= Saldo?" que verifica se o valor solicitado pelo cliente está disponível no ATM para saque. Neste ponto da modelagem percebeu-se que o HD "Valor <= Saldo?" necessitava do valor de saque informado pelo cliente. Então, criou-se o fluxo do HE "Valor" e o digrama apresentado na Figura 80 deverá ser atualizado.

Figura 82 - Decomposição do HD "O ATM tem dinheiro?"

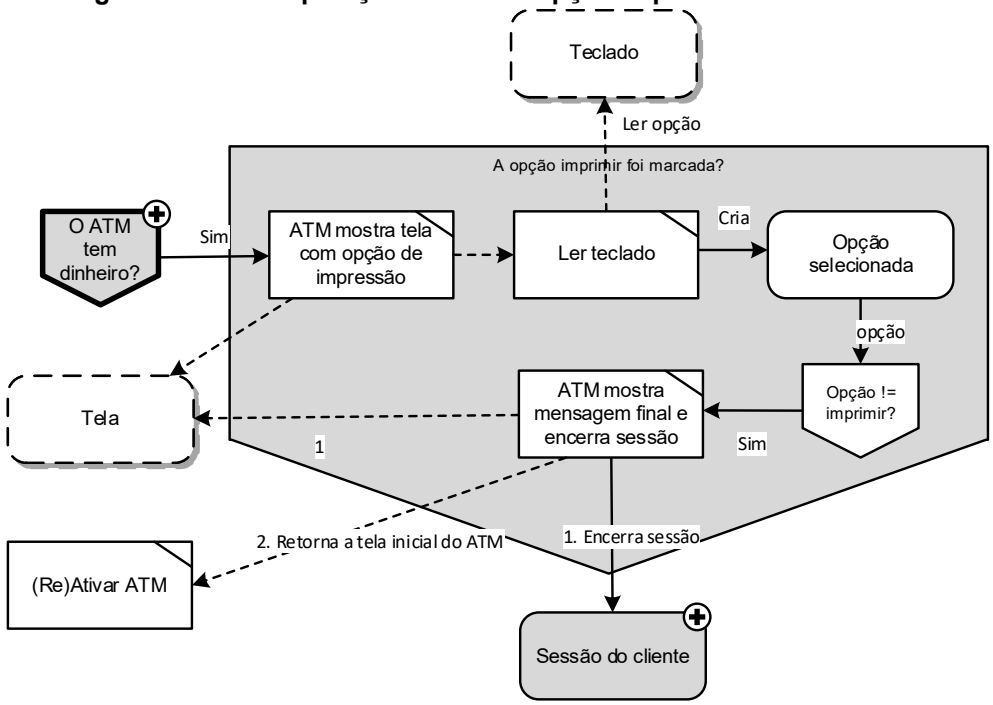


Fonte: Autoria própria

Nos próximos passos do fluxo principal do caso de uso (do passo 16 ao 19), o ATM mostra a opção de imprimir o recibo da transação, o cliente opta por não imprimir e a sessão do cliente é encerrada antes de retornar para a tela inicial do ATM. A decomposição do HD "A opção imprimir foi marcada?" é mostrado na Figura 83. O fluxo inicia com o HT "ATM mostra tela com opção de impressão" e a

consequente leitura do teclado para recuperar a opção do cliente. No fluxo principal do caso de uso a opção selecionada é a de não imprimir. Assim, o ATM mostra a mensagem final da transação ao mesmo tempo em que encerra a sessão do cliente. Na sequência o fluxo é direcionado para a tela inicial do ATM, representada pelo HT "(Re)Ativar ATM". Este último hólon é, na verdade, o HT "Ativar ATM" exibido na Figura 72 que foi renomeado pois passou a ter a dupla função de ativar o ATM quando é iniciado e reativá-lo quando alguma operação terminar.

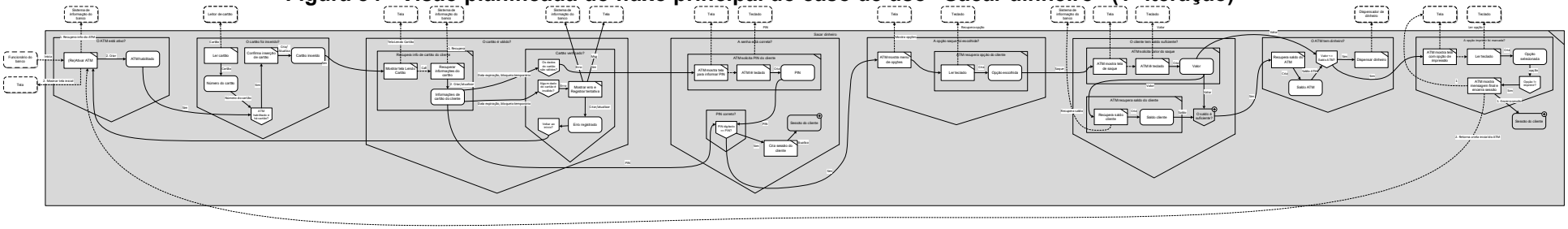
Figura 83 - Decomposição do HD "A opção imprimir foi marcada?"



Fonte: Autoria própria

Modelou-se, até este ponto, os 19 passos do fluxo principal do caso de uso "Sacar dinheiro". A Figura 84 ilustra a visão planificada do modelo, na qual pode-se perceber claramente os oito Hólons Decisionais criados nas primeiras iterações da NOM (cf. Figura 71). A intenção desta figura não é exibir detalhes, mas apresentar uma visão geral dos elementos e relações criadas até o momento.

Figura 84 - Visão planificada do fluxo principal do caso de uso “Sacar dinheiro” (1ª iteração)

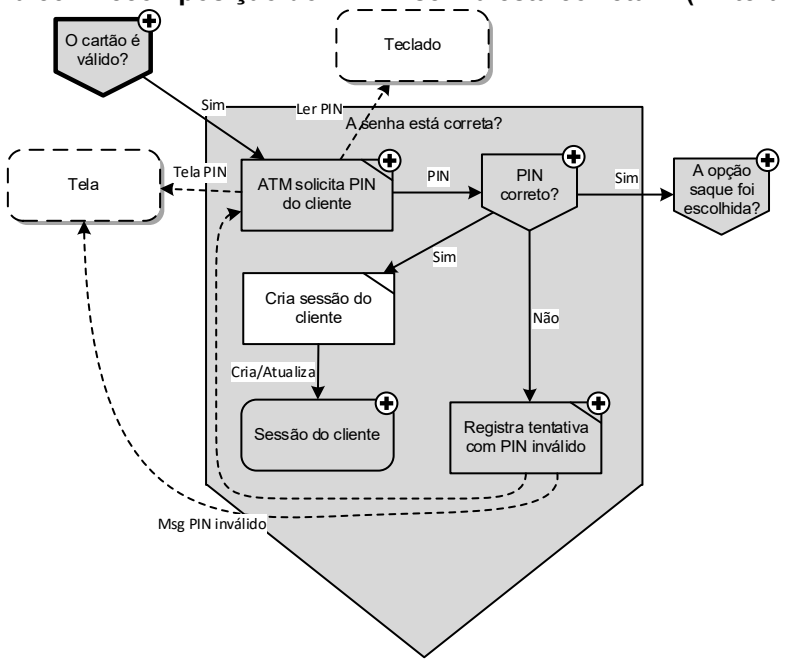


Fonte: Autoria própria

Na sequência, inicia-se a modelagem dos fluxos de exceção do caso de uso. O primeiro fluxo de exceção ainda não modelado, apresentado no Quadro 11, diz respeito a ações somente do cliente que, ao identificar algo suspeito no caixa eletrônico, dirige-se a outro terminal. Assim, não há modelagem a ser criada para esse fluxo.

O Quadro 13 apresenta o próximo fluxo de exceção, cujo nome é “PIN incorreto”, e descreve como exceção a digitação incorreta de senha pelo cliente. Basicamente, quando o cliente informa a senha incorretamente, o sistema do ATM registra a tentativa, mostra uma mensagem na tela e retorna o fluxo para que o cliente faça uma nova tentativa. Esta exceção inicia após o passo “7. O cliente digita sua senha.”, cuja localização no DFH é o HD “A senha está correta?” (cf. Figura 75 e decomposição na Figura 77). Assim, para o HD “PIN correto?” criou-se um fluxo de saída para o novo HT “Registra tentativa com PIN inválido”. Esse HT é responsável pelos passos 8, 9 e 10 do fluxo de exceção “PIN incorreto”.

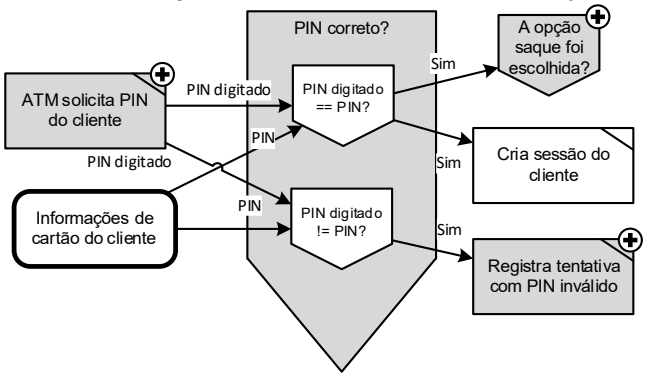
Figura 85 - Decomposição do HD “A senha está correta?” (2ª iteração)



Fonte: Autoria própria

Antes de decompor o HT “Registra tentativa com PIN inválido”, efetuou-se uma nova iteração no HD “PIN correto?” (Figura 77) para que também reflita a condição de PIN incorreto. A Figura 86 ilustra o novo HD “PIN digitado != PIN” que direciona o fluxo para o registro de tentativa com PIN inválido.

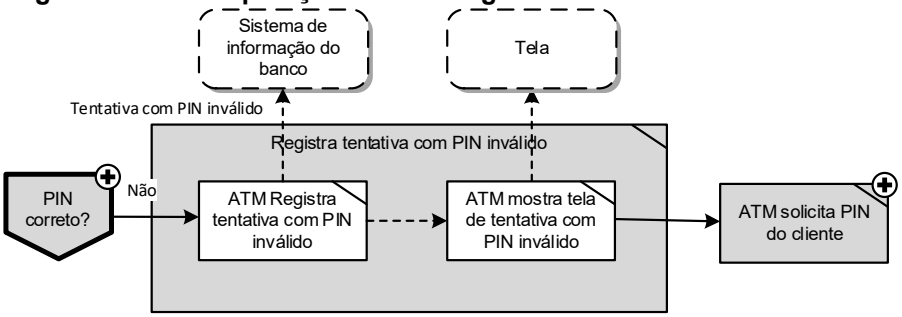
Figura 86 - Decomposição do HD “PIN correto?” (2ª iteração)



Fonte: Autoria própria

A decomposição do HT “Registra tentativa com PIN inválido” é ilustrado na Figura 87. Nele, os HTs “ATM registra tentativa com PIN inválido” e “ATM mostra tela de tentativa com PIN inválido” são responsáveis pelos passos 8 e 9, respectivamente, do fluxo de exceção “PIN incorreto”, antes do direcionamento do fluxo de volta ao passo 6, em que o ATM solicita novamente a senha do cliente.

Figura 87 - Decomposição do HT “Registra tentativa com PIN inválido”

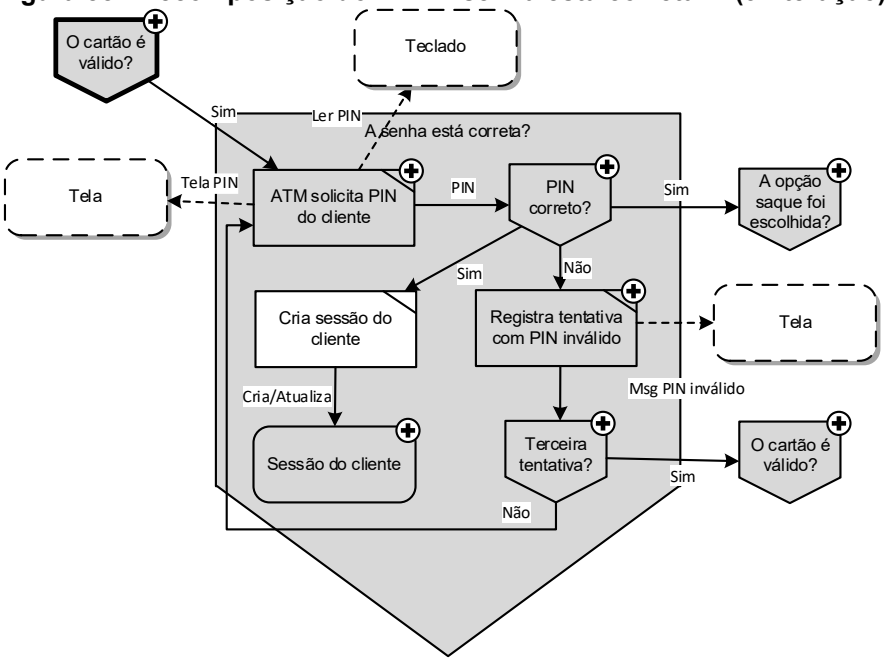


Fonte: Autoria própria

O próximo fluxo de exceção “Número de tentativas com PIN incorreto excedido” é apresentado no Quadro 14. Sua ocorrência depende de o número de tentativas de digitar a senha ter excedido certo limite, neste exemplo dado 3 tentativas. Após sua ocorrência o fluxo é direcionado para o passo 5 de validação do cartão, para caso o banco tenha política de bloqueio do cartão, impedindo que o cartão continue a fazer operações no ATM. Os passos que descrevem a exceção são, cf. Quadro 14, os que seguem: “8. O ATM registra a tentativa com senha incorreta”, “9. O ATM exibe uma mensagem de tentativa com senha incorreta”, “10. O ATM exibe uma mensagem de terceira tentativa com senha incorreta” e “11. Retorna ao passo 5”.

Esta exceção, acima descrita, ocorre no mesmo passo da exceção anterior, ou seja, após o cliente digitar a senha (HD “A senha está correta?”). Desta forma, os hólons serão criados no mesmo HD. Os passos 8 e 9 são os mesmos da exceção anterior, porém ao invés do fluxo ser direcionado ao passo 6, em que o ATM solicita novamente a senha do cliente, o fluxo é direcionado ao passo 5 em que o processo de validação do cartão é realizado novamente. Assim, ao modelar esta etapa, o projetista precisa de uma nova dinâmica de fluxo, mas que mantenha a lógica da exceção anterior. Neste caso, optou-se por criar um HD “Terceira tentativa?” que irá verificar se é a terceira ocorrência de PIN incorreto para decidir sobre o direcionamento para o passo 5 ou para o passo 6 do fluxo de exceção anterior. A Figura 88 ilustra a nova iteração no HD “A senha está correta?” com a inclusão do novo HD e suas relações de chamada.

