UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA

RODRIGO SOUZA CAVALCANTI DE OLIVEIRA

ABORDAGEM PARA ESPECIFICAÇÃO DAS OCORRÊNCIAS DE CARACTERÍSTICAS TRANSVERSAIS EM SOFTWARE EMBARCADO

DISSERTAÇÃO

CURITIBA
2020

RODRIGO SOUZA CAVALCANTI DE OLIVEIRA

ABORDAGEM PARA ESPECIFICAÇÃO DAS OCORRÊNCIAS DE CARACTERÍSTICAS TRANSVERSAIS EM SOFTWARE EMBARCADO

An Approach for the Specification of Crosscutting Concerns Occurrence in Embedded Software

Dissertação apresentada ao Programa de Pós-Graduação em Computação Aplicada da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do título de "Mestre em Computação Aplicada" – Área de Concentração: Engenharia de Sistemas Computacionais.

Orientador(a): Prof(a). Dr(a). Marco Aurélio Wehrmeister
Coorientador(a): Prof(a). Dr(a). Douglas Paulo Bertrand Renaux

CURITIBA

2020

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Curitiba

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

RODRIGO SOUZA CAVALCANTI DE OLIVEIRA

ABORDAGEM PARA ESPECIFICAÇÃO DAS OCORRÊNCIAS DE CARACTERÍSTICAS TRANSVERSAIS EM SOFTWARE EMBARCADO

Trabalho de pesquisa de mestrado apresentado como requisito para obtenção do título de Mestre Em Computação Aplicada da Universidade Tecnológica Federal do Paraná (UTFPR). Área de concentração: Engenharia De Sistemas Computacionais.

Data de aprovação: 18 de Dezembro de 2020

Prof Marco Aurelio Wehrmeister, Doutorado - Universidade Tecnológica Federal do Paraná

Prof Adolfo Gustavo Serra Seca Neto, Doutorado - Universidade Tecnológica Federal do Paraná

Prof Douglas Paulo Bertrand Renaux, Doutorado - Universidade Tecnológica Federal do Paraná

Prof Edison Pignaton De Freitas, Doutorado - Universidade Federal do Rio Grande do Sul (Ufrgs)

Prof Jean Marcelo Simao, Doutorado - Universidade Tecnológica Federal do Paraná

Documento gerado pelo Sistema Acadêmico da UTFPR a partir dos dados da Ata de Defesa em 24/03/2021.

Dedico este trabalho à minha amada esposa
Carina e à minha amada filha Martina, pela
paciência, apoio e compreensão, nos momentos
em que estive ausente.

AGRADECIMENTOS

À Deus por tudo o que Ele tem feito em minha vida. Este trabalho não poderia ser terminado sem a inspiração do Espírito Santo.

Ao Prof. Dr. Marco Aurélio Wehrmeister pela orientação e ao Prof. Dr. Douglas Paulo Bertrand Renaux pela coorientação. Os seus conselhos e ensinamentos foram essenciais para eu chegar até aqui.

Aos membros da banca examinadora: Prof. Dr. Adolfo Gustavo Serra Seca Neto, Prof. Dr. Jean Marcelo Simão e Prof. Dr. Edson Pignaton de Freitas por avaliar meu trabalho e por suas valiosas críticas construtivas. Os senhores contribuíram de forma crucial para o enriquecimento desta dissertação.

À UTFPR, pelo suporte a todas as atividades acadêmicas realizadas e sobretudo pelo apoio na conclusão do Mestrado.

Aos meus familiares, especialmente meus pais, Demóstenes e Rosângela que sempre me apoiaram. A minha esposa Carina, que esteve sempre ao meu lado, seu apoio foi fundamental para eu chegar até aqui. Aos meus sogros, Erwin e Karin, por me incentivarem, e aos meus tios, Henio e Sandra, por vibrarem com o meu humilde exercício na ciência.

Por fim, a todos aqueles que contribuíram direta ou indiretamente para este trabalho, o meu muito obrigado! Que Deus os abençoe abundantemente em todas as áreas de suas vidas!

"O temor do Senhor é o princípio da ciência; os loucos desprezam a sabedoria e a instrução."

Provérbios 1:7

RESUMO

OLIVEIRA, Rodrigo Souza Cavalcanti. **Abordagem para Especificação das Ocorrências de Características Transversais em Software Embarcado**. 2020. 140 f. Dissertação (Mestrado em Computação Aplicada) – Universidade Tecnológica Federal do Paraná. Curitiba, 2020.

A engenharia de software embarcado vem evoluindo rapidamente ao longo dos últimos anos, em especial para lidar com o aumento da complexidade associada a grande quantidade de requisitos funcionais e, características transversais associadas aos requisitos não-funcionais. O processo de identificar e especificar a seleção de pontos onde as características transversais devem ser tratadas vem sendo amplamente discutido desde meados dos anos 90, especialmente no contexto do *Aspect-Oriented Software Development* (AOSD). No entanto, a compreensão de tais especificações é muitas vezes difícil, pois a sintaxe e a semântica das linguagens utilizadas, principalmente as linguagens visuais, não permitem intuir facilmente o significado dos símbolos usados. Tal situação produz artefatos difíceis de compreender e manter. Portanto, o processo de localizar, identificar, e especificar os pontos onde as características transversais devem ser tratadas é chave no projeto de software embarcado, pois pode levar a problemas no desenvolvimento e, principalmente, na integração dos componentes do software do sistema. A presente dissertação de mestrado propõe uma técnica de modelagem gráfica de especificação de seleção de Pontos de Ocorrências de Características Transversais (POCT) para sistemas embarcados sob o nome de JSD (*Join Point Specification Diagram*). Foi realizada uma avaliação empírica com o objetivo de quantificar a compreensão da especificação de seleção de POCT de forma indireta através da avaliação dos efeitos cognitivos e das propriedades de percepção das notações. Foram avaliadas cinco notações, duas gráficas (JPDD e Theme/UML) e três textuais (AspectJ, Tracematch e AspectOCL), que foram usadas para especificar implícita e explicitamente a seleção de POCT (para 16 pontos distintos) em três projetos de software embarcados diferentes que representam aplicações reais no contexto de sistemas de automação. Este trabalho também apresenta a mesma avaliação para a abordagem proposta, a JSD, usando os mesmos projetos de software embarcado. O propósito da avaliação é verificar a capacidade de compreensão das especificações usando a técnica proposta através da avaliação dos efeitos cognitivos e das propriedades de percepção. Para tal, propõe-se um modelo de qualidade baseado no framework conceitual "Physics of Notation" (PoN). Foram usadas nove métricas para quantificar as propriedades perceptuais na especificação da seleção de POCT, sendo que algumas métricas foram criadas no contexto deste trabalho e outras representam conceitos dentro dos princípios do PoN. Os resultados dos experimentos mostram como as características de cada uma das notações impacta na compreensão das especificações criadas. Foram encontradas evidências empíricas de que a especificação gráfica das seleções dos POCT usando a JSD tem alta discriminação sobre a eficácia cognitiva das representações visuais (conforme o framework PoN) e, portanto, podemos concluir que o uso da JSD para especificar a seleção de POCT em sistemas embarcados é mais eficaz quanto a compreensão e interpretação de suas propriedades perceptivas (sintaxe) do que a JPDD e as outras técnicas avaliadas.

Palavras-chave: Software Embarcado. Características Transversais. Notação Visual. Propriedades Cognitivas. Métricas de Software.

ABSTRACT

OLIVEIRA, Rodrigo Souza Cavalcanti. **An Approach for the Specification of Crosscutting Concerns Occurrence in Embedded Software**. 2020. 140 p. Dissertation (Master's Degree in Applied Computing) – Universidade Tecnológica Federal do Paraná. Curitiba, 2020.

Embedded software engineering has evolved rapidly over the past few years, especially to deal with the increased complexity associated with a large number of functional requirements and crosscutting concerns associated with non-functional requirements. The process of identifying and specifying the selection of points where crosscutting concerns should be addressed has been widely discussed since the mid-1990s, especially in the context of Aspect-Oriented Software Development (AOSD). However, understanding these specifications is often difficult, since the syntax and semantics of the languages used, especially visual languages, do not allow you to easily deduce the meaning of the symbols used. Such a situation produces artifacts that are difficult to understand and maintain. Therefore, the process of locating, identifying, and specifying the points where the crosscutting concerns must be handled is a key in the embedded software design, as it can lead to problems in the development and, mainly, in the integration of the system's software components. This master's thesis proposes a graphical modeling technique for specifying the selection of Crosscutting Concerns Occurrence Points (CCOP) for embedded systems under the name of JSD (Join Point Specification Diagram). An empirical evaluation was carried out to assess the understanding of the CCOP selection specification indirectly through the evaluation of the cognitive effects and the perception properties of the notations. Five notations were evaluated, two graphical (JPDD and Theme/UML) and three textual (AspectJ, Tracematch, and AspectOCL), which were used to specify implicitly and explicitly the selection of CCOP (for 16 different points) in three different embedded software projects that represent real applications in the context of automation systems. This work also presents the same evaluation for the proposed approach, JSD, using the same embedded software projects. The purpose of the assessment is to verify the ability to understand the specifications using the proposed technique through the assessment of cognitive effects and perception properties. A quality model based on the conceptual framework "Physics of Notation" (PoN) is proposed. Nine metrics were used to quantify the perceptual properties in the specification of the CCOP selection, with some metrics created in the context of this work and others representing concepts within the principles of PoN. The results of the experiments show how the characteristics of each of the notations impact the understanding of the specifications created. Empirical evidence was found that the graphic specification of CCOP selections using JSD has high discrimination on the cognitive effectiveness of visual representations (according to the PoN framework) and, hence, it seems that the use of JSD to specify the selection of CCOP in embedded systems is more effective in understanding and interpreting its perceptual properties (syntax).

Keywords: Embedded Software. Crosscutting Concerns. Visual/Graphical notation. Perceptual Properties. Software Metrics.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

AOSD	Aspect-Oriented Software Development
POCT	Pontos de Ocorrências de Características Transversais
JSD	Join Point Specification Diagram
JPDD	Join Point Designation Diagram
UML	Unified Modeling Language
OCL	Object Constraint Language
PoN	Physics of Notation
CCOP	Crosscuttting Concerns Ocurrence Points
OMG	Object Management Group
AMoDE-RT	Aspect-Oriented Model-Driven Engineering Approach for Distributed Embedded Real-Time Systems
DERAF	Distributed Embedded Real-Time Aspects Framework
ACOD	Aspects Crosscutting Overview Diagram
OO	Orientação a Objetos
MARTE	Modeling and Analysis of Real-Time and Embedded systems
GenERTiCA	Generation of Embedded Real-Time Code based on Aspects
AORE	Aspect-Oriented Requirements Engineering
SLR	Sistematic Literature Mapping
GORE	Goal-Oriented Requirements Engineering
FCA	Formal Concept Analysis
CBFA	Clustering-Based Fan-in Analysis
MS4PoN	Metrics Suite for PoN
NE	Number of Elements
NLE	Number of Logical Expressions
NVD	Number of Visual Distance
NSIE	Number of Semantically Immediate Elements
NSOE	Number of Semantically Opaque Elements
NSPE	Number of Semantically Perverse Elements
LVE	Level of Expressiveness
NS	Number of Synographs
NH	Number of Homographs

SUMÁRIO

1 INTRODUÇÃO

O desenvolvimento de software embarcado geralmente inclui características funcionais e não-funcionais que estão vinculadas a um único serviço ou função. Nos paradigmas tradicionais de decomposição de software, e.g., paradigma de orientação a objetos e/ou procedimental, tais responsabilidades são disseminadas por várias unidades de modularização, tanto nas especificações usando modelos visuais quanto na codificação do software embarcado. As características transversais são características que traspassam outras características do desenvolvimento de software, ferindo a responsabilidade de um método ou função no desenvolvimento de software orientado a objetos. Posto isto, as características transversais afetam os requisitos funcionais e os não-funcionais levando a um entrelaçamento e espalhamento do código por vários módulos do software, afetando negativamente o princípio de responsabilidade única e também a construção de um código desacoplado e coeso.