Figura 88 - Decomposição do HD “A senha está correta?” (3ª iteração)

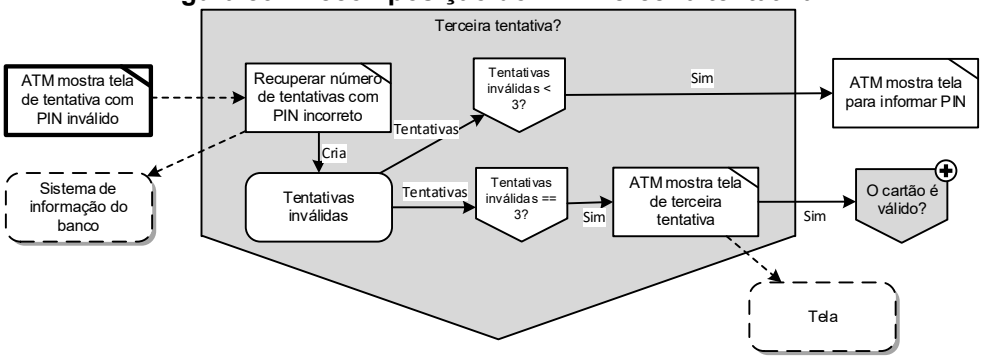


Fonte: Autoria própria

O HD “Terceira tentativa” é responsável pela verificação de ocorrência da terceira tentativa com senha incorreta e direcionamento do fluxo, além de mostrar a tela com mensagem de terceira tentativa inválida. Assim, a Figura 89 ilustra o HD detalhado, no qual criou-se o HT “Recuperar número de tentativas com PIN incorreto” que interage com o HX “Sistema de informação do banco” e cria o HE “Tentativas inválidas” que contém a informação para que o HD “Tentativas inválidas < 3?” ou o HD “Tentativas inválidas == 3?” direcionem o fluxo. O último HD trata o

fluxo de exceção sendo modelado em que o passo 10. “O ATM exibe uma mensagem de terceira tentativa com senha incorreta.” É efetuado pelo HT “ATM mostra tela de terceira tentativa”. Na sequência o fluxo é direcionado para o HD “O cartão é valido”, referente ao passo “11. Retorna ao passo 5”.

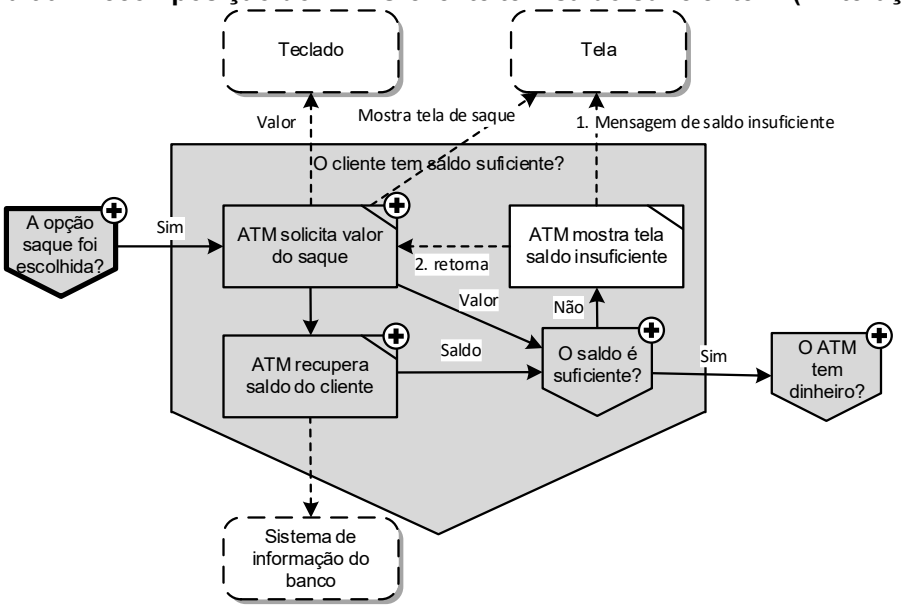
Figura 89 - Decomposição do HD “Terceira tentativa?”



Fonte: Autoria própria

O próximo fluxo de exceção a ser modelado é apresentado no Quadro 15 e seu nome é “Cliente com saldo insuficiente”. A exceção ocorre após o passo 13 em que “O ATM verifica se o cliente tem saldo suficiente”. Os passos do caso de exceção são: “14. O ATM mostra mensagem de saldo insuficiente” e “15. Retorna ao passo 11”. O retorno ao passo 11 permite que o cliente informe novo valor para saque. A modelagem desta exceção se dá no HD “O cliente tem saldo suficiente?”. A Figura 79 já ilustrava que o HD “O saldo é suficiente?” possui decomposição, porém naquele momento da modelagem não foi mostrado. Além disso, havia somente um fluxo saindo deste HD. Então, criou-se um novo fluxo a partir deste HD para representar o saldo insuficiente do cliente e a exibição da tela pelo HT “ATM mostra tela saldo insuficiente”. Esta nova iteração é ilustrada na Figura 90.

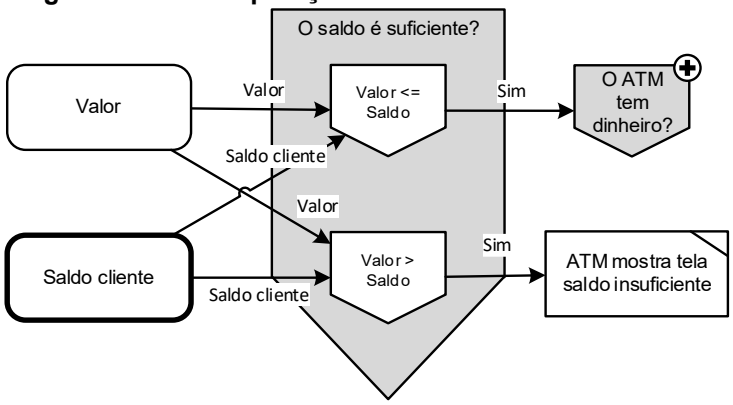
Figura 90 - Decomposição do HD "O cliente tem saldo suficiente?" (2ª iteração)



Fonte: Autoria própria

O HD "O saldo é suficiente" é decomposto na Figura 91. A partir do valor solicitado pelo cliente e do saldo em conta (recuperados em HTs anteriores), o HD "Valor > Saldo?" decide por direcionar o fluxo para o HT "ATM mostra tela saldo insuficiente" criado anteriormente (passo 14 deste caso de exceção). O HD "Valor <= Saldo?" diz respeito ao fluxo principal do caso de uso. O HT "ATM mostra tela saldo insuficiente", além de mostrar a tela de saldo insuficiente, direciona o fluxo para o HT "ATM solicita valor do saque", conforme determina o passo 15 deste fluxo de exceção (ilustrado na Figura 90). É importante salientar que o HT "ATM mostra tela saldo insuficiente" é exibido fora do HE "O saldo é suficiente?", assim como outros hólons, pois possuía relação prévia com o HE.

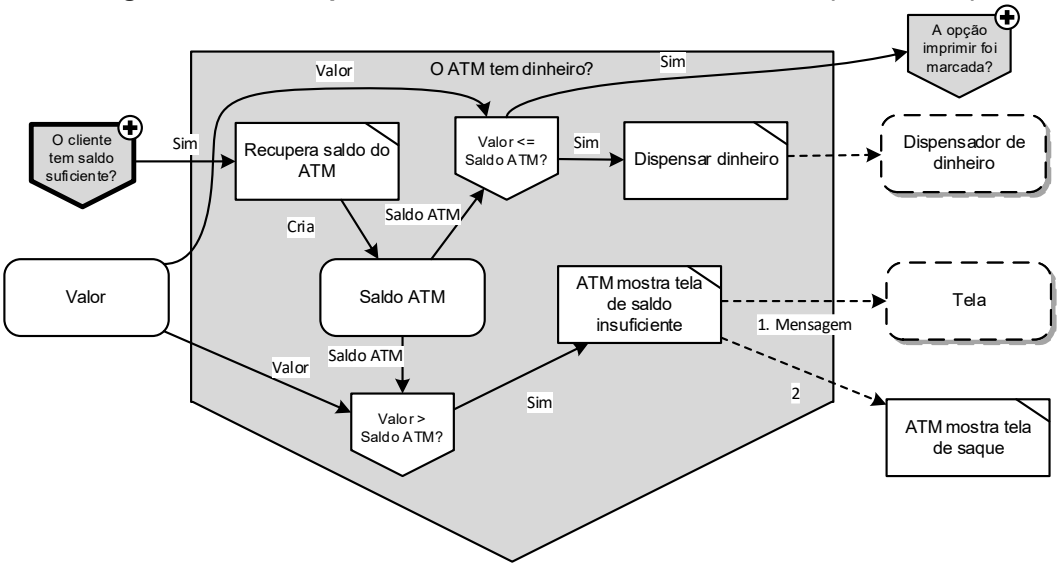
Figura 91 - Decomposição do HD "O saldo é suficiente?"



Fonte: Autoria própria

O próximo fluxo de exceção do caso de uso é ilustrado no Quadro 16 e seu nome é “ATM com saldo insuficiente”. Este caso de exceção inicia após o passo “14. O ATM verifica se possui saldo suficiente na máquina” e trata a situação em que o cliente tem saldo para realizar o saque, porém a máquina não tem dinheiro suficiente. A exceção é modelada no HD “O ATM tem dinheiro?”, cuja Figura 82 apresentou a primeira iteração. Para esta exceção, criou-se um fluxo a partir dos HEs “Saldo ATM” e “Valor” em que o HD “Valor > Saldo ATM?” compara o valor solicitado pelo cliente com o saldo do ATM, sendo essas informações recuperadas em HTs anteriores. A iteração com o novo HD é mostrada na Figura 92, cujo fluxo segue para o HT “ATM mostra tela de saldo insuficiente” (passo 15 deste caso de exceção) e, na sequência, retorna para a tela em que o cliente pode digitar um novo valor para saque conforme determina o passo 16 deste caso de exceção.

Figura 92 - Decomposição do HD “O ATM tem dinheiro?” (2ª iteração)



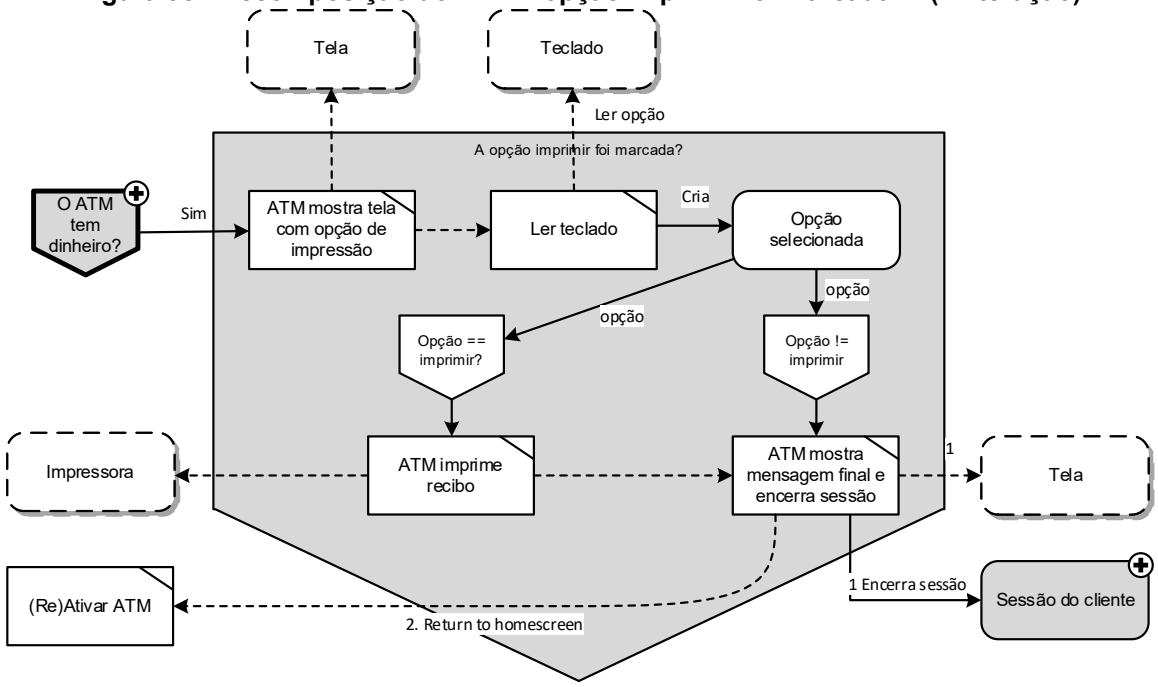
Fonte: Autoria própria

O próximo caso de exceção, denominado “Imprimir recibo de transação”, trata a situação em que o cliente optar por imprimir o recibo. Os passos são apresentados no Quadro 17 e iniciam após o passo 16 em que o “O ATM mostra a opção de imprimir recibo da transação”. Basicamente, este caso de exceção adiciona o fluxo principal a impressão do recibo, passo “17. O ATM imprime recibo de transação”. Os passos 18 e 19 já são os passos do fluxo principal.

A Figura 83 apresenta a primeira iteração no HD “A opção imprimir foi marcada?” em que o HD “Opção != imprimir” direciona o fluxo principal do caso de uso. Para o fluxo de exceção, como ilustrado na Figura 93, optou-se pela mesma

técnica do fluxo de exceção anterior. Assim, criou-se o HD “Opção == imprimir”, cujo fluxo vem do HE “Opção selecionada” e direciona o fluxo para o HT “ATM imprime recibo”, retornando na sequência para o HT “ATM mostra mensagem final e encerra sessão” previamente criado.

Figura 93 - Decomposição do HD “A opção imprimir foi marcada?” (2ª iteração)

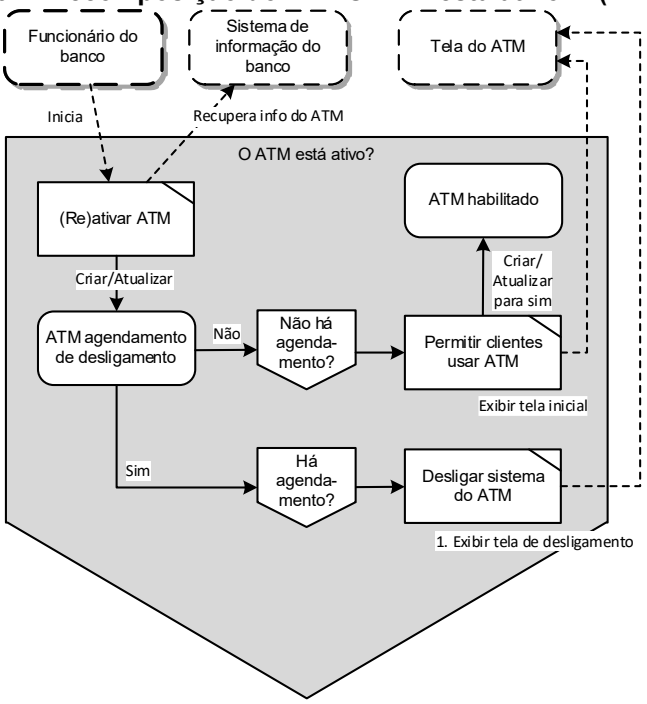


Fonte: Autoria própria

O último fluxo de exceção do caso de uso “Sacar dinheiro” é o “Desligamento programado do ATM”. Esta exceção é ilustrada no Quadro 18 e pode ocorrer imediatamente após o ATM ser ligado ou quando a operação de sacar dinheiro for finalizada. A Figura 72 mostra a primeira iteração do HD “O ATM está ativo?” e, sendo ele o início do caso de uso, é nele, também, que a exceção deverá ser criada.