A literatura apresenta muitas abordagens com o objetivo de resolver esses problemas de modularização e também da quebra de proposições chave da engenharia de software, e.g., a premissa de responsabilidade única [Filman *et al.* 2004, Khan *et al.* 2019]. Uma das técnicas mais populares é o Desenvolvimento de Software Orientado a Aspectos (em inglês, *Aspect-oriented Software Development* - AOSD) [Filman *et al.* 2004]. A AOSD propõe tratar uma característica transversal ao requisito através de unidades de modularização denominadas aspectos. Os aspectos encapsulam toda a implementação responsável por interceptar o fluxo de execução do software e injetar o tratamento da característica transversal nele. Assim, os problemas associados ao entrelaçamento e espalhamento da característica transversal pelo software embarcado são atenuados. Para interceptar o fluxo de execução do software o aspecto precisar saber exatamente os pontos do código e/ou especificação onde devem ocorrer as características transversais.

A especificação da seleção de *Pontos de Ocorrências das Características Transversais* (POCT) indica onde, na especificação e no código fonte, podem ou deve haver características transversais que são inerentes aos requisitos funcionais ou não-funcionais do sistema embarcado. Existem várias linguagens de programação e também notações visuais orientadas a aspectos que apresentam diversos meios linguísticos para identificação e especificação dessas ocorrências. As formas mais comuns de designar a seleção de POCT são baseadas em propriedades léxicas aplicadas em um contexto comportamental ou estrutural das características transversais [Stein *et al.* 2006] e embora exista um número significativo de linguagens, seja em forma gráfica ou

textual, a compreensão dos elementos relacionados à AOSD geralmente requer um conhecimento profundo não só dos seus conceitos, mas também da sintaxe e da semântica da linguagem.

Existem muitas extensões de linguagens consolidadas, e.g., Java e UML, que empregam uma semântica diferente uma da outra para indicar POCT. Assim, é difícil entender a especificação de seleções de POCT escritas em uma linguagem adaptada ou em notações visuais que são geralmente adotadas nas abordagens de modelagem da AOSD [Elrad *et al.* 2002, Kienzle *et al.* 2009, Yang e Wei-Dong 2019]. Tal situação compromete o entendimento de "como" e "onde" as características transversais ocorrem e são tratadas, impactando negativamente no desenvolvimento de software embarcado, e.g., projetos de sistemas embarcados de tempo-real, que apresentam vários tipos distintos de características transversais inerentes não somente aos requisitos funcionais e não-funcionais como também as suas restrições [Freitas *et al.* 2007], e.g., restrições de tempo, concorrência, monitoramento de recursos, segurança, confiabilidade e outros.

O estado da arte atual apresenta dois tipos de abordagens para especificar o tratamento das características transversais: as notações gráficas e as textuais. A especificação da seleção de POCT através da notação gráfica é uma área de pesquisa que tem sido bastante explorada desde o início dos anos 2000 [Mongiovi *et al.* 2017, Acreţoaie *et al.* 2018, Khan *et al.* 2019]. Por exemplo, o *Join Point Designation Diagram* (JPDD) [Stein *et al.* 2006] é uma técnica que foi criada para identificar e especificar a seleção de POCT através da forma gráfica. O JPDD propõe uma extensão da UML [Object Management Group (OMG) 2017] cujo objetivo é especificar as seleções de POCT de uma forma independente do sistema ou da linguagem orientada a aspectos. Outra abordagem gráfica a ser destacada é a *Theme/UML* [Clarke 2002], que também é baseada na UML, mas que atende a separação simétrica das características transversais; no entanto a separação assimétrica é suportada pela maioria das abordagens AOSD. Os desenvolvedores especificam ligações (*bindings*) entre os temas para indicar onde a seleção de POCT deve ser inserida no fluxo de execução do software.

Quanto a abordagem textual, destaca-se a AspectJ [Kiczales Erik Hilsdale 2001]. Os autores defendem que as características transversais de um sistema não podem ser propriamente encapsuladas em seus módulos de implementação usando técnicas convencionais de desenvolvimento de software como por exemplo, a orientação a objetos ou o paradigma procedimental. A decomposição hierárquica se faz necessária no desenvolvimento de software com o intuito de reduzir a complexidade, subdividindo sucessivamente o problema principal em subproblemas

de menor complexidade. No entanto, o uso dessa abordagem hierárquica, que é uma prática comum na orientação a objetos, não resolve o problema de redução de complexidade quando se trata de características transversais inerentes a requisitos funcionais ou não-funcionais, e.g., sistemas embarcados de tempo-real compreendem um conjunto de tarefas periódicas, que são implementadas como threads ou processos. O controle da execução periódica dessas tarefas é uma característica transversal, pois os mesmos pedaços de código são encontrados em todas as tarefas implementadas [Wehrmeister 2009].

Contudo, mesmo sendo a UML amplamente aceita na indústria e na academia, ainda não existe uma semântica precisa quando se trata da especificação de seleções de POCT, até mesmo porque a UML foi criada para atender ou dar suporte ao paradigma orientado a objetos [Wehrmeister *et al.* 2014]. Consequentemente, embora existam abordagens que estendem a semântica da UML para suportar a AOSD [Wehrmeister *et al.* 2014, Clarke 2002, Stein *et al.* 2006], o resultado é uma especificação imprecisa e ambígua das seleções de POCT, devido a pouca discriminação sobre a eficácia cognitiva das representações visuais [Moody 2009]. Isso aumenta de forma indireta a dificuldade no desenvolvimento, compreensão e manutenção dos artefatos gerados para sistemas embarcados.

Sendo assim, o presente trabalho procura contribuir para qualidade da compreensão da especificação de seleção de POCT em sistemas embarcados através da proposição de uma nova técnica de modelagem e um modelo de qualidade que permita avaliar empiricamente técnicas e notações gráficas ou textuais para especificar implícita ou explicitamente a seleção de POCT no software embarcado.

Outro objetivo do presente trabalho é avaliar a compreensão da especificação das seleções de POCT em sistemas embarcados, seja ela gráfica ou textual, de forma indireta através da avaliação dos efeitos cognitivos e das propriedades de percepção das notações e assim, a partir dos resultados dessa avaliação propor uma nova técnica de especificação de seleção de POCT para sistemas embarcados, contribuição principal deste trabalho. Para isso, este trabalho propõe um modelo de qualidade baseado nas premissas e conceitos do framewok *Physics of Notations* (PoN) [Moody 2009]. Tal modelo inclui nove métricas com o objetivo de quantificar propriedades físicas perceptivas (i.e., sintaxe) das notações usadas para especificar a seleção de POCT em software embarcado. Tal modelo de qualidade e suas métricas também são contribuições deste trabalho.

1.1 OBJETIVO GERAL

O objetivo geral deste trabalho de pesquisa é contribuir com a qualidade da especificação de seleção de Pontos de Ocorrências de Características Transversais (POCT) aumentando a eficácia cognitiva das representações visuais através da proposição de uma nova técnica para especificar seleções de POCT.

1.2 OBJETIVOS ESPECÍFICOS

Para atingir o objetivo geral, é necessário atingir os seguintes objetivos específicos:

- Criar uma técnica para tratar os requisitos não-funcionais através de um arcabouço conceitual de alto nível de expressão que pode ser usado tanto na modelagem quanto na implementação de sistemas embarcados de tempo-real
- Criar uma técnica de especificação de seleção de POCT para sistemas embarcados seguindo as diretrizes da Física das Notações (em inglês *Physics of Notations - PoN*))
- Criar um modelo de qualidade baseado nos conceitos da Física das Notações que permita avaliar uma notação de forma quantitativa e não qualitativa
- Realizar experimentos aplicando o modelo de qualidade para verificar algumas técnicas de especificação de seleção de POCT. O objetivo é verificar a expressividade e precisão de linguagem de modelagem gráfica sob a perspectiva do PoN.
- Comparar a linguagem de modelagem proposta em relação a outras notações gráficas e textuais usadas para especificar a seleção de POCT.

1.3 CONTRIBUIÇÕES E DELIMITAÇÃO DE ESCOPO

Esta dissertação foi desenvolvida como uma extensão da abordagem AMoDE-RT (Em inglês, *Aspect-Oriented Model-Driven Engineering Approach for Distributed Embedded Real-Time Systems*) [Wehrmeister *et al.* 2014]. Um dos objetivos principais da AMoDE-RT é a manipulação de requisitos não-funcionais através de um arcabouço conceitual de alto nível de expressão que pode ser usado tanto na modelagem quanto na implementação de aspectos para sistemas embarcados de tempo real. Partindo desse contexto, aborda-se um problema na AMoDE-

RT quanto a compreensão da especificação da seleção de POCT em sistemas embarcados, tendo em vista a necessidade de uma especificação clara e concisa para aplicação dos mecanismos inerentes ao AOSD que tem por objetivo o gerenciamento do aumento da complexidade dos requisitos funcionais e não-funcionais de um sistema embarcado de tempo-real. Sendo assim, o escopo dessa dissertação se limita a uma parte da AMoDE-RT, sendo a mesma referente a modelagem e especificação de seleção de POCT, substituindo a JPDD (*Join Point Designation Diagram*) por outra representação visual mais concisa, que tem suas ambiguidades removidas e também possa ser incluída por referência a outros diagramas da AMoDE-RT (e.g., DERAF, ACOD).

Posto isto, seguem as contribuições desse trabalho de acordo com escopo delimitado acima:

- **Modelo de Qualidade baseado no PoN**: Esta dissertação contribui com o estado da arte da área através da proposta de um modelo de qualidade que tem por objetivo quantificar a compreensão da especificação da seleção de POCT de forma indireta através da avaliação dos efeitos cognitivos e das propriedades de percepção das notações textuais e gráficas sob a ótica da Física das Notações (Em inglês, *Physics of Notations* - PoN)

- **Técnica de especificação de seleção de POCT com codificação dupla**: Este trabalho contribui também na criação de uma nova técnica de especificação da seleção de POCT dentro do modelo UML, que faz uso do princípio de codificação dupla (em inglês, *dual coding*) [Moody 2009]. Este princípio prega que um modelo visual pode ser melhor compreendido se fizer uso tanto de símbolos gráficos quanto de símbolos textuais, contudo deve-se dar ênfase ao significado dos símbolos visuais através de símbolos textuais.

- **Uso da Física das Notações (PoN) em sistemas embarcados**: O uso de linguagens de modelagem visuais na engenharia de software embarcado não chega a ser um tópico novo de pesquisa. No entanto, a aplicação de princípios da Física das Notações (PoN) para avaliar os efeitos cognitivos e as propriedades de percepção da especificação de seleção de POCT no projeto de sistemas embarcados pode ser considerado um tópico de pesquisa recente.

1.4 ORGANIZAÇÃO DO TEXTO

O presente trabalho está organizado da seguinte forma. O Capítulo 2 apresenta um referencial teórico e revisão de conceitos usados no contexto deste trabalho. O Capítulo 3 faz uma análise do estado da arte e discute os trabalhos relacionados. O Capítulo 4 apresenta a técnica proposta para a especificação da seleção de POCT e como aplicá-la no contexto da AMoDE-RT. O Capítulo 5 apresenta o modelo de avaliação proposto para a especificação da seleção de POCT. O Capítulo 6 apresenta a avaliação empírica e analisa seus resultados e também discute as possíveis ameaças à validade dos resultados e da avaliação empírica. Finalmente, o Capítulo 7 apresenta as conclusões e indica direções para os trabalhos futuros.

2 REFERENCIAL TEÓRICO

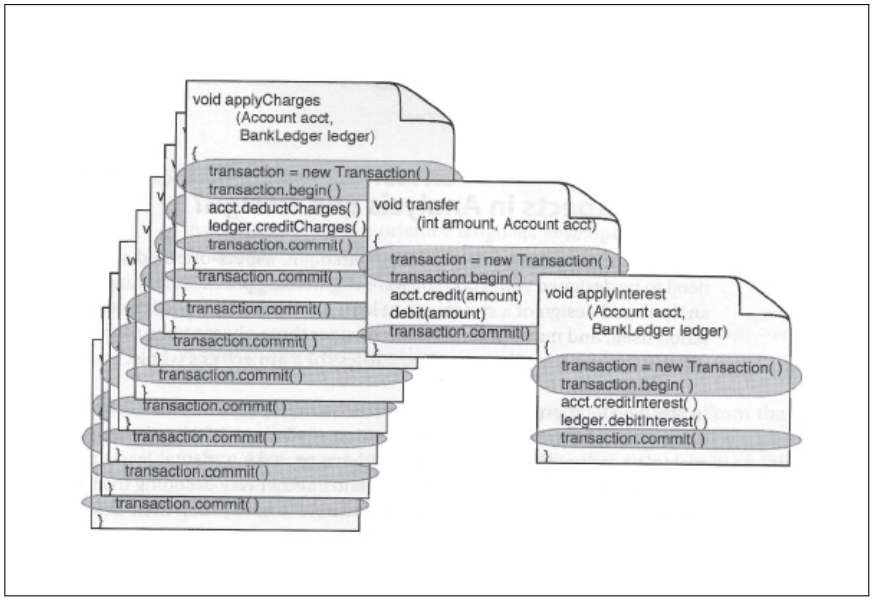
Esta seção apresenta um referencial teórico dos conceitos usados no contexto deste trabalho. O objetivo aqui é fornecer um entendimento para conceitos relevantes abordados neste texto, como, o que são as Características Transversais e os conceitos fundamentais quanto ao tratamento das Características Transversais. Para isso, é apresentada a abordagem *Aspect-Oriented Model-Driven Engineering for Real-time Systems* (AMoDE-RT), o framework *Distributed Embedded Real-time Aspects Framework* - (DERAF) e um exemplo de aplicação para ilustração quanto ao entendimento de ocorrências de características transversais. Além disso, é apresentada a Física das Notações (PoN) e seus 9 princípios.

2.1 CARACTERÍSTICAS TRANSVERSAIS

Uma característica transversal consiste em uma parte do código ou modelo visual que expressa uma responsabilidade específica que, em uma decomposição Orientada a Objetos (OO), é necessariamente espalhada por várias classes. São características que afetam ou interferem outras características do software levando a um entrelaçamento e espalhamento do código ou modelo visual. O comportamento de entrelaçamento e espalhamento viola os princípios básicos da orientação a objetos como desacoplamento e coesão. Exemplo de características transversais: armazenamento em cache de persistência, gerenciamento de memória, sincronização, segurança, confiabilidade, restrições de tempo, concorrência, monitoramento de recursos e outros. Os requisitos não-funcionais podem ser vistos como características transversais que impactam nos requisitos funcionais principalmente no desenvolvimento de sistemas embarcados. A Figura 1 apresenta um exemplo de uma característica transversal relacionadas ao gerenciamento de transações de um sistema. Neste exemplo, há um espalhamento do mesmo bloco de código entre vários módulos do sistema, ferindo o princípio de responsabilidade única de um método ou função.

Alguns autores, como [Kiczales *et al.* 1997], [Clarke e Baniassad 2005], [Wehrmeister 2009] afirmam que abordagens tradicionais como OO não lidam com características transversais de maneira adequada. Em outras palavras, a decomposição hierárquica da OO é incapaz de encapsular requisitos não-funcionais que possuem características transversais levando a um entrelaçamento e dispersão (i.e espalhamento) no manuseio desses requisitos. A consequência

Figura 1 – Espalhamento: o mesmo código em vários lugares



Fonte: [Clarke e Baniassad 2005]

de ter características transversais no código ou modelo visual é que ela leva a problemas de manutenção e qualidade devido à complexidade implícita na compreensão de sua implementação. A mitigação desse problema surgiu no Desenvolvimento de Software Orientado a Aspectos (em inglês, *Aspect-Oriented Software Development* - AOSD) [Filman *et al.* 2004]. O AOSD tem como objetivo encapsular a característica transversal em uma unidade modular de código e separar a funcionalidade principal de um sistema das características que possuem um comportamento mais abrangente, aumentando assim sua modularidade e reuso de código ou modelo visual. Sendo assim, é importante destacar alguns conceitos mais fundamentais relacionados ao tratamento das características transversais.

- **Tangling** indica que vários *concerns* são misturadas em um módulo;
- **Scattering** indica que um *concern* está espalhado por vários módulos;
- **Concerns** são características ou interesses que pertencem ao desenvolvimento do sistema, sua operação ou quaisquer outros aspectos que sejam críticos ou importantes para um ou mais interessados. Eles estão relacionados a requisitos funcionais e não-funcionais;
- **Separation of concerns** significa lidar com cada *concern* isoladamente, a fim de permitir a criação de artefatos modulares que os manipulem.
- **Modularization** significa a capacidade de agrupar ou particionar artefatos em entidades que idealmente devem ser fracamente acopladas e altamente coesas;

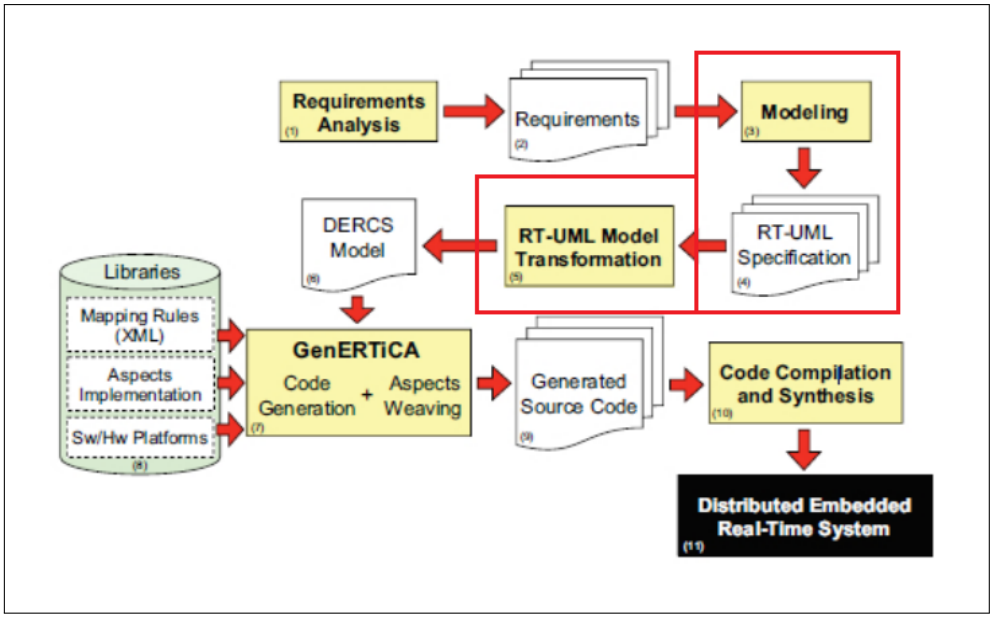
- **Composition** é a capacidade de integrar vários artefatos modulares em um todo coerentemente;
- **Decomposition** é a divisão do problema maior em problemas menores, que podem ser tratados separadamente um do outro;
- **Crosscutting** representa a ocorrência de *tangling* e *scattering* que ocorre quando a decomposição selecionada é incapaz de modularizar de forma eficaz os *concerns*;
- **Crosscutting concerns** são características que não podem ser mapeadas para módulos únicos, levando a um entrelaçamento e espalhamento do código ou modelo visual. Os requisitos não-funcionais podem ser vistos como características transversais, porque geralmente são misturados aos requisitos funcionais em vários módulos.
- **Pontos de Ocorrências de Características Transversais - POCT** são pontos bem definidos na estrutura ou no comportamento dos *concerns* que indicam ocorrências de características transversais.

2.2 ASPECT-ORIENTED MODEL-DRIVEN ENGINEERING FOR REAL TIME SYSTEMS (AMODE-RT)

A técnica de Engenharia Orientada a Aspectos para Sistemas em Tempo-Real (AMoDE-RT) [Wehrmeister *et al.* 2014] foi criada para abordar a "complexidade" associada ao design de um sistema embarcado de tempo-real. A AMoDE-RT permite a possibilidade de aumentar o nível de abstração durante o projeto de sistemas embarcados de tempo-real e também permite uma transição suave das fases iniciais de especificação e modelagem para as fases de implementação. Para atingir esses objetivos, a AMoDE-RT usa técnicas da Engenharia Orientada a Modelos (em inglês, *Model Driven Engineering* - MDE) [Schmidt 2006], combinadas com o AOSD [Filman *et al.* 2004] e com a Linguagem de Modelagem Unificada (em inglês *Unified Modeling Language* - UML) [Object Management Group (OMG) 2017].

A Figura 2 apresenta uma visão geral do design da abordagem AMoDE-RT. Dentre os domínios apresentados na figura, é possível destacar aqueles relacionados ao contexto desta dissertação. Na Figura 2 destaca-se em vermelho o escopo deste trabalho que está delimitado ao domínio *Modeling*, *RT-UML Specification* e o domínio *RT-UML Model Transformation*.

Figura 2 – AMoDE-RT Design



Fonte:[Wehrmeister 2009]

O objetivo deste trabalho é ser uma extensão a AMoDE-RT e especificamente modificar a abordagem de especificação de seleção de POCT.

A primeira etapa da AMoDE-RT é de reunir os requisitos e restrições com o objetivo de aplicá-los no domínio de sistemas embarcados de tempo-real distribuídos. A etapa seguinte é a de identificação de requisitos que é dividida em duas atividades que podem ser realizadas em paralelo: especificação de requisitos funcionais e especificação de requisitos não-funcionais. Primeiramente, um diagrama de caso de uso é criado. Ele descreve todas as funcionalidades esperadas para o sistema de tempo-real embarcado distribuído, e também os elementos externos que interagem com essas funcionalidades. No final dessas duas etapas, os projetistas produziram um conjunto de documentos especificando requisitos funcionais e não-funcionais com os quais o sistema em desenvolvimento deve lidar e também as relações entre esses requisitos. Esses documentos são então usados na próxima fase: modelagem do sistema. Os diagramas UML anotados com o estereótipo do perfil MARTE [Object Management Group (OMG) 2018] são usados para modelar a estrutura e o comportamento de sistemas embarcados de tempo-real distribuídos. Nesta fase, os modelos UML são criados e sucessivamente refinados para atingir o nível de detalhe desejado, fornecendo informações suficientes para permitir a realização do sistema. No modelo UML inicial, os elementos descrevem conceitos que estão mais próximos do domínio do aplicativo de destino, por exemplo, sensores, dispositivos de direção, turbinas, informações de velocidade e trajetória, braços de robôs etc. Esses elementos representam conceitos do domínio do problema, ocultando detalhes sobre sua implementação.