Basicamente, o ATM deve verificar no sistema de informação do banco se há desligamento programado e desligá-lo. Como em situações anteriores, este caso de exceção não pode interromper os fluxos já modelados. Para isso, criou-se um fluxo intermediário e o HT “(Re)ativar ATM” não pode mais habilitar o ATM diretamente. Nesta nova iteração, o HT recupera do HX “Sistema de informação do banco” se há agendamento programado, criando, por consequência, o HE “ATM agendamento de desligamento”, como pode ser observado na Figura 94.

Figura 94 - Decomposição do HD "O ATM está ativo?" (2ª iteração)

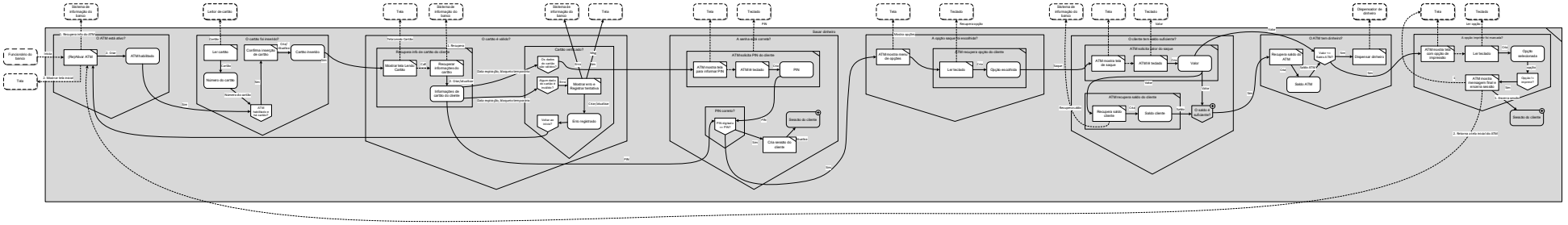


Fonte: Autoria própria

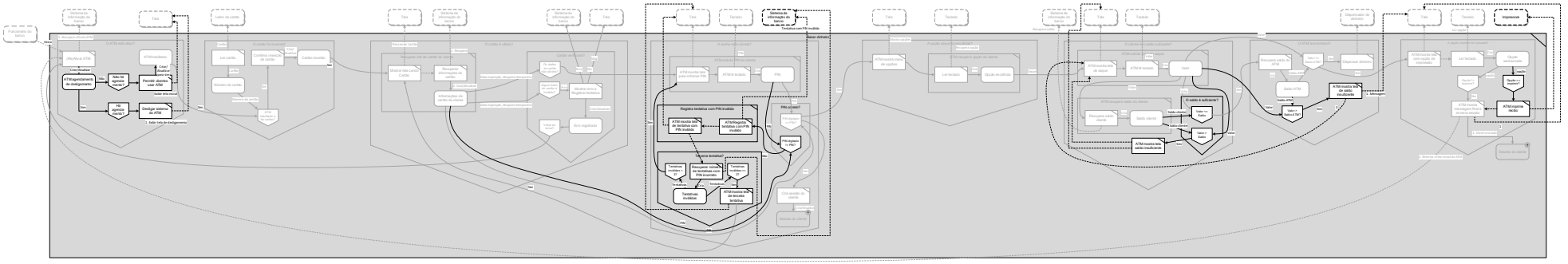
Caso haja programação, o HD "Há agendamento?" direcional o fluxo para o HT "Desliga sistema do ATM", que é responsável por mostrar a tela de desligamento e efetivar o desligamento do ATM. Para manter o fluxo principal que havia, criou-se o HD "Não há agendamento?" e o subsequente HT "Permitir clientes usar ATM" que, a partir desta iteração, é responsável por mostrar a tela inicial do ATM e criar ou atualizar o HE "ATM habilitado", habilitando-o para clientes.

O processo de modelagem se encerra após a criação dos hólons e relações para os fluxos de exceção do caso de uso. A Figura 95 ilustra, novamente, a visão planificada, agora com destaque para os novos hólons e relações criados a partir dos fluxos de exceção. O objetivo da figura não é apresentar detalhes, mas oferecer uma visão geral das modificações ocorridas no modelo. Desta forma, é possível fazer uma comparação visual do número de novos elementos e relações em comparação com a Figura 84 que apresenta somente os elementos para o fluxo principal do caso de uso.

Figura 95 - Visão planificada do modelo antes (a) e após (b) a modelagem dos fluxos de exceção do caso de uso “Sacar dinheiro” com destaque para os novos hólons.



(a)



(b)

Fonte: Autoria própria

Após a modelagem do caso de uso com o Diagrama de Fluxo Holônico (DFH), faz-se a verificação e validação dos modelos. As etapas desse processo foram explicadas na seção “3.7 Verificação e validação dos modelos”, que apresentou, na Figura 65, o diagrama de estados do caso de uso. Adicionalmente, pode-se realizar a geração semiautomatizada de código, conforme mostrado na seção 3.8, para verificar se o modelo se comporta como esperado em uma determinada implementação do PON. Por fim, salienta-se que o modelo de fluxo holônico concebeu a lógica para o caso de uso “Sacar dinheiro” compatível com o Paradigma Orientado a Notificação. Assim, os próximos passos são a criação dos modelos lógicos de implementação, ou a implementação direta em uma plataforma alvo do paradigma. Estas etapas estão fora do escopo deste trabalho e podem usar os trabalhos de referência apresentados na fundamentação teórica.

4.3 GRUPO FOCAL

Para avaliar a proposta de metodologia apresentada neste trabalho, aproveitou-se do grupo de pesquisa em PON, especialistas no paradigma. Os oito especialistas que participaram do grupo focal são identificados no texto 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 documento de transcrição da sessão ficou com 56 páginas. A seguir são apresentadas somente as principais contribuições, que foram incorporadas na metodologia.

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 usando uma versão preliminar do DFH, e, à medida que o diagrama e sua modelagem foram sendo apresentados, abria-se tempo para discussão e contribuições. Como esperado, o grupo focal estimulou a interação entre os participantes e diversas discussões surgiram.

Das principais contribuições práticas dos participantes, que foram incorporadas à metodologia, uma foi a adição do elemento Hólon FBE (HF) ao DFH. Percebeu-se que a existência desse elemento lógico do PON permitiria uma maior capacidade 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 decomposição. 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 (cf. Figura 55). O E1 destacou 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 Figura 55, 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. Em relação a esta visão, houve uma discussão sobre limitar o uso de alguns tipos de hólons nos níveis mais abstratos, por exemplo os Hólons Decisionais (HD) que são os hólons que possuem a semântica de decisão. Porém, esta ideia foi abandonada após os participantes concordarem que isto poderia limitar a capacidade de modelagem do projetista.

Além disso, é pertinente salientar que o projetista pode aplicar esta abordagem de “modelagem por questionamentos”, usando somente Hólons Decisionais nos níveis mais abstratos. Isto porque a definição formal do DFH não determina obrigatoriedade de uso de todos os tipos de hólons em todos os níveis, ao contrário disso, a definição é flexível para que o projetista use a abordagem que for mais adequada aos seus objetivos de modelagem de software para o PON.

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 com os modelos criados com o DFH, questionando se não ficariam demasiadamente complexos, mas concordou que não foi o caso para a modelagem apresentada “Sacar dinheiro” (cf. Figura 59). 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 nas seções “Verificação e validação dos modelos” e “Geração semiautomatizada de código”, pode-se, a partir da visão planificada do modelo, realizar o mapeamento para os elementos do PON. Desta forma, mesmo que esta transição não esteja devidamente marcada, as etapas da NOM garantem a compatibilidade com o PON e, consequentemente, a definição dos elementos para o paradigma.

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.

4.4 REQUISITOS DE QUALIDADE PARA LINGUAGENS DE MODELAGEM

Esta seção retoma os requisitos de qualidade para linguagens de modelagem apresentados na fundamentação teórica e os discute em relação ao que foi alcançado nas propostas para a Metodologia de Projeto de Software Orientado a Notificações. Devido ao estágio atual de desenvolvimento da NOM, a avaliação desses requisitos se realizou na forma de discussões a partir da percepção dos proponentes da metodologia e interações com pesquisadores do Paradigma Orientado a Notificações.

O *Guidelines of Modeling* (GoM) (SCHUETTE; ROTTHOWE, 1998) define seis princípios de projeto que visam direcionar esforços de modelagem para criação de modelos de software com qualidade. Os princípios de *adequação de construção*, de *adequação de linguagem* e de *eficiência econômica* são apontados pelos autores do GoM como obrigatórios para garantia mínima de qualidade dos modelos e os outros três, nomeadamente *princípios de clareza*, *de projeto sistemático* e *de comparabilidade* são considerados desejáveis.

O primeiro princípio obrigatório do GoM, i.e. *princípio de adequação de construção*, diz respeito à qualidade do modelo em representar a realidade. Porém, conforme os próprios autores da GoM argumentam, este princípio é de difícil avaliação pois cada modelo é a representação da visão do projetista. Desta forma, avaliar este princípio é avaliar a percepção de projetistas, e dos envolvidos no processo de modelagem, em relação ao poder da linguagem para representar a realidade.

No estágio atual de desenvolvimento da NOM, somente os proponentes da metodologia fizeram esforços no seu uso, o que dificulta a avaliação pela visão de outros projetistas. Por outro lado, a experiência com o grupo focal aplicado com os pesquisadores do PON, mostrou que os participantes não tiveram dificuldade para interpretar o modelo criado para o caso de uso “Sacar dinheiro”. O exemplo foi escolhido por ser real e de amplo conhecimento. Assim, pode-se inferir que é possível criar modelos usando a linguagem de modelagem proposta (DFH), que represente a realidade. Para avaliar efetivamente este princípio em relação à NOM será necessário o seu uso extensivo. Entretanto, as experiências conduzidas até o momento indicam que as escolhas para a NOM a conduzem no atendimento do princípio de adequação de construção do GoM.

O segundo princípio do GoM é o princípio de adequação de linguagem. Este princípio é subdividido em duas formas. Primeiramente, visa assegurar a conformidade de linguagem. Segundo os autores da GoM, esta conformidade está ligada à seleção de técnicas e linguagens de modelagem que possam representar a realidade em que se encontra o problema.

A realidade da NOM é o Paradigma Orientado a Notificações (PON), assim o contexto para o qual a metodologia foi proposta, requereu esforços no desenvolvimento de técnicas e linguagens próprias. Isto se deu, após a não identificação de abordagens existentes na literatura que fossem adequadas ao problema, via mapeamento sistemático da literatura.

Para obter a conformidade de linguagem, buscou-se fundamentar a NOM no arcabouço teórico que inspirou o PON, isto é, os Sistemas Holônicos. Desta forma, a linguagem proposta na metodologia se apropria dos benefícios de generalidade da abordagem holônica para a modelagem de todo tipo de sistema e direciona esforços para concepção de aplicações para o PON.

A segunda forma do princípio de adequação, chamada de correção da linguagem, está ligada ao uso correto da sintaxe da linguagem. Os autores do GoM apontam o papel fundamental do metamodelo e de uma ferramenta de modelagem para garantir a corretude dos modelos. Para garantia de correção da linguagem definida na NOM, o seu metamodelo foi definido matematicamente (ver seção 3.4).

Não está no escopo deste trabalho de pesquisa o desenvolvimento de uma ferramenta de modelagem e, assim, optou-se pelo uso do software Microsoft Visio para criação dos modelos da NOM. Essa ferramenta não permite a verificação dos modelos em relação ao metamodelo, mas permite o uso de elementos personalizados. Outras ferramentas de modelagem que permitem o uso de elementos personalizados foram testadas informalmente no decorrer da pesquisa, porém, nenhuma delas atendia às características de modelagem proposta neste trabalho. Assim, o desenvolvimento de uma ferramenta própria de modelagem será objeto de trabalhos futuros.

O terceiro princípio do GoM é o princípio de eficiência econômica. Segundo os autores da GoM, este princípio afere os benefícios da modelagem de software em relação aos custos envolvidos. Este princípio somente poderá ser verificado após uso extensivo da metodologia proposta. Porém, acredita-se que os esforços empregados no desenvolvimento da NOM também direcionam a metodologia para o

atendimento deste princípio. Isto pôde ser observado no grupo focal aplicado com os pesquisadores PON, no qual foi evidenciada a dificuldade em desenvolver softwares no PON, sobretudo para projetistas acostumados com o paradigma dominante (i.e. o Paradigma Orientado a Objetos). Os participantes do grupo focal destacaram o aspecto do controle do fluxo do software no PON, que é diferenciado em relação a outras abordagens algorítmicas. Sobre esse aspecto, os pesquisadores destacaram como qualidade do Diagrama de Fluxo Holônico (DFH) da NOM a evidência que o diagrama confere aos fluxos que estão sendo projetados. A partir das falas dos pesquisadores, pode-se supor que o uso da linguagem de modelagem tem potencial para melhorar as condições de desenvolvimento de aplicações para o PON, assim, promove-se a eficiência econômica.

O quarto princípio é desejável, mas não obrigatório. Chamado de princípio de clareza, os autores do GoM atribuem a este princípio os aspectos de organização hierárquica dos modelos, organização dos elementos da linguagem e filtros de informações. A organização hierárquica dos modelos permite aos projetistas, segundo os autores da GoM, a criação de visões em diferentes níveis de abstração, facilitando a compreensão do sistema. A organização hierárquica dos modelos na linguagem da NOM é parte estrutural dela, isto é, a linguagem é baseada na teoria dos sistemas holônicos e oferece aos projetistas a organização por meio de holarquias.

As seções “3.6 Exemplo de modelagem usando o Modelo de Fluxo Holônico” e “4.2 Estudo de caso” evidenciam o uso da hierarquia dos hólons, nos quais são apresentados alguns níveis de decomposição. Neles, os envolvidos na modelagem do projeto podem ter uma visão macro do modelo do software até elementos mais detalhados que descrevem as funcionalidades pretendidas em um nível mais próximo da lógica do PON.

A organização dos elementos da linguagem, segundo os autores da GoM, diz respeito a questões gráficas, ou seja, como os elementos da linguagem são apresentados. Neste aspecto, são consideradas características como simetria, alinhamento e distribuição dos elementos do modelo, por exemplo. Neste aspecto é necessário o uso mais extensivo da linguagem, além de uma ferramenta gráfica. Porém, a receptividade dos especialistas do grupo focal em relação aos elementos da linguagem são indícios de que as escolhas para a metodologia direcionam os esforços para atendimento destes requisitos. Inclusive, os elementos gráficos

receberam melhorias apresentadas pelos especialistas que participaram do grupo focal.

Por último, sobre o aspecto de filtro de informações, os autores do GoM destacam a possibilidade de o usuário escolher o nível de informações que deseja visualizar. Este item depende, novamente, do desenvolvimento de ferramenta específica. Porém, o projeto da linguagem da NOM (o DFH), prevê diferentes visões com o uso das hierarquias de hólons, em que o projetista pode ter uma visão macro dos softwares ou casos de uso na forma da visão planificada até o detalhamento de um único hólon. Desta forma, empregou-se esforços de concepção no DFH para atender este aspecto de qualidade para criação de modelos.