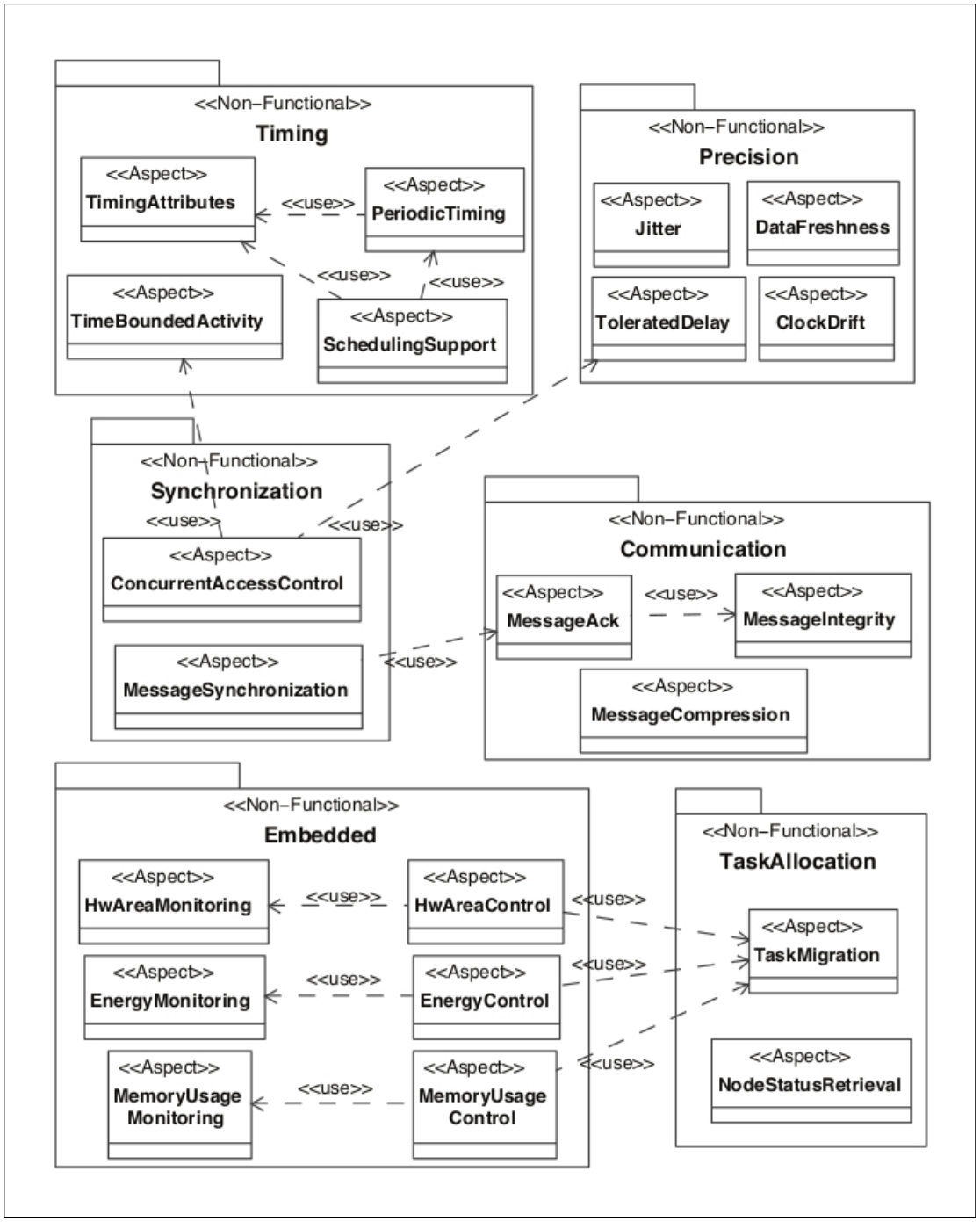
Níveis de abstração mais altos são mais fáceis de entender e permitem que o projetista se concentre nas bases dos aplicativos em vez de se preocupar com questões de implementação. A especificação do tratamento de requisitos não-funcionais é feita com auxílio de uma outra técnica chamada DERAF, explicada na próxima seção. No final da fase de modelagem, os projetistas criam um modelo UML que especifica os elementos para lidar com os requisitos funcionais e não-funcionais, usando, respectivamente, os conceitos de Orientação a Objetos e AOSD. Durante essa fase de modelagem, diagramas são criados com o objetivo de especificar a seleção de elementos do modelo que são afetados por características transversais de requisitos funcionais e não funcionais. Para isso, os engenheiros de software criam diagramas de especificação de pontos de junção (Em inglês, *Join Point Designation Diagram - JPDD*). É nessa fase que a presente dissertação propõe uma contribuição na criação de uma nova técnica de especificação de POCT chamada *Join Point Specification Diagram - JSD*, a ser usada no processo de modelagem da AMoDE-RT, subistituindo assim a JPDD, devido a melhoria na aplicação da JSD quanto a compreensão das propriedades de percepção (sintaxe) na especificação de seleção de POCT para sistemas embarcados, diminuindo assim a complexidade da abordagem AMoDE-RT.

A próxima etapa da AMoDE-RT realiza a transformação do modelo UML anotado com estereótipos de perfil MARTE em uma instância do *Distributed Embedded Real-Time Compact Specification* (DERCS) [Wehrmeister 2009], que é usado para fins de geração de código e execução de modelo. A etapa seguinte é a geração do código-fonte do modelo DERCS através de uma ferramenta de geração de código chamada *Generation of Embedded Real-Time Code based on Aspects* (GenERTiCA) o qual foi desenvolvida em [Wehrmeister 2009]. A última etapa da AMoDE-RT é o uso de uma ferramenta de terceiros para compilar e sintetizar o código do aplicativo gerado. Além disso, os arquivos de configuração da plataforma gerados são usados para configurar a plataforma final que será implementada. Depois disso, o sistema embarcado de tempo-real distribuído está pronto para ser executado ou testado.

2.3 FRAMEWORK DE ASPECTOS PARA SISTEMAS DISTRIBUÍDOS

O Framework de Aspectos para Sistemas Distribuídos (Em inglês, *Distributed Embedded Real-time Aspects Framework* - DERAF) [Wehrmeister 2009], tem por objetivo modularizar a manipulação de requisitos não-funcionais encapsulando em um único elemento todas as questões relacionadas a manipulação de requisitos não-funcionais. Esse framework é importante para dissertação devido aplicação da técnica proposta (JSD), onde é necessário o uso do DERAF

Figura 3 – Características Transversais fornecidas por DERAF



Fonte: [Wehrmeister 2009]

para o tratamento da seleção de POCT. A Figura 3 apresenta uma visão geral das características transversais manipuladas por DERAF. Como pode ser visto, são apresentados seis pacotes de requisitos não-funcionais com suas características transversais agrupadas de acordo com seus objetivos. Nas seções seguintes apresentamos um breve resumo das questões semânticas de cada características transversal.

2.3.1 Pacote Timing

Este pacote contém características transversais para lidar com os requisitos relacionados ao tempo, como prazos para execução de atividades, informações de complexidade de pior caso (Em inglês, *Worst-Case Execution Time - WCET*), ativação de tarefas periódicas e outros. A características transversal *TimingAttributes* é responsável por lidar com as características dos objetos ativos como deadline, prioridade, WCET e instantes de tempo absolutos nos quais seu comportamento deve iniciar e terminar a execução. Atributos que representam as características mencionadas são inseridos nas classes de objetos ativos afetados, bem como métodos e comportamentos para inicializar e tratar esses atributos.

2.3.2 Pacote Precision

A precisão no cumprimento dos requisitos de tempo é tratada pelos aspectos deste pacote, que concentra esforços em características como o atraso máximo tolerado no início das atividades, variação na pontualidade das atividades, duração da validade da informação ou o desvio da referência do relógio local em comparação com o global. O aspecto de *jitter* mede a variação da precisão nas atividades realizadas pelo sistema. Este aspecto fornece meios para medir o tempo antes (ou depois) de uma atividade observada acontecer, armazenando esta informação (o histórico deve fornecer informações de pelo menos uma amostra de tempo) para calcular a variância entre os instantes de tempo observados. Este aspecto pode ser usado, por exemplo, para calcular o *jitter* em uma ativação ou execução periódica de objeto ativo, ou para calcular a variação de tempo de um envio de mensagem periódica.

2.3.3 Pacote Synchronization

A sincronização e o controle de acesso simultâneo aos requisitos de recursos compartilhados são tratados pelos aspectos deste pacote. O aspecto *ConcurrentAccessControl* fornece meios para controlar o acesso simultâneo a objetos, que compartilham suas informações de atributos com outros objetos.

2.3.4 Pacote Communication

Este pacote fornece aspectos para lidar com a comunicação de objetos em termos de envio de mensagens. A intenção é cobrir a comunicação entre objetos que estão localizados em dispositivos de computação fisicamente separados.

2.3.5 Pacote TaskAllocation

Os aspectos fornecidos por este pacote tratam de requisitos não funcionais relacionados à distribuição de objetos em diferentes dispositivos de computação em tempo de execução. Esses aspectos são normalmente relacionados a nós do sistema distribuído que são fisicamente separados.

2.3.6 Pacote Embedded

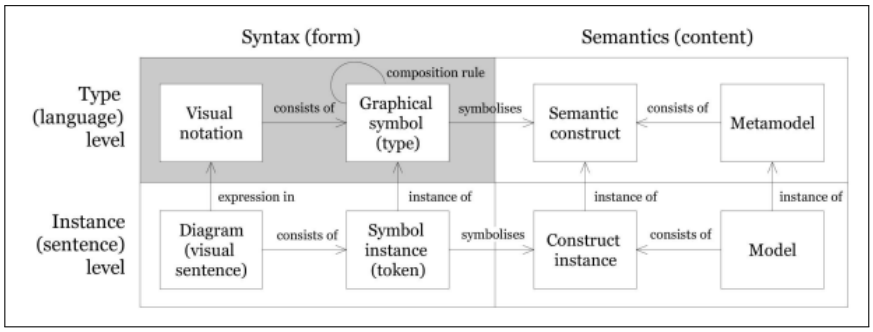
Requisitos não-funcionais relacionados à disponibilidade de recursos físicos, que são preocupações muito comuns no projeto de sistemas embarcados, são tratados pelos aspectos deste pacote. O consumo de energia, o uso de memória e a área reconfigurável de hardware podem ser citados como exemplos de tais preocupações.

2.4 A FÍSICA DAS NOTAÇÕES

O framework conceitual Física das Notações (em inglês, *Physics of Notations* - PoN) [Moody 2009] fornece uma base lógico-científica para o design das notações visuais na engenharia de software. Moody afirma que as notações visuais são parte integrante da engenharia de software, mas os pesquisadores e designers de notações têm historicamente ignorado ou subestimado questões importantes quanto a eficácia cognitiva da representação visual. Ele afirma que, embora vários trabalhos na literatura avaliem e comparam as notações, os detalhes quanto a sintaxe das representações visuais são raramente discutidos. Para preencher essa lacuna, o PoN aborda as propriedades físicas e perceptivas das notações (sintaxe), em vez de suas propriedades lógicas (semânticas). As propriedades perceptivas são analisadas à luz da Teoria do Design apresentada em [Jones e Gregor 2007].

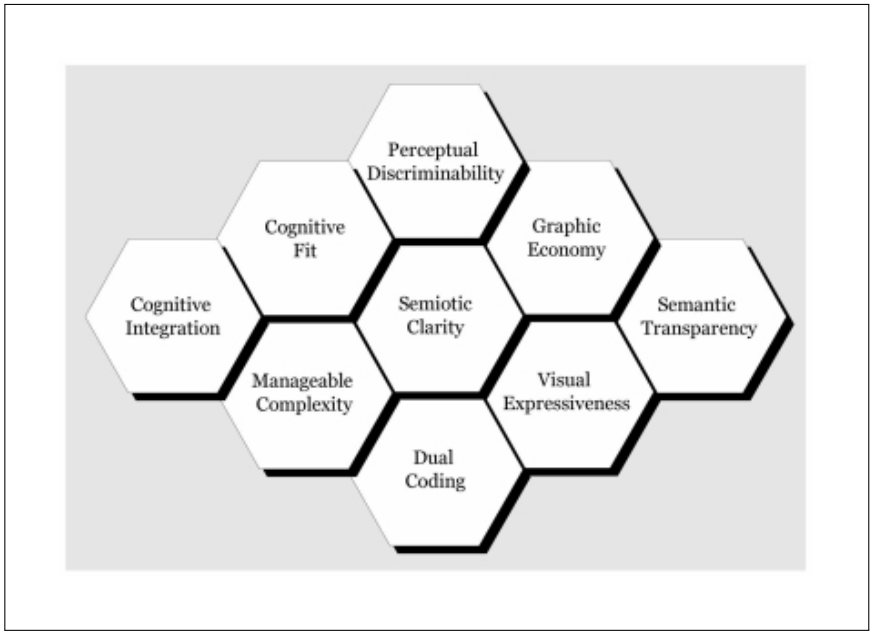
A Figura 4 apresenta um resumo do escopo do PoN, demonstrando que o propósito da criação do framework não está relacionado com avaliação de construções semânticas apropriadas ou a definição de significados de símbolos e sim com suas propriedades físicas e perceptivas (sintaxe). A teoria associada ao PoN envolve nove princípios de design, que são embasados em teorias e evidências empíricas sobre a efetividade cognitiva e perceptivas das notações visuais. [Moody 2009]

Figura 4 – Escopo PoN



Fonte: [Moody 2009]

Figura 5 – Princípios PoN



Fonte: [Moody 2009]

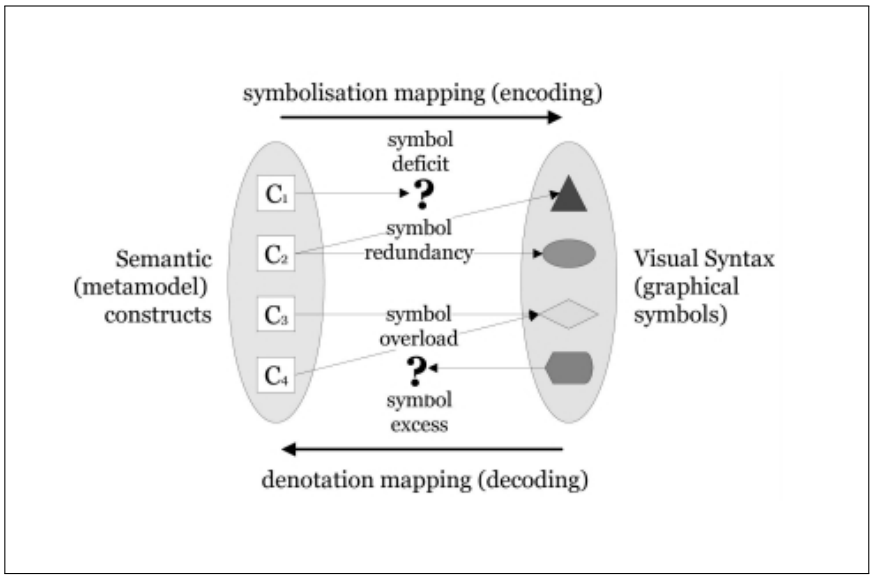
Os princípios apresentados na Figura 5 formam a teoria prescritiva para notações visuais ou a Teoria do Design e Ação que define os princípios explícitos para transformar o design de uma notação visual de um processo inconsciente (*craft*) para um processo consciente (*design discipline*) [Jones e Gregor 2007]. Nesta dissertação, 5 princípios foram escolhidos. Um resumo

de todos os princípios da Física das Notações é explicado abaixo e o porquê apenas 5 princípios foram selecionados para essa dissertação:

- ***Clareza Semiótica*:** define que deve haver uma correspondência 1:1 entre construções semânticas e símbolos gráficos numa notação. De acordo com a teoria dos símbolos para que uma notação satisfaça os requisitos de um sistema notacional, é uma condição *sine qua non* a correspondência 1:1 dos símbolos e seus conceitos os quais referenciam; e.g., linguagens naturais não são sistemas notacionais pois as mesmas contêm sinônimos e homônimos. Quando não há uma correspondência 1:1 entre os símbolos pode-se ocorrer umas das anomalias apresentadas na Figura 6. Conforme [Moody 2009] essas anomalias podem ser: (a) défict de símbolos quando existem construções semânticas que não são representadas por nenhum símbolo; (b) redundância de símbolos quando uma construção semântica é representada por vários símbolos; (c) sobrecarga de símbolos quando diferentes construções semânticas podem ser representadas pelo mesmo símbolo; e por fim, (d) o excesso de símbolos que pode ocorrer quando símbolos do modelo computacional não correspondem a nenhuma construção semântica.

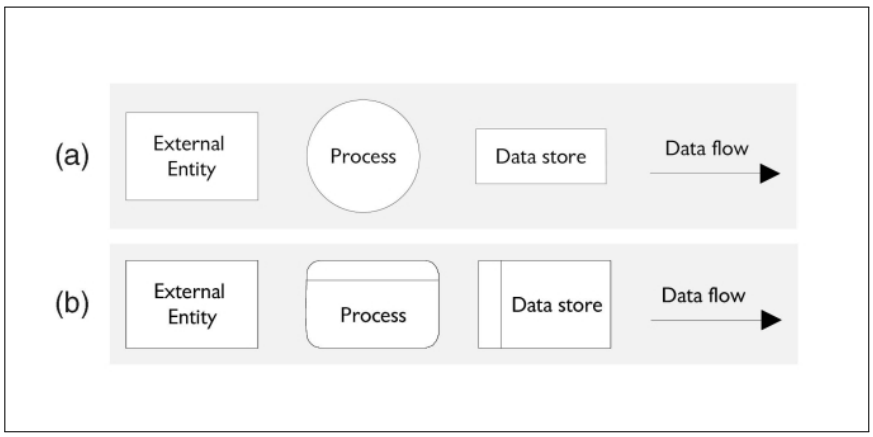
- ***Discernimento de Percepção*:** define que os símbolos diferentes devem ser claramente distinguíveis uns dos outros. Esse princípio parte da premissa que uma diferenciação precisa entre símbolos é um pré-requisito para uma maior acuracidade na interpretação dos mesmos [Moody 2009]. A distância visual entre os símbolos é um fator determinante no discernimento de percepção. Para medir a distância visual em um modelo é necessário quantificar o número de símbolos diferentes entre si. No geral, quanto maior for a distância visual entre os símbolos, mais rápido e com mais precisão será o reconhecimento dos símbolos em um modelo ou diagrama. Em contrapartida, diferenças sutis entre os símbolos podem resultar em erros de interpretação [Moody 2009]. O esforço para discriminação perceptiva é mais alto para engenheiros em começo de carreira do que para engenheiros mais experientes até porque a experiência os habilita em fazer distinções mais finas e precisas. A maioria das notações de Engenharia de Software usam um repertório limitado de formas, na maioria variações do retângulo [Moody 2009]. Figura 7 apresenta uma comparação entre uma especificação de um modelo com excelência e uma especificação medíocre, onde em (a) o autor usa uma clara distinção entre as formas para todas as contruções semânticas do modelo, equanto em (b) o autor usa variantes de retângulo.

Figura 6 – Princípio da Clareza Semiótica



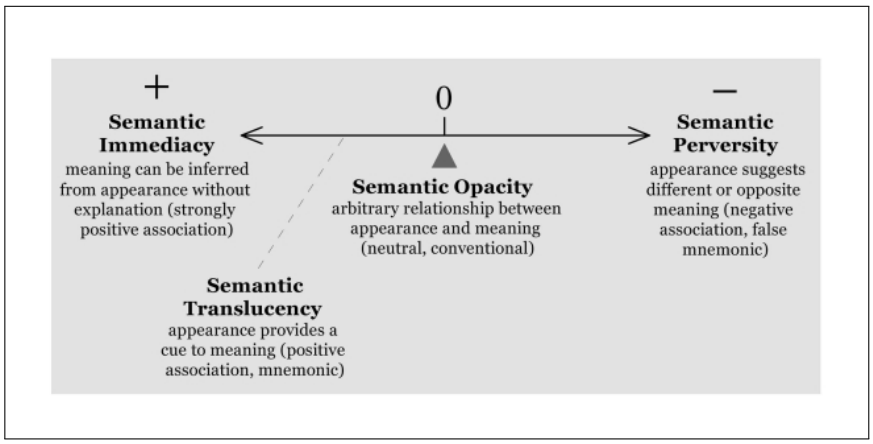
Fonte: [Moody 2009]

Figura 7 – Princípio do Discernimento de Percepção



Fonte: [Moody 2009]

Figura 8 – Princípio da Transparência Semântica



Fonte: [Moody 2009]

- ***Transparência Semântica*:** Esse princípio define que a aparência dos símbolos deve sugerir seu significado de forma que conduza o programador/engenheiro a um entendimento mais rápido e preciso, especialmente para engenheiros iniciantes [Moody 2009]. Enquanto o princípio de discernimento de percepção enfatiza que símbolos devem ser diferentes uns dos outros, este princípio requer que os símbolos forneçam pistas de seus significados. É uma tratativa de formalizar a naturalidade e a intuição nas notações visuais gerando elementos mnemônicos que reduzem a complexidade cognitiva de entendimento do modelo computacional. A Figura 8 define o grau de associação entre a forma dos símbolos e seus respectivos conteúdo.

- ***Expressividade Visual*:** define o uso de todas as possibilidades e capacidades das variáveis visuais usadas numa notação. O princípio de expressividade visual mede a utilização da capacidade de variação visual através do vocabulário visual. Uma representação visual perceptualmente rica é consequência da exploração dos múltiplos canais visuais de comunicação, e.g., forma, posição horizontal, posição vertical, tamanho, orientação, brilho, cor e textura. A Figura 9 apresenta uma escala referente as variáveis visuais. O princípio de expressividade visual divide o conjunto de variáveis visuais em dois subconjuntos:

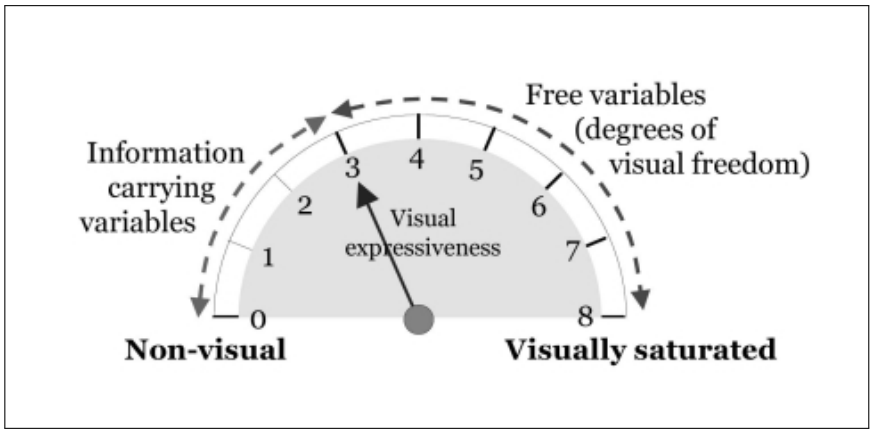
  - **Variáveis-portadoras de informação** são variáveis que encapsulam informações numa notação.
  - **Variáveis Livres** são variáveis de comunicação não usadas formalmente.

  Uma notação sem variáveis-portadoras de informação (e.g., variáveis visuais), possuem uma expressividade zero e oito graus de liberdade visual, sendo chamados de notações não-visuais ou textuais, enquanto o inverso, no caso, uma notação que usa todas as variáveis visuais possui uma expressividade visual igual a oito e zero grau de liberdade visual, sendo definida como uma notaçao visualmente saturada.

- ***Economia Gráfica*:** define que o número de símbolos gráficos diferentes deve ser cognitivamente gerenciável. Uma notação visual pode possibilitar um número imenso de combinações de variáveis visuais de diferentes formas. Contudo, se compararmos com linguagens textuais, essa estratégia será eficaz até certo ponto, visto que há um limite cognitivo no número de variáveis visuais que podem ser reconhecidas de forma eficiente [Moody 2009]. Diagramas ou modelos visuais são bem aplicados quando fazem uso de abstrações que resumem todo um contexto mais detalhado, sendo assim, o uso de muita

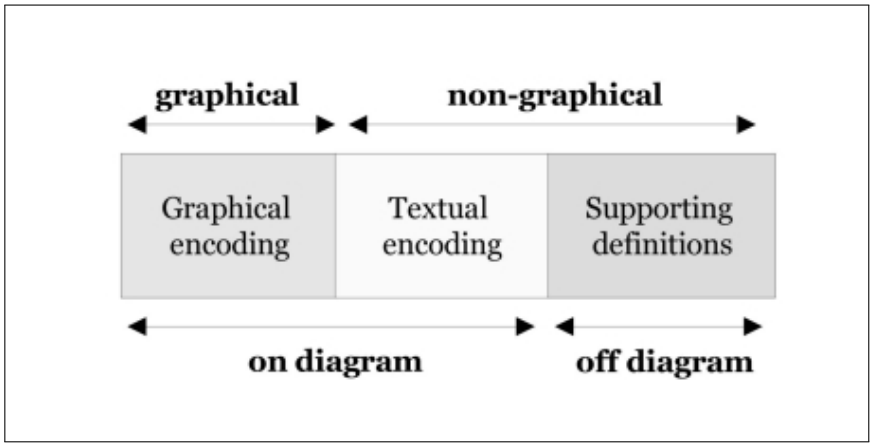
informação gráfica pode se tornar improdutivo [Moody 2009]. Além disso, algumas informações de um modelo computacional são melhores descritas na forma textual [Moody 2009]. A Figura 10 apresenta uma ação de equilíbrio para manter a complexidade de um diagrama gerenciável onde, numa especificação de uma notação, deve-se verificar quais informações precisam ser gráficas, quais precisam ser textuais, como também o que incluir nas definições de suporte.