O quinto princípio, de projeto sistemático, diz respeito à capacidade do modelo em apresentar, de forma consistente, as diferentes visões da modelagem. Os autores do GoM destacam, em especial, a necessidade desta consistência entre as visões estruturais e comportamentais. No caso do DFH, o uso dos sistemas holônicos direcionou a criação de um diagrama com dupla função: estrutural e comportamental. Porém, vale ressaltar a importância da consistência nas diferentes visões. Assim, acredita-se que a definição matemática do metamodelo do DFH seja suficiente para atender a este princípio.

O último princípio do GoM, de comparabilidade, foca na comparação semântica entre dois modelos e os autores do guia apontam a convenção de nomes e *layouts* pré-configurados como características que melhoram a qualidade deste princípio nos modelos. Este princípio precisa ser melhor explorado na NOM. Apesar da NOM não apresentar *layouts* pré-configurados, pode-se perceber no caso de estudo, apresentado na seção 4.2, que algumas situações se repetem e poderiam ter *layouts* pré-configurados.

Um exemplo é a modelagem da leitura do teclado do terminal ATM, ilustrado na Figura 80 e na Figura 83. Sobre a convenção de nomes, a NOM sugere que os elementos decisionais sejam nomeados na forma de questionamentos, porém para os outros elementos, não há sugestões diretas. É importante frisar que a linguagem criada para a NOM (o DFH), atua na concepção de software e, assim, optou-se por direcionar os esforços em um menor número de tipos de elementos e em uma maior liberdade na definição dos nomes dos elementos como forma de oferecer flexibilidade. Todavia, estas escolhas carecem de maior uso da linguagem para melhor avaliação em relação ao princípio de comparabilidade.

Os princípios da GoM dizem respeito à qualidade de modelos em geral, ou seja, às qualidades que quaisquer modelos deveriam ter. A proposta da NOM é de atuar em um domínio específico, o Paradigma Orientado a Notificações. Assim, avalia-se a metodologia, na sequência, segundo diretrizes elaborados por Ulrich Frank (2011, 2013) como recomendações para linguagens de modelagem de domínios específicos (DSML – do inglês *Domain-Specific Modeling Language*). Iniciou-se com os requisitos/recomendações definidos por Frank (2013) para criação da DSMLs.

O primeiro requisito, ou recomendação, chamado P1, diz respeito à familiaridade dos usuários da linguagem com os conceitos nela usados. Este requisito está relacionado com a primeira diretriz definida para este trabalho de pesquisa (ver seção 1.2) “A NOM deverá ser orientada aos conceitos e primitivas do PON.”. Assim, o DFH, linguagem criada na NOM, segue esta recomendação pois os elementos criados para o diagrama foram baseados nos princípios do PON de forma que os HDs se relacionam com as condições das futuras regras, os HTs e HEs se relacionam com os métodos e atributos dos futuros FBEs e os fluxos entre hólons se relacionam com as notificações do software PON.

O segundo requisito (P2), sugere fornecer conceitos específicos do domínio, sem que se perca a semântica variável dentro do escopo de aplicação da linguagem. Esta recomendação tem por objetivo maximizar o auxílio ao projetista com a linguagem, porém sem limitá-lo. A opção de projeto para o DFH neste sentido foi usar a abordagem holônica, que é generalista no que diz respeito à representação de sistemas, porém definiu-se elementos baseados nos princípios do PON para oferecer a semântica necessária para auxiliar os projetistas na concepção de aplicações neste domínio. O grupo focal validou com especialistas PON as escolhas para o DFH e, na ocasião, um novo elemento foi incorporado à linguagem (Hólon FBE). Conforme foi relatado na seção anterior, os especialistas, por vezes, sentiram a necessidade de novos elementos para o DFH, porém, ao final da seção de três horas de avaliação da NOM e DFH, concluíram que os elementos apresentados, com a inclusão do HF, mostravam-se suficientes para modelagem de software no Paradigma Orientado a Notificações.

O terceiro requisito (P3), diz respeito à linguagem permitir a modelagem em um nível de detalhe suficiente para todas as aplicações previsíveis. Para cobrir outras aplicações possíveis, ela deve fornecer mecanismos de extensão. Para

validar esta recomendação é necessário maior uso da linguagem em diferentes cenários de aplicação. Porém, o estudo de caso sobre a modelagem de parte de um ATM, exposto na seção 4.2, é generalista, e permite a interpretação de que a linguagem pode ser usada para muitas aplicações. O estudo de caso expõe interação com usuário, com dispositivos externos, assim como operações internas, sendo possível interpretar que estes tipos de funcionalidades são comuns à maior parte das aplicações. No que diz respeito a mecanismos de extensão da linguagem, não houve formalização de tais mecanismos no metamodelo do DFH.

O quarto requisito (P4), sugere que uma linguagem de modelagem deve fornecer conceitos que permitam distinguir claramente diferentes níveis de abstração dentro de um modelo. No projeto do DFH, esta recomendação é tratada pela abordagem holônica, na qual a visão mais abstrata é mostrada primeiro e os projetistas navegam (*zoom-in*) para níveis mais detalhados a partir dos elementos existentes. No grupo focal com os especialistas em PON foi identificado certa falta de clareza na identificação dos elementos que possuem decomposição. Assim, por sugestão dos próprios especialistas, a representação visual da linguagem foi melhorada para deixar mais evidente quando um elemento possui decomposições.

O quinto requisito (P5), informa que deve haver um mapeamento claro dos conceitos da linguagem com o que se pretende representar. Em um caso ideal, todas as informações que se deseja representar podem ser extraídas do modelo. Deixar para que os potenciais usuários mapeiem os conceitos poderá contribuir para aumentar os riscos e os custos. O processo de modelagem da NOM prevê etapas de verificação e validação dos modelos criados pelos projetistas para compatibilizá-los com a lógica do Paradigma Orientado a Notificações e com os casos de uso que os originaram. Estas etapas visam auxiliar o projetista, direcionando esforços para modelos compatíveis com o PON e que atendam aos requisitos de software definidos anteriormente nos casos de uso.

Os próximos requisitos delineados por Frank (2013), dizem respeito a requisitos de seleção para a linguagem de metamodelagem escolhida para criação da DSML. São oito requisitos, nomeados de MM1 até MM8.

O requisito MM1 sugere que a linguagem de metamodelagem deve ser suportada por um ambiente de software. O DFH, linguagem da NOM, não é suportado totalmente por software atualmente. A criação ou uso de uma ferramenta

de modelagem estava fora do escopo deste trabalho. Assim, usou-se a ferramenta Microsoft Visio para criação dos modelos.

O requisito MM2 enfatiza a necessidade de que a linguagem de metamodelagem ofereça clara distinção entre os níveis do metamodelo, como M1 ou M2. Neste trabalho, a apresentação da linguagem criada para a NOM, o DFH, não evidencia níveis de modelo e metamodelos, se visto sob a ótica da arquitetura de quatro níveis (M0-M3) descrita pelo *Object Manage Group* (OMG, 2016). Assim, para evidenciar esta característica no DFH, a camada M3 diz respeito ao conceito de hólon, usado na teoria dos sistemas holônicos. A partir da camada M3, a camada M2 foi especificada matematicamente na seção 3.4 e é a linguagem de modelagem de domínio específico criada para o PON. A camada M1, consequentemente, envolve os modelos criados com o DFH. A seção 3.6 e a seção 4.2 apresentam modelos da camada M1. A camada M0 não tem representação definida no DFH, porém, como se pode observar nas seções de verificação e validação dos modelos e geração semiautomatizada de códigos, por vezes há mapeamento de 1 para 1 entre elementos da camada M1 e M0. Assim, sob as recomendações deste requisito, é importante que sejam realizados mais estudos para avaliar se o nível de distinção entre as camadas M1 e M0 é satisfatório.

O requisito MM3 está relacionado à notação gráfica da linguagem de metamodelagem. No caso do DFH, desenvolveu-se uma nova linguagem, porém, os elementos gráficos foram projetados para ter certa similaridade com elementos gráficos de outras abordagens. Por exemplo, o elemento decisional (HD) possui a forma com similaridade ao losango que é elemento de decisão nos diagramas de fluxogramas, diagramas de atividade da UML e diagramas de modelagem de processos da BPMN. Os outros elementos possuem forma retangular, assemelhando-se a elementos que representam atividades, funções e dados nos mesmos diagramas citados. Assim, buscou-se uma diferenciação nos elementos, mas sem causar grande estranheza. No grupo focal com os especialistas não foi observada críticas em relação à notação gráfica. Ao contrário, ela foi elogiada pelos participantes.

O requisito MM4 sugere que os conceitos de uma linguagem de metamodelagem devem ser complementados por uma linguagem para especificar restrições. No projeto do DFH não foi prevista linguagem complementar para especificar restrições na modelagem. As restrições são descritas em linguagem

natural. Porém, como o DFH é um diagrama de fluxo que possui elementos e relações entre eles, não parece haver óbice ao uso de uma linguagem complementar, como a *Object Constraint Language* (OCL) do Object Management Group (2014), caso o projetista desejar.

O requisito MM5 sugere que deve existir um mapeamento claro dos conceitos utilizados para o desenvolvimento de software. Na abordagem proposta neste trabalho, houve o desenvolvimento da própria linguagem de metamodelagem e, assim, foram empreendidos esforços para que a linguagem oferecesse os recursos necessários para o desenvolvimento de software no Paradigma Orientado a Notificações, ou seja, o domínio específico da linguagem desenvolvida.

O requisito MM6 sugere que uma linguagem de metamodelagem deve permitir a distinção entre diferentes níveis de abstrações. Este requisito é alcançado pela linguagem de metamodelagem pois herda as características de abstração dos sistemas holônicos em múltiplos níveis.

Segundo o requisito MM7, uma linguagem de metamodelagem deve fornecer conceitos que permitam representar instâncias. Este requisito não é completamente atendido pelo DFH. Mesmo que em algumas circunstâncias um elemento do nível M1 seja mapeado para um elemento do nível M0, a especificação do DFH não tem uma definição formal sobre isso. Isto se deve ao fato de os esforços da proposta deste trabalho serem direcionados a aspectos mais abstratos de concepção de software para o PON. É importante ressaltar que, mesmo o DFH não atendendo completamente este requisito, a linguagem de metamodelagem não é fator limitante. Ademais, sugere-se que após a etapa de modelagem com o DFH sejam empreendidos esforços para modelagem lógica usando abordagens existentes como o DON.

O último requisito, MM8, diz respeito à disseminação e padronização da linguagem de metamodelagem. Segundo o autor das diretrizes, quanto maior a disseminação da linguagem, mais ela será atrativa em termos de economia e escala. A linguagem de metamodelagem DFH foi desenvolvida no âmbito deste trabalho e, assim, para fins de atendimento deste requisito, não houve tempo de uso extensivo para disseminação da linguagem.

Além dos requisitos para linguagens e metamodelos para modelagem de domínios específicos, Frank (2013) propôs um método para criação de Linguagens de Modelagem de Domínios Específicos (DSML), que é ilustrado na Figura 40. A

concepção de uma linguagem de modelagem para o PON foi o principal artefato deste trabalho de pesquisa e, assim, é importante destacar que as etapas da pesquisa são compatíveis com o método proposto por Frank (2013) para criação de DSMLs, conforme descrito na seção 3.1 do capítulo Metodologia de Projeto de Software Orientado a Notificações (NOM). Especificamente a etapa de “Projeto da Notação Gráfica” apresenta um conjunto de 9 diretrizes. A seguir, discute-se as diretrizes em relação ao projeto do DFH.

A Diretriz 1 sugere definir categorias semânticas de conceitos. No projeto do DFH optou-se pela categorização por princípios do Paradigma Orientado a Notificações, isto é, recorreu-se aos elementos que fundamentam a lógica do paradigma para propor categorias semânticas compatíveis.

A Diretriz 2 sugere que sejam criados símbolos genéricos por categoria para promover a percepção e interpretação apropriadas. O projeto da notação gráfica do DFH segue esta recomendação e foram criados elementos gráficos distintos, conforme apresentado no Quadro 8.

A Diretriz 3 sugere que a distinção gráfica deverá maior entre elementos com maior diferença semântica. O Quadro 8 também apresenta as diferenças semânticas entre os elementos da linguagem. Buscou-se esta diferenciação no projeto dos elementos gráficos do DFH, porém sem que se causasse conflito com a diretriz MM3.

A Diretriz 4 sugere dar preferência aos ícones às formas. Esta diretriz pode ser observada no projeto gráfico do DFH em que a maior parte dos elementos possuem forma retangular, porém com detalhes que os caracterizam também como ícones. Estas escolhas de projeto refletem no atendimento da próxima diretriz, número 5, em que se sugere a combinação de forma, ícones, cor e texto.

A Diretriz 6 chama atenção para que se evite a sobrecarga de símbolos. No projeto do DFH, o próprio número de elementos semânticos é limitado, o que o leva em direção ao atendimento também da Diretriz 7 que sugere evitar o uso de símbolos redundantes.

A Diretriz 8 sugere representar características semânticas monotônicas de um conceito por meio de composições de símbolos. Esta diretriz é suportada pelas proposições deste trabalho pois, ao mesmo tempo em que a abordagem holônica confere autonomia, confere também interatividade e comunicação entre os

elementos da linguagem, proporcionando meios para composição de elementos de diferentes formas.

A Diretriz 9 sugere que a notação gráfica deve incluir símbolos que permitam reduzir a complexidade do diagrama e cita como exemplo um símbolo usado para representar processos agregados. A abordagem holônica usada no projeto da linguagem apresentada neste trabalho segue esta diretriz e a expande interpretando que, não somente processos (i.e., Hólons Transacionais) podem ser decompostos para melhorar a interpretação dos modelos, mas que isso se aplica, também, a elementos de decisão que podem ser decompostos em decisões mais detalhadas.

O objetivo desta seção foi ampliar a discussão a respeito das escolhas de projeto para a Metodologia de Projeto Orientado a Notificações (NOM). A característica propositiva desta pesquisa, limita até certo ponto a sua avaliação. Assim, espera-se que o uso extensivo da metodologia e da linguagem criada para ela possam evoluir e ser reavaliada.

5 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho de pesquisa tem como contexto o desenvolvimento de software dentro do Paradigma Orientado a Notificações (PON), que é objeto de pesquisa na UTFPR. O PON foi proposto como uma abordagem para desenvolvimento de software com o intuito de melhorar o desempenho das aplicações e facilitar o desenvolvimento delas, contribuindo, tendo assim potencial para contribuir para com o aumento da produtividade, melhoria da qualidade e redução da complexidade em engenharia de software. O PON, entretanto, é recente, se comparado com outras abordagens de desenvolvimento de software. Grande parte dos esforços de pesquisa atuais sobre o paradigma se concentram na comprovação dos seus benefícios e na evolução de sua teoria e técnicas, além de estudos sobre seu uso prático.

O trabalho de pesquisa apresentado neste documento de tese, focou em aspectos que foram menos explorados em trabalhos anteriores do PON. Isto é, focou-se em aspectos de concepção de software PON, buscando, a partir de uma especificação de requisitos para um software, oferecer ao projetista meios para que ele possa desenvolver uma solução de software no PON. A revisão bibliográfica e o mapeamento sistemático da literatura, desenvolvidos no âmbito deste trabalho, evidenciou que processos de desenvolvimento de software baseado em regras não evoluíram expressivamente, limitando-se a poucas abordagens. Não possuir um processo de desenvolvimento de software eficaz para o PON pode afetar a evolução do próprio paradigma, sobretudo quando os softwares sendo desenvolvidos se tornam mais extensos e complexos.

A partir desta problematização e da hipótese de que uma nova metodologia de projeto de software explicitamente orientada aos conceitos do PON possa minimizar as dificuldades na concepção e desenvolvimento de aplicações PON, traçou-se como objetivo desta pesquisa de doutorado a criação de uma metodologia, que foi denominada Metodologia de Projeto de Software Orientado a Notificações (NOM). A NOM diferencia-se das demais abordagens encontradas na literatura e dos esforços anteriores do grupo de pesquisas no PON pois atua com maior ênfase em aspectos de concepção de software de acordo com o paradigma. Neste sentido, a NOM não concorre com as outras abordagens e, sim, atua complementando-as.

Para cumprir o objetivo de pesquisa, buscou-se, inicialmente, fundamentá-la a partir dos esforços empregados no grupo de pesquisas do PON a respeito das abordagens de modelagem usadas no paradigma. Para a pesquisa na literatura, optou-se pela realização de um mapeamento sistemático (ver seção 2.2). Além disso, como o objetivo desta pesquisa era o de criar uma metodologia, que envolveria, também, a criação de uma linguagem de modelagem para concepção de aplicações para o PON, buscou-se fundamentação na literatura sobre as melhores práticas e recomendações para criação de linguagens de propósito específico (resultados apresentados na seção 2.3). Ainda, no que tange a fundamentação teórica para atingir o objetivo da pesquisa, buscou-se inspiração nas teorias que influenciaram a criação do próprio PON, os sistemas holônicos (seção 2.4).

Na sequência, criou-se a primeira versão da metodologia NOM, cujos esforços se concentraram na definição de um processo e linguagem de modelagem direcionada para o PON. A NOM é apresentada no capítulo 3 deste documento.

A avaliação das propostas apresentadas neste trabalho usou as técnicas de estudo de caso e grupo focal. Adicionalmente, apresentou-se uma discussão que avaliou quais requisitos e recomendações para criação de linguagens de modelagem de propósito específico a metodologia NOM atendeu. No estudo de caso, apresentou-se um exemplo completo do processo de modelagem da NOM para um caso de uso. Sabe-se da limitação do uso de exemplos como forma de avaliação e que a sua significância está associada à complexidade deles. Assim, optou-se por apresentar a modelagem da operação de sacar dinheiro em um caixa eletrônico (ATM) que, além de envolver diversas decisões, possui lógica sequencial de execução, sabidamente difícil de implementar em sistemas baseados em regras. O grupo focal com especialistas em PON, além de ter avaliado a metodologia proposta, evidenciou melhorias necessárias. Assim, a partir das sugestões dos especialistas, a NOM recebeu um incremento e a versão final da metodologia foi apresentada no capítulo 3 deste documento.

5.1 PRINCIPAIS CONTRIBUIÇÕES DA TESE

A principal contribuição desta tese de doutorado foi a definição da Metodologia de Projeto de Software Orientado a Notificações (NOM). A NOM é composta de um abrangente conjunto de atividades, que apoia projetistas no processo de modelagem de software para o Paradigma Orientado a Notificações (PON). O processo iterativo e incremental da NOM permite que o modelo de software PON seja ampliado e detalhado sucessivamente a cada nova iteração. As atividades de verificação e validação direcionam os esforços do projetista para que o modelo do software seja aderente ao PON ao mesmo tempo que visa garantir que o modelo do software atenda aos requisitos, i.e. o caso de uso.

A NOM diferencia-se de outras abordagens por apresentar um processo e uma notação de modelagem próprios para o PON. O processo de modelagem na NOM é realizado com uma abordagem denominada **Modelagem de Fluxo Holônico (MFH)**. Nessa abordagem, o projetista usa uma linguagem de modelagem (**Diagrama de Fluxo Holônico - DFH**), desenvolvida no âmbito deste trabalho, para criar os modelos do software PON, a partir de um modelo de requisitos na forma de casos de uso (i.e., Modelo de Casos de Uso). Ressalta-se os esforços empreendidos na criação da MFH para que fossem privilegiados aspectos de concepção de software PON, ou seja, que apoiasse o projetista de software PON, a partir dos requisitos do software, definidos no âmbito dos casos de uso, na criação de modelos de software que pudessem ser implementados no paradigma. Na seção 3.7 é apresentada uma técnica de verificação de modelos, cuja função é verificar se os níveis mais detalhados de um modelo criado com o DFH são compatíveis com a lógica do PON. Esta compatibilização permite que seja realizada a geração semiautomatizada de código para a Tecnologia LingPON, apresentada na seção 3.8. A seção 3.9 complementa o conjunto de evidências, discutindo e apresentando aspectos de criação de código PON a partir de modelos criados na MFH.

A MFH incorpora, com seu processo e linguagem de modelagem, uma visão diagramática unificada, ausente nas abordagens anteriores. A visão planificada do modelo, por exemplo, permite ao projetista ter um panorama global da solução criada por ele. Além disso, a abordagem holônica orientada a fluxos permite que o projetista tenha uma referência visual de causa e efeito, ou seja, de sequência entre

os elementos do diagrama. Estas contribuições são especialmente úteis para projetistas de software que estejam iniciando no PON.

Outro aspecto da NOM foi o uso da teoria dos Sistemas Holônicos para criar uma abordagem de modelagem. Isto, em si, não é novidade. Todavia, uma contribuição importante, dos esforços deste trabalho de pesquisa, foi a introdução do conceito de decomposição de decisões, com a criação do Hólon Decisional (HD). Comumente, na modelagem de processos e de softwares, elementos decisionais não podem ser decompostos. É o caso do diagrama de atividade da UML em que há subatividades, mas não há subdecisões. Similarmente, na modelagem de processos usando BPMN há o conceito de subprocessos, mas não há o conceito de *subgateways* (gateway é o elemento de decisão da BPMN). Assim, esta contribuição pode oferecer novas possibilidades de modelagem, em especial, permitir criar cenários em que decisões mais abstratas podem ser decompostas em subdecisões (conforme ilustrado na seção 3.5.1).

O trabalho de pesquisa, apresentado neste documento de tese, buscou mitigar dificuldades no desenvolvimento de software para o Paradigma Orientado a Notificações (PON) com a definição de uma nova metodologia. Para isso, fundamentou-se no PON e nas recomendações para criação de linguagens de modelagem para contemplar o seu aspecto de ser uma pesquisa propositiva. Sabe-se que a avaliação de pesquisas que criam metodologias ou métodos são, muitas vezes, subjetivas e necessitam de um tempo e esforço maiores para sua comprovação definitiva. Todavia, acredita-se que os esforços empreendidos nesta pesquisa, apresentados neste documento de tese, auxiliam no amadurecimento de aspectos de modelagem de software PON.

5.2 TRABALHOS FUTUROS

5.2.1 Desenvolvimento de uma ferramenta de modelagem para a MFH

No decorrer deste trabalho de pesquisa, usou-se a ferramenta Visio, da Microsoft, para criação dos diagramas da MFH. Não foram encontradas ferramentas compatíveis, ou que pudessem ser adaptadas, com a proposta de Modelagem de Fluxo Holônico. Em especial, as ferramentas encontradas não são compatíveis com

o mecanismo de decomposição (*zoom-in* e *zoom-out*) na forma proposta pela MFH. A falta de uma ferramenta de modelagem para a MFH não prejudicou o desenvolvimento da tese, porém, é fato que se houvesse uma ferramenta, muitos problemas seriam evitados. Um exemplo simples é a modificação do nome de um hólon. Na ferramenta Visio é necessário buscar todas as ocorrências deste elemento para modificar o nome. Em uma ferramenta de modelagem apropriada, a modificação de um elemento aplica-se a todo o modelo.

É fato que o desenvolvimento de uma ferramenta de modelagem para a MFH é essencial para que a metodologia NOM possa ser usada por projetistas do PON. Inclusive, o segundo princípio definido pelo GoM para criação de modelos com qualidade, princípio de adequação de linguagem, indica a necessidade de uma ferramenta de modelagem.

O capítulo 3, deste documento, apresenta a descrição da metodologia NOM e a especificação da MFH. Assim, o desenvolvimento da ferramenta de modelagem para a NOM pode partir dessas especificações. A NOM define uma série de atividades, que envolvem verificação e validação dos modelos, porém, inicialmente, o desenvolvimento da ferramenta pode se concentrar em criar um ambiente de modelagem que suporte as definições apresentadas na seção 3.4 e, em especial, o mecanismo de decomposição (*zoom-in* e *zoom-out*) exemplificado na seção 3.5, além da funcionalidade que apresenta a visão planificada do modelo.

Os demais recursos apresentados no capítulo 3, como a verificação do modelo em relação à lógica do PON, a criação de diagramas de estados a partir de um DFH ou a geração de código semiautomatizada podem ser implementados futuramente.

5.2.2 Novos exemplos de uso da MFH

Este documento de tese apresentou alguns exemplos da MFH, em especial, um exemplo mais complexo, da operação de sacar dinheiro em um caixa eletrônico, para demonstrar a efetividade e viabilidade da metodologia. Todavia, se os projetistas tiverem ao seu dispor um conjunto com maior número de exemplos, facilitará o uso da metodologia e seu aprendizado. O desenvolvimento de uma

ferramenta de modelagem para a MFH, conforme indicado na seção anterior, facilitará a criação de novos exemplos.

5.2.3 Testar a NOM fora do contexto do PON

A NOM foi concebida para uso no Paradigma Orientado a Notificações. O PON compartilha conceitos de outras abordagens, por exemplo o uso de regras. Assim, um trabalho futuro pode ser avaliar o uso da NOM para modelagem em Sistemas Baseados em Regras (SBRs). As pesquisas realizadas no âmbito deste trabalho evidenciaram a falta de abordagens de processos de desenvolvimento de software para essa classe de sistemas.

Em especial, há uma classe de SBRs, denominada Sistemas de Produção (SPs), que possui maior similaridade com o PON. As regras nos SPs são, assim como no PON, elementos decisionais, ensejando o uso do elemento HD. Nos SPs também há o conceito de FBE, o que enseja o uso dos elementos HE e HT. Ademais, a primitiva de modelagem HX é comum a todo sistema de software que, em geral, necessita relação com entidades externas a ele.

5.2.4 MFH como linguagem de metamodelagem para DSML

As primitivas de modelagem criadas para a MFH inspiraram-se em elementos do PON. Todavia, a dinâmica de modelagem empregada no processo é generalista, ou seja, usa uma abordagem de encadeamento de ideias por meio de fluxos. Assim, uma investigação futura pode ser a generalização da MFH de forma a defini-la como linguagem de metamodelagem. Neste caso, aproveita-se os mecanismos de organização e decomposição, herdados dos sistemas holônicos, e, permite-se a definição de primitivas de modelagem para uma nova linguagem de modelagem de domínio específico.

REFERÊNCIAS

7GRAUS. **Sinônimos.com.br**. Disponível em: http://www.sinonimos.com.br/fato/. Acesso em: 13 ago. 2015.

ABDULLAH, Mohd Syazwan *et al*. A UML Profile for Knowledge-Based Systems Modelling. In: 5TH ACIS INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING RESEARCH, MANAGEMENT & APPLICATIONS (SERA 2007), 5., 2007, Busan. **Proceedings [...] .** [S. L.]: Ieee, 2007a. p. 871-878.

ABDULLAH, Mohd Syazwan *et al*. Using Unified Modeling Language for Conceptual Modelling of Knowledge-Based Systems. In: CONCEPTUAL Modeling - ER 2007: 26th International Conference on Conceptual Modeling, Auckland, New Zealand, November 5-9, 2007. Proceedings. Auckland: Springer, 2007b. p. 438-453. (Lecture Notes in Computer Science 4801).

AGARAM, Mukundan K.; LIU, Chang. Vocabulary Model Requirements for Production Rule Systems. In: IEEE 16TH INTERNATIONAL ENTERPRISE DISTRIBUTED OBJECT COMPUTING CONFERENCE WORKSHOPS, 16., 2012, Beijing. **Proceedings [...] .** [S. L.]: I, 2012. p. 132-139.

AGÜERO, Jorge *et al*. Agent Capability Taxonomy for Dynamic Environments. In: HYBRID Artificial Intelligent Systems: 7th International Conference, HAIS 2012, Salamanca, Spain, March 28-30th, 2012. Proceedings, Part I. Salamanca: Springer, 2012. p. 37-48. (Lecture Notes in Computer Science 7208).

ASSOCIAÇÃO BRASILEIRA DAS EMPRESAS DE SOFTWARE (ABES) (São Paulo). **Mercado Brasileiro de Software**: panorama e tendências. São Paulo: Associação Brasileira das Empresas de Software, 2019. 28 p. Disponível em: https://central.abessoftware.com.br/Content/UploadedFiles/Arquivos/Dados%202011/ABES-EstudoMercadoBr. Acesso em: 04 jan. 2021.

ASSOCIAÇÃO PARA PROMOÇÃO DA EXCELÊNCIA DO SOFTWARE BRASILEIRO (SOFTEX) (Brasília). **Relatório de Atividades 2016**. Brasília: Comunicação Softex, 2017. 187 p. Disponível em: http://ftp.softex.br/relatorio2016/book-softex-2016.pdf. Acesso em: 04 jan. 2021.

ASSOCIAÇÃO PARA PROMOÇÃO DA EXCELÊNCIA DO SOFTWARE BRASILEIRO (SOFTEX). **A Indústria Brasileira de Software e Serviços de TI (IBSS)**. Brasília: Softex Comunicação, 2013. Disponível em: http://www.softex.br/ti-brasileira/. Acesso em: 08 mar. 2016.

BABA-HAMED, Latifa. Rules Termination Analysis Based on Petri Nets: implementation issues. In: 2006 2ND INTERNATIONAL CONFERENCE ON INFORMATION & COMMUNICATION TECHNOLOGIES, 2., 2006, Damascus. **Proceedings [...] .** [S. L.]: Ieee, 2006. p. 3540-3545.

BACKUS, John. Can programming be liberated from the von Neumann style?: a functional style and its algebra of programs. **Communications Of The Acm**, [S. L.], v. 21, n. 8, p. 613-641, ago. 1978.

BAMBINI, Martha Delphino; MENDES, Cássia Isabel Costa; MOURA, Maria Fernanda; OLIVEIRA, Stanley Robson de Medeiros. Software para Agropecuária: panorama do mercado brasileiro. **Parcerias Estratégicas**, Brasília, v. 18, n. 36, p. 175-198, jun. 2013. Ed. Especial. Disponível em: http://ainfo.cnptia.embrapa.br/digital/bitstream/item/103027/1/SW-Bambini.pdf. Acesso em: 04 jan. 2021.

BANASZEWSKI, Roni Fabio. **Paradigma Orientado a Notificações**: avanços e comparações. 2009. 283 f. Dissertação (Mestrado) - Curso de Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial - Cpgei, Universidade Tecnológica Federal do Paraná - Utfpr, Curitiba, 2009.

BANASZEWSKI, Roni Fabio; STADZISZ, Paulo Cézar; TACLA, Cesar Augusto; SIMÃO, Jean Marcelo. Notification Oriented Paradigm (NOP): a software development approach based on artificial intelligence concepts. In: CONGRESS OF LOGIC APPLIED TO TECHNOLOGY, 6., 2007, Santos. **Anais [...] .** Santos: Laptec, 2007. p. 1-8.