Figura 9 – Princípio da Expressividade Visual



Fonte: [Moody 2009]

Figura 10 – Princípio da Economia Gráfica

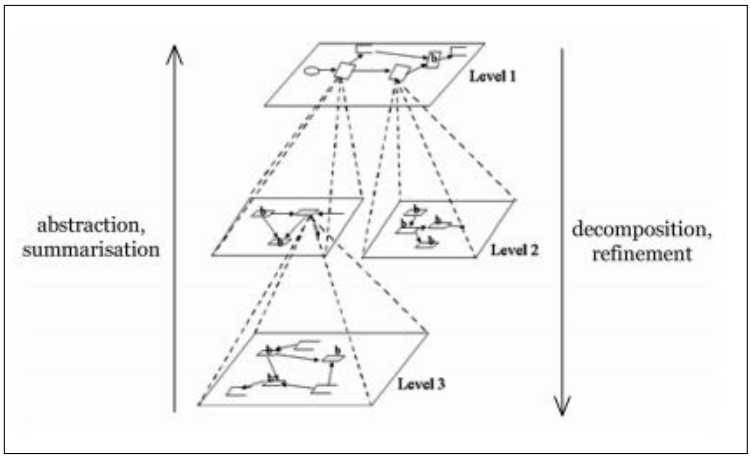


Fonte: [Moody 2009]

- ***Gerenciamento de Complexidade*:** se refere à capacidade de uma notação visual de representar informações sem sobrecarregar a mente humana. Esse é um dos problemas mais complexos no projeto de uma notação visual. O gerenciamento de complexidade tem um efeito importante sobre a eficácia cognitiva, pois a quantidade de informação que pode ser transmitida de forma eficaz por um único diagrama é limitada pelas capacidades perceptivas e cognitivas humanas [Moody 2009]. A Figura 11 apresenta uma modelagem de um

sistema representado em múltiplos níveis de abstração. Verifica-se que a complexidade do modelo é gerenciada em cada nível (L1, L2, L3).

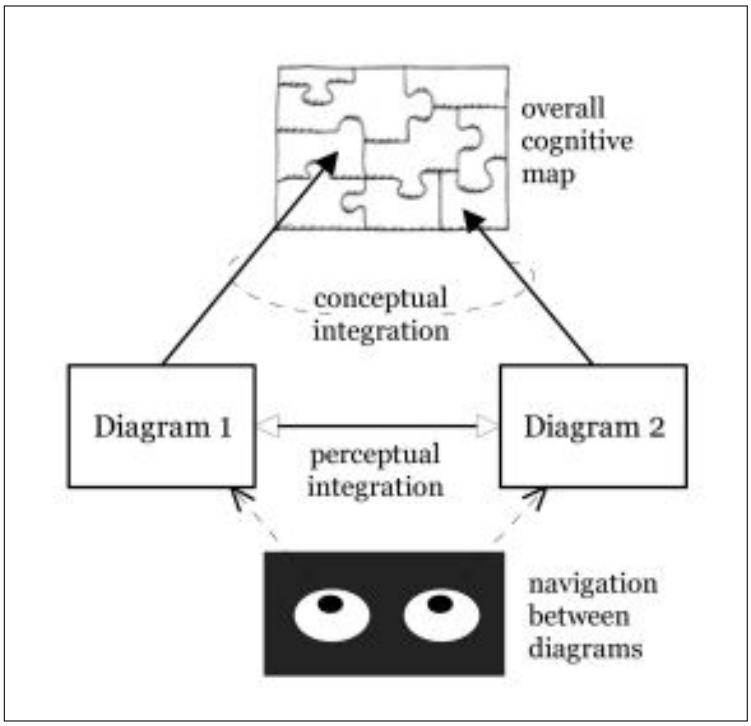
Figura 11 – Princípio do Gerenciamento de Complexidade



Fonte: [Moody 2009]

- ***Integração Cognitiva*:** este princípio aplica-se somente quando vários diagramas são usados para representar o sistema. Moody afirma que este é um problema crítico na Engenharia de Software, onde os problemas são normalmente representados por sistemas de vários diagramas em vez de diagramas únicos. A Figura 12 apresenta uma ilustração de integração cognitiva, quando vários diagramas são usados para representar um domínio.

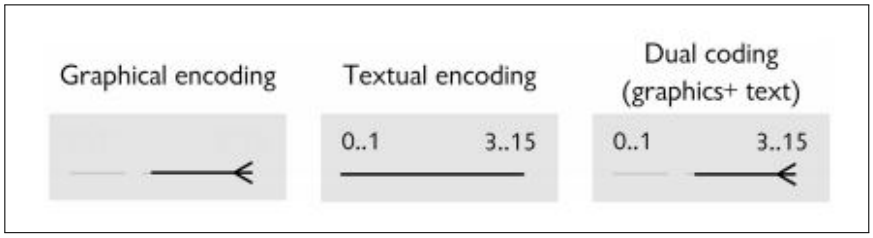
Figura 12 – Princípio da Integração Cognitiva



Fonte: [Moody 2009]

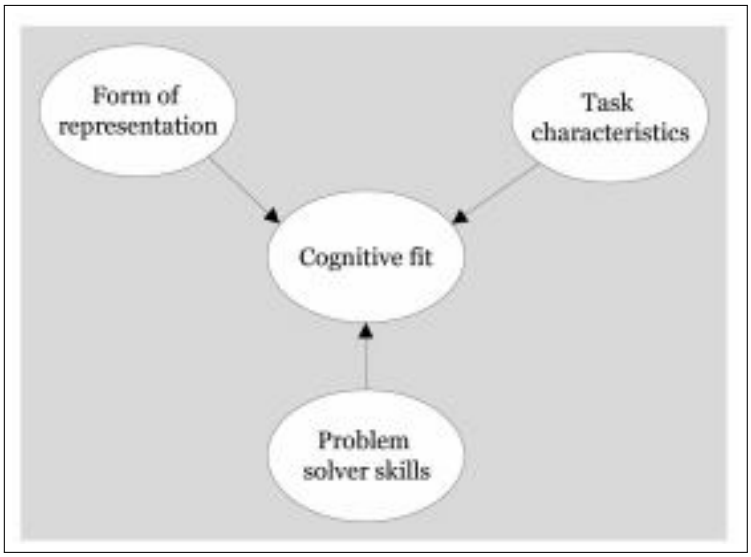
- ***Codificação Dupla*:** Os princípios de discernimento de percepção e expressividade visual desaconselham o uso de texto para codificar informações em notações visuais. No entanto, isso não significa que o texto não tenha lugar no projeto de uma notação visual. Imagens e palavras não são inimigas e não devem ser mutuamente exclusivas [Moody 2009]. De acordo com a teoria da codificação dupla, usar textos e gráficos juntos para transmitir informações é mais eficaz do que usar apenas um deles. Quando as informações são apresentadas de forma textual e visual, suas representações são codificadas em sistemas separados na memória e as conexões entre as duas são fortalecidas. A Figura 13 apresenta três especificações, onde a última se refere ao uso de símbolos visuais e textuais, representando o melhor dos dois mundos.

Figura 13 – Princípio da Codificação Dupla



Fonte: [Moody 2009]

Figura 14 – Princípio do Ajuste Cognitivo



Fonte: [Moody 2009]

- ***Ajuste Cognitivo*:** A teoria do ajuste cognitivo é uma teoria amplamente aceita no campo dos sistemas de informação, sendo validada em diversos domínios, desde a tomada de decisão gerencial até a manutenção do programa. [Moody 2009]. Este princípio do ajuste cognitivo afirma que diferentes representações da informação são adequadas para diferentes

tarefas e diferentes públicos. O desempenho na resolução de problemas (que corresponde aproximadamente à eficácia cognitiva) é determinado por um ajuste de três vias entre a representação do problema, as características da tarefa e as habilidades de resolução de problemas. A Figura 14 apresenta uma ilustração do princípio de ajuste cognitivo.

2.5 EXEMPLO DE APLICAÇÃO: CADEIRA DE RODAS AUTOMATIZADA

Para ilustração quanto ao entendimento de ocorrências de características transversais em sistemas embarcados, esta seção apresenta um exemplo de aplicação que consiste em um projeto de um sistema embarcado de tempo-real de automação e controle distribuído de uma cadeira de rodas. Este exemplo se refere a um subsistema de controle de movimento de uma cadeira de rodas automatizada, que é composto por um joystick de alavanca convencional, dois sensores de velocidade de rotação da roda e dois motores. O sistema deve realizar as seguintes atividades simultâneas:

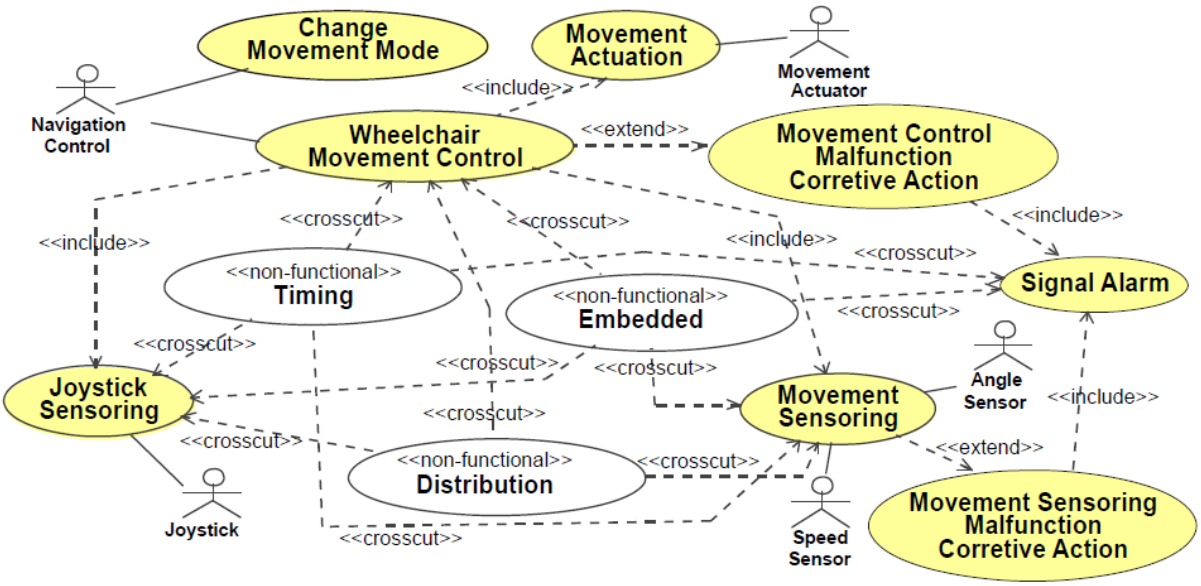
- Manipular os dados dos sensores de roda a cada 10 milissegundos para determinar o movimento, velocidade e direção;
- Determinar a posição atual do joystick no mesmo período, ou seja, 10 ms;
- Executar o algoritmo de controle da cadeira de rodas a cada 50 ms, aplicando o valor calculado nos motores da roda esquerda e direita;
- Monitorar as mudanças no modo de operação. O modo de operação influencia em como tratar os atrasos de prazos: (i) sinaliza a ocorrência de prazos perdidos; (ii) sinaliza a ocorrência de prazos perdidos e aplica o último valor válido; ou (iii) sinaliza a ocorrência de prazos perdidos e para o movimento da cadeira de rodas.

A Figura 15 apresenta um diagrama de caso de uso descrevendo os requisitos funcionais (RFs) e os não-funcionais (RNFs). Os casos de usos dos requisitos não-funcionais apresentam a mesma sintaxe dos funcionais, porém com um estereótipo aplicado sobre ele. (<<non-functional>>). Para simplificar a representação dos pontos de ocorrências de características transversais (POCT), optamos por apresentar apenas o RNF relacionado a periodicidade que é representada pelo RNF de temporização. (i.e., Timing), devido a ser um RNF que possui uma natureza transversal, i.e., existem POCT que são inerentes ao requisito, e que devem ser

tratados. Para representar como esses RNFs interferem nos RFs do sistema, uma seta vai do elemento que representa os RNFs até o caso de uso afetado. Esta seta é indicada com o estereótipo <<crosscut>>.

Sendo assim, é possível visualizar no diagrama de caso de uso apresentado na Figura 15 que os RNFs deste exemplo ferem o princípio de responsabilidade única da Engenharia de Software quando interferem nos RFs, e.g., o requisito não-funcional *Timing* interfere nos requisitos funcionais: (a) Joystick Sensoring, Wheelchair Movement Control, Movent Sensoring e Signal Alarming gerando um espalhamento e emaranhamento dos conceitos, prejudicando assim no entendimento, manutenção e qualidade do projeto de desenvolivmento do sistema embarcado de tempo-real.