BASKERVILLE, Richard; LEVINE, Linda; PRIES-HEJE, Jan; SLAUGHTER, Sandra. How Internet software companies negotiate quality. **Computer**, [S.L.], v. 34, n. 5, p. 51-57, maio 2001. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/2.920612.

BASSILIADES, Nick; ANTONIOU, Grigoris; VLAHAVAS, Ioannis. A Defeasible Logic Reasoner for the Semantic Web. In: RULES and Rule Markup Languages for the Semantic Web: Third International Workshop, RuleML 2004, Hiroshima, Japan, November 8, 2004. Proceedings. Hiroshima: Springer, 2004. p. 49-64. (Lecture Notes in Computer Science 3323).

BATISTA, Márcio Venâncio. **Proposta de um Método de Aplicação da Teoria de Projeto Axiomático ao Desenvolvimento de Software PON-POR**: from theory to implementation. 2013. 241 f. Dissertação (Mestrado) - Curso de Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial - Cpgei, Universidade Tecnológica Federal do Paraná - Utfpr, Curitiba, 2013.

BECK, Kent. **Extreme Programming Explained**: embrace change. [S.L]: Addison-Wesley Professional, 2000. 190 p.

BOGGINO, Adriana Giret. **ANEMONA**: una metodología multi agente para sistemas holónicos de fabricación. 2005. 295 f. Tese (Doutorado) - Curso de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Valência, 2005.

BOUKHEBOUZE, Mohamed *et al*. A Rule-Based Modeling for the Description of Flexible and Self-healing Business Processes. In: ADVANCES in Databases and Information Systems. [S.L]: Springer, 2009. p. 15-27. (Lecture Notes in Computer Science 5739).

CAÑADAS, Joaquín; PALMA, José; TÚNEZ, Samuel. Defining the semantics of rule-based Web applications through model-driven development. **International Journal Of Applied Mathematics And Computer Science**, [S.L.], v. 21, n. 1, p. 41-55, 1 mar. 2011. Walter de Gruyter GmbH. http://dx.doi.org/10.2478/v10006-011-0003-4.

CAÑADAS, Joaquín; PALMA, Jose; TÚNEZ, Samuel. A Model-Driven Method for automatic generation of Rule-based Web Applications. In: 5TH WORKSHOP ON KNOWLEDGE ENGINEERING AND SOFTWARE ENGINEERING, 5., 2009, Paderborn. **Proceedings [...] .** [S. L.]: Ceur-Ws.Org, 2009. p. 1-12.

CARDOSO, Janette; VALETTE, Robert. **Redes de Petri**. Florianópolis: Universidade Federal de Santa Catarina, 1997. 157 p.

CORRADINI, Flavio *et al*. A Constrained ECA Language Supporting Formal Verification of WSNs. In: IEEE 29TH INTERNATIONAL CONFERENCE ON ADVANCED INFORMATION NETWORKING AND APPLICATIONS WORKSHOPS, 29., 2015, Gwangju. **Proceedings [...] .** Gwangju: Ieee, 2015. p. 187-192.

DANIEL, Florian; MATERA, Maristella; POZZI, Giuseppe. Combining conceptual modeling and active rules for the design of adaptive web applications. In: SIXTH INTERNATIONAL CONFERENCE ON WEB ENGINEERING, 6., 2006, Palo Alto. **Proceedings [...] .** New York: Association For Computing Machinery, 2006. p. 1-8.

DERMEVAL, Diego; COELHO, Jorge A. P. de M.; BITTENCOURT, Ig I.. Mapeamento Sistemático e Revisão Sistemática da Literatura em Informática na Educação. In: JAQUES, Patricia; PIMENTEL, Mariano; SIQUEIRA, Sean; BITENCOURT, Ig. **Metodologia de Pesquisa Científica em Informática na Educação**: abordagem quantitativa. Porto Alegre: Sbc, 2020. p. 1-26. (Série Metodologia de Pesquisa em Informática na Educação, v. 2). Disponível em: https://metodologia.ceie-br.org/livro-2. Acesso em: 04 jan. 2021.

DIACONESCU, Ion-Mircea. Methodologies and Tools for Modeling Rule-based Web Services. In: INAUGURAL IEEE-IES DIGITAL ECOSYSTEMS AND TECHNOLOGIES CONFERENCE, 1., 2007, Cairns. **Proceedings [...] .** [S. L.]: Ieee, 2007. p. 471-476.

DRESCH, Aline Lacerda; ANTUNES, Daniel Pacheco; VALLE JÚNIOR, José Antonio. **Design Science Research**. Porto Alegre: Bookman, 2014. 204 p.

DÖLL, Luciano Mathias. **Proposta de uma metodologia para a modelagem da dinâmica de sistemas orientados a objetos usando redes de petri predicado/transição**. 2002. 121 f. Dissertação (Mestrado) - Curso de Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial - Cpgei, Universidade Tecnológica Federal do Paraná - Utfpr, Curitiba, 2002.

EDMONDS, Jeff. **How to Think About Algorithms**. Cambridge: Cambridge University Press, 2008. 472 p.

EITER, Thomas; FEIER, Cristina; FINK, Michael. Simulating Production Rules Using ACTHEX. In: CORRECT Reasoning: Essays on Logic-Based AI in Honour of Vladimir Lifschitz. [S.L]: Springer, 2012. p. 211-228. (Lecture Notes in Computer Science 7265).

ERICSSON, Annemarie *et al*. Verification of an industrial rule-based manufacturing system using REX. In: 1ST INTERNATIONAL WORKSHOP ON COMPLEX EVENT

PROCESSING FOR FUTURE INTERNET, 1., 2008, Vienna. **Proceedings [...] .** Vienna: Sun Site Central Europe Ceur-Ws, 2008. p. 1-10.

EVERMANN, Joerg Magnus. **Using design languages for conceptual modelling**: the uml case. 2003. 395 f. Tese (Doutorado) - Curso de Sauder School Of Business, Universidade da Colúmbia Britânica, Vancouver, 2003.

FERREIRA, Cleverson Avelino. **Linguagem e compilador para o paradigma orientado a notificações (PON)**: avanços e comparações. 2015. 245 f. Dissertação (Mestrado) - Curso de Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial - Cpgei, Universidade Tecnológica Federal do Paraná - Utfpr, Curitiba, 2015.

FERREIRA, Cleverson Avelino; RONSZCKA, Adriano Francisco; IORIS, P. A. M.; KOSSOSKI, Clayton. **Compilador para o Paradigma Orientado a Notificações**. Curitiba: Relatório Técnico da Disciplina Compiladores Ppgca/utfpr, 2013.

FORGY, Charles L.. Rete: a fast algorithm for the many pattern/many object pattern match problem. **Artificial Intelligence**, [S.L.], v. 19, n. 1, p. 17-37, set. 1982. Elsevier BV. http://dx.doi.org/10.1016/0004-3702(82)90020-0.

FOWLER, Martin. **Refatoração**: aperfeiçoando o projeto de código existente. Porto Alegre: Bookman, 2004. 365 p.

FRANK, Ulrich. Domain-Specific Modeling Languages – Requirements Analysis and Design Guidelines. In: REINHARTZ-BERGER, Iris; STURM, Arnon; CLARK, Tony; COHEN, Sholom; BETTIN, Jorn. **Domain Engineering**: product lines, languages, and conceptual models. [S.L]: Springer, 2013. p. 133-157.

FRANK, Ulrich. Some guidelines for the conception of domain-specific modelling languages. In: 4TH INTERNATIONAL WORKSHOP ON ENTERPRISE MODELLING AND INFORMATION SYSTEMS ARCHITECTURES, EMISA 2011, 4., 2011, Hamburg. **Proceedings [...] .** [S. L.]: Gi Lni, 2011. p. 93-106.

FRIEDMAN-HILL, Ernest. **Jess in Action**: rule based system in java. Greenwich: Manning Publications Co, 2003. 480 p.

FRITZEN, Oliver; MAY, Wolfgang; SCHENK, Franz. Markup and Component Interoperability for Active Rules. In: WEB Reasoning and Rule Systems: Second International Conference, RR 2008, Karlsruhe, Germany, October 31-November 1, 2008. Proceedings. Karlsruhe: Springer, 2008. p. 197-204. (Lecture Notes in Computer Science 5341).

FUGGETTA, Alfonso; NITTO, Elisabetta di. Software Process. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 36., 2014, Hyderabad. **Proceedings [...] .** New York: Association For Computing Machinery, 2014. p. 1-12. Disponível em: https://dl.acm.org/doi/10.1145/2593882.2593883. Acesso em: 04 jan. 2021.

GALIN, Daniel. **Software Quality Assurance**: from theory to implementation. Harlow: Pearson Education Limited, 2004. 590 p.

GAMMA, Erich *et al*. **Design patterns**: elements of reusable object-oriented software. Reading: Addison-Wesley, 1995. 395 p.

GIL, Antonio Carlos. **Como Elaborar Projetos de Pesquisa**. 5. ed. São Paulo: Atlas, 2010. 184 p.

GIURCA, Adrian; WAGNER, Gerd. Rule Modeling and Interchange. In: NINTH INTERNATIONAL SYMPOSIUM ON SYMBOLIC AND NUMERIC ALGORITHMS FOR SCIENTIFIC COMPUTING (SYNASC 2007), 9., 2007, Timisoara. **Proceedings [...] .** [S. L.]: Ieee, 2007. p. 485-491.

GONÇALVES, Eder Mateus Nunes. Specifying Knowledge in Cognitive Multiagent Systems Using a Class of Hierarchical Petri Nets. **Journal Of Software**, [S.L.], v. 7, n. 11, p. 2405-2414, 1 nov. 2012. International Academy Publishing (IAP). http://dx.doi.org/10.4304/jsw.7.11.2405-2414.

GOUES, Claire Le; NGUYEN, Thanhvu; FORREST, Stephanie; WEIMER, Westley. GenProg: a generic method for automatic software repair. **Ieee Transactions On Software Engineering**, [S.L.], v. 38, n. 1, p. 54-72, jan. 2012. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/tse.2011.104.

HAWRYSZKIEWYCZ, Igor T.. **Introduction to systems analysis and design**. New York: Prentice Hall, 1991.

HU, Zhifang; LU, Tao; ZHAO, Zhuo. Context-aware service system modeling using timed CPN. In: 2013 10TH INTERNATIONAL CONFERENCE ON SERVICE SYSTEMS AND SERVICE MANAGEMENT, 10., 2013, Hong Kong. **Proceedings [...] .** [S. L.]: Ieee, 2013. p. 1-6.

HUANG, Hongtao; HUANG, Shaobin; ZHANG, Tao. A Formal Method for Verifying Production Knowledge Base. In: FOURTH INTERNATIONAL CONFERENCE ON INTERNET COMPUTING FOR SCIENCE AND ENGINEERING, 4., 2009, Harbin. **Proceedings [...] .** [S. L.]: Ieee, 2010. p. 19-23.

IASSINOVSKI, Serguei; ARTIBA, Abdelhakim; FAGNART, Christophe. A generic production rules-based system for on-line simulation, decision making and discrete process control. **International Journal Of Production Economics**, [S.L.], v. 112, n. 1, p. 62-76, mar. 2008a. Elsevier BV. http://dx.doi.org/10.1016/j.ijpe.2006.08.028.

IASSINOVSKI, Serguei; ARTIBA, Abdelhakim; FAGNART, Christophe. SD Builder®: a production rules-based tool for on-line simulation, decision making and discrete process control. **Engineering Applications Of Artificial Intelligence**, [S.L.], v. 21, n. 3, p. 406-418, abr. 2008b. Elsevier BV. http://dx.doi.org/10.1016/j.engappai.2007.05.005.

JACKSON, Michael. Aspects of abstraction in software development. **Software & Systems Modeling**, [S.L.], v. 11, n. 4, p. 495-511, 11 ago. 2012. Springer Science and Business Media LLC. http://dx.doi.org/10.1007/s10270-012-0259-7.

JANIESCH, Christian; FISCHER, Marcus; WINKELMANN, Axel; NENTWICH, Valentin. Specifying autonomy in the Internet of Things: the autonomy model and notation. **Information Systems And E-Business Management**, [S.L.], v. 17, n. 1, p.

159-194, 27 nov. 2018. Springer Science and Business Media LLC. http://dx.doi.org/10.1007/s10257-018-0379-x.

JASINSKI, Ricardo Pereira. **Framework para Geração de Hardware em VHDL a Partir de Modelos em PON (Paradigma Orientado a Notificações)**. Curitiba: Relatório da Disciplina de Lógica Reconfigurável Por Hardware, 2012. 33 p.

KAISLER, Stephen H.. **Software paradigms**. Hoboken: John Wiley & Sons, Inc., 2005. 458 p.

KERSCHBAUMER, Ricardo *et al*. Paradigma Orientado a Notificações para a Síntese de Lógica Reconfigurável Paradigma Orientado a Notificações para a Síntese de Lógica Reconfigurável. In: CONGRESSO BRASILEIRO DE INTELIGêNCIA COMPUTACIONAL, 12., 2015, Curitiba. **Anais [...] .** [S. L.]: Abricom, 2015. p. 1-6.

KERSCHBAUMER, Ricardo. **Proposição do paradigma orientado a notificações no desenvolvimento de circuitos lógico-digitais reconfiguráveis**. 2018. 378 f. Tese (Doutorado) - Curso de Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial - Cpgei, Universidade Tecnológica Federal do Paraná - Utfpr, Curitiba, 2018.

KOESTLER, Arthur. **O fantasma da máquina**. Rio de Janeiro: Zahar Editores, 1969. 385 p.

KOSSOSKI, Clayton. **Proposta de Um Método de Teste para Processos de Desenvolvimento de Software usando o Paradigma Orientado a Notificações**. 2015. 270 f. Dissertação (Mestrado) - Curso de Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial - Cpgei, Universidade Tecnológica Federal do Paraná - Utfpr, Curitiba, 2015.

KOSSOSKI, Clayton; SIMÃO, Jean Marcelo; STADZISZ, Paulo Cézar. Introdução ao teste funcional de software no Paradigma Orientado a Notificações Paradigma Orientado a Notificações Mecanismo de Notificações do PON. In: CONGRESO INTERNACIONAL DE COMPUTACIÓN Y TELECOMUNICACIONES, 6., 2014, Lima. **Anais [...] .** Lima: Fondo Editorial de La Uigv, 2014. p. 136-146.

KROGSTIE, John. Integrated Goal, Data and Process Modeling: from tempora to model-generated work-places. In: INFORMATION Systems Engineering: From Data Analysis to Process Networks. [S.L]: Igi Global, 2008. p. 43-65.

LAL, Indu Bhushan; PATHAK, Vibhawendra; KUMAR, Shiv Kant. A Study of Job Satisfaction in Software Industry: myths and realities. **International Journal Of Emerging Research In Management &technology**: panorama e tendências, [S. L.], v. 4, n. 5, p. 43-49, maio 2015.