Figura 15 – Diagrama de caso de uso para cadeira de rodas automatizada



Fonte: [Wehrmeister 2009]

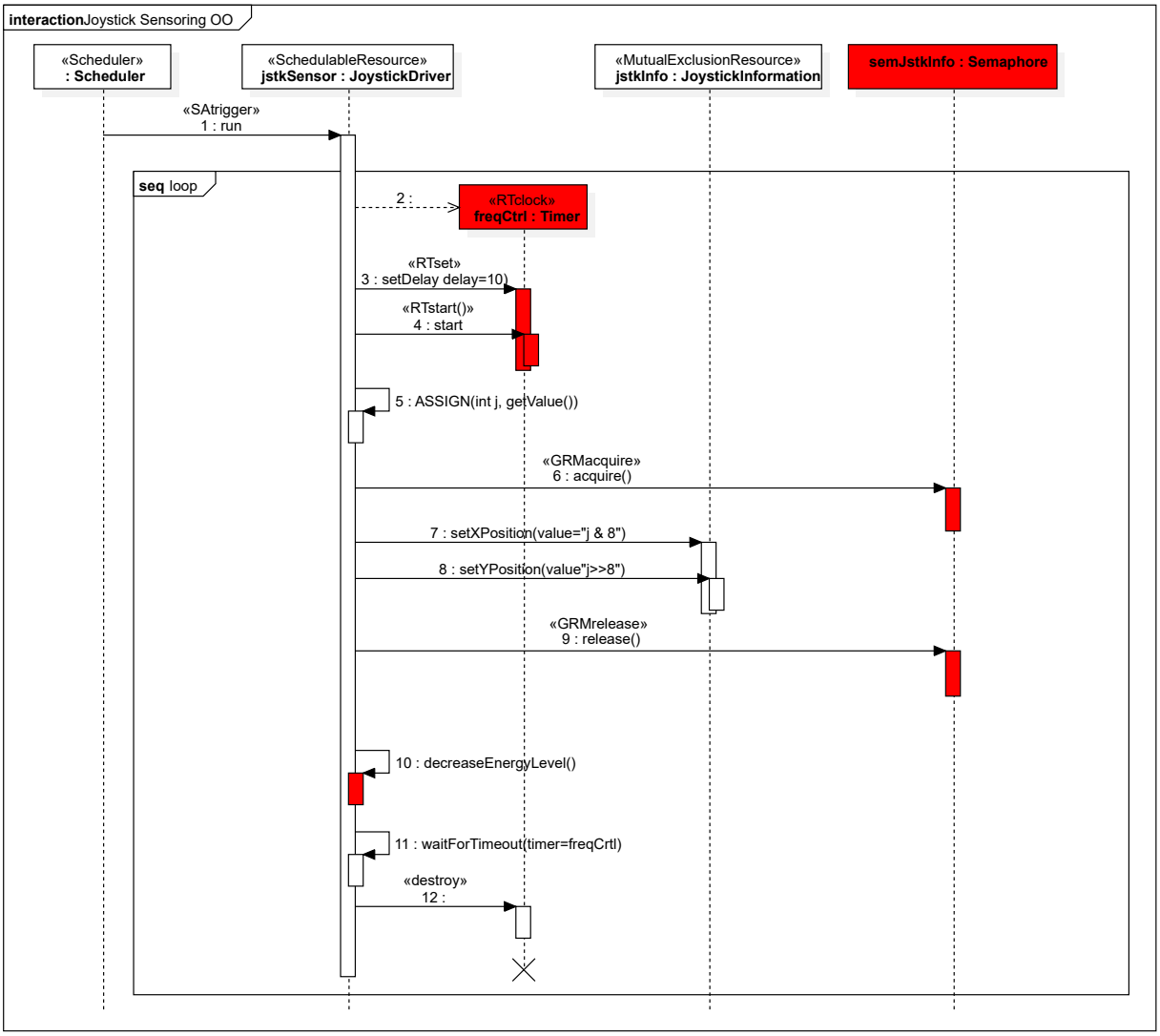
Posto isto, para propósito de ilustração, a Figura 16 apresenta um diagrama de sequência relacionada a funcionalidade de controle de movimento de uma cadeira de rodas automatizada. Nesse diagrama, existem três pontos de ocorrências de características transversais (POCT) destacadas em vermelho: (a) Timer, responsável pelas tarefas periódicas (anotada com o estereótipo RTclock), (b) o acesso exclusivo a recursos compartilhados, que é representado pelo semáforo (*Semaphore*), e (c) a diminuição do nível de energia. Basicamente, esse diagrama de sequência além de especificar as operações funcionais necessárias para o controle de movimento de uma cadeira de rodas automatizada, também especifica o tratamento dos POCT naturais ao requisito não-funcional de controle de tarefas periódicas (*Timing*). A ativação periódica da tarefa é

implementada a partir de um loop infinito, dentro do loop infinito se inicia um temporizador (mensagem 2), se determina o delay do temporizador em 10 milissegundos (mensagem 3), se inicia o temporizador (mensagem 4), se acessa o *joystickInformation* (que é um objeto compartilhado), através do semáforo (mensagem 6,7,8,9), monitora o nível de energia (mensagem 10), e mais adiante tem que aguardar pelo próximo temporizador, ficando entendido que todas essas mensagens vão terminar antes de gerar o *Timeout*, i.e., o tempo de execução da tarefa tem que ser menor que o período. Finalmente na mensagem 12 o temporizador é destruído. Todas essas operações são necessárias para que haja garantia de que as tarefas serão executadas de forma periódica.

Como mencionado acima, esses POCT são inerentes (i.e., naturais) ao RNF de temporização e traspassam a responsabilidade única de um requisito funcional, i.e., afetam o RF implicando em uma descentralização no tratamento dos requisitos do sistema embarcado de tempo-real, dificultando o reuso, a compreensão e a manutenção dos requisitos do sistema.

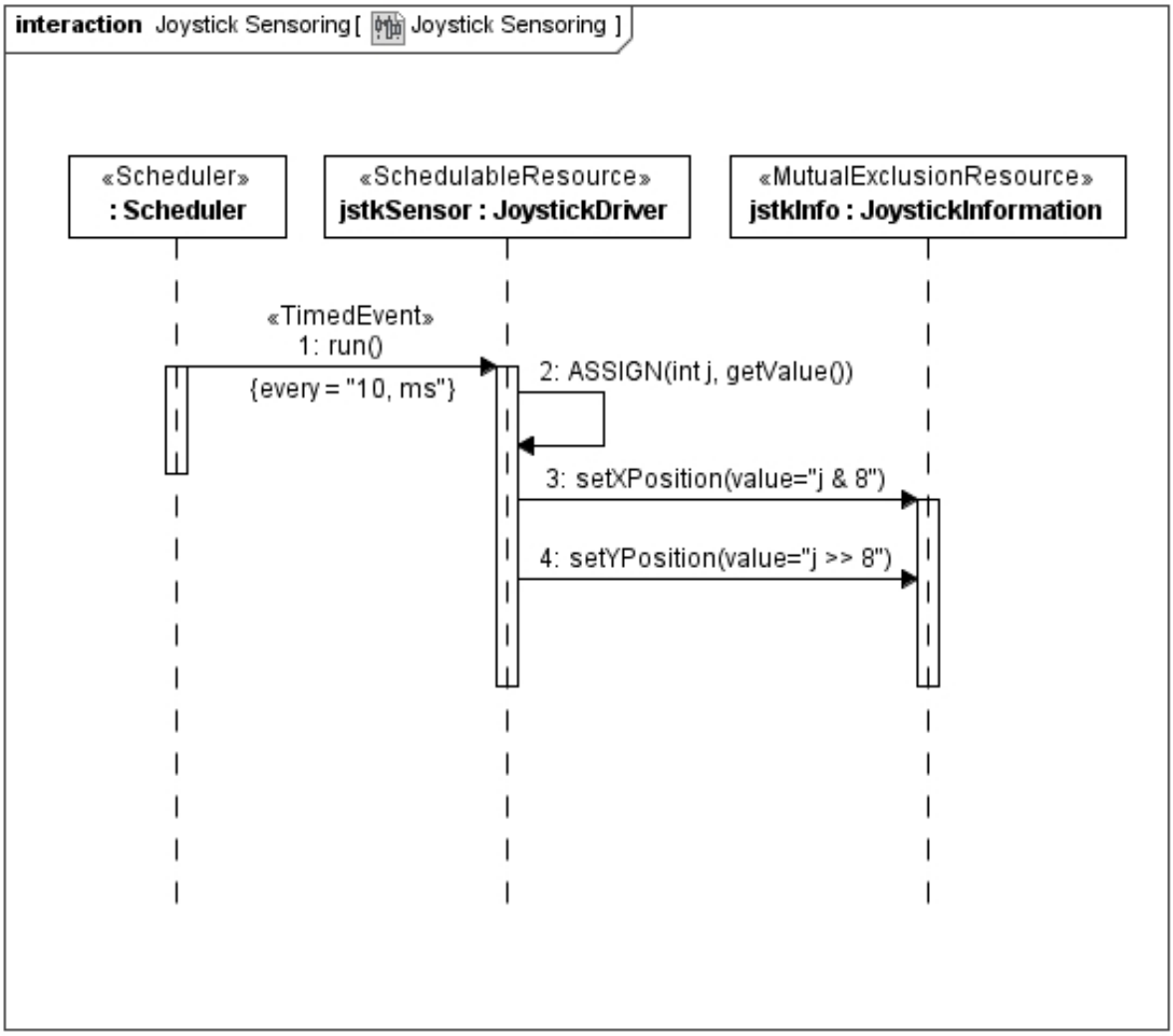
É possível atenuar esses problemas através da modularização e separação de conceitos aplicadas no desenvolvimento de software embarcado, i.e., isolando e modularizando o tratamento dos POCT (e.g., código de controle de tarefas periódicas). A Figura 17 apresenta um diagrama de sequência onde é especificado o que realmente é necessário para o desenvolvimento dos requisitos funcionais, i.e., o que está relacionado necessariamente com a aplicação em si: (i) executar o controle de tarefas periódicas, que são ativas a cada 10 milissegundos (o método run() está anotado com o estereótipo <<TimedEvent>>), (ii) coletar as informações dos sensores, (iii) calcular a lei dos controles e (iv) fazer a atuação, i.e., mudar o estado do objeto através dos métodos setXPosition(..) e setYPosition(..). Dessa forma, é possível verificar que todos os RNFs desaparecem do modelo, i.e., os POCT continuam a existir, a diferença é que nessa especificação está sendo usada a biblioteca do Desenvolvimento de Software Orientado a Aspectos (AOSD) que chama o tratamento dos POCT de forma modularizada, deixando só o que é principal no modelo. Sendo assim, a especificação fica mais clara, concisa, desacoplada e não fere o princípio de responsabilidade única da engenharia de software.

Figura 16 – Diagrama de sequência sem tratamento de ocorrências de características transversais



Fonte: o autor

Figura 17 – Diagrama de sequência com tratamento de ocorrências de características transversais



Fonte: [Wehrmeister 2009]

2.6 DISCUSSÃO

O objetivo deste capítulo foi de fornecer uma revisão dos conceitos principais relacionados com o objetivo desta dissertação. O motivo da apresentação da abordagem AMoDe-RT e do framework DERAF, é devido a associação da presente dissertação com aquela abordagem, tendo em vista que este trabalho visa fazer uma modificação do trabalho original, através da proposição de uma nova abordagem para especificar a seleção de POCT em sistemas embarcados. O entendimento quanto ao que significam as características transversais se faz necessário para a compreensão da seleção de POCT em sistemas embarcados. Para melhor compreensão da contribuição deste trabalho, se faz necessário também apresentar o framework PoN. Em especial, foram apresentados os cinco princípios do PoN usados na avaliação empírica apresentada neste trabalho que compara seis notações usadas para especificação de seleção de POCT através de métricas quantitativas que medem de forma indireta a compreensão da seleção de POCT através da avaliação dos efeitos cognitivos e das propriedades de percepção das notações. Outro ponto apresentado foi uma ilustração de um exemplo único de um sistema embarcado de tempo-real relacionado a automatização de uma cadeira de rodas. Foram apresentados os pontos de ocorrências de características transversais associadas ao requisito não-funcional de temporização (i.e., controle de tarefas periódicas), uma especificação de um diagrama de sequência sem tratamento dos POCT e outra com o tratamento através da separação de conceitos (Em inglês, *Separation of Concerns (SoC)*). Dessa forma, o conteúdo deste capítulo fundamenta a contribuição desta dissertação quanto a elaboração de uma nova técnica para especificação de seleção de POCT para sistemas embarcados

3 ANÁLISE DO ESTADO DA ARTE

Este capítulo apresenta uma análise do estado da arte no que diz respeito a especificação de seleção de Pontos de Ocorrências de Características Transversais (POCT). Foi realizado um mapeamento sistemático da literatura (Em inglês, *Systematic Literature Mapping* - SLM) com o objetivo de se identificar as principais técnicas e métodos usados na especificação da seleção de POCT, bem como identificar os pontos relevantes e lacunas dos trabalhos relacionados. O método de pesquisa do mapeamento sistemático foi desenvolvido com base nas diretrizes sugeridas por [Kitchenham 2004] e é apresentado no anexo B.