LE, Jia-Jin; HE, Feng. Automatic Web Services Composition Based on Reasoning Petri Net. In: 2008 INTERNATIONAL CONFERENCE ON ADVANCED LANGUAGE PROCESSING AND WEB INFORMATION TECHNOLOGY, 7., 2008, Dalian Liaoning. **Proceedings [...] .** [S. L.]: Ieee, 2008. p. 569-574.

LEE, Minsoo; SU, Stanley Y. W.; LAM, Herman. An Event-Trigger-Rule Model for Supporting Collaborative Knowledge Sharing Among Distributed Organizations. In: ONTHE Move to Meaningful Internet Systems 2006: OTM 2006 Workshops: OTM Confederated International Workshops and Posters, AWeSOMe, CAMS, COMINF, IS, KSinBIT, MIOS-CIAO, MONET, OnToContent, ORM, PerSys, OTM Academy Doctoral Consortium, RDDS, SWWS, and SeBGIS 2006, Montpellier, France, October 29 - November 3, 2006. Proceedings, Part I. 3. ed. Montpellier: Springer, 2006. p. 780-791. (Lecture Notes in Computer Science 4277).

LEE, Pou-Yung; CHENG, A.M.K.. HAL: a faster match algorithm. **Ieee Transactions On Knowledge And Data Engineering**, [S.L.], v. 14, n. 5, p. 1047-1058, set. 2002. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/tkde.2002.1033773.

LI, Xiaoou; MEDINA, Joselito M.; CHAPA, Sergio V.. Applying Petri Nets in Active Database Systems. **Ieee Transactions On Systems, Man And Cybernetics, Part C (Applications And Reviews)**, [S.L.], v. 37, n. 4, p. 482-493, jul. 2007. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/tsmcc.2007.897329.

LINHARES, Robson R.; PORDEUS, Leonardo F.; SIMAO, Jean M.; STADZISZ, Paulo C.. NOCA — A Notification-Oriented Computer Architecture: prototype and simulator. **Ieee Access**, [S.L.], v. 8, p. 37287-37304, 2020. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/access.2020.2975360.

LINHARES, Robson Ribeiro *et al*. Comparações entre o Paradigma Orientado a Objetos e o Paradigma Orientado a Notificações sob o contexto de um simulador de sistema telefônico. In: CONGRESO INTERNACIONAL DE COMPUTACIÓN Y TELECOMUNICACIONES, 3., 2011, Lima. **Anais [...] .** Lima: Fondo Editorial de La Uigv, 2011. p. 103-118.

LINHARES, Robson Ribeiro. **Contribuição para o desenvolvimento de uma arquitetura de computação própria ao Paradigma Orientado a Notificações**. 2015. 340 f. Tese (Doutorado) - Curso de Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial - Cpgei, Universidade Tecnológica Federal do Paraná - Utfpr, Curitiba, 2015.

LINHARES, Robson Ribeiro; SIMAO, Jean Marcelo; STADZISZ, Paulo Cezar. NOCA: A Notification-Oriented Computer Architecture. **Ieee Latin America Transactions**, [S.L.], v. 13, n. 5, p. 1593-1604, maio 2015. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/tla.2015.7112020.

LIU, Dong *et al*. Automating transition from use-cases to class model. In: CANADIAN CONFERENCE ON ELECTRICAL AND COMPUTER ENGINEERING, 2., 2003, Montreal. **Proceedings [...] .** [S. L.]: Ieee, 2003. p. 831-834.

LUCCA, Jardel; SIMÃO, Jean Marcelo. **Módulo de Interface Amigável (Wizard) sobre Meta-modelo de Controle de um Simulador de Sistemas de Manufatura Holônico e Testes Comparativos Decorrentes entre Políticas de Controle**. Curitiba: Utfpr, 2008. 28 p.

LUKICHEV, Sergey; GIURCA, Adrian; WAGNER, Gerd. An Integrated Rule Modeling Framework. In: INFORMATIK 2007: INFORMATIK TRIFFT LOGISTIK . BAND 1, 37., 2007, Bremen. **Proceedings [...] .** [S. L.]: Gesellschaft Für Informatik, 2007. p. 217-221.

LUKICHEV, Sergey; WAGNER, Gerd. Visual Rules Modeling. **Perspectives Of Systems Informatics**, [S.L.], p. 467-473, 2006. Springer Berlin Heidelberg. http://dx.doi.org/10.1007/978-3-540-70881-0_42.

MALINOVA, Monika. **A Language for Designing Process Maps**. 2016. 266 f. Tese (Doutorado) - Curso de Wirtschaftsinformatik U. Operations Mgmt, Wu Vienna University Of Economics And Business, Vienna, 2016.

MARCONI, Maria de Andrade; LAKATOS, Eva Maria. **Fundamentos de metodologia científica**. 5. ed. São Paulo: Atlas, 2003. 310 p.

MARTÍNEZ-FERNÁNDEZ, José L.; MARTÍNEZ, Paloma; GONZÁLEZ-CRISTÓBAL, José C.. Towards an Improvement of Software Development Processes through Standard Business Rules. In: RULE Interchange and Applications: International Symposium, RuleML 2009, Las Vegas, Nevada, USA, November 5-7, 2009. Proceedings. 3. ed. Las Vegas: Springer, 2009. p. 159-166. (Lecture Notes in Computer Science 5858).

MEDVIDOVIC, Nenad; TAYLOR, Richard N.. Software architecture: foundations, theory, and practice. In: 32ND ACM/IEEE INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 32., 2010, Cape Town. **Proceedings [...] .** [S. L.]: Acm Press, 2010. p. 471-472.

MELO, Luiz Carlos Viana; SIMÃO, Jean Marcelo; FABRO, João A.. Adaptação do Paradigma Orientado a Notificações para desenvolvimento de sistemas fuzzy. In: TERCEIRO CONGRESSO BRASILEIRO DE SISTEMAS FUZZY, 3., 2014, João Pessoa. **Anais [...] .** João Pessoal: Cbsf, 2014. p. 1-12.

MENDONÇA, Igor Thiago Marques; SIMÃO, Jean Marcelo; WIECHETECK, Luciana Vilas Boas; STADZISZ, Paulo Cézar. Método para Desenvolvimento de Sistemas Orientados a Regras utilizando o Paradigma Orientado a Notif. In: CONGRESSO BRASILEIRO DE INTELIGêNCIA COMPUTACIONAL, 12., 2015, Curitiba. **Anais [...] .** Curitiba: Abricom, 2015. p. 1-6. CD-ROM.

MIRANKER, Daniel P. *et al*. On the Performance of Lazy Matching in Production Systems. In: 8TH NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, 8., 1990, Boston. **Proceedings [...] .** Boston: Association For The Advancement Of Artificial Intelligence, 1990. p. 685-692.

MIRANKER, Daniel P.. TREAT: a better match algorithm for ai production systems. In: SIXTH NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, 6., 1987, Seatle. **Proceedings [...] .** Seatle: Association For The Advancement Of Artificial Intelligence, 1987. p. 42-47.

MOGHADAM, Mahshid Helali; MOZAYANI, Nasser. A street lighting control system based on holonic structures and traffic system. In: INTERNATIONAL CONFERENCE

ON COMPUTER RESEARCH AND DEVELOPMENT, 3., 2011, Shanghai. **Proceedings [...] .** [S. L.]: Ieee, 2011. p. 92-96.

MOODY, D.. The “Physics” of Notations: toward a scientific basis for constructing visual notations in software engineering. **Ieee Transactions On Software Engineering**, [S.L.], v. 35, n. 6, p. 756-779, nov. 2009. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/tse.2009.67.

MOORE, Kevin; WERMELINGER, Michel. The challenge of software complexity. In: SPRINGER. **Proceedings of the European Conference on Complex Systems 2012**. Dordrecht: Springer, 2013. p. 179-187.

MORE, Priyanka; PHALNIKAR, Rashmi. Generating UML Diagrams from Natural Language Specifications. **International Journal Of Applied Information Systems**, [S.L.], v. 1, n. 8, p. 19-23, 10 abr. 2012. Foundation of Computer Science. http://dx.doi.org/10.5120/ijais12-450222.

MOULIN, Mark. A Formal Verification Analysis of a Bayesian Inference-Based Sensors and Actuators Control System. In: IECON 2006 - 32ND ANNUAL CONFERENCE ON IEEE INDUSTRIAL ELECTRONICS, 32., 2006, Paris. **Proceedings [...] .** [S. L.]: Ieee, 2006. p. 2939-2943.

NALEPA, Grzegorz J.; BOBEK, Szymon; LIGĘZA, Antoni; KACZOR, Krzysztof. Algorithms for Rule Inference in Modularized Rule Bases. **Rule-Based Reasoning, Programming, And Applications**, [S.L.], p. 305-312, 2011. Springer Berlin Heidelberg. http://dx.doi.org/10.1007/978-3-642-22546-8_24.

NALEPA, Grzegorz; LIGĘZA, Antoni. The HeKatE methodology. Hybrid engineering of intelligent systems. **International Journal Of Applied Mathematics And Computer Science**, [S.L.], v. 20, n. 1, p. 35-53, 1 mar. 2010. Walter de Gruyter GmbH. http://dx.doi.org/10.2478/v10006-010-0003-9.

NEGERI, Ebisa; BAKEN, Nico; POPOV, Marjan. Holonic Architecture of the Smart Grid. **Smart Grid And Renewable Energy**, [S.L.], v. 04, n. 02, p. 202-212, 2013. Scientific Research Publishing, Inc.. http://dx.doi.org/10.4236/sgre.2013.42025.

NEGRINI, Fabio. **Tecnologia NOPL Erlang-Elixir – paradigma orientado a notificações via uma abordagem orientada a microatores assíncronos**. 2019. 298 f. Dissertação (Mestrado) - Curso de Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial - Cpgei, Universidade Tecnológica Federal do Paraná - Utfpr, Curitiba, 2019.

NOVAES, Paulo José Dantas. **Método e linguagem para modelagem gráfica de requisitos de software e sistemas**. 2019. 347 f. Dissertação (Mestrado) - Curso de Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial - Cpgei, Universidade Tecnológica Federal do Paraná - Utfpr, Curitiba, 2019.

OBJECT MANAGEMENT GROUP. **OMG MOF 2.5.1**: Meta Object Facility. 2.5.1 ed. [S.L]: Omg, 2016. 76 p.

OBJECT MANAGEMENT GROUP. **OMG OCL 2.4**: Object Constraint Language. 2.4 ed. [S.L]: Omg, 2014. 262 p.

OBJECT MANAGEMENT GROUP. **OMG PRR 1.0**: Production Rule Representation (PRR). 1 ed. [S.L]: Omg, 2009. 62 p.

OBJECT MANAGEMENT GROUP. **OMG UML 2.5**: Unified Modeling Language (UML). 2.5 ed. [S.L]: Omg, 2013. 786 p. Disponível em: http://www.omg.org/spec/UML/2.5. Acesso em: 05 jan. 2021.

OKTABA, Hanna; GONZÁLEZ, Guadalupe Ibargüengoitia. Software Process Modeled with Objects: static view. **Computación y Sistemas**, [S. L.], v. 1, n. 4, p. 228-238, mar. 1998.

OLIVEIRA, Rodrigo Nunes. **Assistência à autonomia domiciliar empregando Paradigma Orientado a Notificações**. 2019. 95 f. Dissertação (Mestrado) - Curso de Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial - Cpgei, Universidade Tecnológica Federal do Paraná - Utfpr, Curitiba, 2019.

OLMEDO-AGUIRRE, José Oscar; LAROSA, Mónica Rivera de; MORALES-LUNA, Guillermo. ECA-Rule Visual Programming for Ubiquitous and Nomadic Computing. In: MICAI 2008: Advances in Artificial Intelligence: 7th Mexican International Conference on Artificial Intelligence, Atizapán de Zaragoza, Mexico, October 27-31, 2008 Proceedings. Zaragoza: Springer, 2008. p. 925-935. (Lecture Notes in Computer Science 5317).

PETERS, Eduardo. **Coprocessador para aceleração de aplicações desenvolvidas utilizando Paradigma Orientado a Notificações**. 2012. 96 f. Dissertação (Mestrado) - Curso de Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial - Cpgei, Universidade Tecnológica Federal do Paraná - Utfpr, Curitiba, 2012.

PIMENTEL, Andrey Ricardo. **Uma abordagem para projeto de software orientado a objetos baseado na teoria de projeto axiomático**. 2007. 189 f. Tese (Doutorado) - Curso de Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial - Cpgei, Universidade Tecnológica Federal do Paraná - Utfpr, Curitiba, 2007.

PORDEUS, Leonardo Faix. **Simulação de uma arquitetura de própria ao paradigma orientado a notificações**. 2017. 364 f. Dissertação (Mestrado) - Curso de Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial - Cpgei, Universidade Tecnológica Federal do Paraná - Utfpr, Curitiba, 2017.

PRANEVICIUS, Henrikas; BUDNIKAS, Germanas. Representing and checking consistency and dynamic constraints of business rules. In: 20TH EURO MINI CONFERENCE CONTINUOUS OPTIMIZATION AND KNOWLEDGE-BASED TECHNOLOGIES, 20., 2008, Neringa. **Proceedings [...] .** Neringa: Europt-2008, 2008. p. 468-473.

PRAT, Nicolas; AKOKA, Jacky; COMYN-WATTIAU, Isabelle. An MDA approach to knowledge engineering. **Expert Systems With Applications**, [S.L.], v. 39, n. 12, p. 10420-10437, set. 2012. Elsevier BV. http://dx.doi.org/10.1016/j.eswa.2012.02.010.

PRAT, Nicolas; AKOKA, Jacky; COMYN-WATTIAU, Isabelle. Mapping CommonKADS Knowledge Models into PRR. In: SEKE 2011 : 23RD

INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING KNOWLEDGE ENGINEERING, 23., 2011, Miami Beach. **Proceedings [...] .** [S. L.]: Eden Roc Renaissance, 2011. p. 143-148.

PRAT, Nicolas; COMYN-WATTIAU, Isabelle; AKOKA, Jacky. Combining objects with rules to represent aggregation knowledge in data warehouse and OLAP systems. **Data & Knowledge Engineering**, [S.L.], v. 70, n. 8, p. 732-752, ago. 2011. Elsevier BV. http://dx.doi.org/10.1016/j.datak.2011.03.004.

PRAT, Nicolas; WATTIAU, Isabelle; AKOKA, Jacky. Representation of aggregation knowledge in OLAP systems. In: 18TH EUROPEAN CONFERENCE ON INFORMATION SYSTEMS, ECIS 2010, 18., 2010, Pretoria. **Proceedings [...] .** [S. L.]: Association For Information Systems, 2010. p. 1-12.

PRESSMAN, Roger S.. **Engenharia de software**: uma abordagem profissional. 7. ed. Porto Alegre: McGraw Hill Brasil, 2011. 780 p.

REINHARTZ-BERGER, Iris; STURM, Arnon. Enhancing UML Models: a domain analysis approach. In: HALPIN, Terry. **Selected Readings on Database Technologies and Applications**. [S.L]: Igi Global, 2009. p. 369-394.

RIBARIĆ, Marko *et al*. Model-Driven Engineering of Rules for Web Services. In: GENERATIVE and Transformational Techniques in Software Engineering II: International Summer School, GTTSE 2007, Braga, Portugal, July 2-7, 2007. Revised Papers. [S.L]: Springer, 2007. p. 377-395. (Lecture Notes in Computer Science 5235).

RONSZCKA, Adriano Francisco *et al*. Notification-Oriented and Rete Network Inference: a comparative study. In: IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN, AND CYBERNETICS, 2015., 2015, Hong Kong. **Proceedings [...] .** Hong Kong: Ieee, 2015. p. 807-814.

RONSZCKA, Adriano Francisco. **Contribuição para a concepção de aplicações no Paradigma Orientado a Notificações (PON) sob o viés de padrões**. 2012. 226 f. Dissertação (Mestrado) - Curso de Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial - Cpgei, Universidade Tecnológica Federal do Paraná - Utfpr, Curitiba, 2012.

RONSZCKA, Adriano Francisco. **Método para a criação de linguagens de programação e compiladores para o paradigma orientado a notificações em plataformas distintas**. 2019. 376 f. Tese (Doutorado) - Curso de Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial - Cpgei, Universidade Tecnológica Federal do Paraná - Utfpr, Curitiba, 2019.

ROSEMANN, Michael; SEDERA, Wasana; SEDERA, Darshana. Testing a Framework for the Quality of Process Models - A Case Study. In: 5TH PACIFIC ASIA CONFERENCE ON INFORMATION SYSTEMS (PACIS 2001), 5., 2001, Seoul. **Proceedings [...] .** [S. L.]: Association For Information Systems, 2001. p. 978-991.

SANTOS, Leonardo Araújo. **Linguagem e compilador para o paradigma orientado a notificações**: avanços para facilitar a codificação e sua validação em

uma aplicação de controle de futebol de robô. 2017. 274 f. Dissertação (Mestrado) - Curso de Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial - Cpgei, Universidade Tecnológica Federal do Paraná - Utfpr, Curitiba, 2017.

SCACCHI, Walt. Process Models in Software Engineering. In: MARCINIAK, J.J.. **Encyclopedia of Software Engineering**. 2. ed. New York: John Wiley & Sons, Inc., 2002. p. 1-24.

SCHUETTE, Reinhard; ROTTHOWE, Thomas. The Guidelines of Modeling: an approach to enhance the quality in information models. In: CONCEPTUAL Modeling – ER '98: 17th International Conference on Conceptual Modeling, Singapore, November 16-19, 1998. Proceeding. Singapore: Springer, 1998. p. 240-254. (Lecture Notes in Computer Science 1507).

SCHÜTZ, Fernando. **NeuroPON**: uma abordagem para o desenvolvimento de redes neurais artificiais utilizando o paradigma orientado a notificações. 2019. 269 f. Tese (Doutorado) - Curso de Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial - Cpgei, Universidade Tecnológica Federal do Paraná - Utfpr, Curitiba, 2019.

SCHÜTZ, Fernando; FABRO, João A.; RONSZCKA, Adriano F.; STADZISZ, Paulo C.; SIMÃO, Jean M.. Proposal of a declarative and parallelizable artificial neural network using the notification-oriented paradigm. **Neural Computing And Applications**, [S.L.], v. 30, n. 6, p. 1715-1731, 4 jun. 2018. Springer Science and Business Media LLC. http://dx.doi.org/10.1007/s00521-018-3517-y.

SEGULJA, Cedomir; ABDELRAHMAN, Tarek S.. What is the cost of weak determinism? In: 23RD INTERNATIONAL CONFERENCE ON PARALLEL ARCHITECTURES AND COMPILATION, 23., 2014, Edmonton. **Proceedings [...] .** New York: Acm Press, 2014. p. 99-112.

SELIC, B.. The pragmatics of model-driven development. **Ieee Software**, [S.L.], v. 20, n. 5, p. 19-25, set. 2003. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/ms.2003.1231146.

SHAW, Mary. Writing good software engineering research papers. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 25., 2003, Portland. **Proceedings [...] .** [S. L.]: Ieee, 2003. p. 726-736.

SILVA, Bruno Berstel-Da; CHNITI, Amina. Detection of Inconsistencies in Rules Due to Changes in Ontologies: let?s get formal. In: WEB Reasoning and Rule Systems: 7th International Conference, RR 2013, Mannheim, Germany, July 27-29, 2013. Proceedings. Mannheim: Springer, 2013. p. 198-203. (Lecture Notes in Computer Science 7994).

SILVA, Edna Lúcia; MENEZES, Estera Muszkat. **Metodologia da Pesquisa e Elaboração de Dissertação**. 4. ed. Florianópolis: Ufsc, 2005. 138 p.

SILVA, Juarez Nuno da; PITASSI, Claudio. Práticas Logísticas nas Pequenas e Médias Empresas Brasileiras. **Revista Adm.Made**, Rio de Janeiro, v. 17, n. 2, p. 29-48, ago. 2013. Disponível

em: http://revistaadmmade.estacio.br/index.php/admmade/article/view/645/387. Acesso em: 04 jan. 2021.

SIMAO, J.M.; STADZISZ, P.C.. Inference Based on Notifications: a holonic metamodel applied to control issues. **Ieee Transactions On Systems, Man, And Cybernetics - Part A**: Systems and Humans, [S.L.], v. 39, n. 1, p. 238-250, jan. 2009. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/tsmca.2008.2006371.

SIMAO, J.M.; TACLA, C.A.; STADZISZ, P.C.. Holonic Control Metamodel. **Ieee Transactions On Systems, Man, And Cybernetics - Part A**: Systems and Humans, [S.L.], v. 39, n. 5, p. 1126-1139, set. 2009. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/tsmca.2009.2022060.

SIMÃO, Jean Marcelo *et al*. Evaluation of the Notification Oriented Paradigm Applied to Sentient Computing. In: IEEE 17TH INTERNATIONAL SYMPOSIUM ON OBJECT/COMPONENT/SERVICE-ORIENTED REAL-TIME DISTRIBUTED COMPUTING, 17., 2014, Reno. **Proceedings [...] .** Reno: Ieee, 2014. p. 253-260.

SIMÃO, Jean Marcelo. **A contribution to the development of a HMS simulation tool and proposition of a meta-model for holonic control**. 2005. 192 f. Tese (Doutorado) - Curso de Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial - Cpgei, Universidade Tecnológica Federal do Paraná - Utfpr, Curitiba, 2005.

SIMÃO, Jean Marcelo. **Proposta de uma Arquitetura de Controle para Sistemas Flexíveis de Manufatura Baseada em Regras e Agentes**. 2001. 201 f. Dissertação (Mestrado) - Curso de Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial - Cpgei, Universidade Tecnológica Federal do Paraná - Utfpr, Curitiba, 2001.

SIMÃO, Jean Marcelo; STADZISZ, Paulo Cezar. **Paradigma Orientado a Notificações (PON) – Uma Técnica de Composição e Execução de Software Orientado a Notificações**. . BR n. PI 0805518-1. Depósito: 26 nov. 2008. Concessão: 27 dez. 2011.

SIMÃO, Jean Marcelo; STADZISZ, Paulo Cezar; WIECHETECK, Luciana Vilas Boas. **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)**. . BR n. 10 2012 026430-7. Depósito: 16 out. 2012. Concessão: 19 maio 2015.

SIMÃO, Jean Marcelo; TACLA, César Augusto; BANASZEWSKI, Roni Fábio; STADZISZ, Paulo Cézar. **Mecanismo de Inferência Otimizado do Paradigma Orientado a Notificações (PON) e Mecanismos de Resolução de Conflitos para Ambientes Monoprocessados e Multiprocessados Aplicados ao PON**. . BR n. PI 1003736-5. Depósito: 25 nov. 2010. Concessão: 14 fev. 2012.

STADZISZ, Paulo Cézar. **Disciplina de Engenharia de Software**: notas de aula. Curitiba: Universidade Tecnológica Federal do Paraná, 2014. Color.

STALLINGS, William. **Arquitetura e Organização de Computadores**. 8. ed. São Paulo: Prentice Hall, 2010. 640 p.

SUH, Nam P.. **The Principles of Design**. New York: Oxford University Press, 1990. 401 p.

TANENBAUM, Andrew S.. **Modern operating systems**. 3. ed. Upper Saddle River: Pearson Education, 2009. 1072 p.

TRIBUNAL DE CONTAS DA UNIÃO (TCU) (Brasília). **Levantamento de Pessoal de TI**. Brasília: Tcu, Secretaria de Fiscalização de Tecnologia da Informação (Sefti), 2015.

VALCKENAERS, Paul; VAN BRUSSEL, Hendrik; HOLVOET, Tom. Fundamentals of Holonic Systems and Their Implications for Self-Adaptive and Self-Organizing Systems. In: IEEE INTERNATIONAL CONFERENCE ON SELF-ADAPTIVE AND SELF-ORGANIZING SYSTEMS WORKSHOPS, 2., 2008, Venice. **Proceedings [...] .** [S. L.]: Ieee, 2008. p. 168-173.

VALENÇA, Glauber Z. *et al*. Framework PON, Avanços e Comparações. In: SIMPÓSIO DE COMPUTAÇÃO APLICADA, 3., 2011, Passo Fundo. **Anais [...] .** Passo Fundo: Sbc, 2011. p. 1-15.

VALENÇA, Glauber Zárate. **Contribuição para a Materialização do Paradigma Orientado a Notificações (PON) via Framework e Wizard**. 2012. 226 f. Dissertação (Mestrado) - Curso de Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial - Cpgei, Universidade Tecnológica Federal do Paraná - Utfpr, Curitiba, 2012.

VAN BRUSSEL, Hendrik; WYNS, Jo; VALCKENAERS, Paul; BONGAERTS, Luc; PEETERS, Patrick. Reference architecture for holonic manufacturing systems: prosa. **Computers In Industry**, [S.L.], v. 37, n. 3, p. 255-274, nov. 1998. Elsevier BV. http://dx.doi.org/10.1016/s0166-3615(98)00102-x.

VAN ROY, Peter. Programming Paradigms for Dummies: what every programmer should know. In: ASSAYAG, Gérard; GERZSO, Andrew. **New computational paradigms for computer music**. [S.L]: Delatour France, 2009. p. 9-47.

VON BARTALANFFY, Ludwig. **General System Theory**: foundations, development, applications. [S.L]: George Braziller Inc., 1969. 296 p.

WALLACE, Andrew. Holons and a Holonic Society. In: WALLACE, Andrew. **Journal of European Technocracy**. [S.L]: Lulu.Com, 2008. p. 89-100.

WANG, Jinheng; WANG, Pu; LI, Yafen. Research and Implementation of the Code Generation System Based on Production Rules. In: 4TH INTERNATIONAL CONFERENCE ON INTELLIGENT HUMAN-MACHINE SYSTEMS AND CYBERNETICS, 4., 2012, Nanchang. **Proceedings [...] .** [S. L.]: Ieee, 2012. p. 224-227.

WEISZFLOG, Walter. **Michaelis Moderno Dicionario Da Lingua Portuguesa**. [S.L]: Melhoramentos, 1998. 2260 p.

WIECHETECK, Luciana Vilas Boas. **Método para projeto de software usando o Paradigma Orientado a Notificações - PON**. 2011. 198 f. Dissertação (Mestrado) - Curso de Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial - Cpgei, Universidade Tecnológica Federal do Paraná - Utfpr, Curitiba, 2011.

WIECHETECK, Luciana Vilas Boas; STADZISZ, Paulo Cézar; SIMÃO, Jean Marcelo. Um Perfil UML para o Paradigma Orientado a Notificações (PON). In: CONGRESSO INTERNACIONAL DE COMPUTACION Y TELECOMUNICACIONES, 3., 2011, Lima. **Anais [...] .** Lima: Universidad Inca Garcilaso de La Vega, 2011. p. 1-16.

WINTERS, G.. Use Case Terminology. **Ieee Software**, [S.L.], v. 22, n. 2, p. 67-67, mar. 2005. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/ms.2005.49.

WITT, Fernando A. de *et al*. Comparação entre o paradigma orientado a objetos (POO) e o Paradigma Orientado a Notificações (PON) em um controle discreto em lógica reconfigurável. In: XVI SEMINÁRIO DE INICIAÇÃO CIENTÍFICA E TECNOLÓGICA DA UTFPR-SICITE, 16., 2011, Curitiba. **Anais [...] .** Curitiba: Utfpr, 2011. p. 1-5.

XAVIER, Robson Duarte. **Paradigmas de desenvolvimento de software**: comparação entre abordagens orientada a eventos e orientada a notificações. 2014. 240 f. Dissertação (Mestrado) - Curso de Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial - Cpgei, Universidade Tecnológica Federal do Paraná - Utfpr, Curitiba, 2014.

YE, Yao *et al*. A Holonic Model in Wireless Sensor Networks. In: INTERNATIONAL CONFERENCE ON INTELLIGENT INFORMATION HIDING AND MULTIMEDIA SIGNAL PROCESSING, 4., 2008, Harbin. **Proceedings [...] .** [S. L.]: Ieee, 2008. p. 491-495.

ZACHARIAS, Valentin. Development and Verification of Rule Based Systems - A Survey of Developers. In: RULE Representation, Interchange and Reasoning on the Web. [S.L]: Springer, 2008. p. 6-16. (Lecture Notes in Computer Science 5321).

ZACHARIAS, Valentin. The Agile Development of Rule Bases. In: WOJTKOWSKI, Wita *et al*. **Information Systems Development**: challenges in practice, theory, and education. [S.L]: Springer, 2009. p. 93-104.

ZAMANI, M. A. *et al*. AAPNES: a petri net expert system realization of adaptive autonomy in smart grid. In: 5TH INTERNATIONAL SYMPOSIUM ON TELECOMMUNICATIONS, 5., 2010, Tehran. **Proceedings [...] .** [S. L.]: Ieee, 2010. p. 1-7.

ZHANG, Guoquan; LI, Wenli. Flexible Holonic Organization Modeling and Cultural Evolution. In: 4TH INTERNATIONAL CONFERENCE ON WIRELESS COMMUNICATIONS, NETWORKING AND MOBILE COMPUTING, 4., 2008, Dalian. **Proceedings [...] .** [S. L.]: Ieee, 2008. v. 17, p. 1-5.

ZHANG, Linda L.; XU, Qianli; HELO, Petri. A knowledge?based system for process family planning. **Journal Of Manufacturing Technology Management**, [S.L.], v. 24, n. 2, p. 174-196, fev. 2013. Emerald. http://dx.doi.org/10.1108/17410381311292296.

ZHANG, Linda; XU, Qianli; SHOU, Yongyi. Planning process families with a knowledge-based system. In: IEEE INTERNATIONAL CONFERENCE ON INDUSTRIAL ENGINEERING AND ENGINEERING MANAGEMENT, 18., 2011, Singapore. **Proceedings [...] .** [S. L.]: Ieee, 2011. p. 1815-1820.

ZHI-XUE, Wang *et al*. ECA rule modeling language based on UML. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER SCIENCE AND AUTOMATION ENGINEERING (CSAE), 2., 2012, Zhangjiajie. **Proceedings [...] .** [S. L.]: Ieee, 2012. p. 623-628.

ZHOU, Guo-Xiang; GAO, De-Ping. ECA Rule and Colored Petri Nets Based Workflow Modeling Research. In: 2010 INTERNATIONAL CONFERENCE ON MANAGEMENT AND SERVICE SCIENCE, 2., 2010, Wuhan. **Proceedings [...] .** [S. L.]: Ieee, 2010. v. 18, p. 1-4.