De acordo com a revisão sistemática, o número de problemas em aberto e desafios a serem resolvidos é alto. O texto mostrou deficiências quanto a identificação e especificação de seleção de POCT tanto usando notações gráficas como textuais, e.g., a falta de formalismo relacionado à fraca representação semântica, o uso dos símbolos da UML para AOSD que diferem bastante do uso tradicional, a falta de disseminação das técnicas, a pouca aceitação na indústria, a falta de conhecimento do AOSD, a semântica complexa (não trivial), a falta de avaliação empírica, entre outras. Um trabalho que mostra muitas promessas em sua aplicabilidade na indústria é apresentado em [Khan *et al.* 2019], que é a criação de uma linguagem responsável por verificar restrições transversais por meio da extensão da OCL. Como a OCL é bem aceita na área de Engenharia de Software, essa limitação poderia ser preenchida por este trabalho, mas de toda forma pelo fato de a AspectOCL trabalhar apenas com as restrições transversais e não com características transversais, isso impossibilitaria a aplicação da técnica no escopo deste trabalho e não teria relevância para resolução dos problemas em aberto relacionados a especificação de seleção de POCT, os quais este trabalho aborda.

Uma grande dificuldade no entendimento de abordagens de especificação de seleção de POCT é a falta de conhecimento da AOSD [Filman *et al.* 2004], uma vez que o conceito de pontos de junção (Em inglês *join points*) são derivados do AOSD. Essa definição não é abrangente o suficiente para o nível de modelagem e há uma complexidade associada a ela, e.g., nas linguagens de modelagem, há uma necessidade de unificar conceitos específicos de linguagem de programação em elementos mais abstratos para poder servir a várias linguagens de programação diferentes. A Tabela 1 apresenta as características das técnicas discutidas na revisão sistemática, fazendo uma comparação das mesmas de acordo com as características do AOSD apresentadas na literatura [Wimmer *et al.* 2011].

Tabela 1 – Comparação das notações visuais e textuais em termos de características

	Join Point Estrutural		Join Point Comportamental		Pointcut Simples		Pointcut Composto		Método de Quantificação			Posição Relativa			Abstração	
	estático	dinâmico	estático	dinâmico	gráfico	textual	gráfico	textual	declarativa	imperativa	enumeração	before	around	after	alto nível	baixo nível
Execution Patterns		✓		✓		✓		✓	✓						✓	✓
Dinamic Analysis		✓		✓		✓		✓	✓						✓	✓
Clone Detection	✓	✓	✓	✓	✓	✓	✓	✓		✓					✓	
Random Walks		✓		✓	✓	✓	✓	✓		✓					✓	
History Based	✓		✓			✓		✓	✓							✓
Method Call Tree	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓				✓	✓
Concern Peers	✓	✓	✓	✓		✓		✓		✓					✓	✓
CBFA	✓	✓	✓	✓		✓		✓		✓			✓		✓	
Tracematch		✓		✓		✓		✓	✓	✓		✓	✓	✓	✓	✓
AspectJ		✓		✓		✓		✓		✓		✓	✓	✓	✓	✓
JAsCo		✓		✓		✓		✓	✓	✓		✓	✓	✓	✓	✓
DataFlow Pointcuts		✓		✓		✓		✓	✓	✓		✓	✓	✓	✓	✓
AspectQuery		✓		✓		✓		✓	✓						✓	✓
JPDD	✓		✓		✓		✓					✓	✓	✓	✓	
AspectOCL									✓	✓	✓				✓	
Theme/UML	✓		✓	✓					✓		✓				✓	

Fonte: Adaptado (traduzido) de [Wimmer *et al.* 2011]

- **Join Point Estrutural** representa elementos estruturais de uma linguagem onde a implementação de uma característica transversal pode ser introduzida. Além disso, os join point estruturais podem ser estáticos ou dinâmicos. Os join points estáticos estruturais são entendidos como classes, os join points dinâmicos estruturais são entendidos como objetos.

- **Join Point Comportamental** representa elementos comportamentais de uma linguagem onde a modificação pode ser introduzida. Além disso, é feita uma distinção entre join points estáticos comportamentais (e.g., atividades) e join points dinâmicos comportamentais (por exemplo, ocorrências de mensagens).

- **Pointcut Simples** representa um conjunto de join points de um certo tipo (por exemplo, estrutural-estático), que são selecionados de acordo com um certo Método de Quantificação. Portanto, representa um meio para selecionar vários elementos, e.g., selecionar todas as operações de modificação para fins de registro de log.

- **Pointcut Composto** representa pointcuts que podem ser compostos de outros pointcuts, para fins de reuso, por meio de operadores lógicos, por exemplo, "AND", "OR", "NOT", para formar pointcuts compostos, e.g., isso permite combinar a seleção de todas as operações de modificação junto com todas as operações de recuperação. Desse modo, todos os filhos de um pointcut composto, ou seja, todos os join points selecionados, referem-se ao mesmo modelo.

- **Método de Quantificação** descreve um mecanismo, por exemplo, um predicado para selecionar a partir dos join points potenciais, i.e., aqueles que deveriam estar disponíveis para a introdução de uma adaptação, e.g., selecione todas as classes abstratas ou todos os métodos que são implementados em uma determinada classe. O método de quantificação corresponde ao que é denominado designador de pointcut no AspectJ [Kiczales Erik Hilsdale 2001]. A seleção dos join points podem ser especificada de forma declarativa, imperativa ou simplesmente por enumeração.

- **Posição Relativa** pode fornecer mais informações sobre onde as características transversais devem ser introduzidas. Representa um tipo de especificação de localização. Esta informação adicional é necessária em alguns casos quando a seleção de join points apenas por pointcuts não é suficiente.

- **Abstração** significa que os join points podem ou não ter sido identificados, i.e., pode acontecer do modelo apenas especificar o fato de que existem join points em um determinado módulo, mas não exatamente onde. No entanto, as linguagens de modelagem que fornecem um baixo nível de abstração permitem especificar os join points de uma forma exata. Portanto, esse critério é avaliado como alto ou baixo.

3.1 UMA VISÃO GERAL DO ESTADO DA ARTE ATUAL DAS CARACTERÍSTICAS TRANSVERSAIS

As características transversais na engenharia de software para sistemas embarcados vêm crescendo em importância devido ao crescente desenvolvimento de softwares cada vez mais complexos para dispositivos conectados à internet, i.e., internet das coisas (Em inglês, *Internet of Things - IoT*). Atualmente, os chamados sistemas ciber-físicos (Em inglês, *Cyber-Physical Systems - CPS*) vem sendo usados em diferentes áreas, e.g., área automotiva, aviônica, medição inteligente de energia, cidades inteligentes entre outras, com objetivo de melhorar a vida das

pessoas e da sociedade. Com a crescente complexidade associada ao desenvolvimento de sistemas ciber-físicos, houve também um aumento significativo do número de requisitos necessários para o seu desenvolvimento. Na Engenharia de Software, existem os requisitos funcionais (RF) e requisitos não-funcionais (RFN). Os RNFs são fundamentais para restrições críticas de um sistema ciber-físico, ou sistema embarcado (e.g., restrições de segurança, temporização, monitoramento de recursos, concorrência, dependência, consumo de energia, entre outros).

Atualmente, na literatura, o interesse sobre esse assunto, vem aumentando significativamente. Existem vários trabalhos relacionados ao tratamento das características transversais na engenharia de software para sistemas embarcados usando abordagens e técnicas diversas. Em [Younas *et al.* 2020], os autores apresentam uma abordagem que manipula a semântica textual dos requisitos funcionais para identificar os requisitos não-funcionais e suas características transversais. Em [Zhou *et al.* 2020], é proposto um método de avaliação quantitativo referente a arquitetura de requisitos-não funcionais e suas características transversais. O método se baseia nas diretrizes da abordagem de requisitos não-funcionais orientados a metas (Em inglês, *Goal-oriented NFR*) aplicado através do uso da ArchiMate [Wierda 2014] que é uma linguagem de modelagem com foco em arquitetura empresarial (Em inglês, *Enterprise Architecture*). Em [Bowers *et al.* 2020], os autores apresentam uma nova técnica para lidar com características transversais e otimizar o cumprimento dos requisitos não-funcionais chamada Proventia. Os autores afirmam que os requisitos não-funcionais introduzem em um sistema características transversais que são difíceis de prever pois o cumprimento de um requisito não-funcional pode impactar o cumprimento de um ou mais requisitos funcionais e vice-versa. Em [DeVries e Cheng 2019], os autores apresentam o método chamado *Soter*, responsável pela modelagem e análise de caracteríticas transversais associadas a requisitos não-funcionais através da Engenharia de Requisitos Orientada a Metas (Em inglês, *Goal-Oriented Requirement Engineering - GORE*) [Yu *et al.* 2004]. Os autores afirmam que todos os requisitos não-funcionais possuem uma natureza transversal devido a especificar um atributo de qualidade de um requisito funcional. (e.g., custo, performance, segurança entre outros.) Em [Hannousse 2019], os autores propõem um novo processo de desenvolvimento de componentes de software com suporte a características transversais usando a orientação a aspectos (Em inglês, *Aspect-oriented Software Development* - AOSD). Os autores afirmam que o processo tradicional de desenvolvimento de software é falho quanto ao tratamento de características transversais relacionado ao desenvolvimento de componentes de software. Em [Roque *et al.* 2019], é apresentada uma avaliação de especificação

de requisitos para dois sistemas críticos de controle de tempo-real que dependem de protocolos de comunicação industriais. Também é proposto o uso da Modelagem Orientada a Aspectos (Em inglês, *Aspect-oriented modeling*) para mapear comportamento de falhas como requisitos não-funcionais nas fases de projeto. Os autores afirmam que falhas que afetam o desempenho de protocolos de comunicação geralmente tem efeitos transversais, e, portanto, a especificação de requisitos não-funcionais usando técnicas de modelagem são muito importantes para um projeto de um sistema embarcado de tempo-real robusto pois é possível capturar as características transversais na fase de projetos. Em [Zubcoff *et al.* 2019], os autores apresentam um experimento que se concentra em otimizar os requisitos não-funcionais ao selecionar os requisitos funcionais para implementação. Para esse propósito, os autores escolheram a GORE e fizeram comparações das notações envolvidas no experimento.

Portanto, a literatura da Engenharia de Software para Sistemas Embarcados vem demonstrando um interesse quanto ao impacto causado pelos RNFs que frequentemente possuem características transversais inerentes a sua execução. Incluir requisitos não-funcionais em uma especificação de requisitos para um sistema embarcado pode ser um grande desafio. Uma vez que os RNFs são frequentemente transversais, eles devem ser aplicados meticulosamente. Para sistemas embarcados de tempo-real, garantir que os requisitos não-funcionais sejam satisfeitos (e.g., requisitos de temporização) é fundamental pois os RNFs podem introduzir características transversais que são difíceis de prever na fase de projetos, e.g., um controle de cruzeiro adaptativo (Em inglês, *Adaptive Cruise Control - ACC*) pode ter um requisito não-funcional de segurança para evitar colisão com outros carros dependendo de sua proximidade, e da mesma forma pode ter um requisito funcional para manter a velocidade desejada do motorista. Dependendo de como os requisitos foram especificados, o requisito de segurança não-funcional para evitar uma colisão, pode ser violado quando a velocidade desejada do motorista é mantida (e.g., quando um obstáculo inesperado se move na frente do carro). [Zhou *et al.* 2020].

Sendo assim, pode-se afirmar que uma especificação clara dos requisitos não-funcionais é muito importante para um projeto de sistemas embarcados e pode contribuir de forma substancial para a melhoria do sistema quanto ao desempenho, confiabilidade, segurança entre outros. Dessa forma, cada vez mais é possível identificar uma natureza transversal (Em inglês, *cross-cutting nature*) em RNFs no desenvolvimento de um sistema embarcado que não se encaixa na decomposição padrão expressa por uma especificação de um requisito funcional.

Tabela 2 – Quantidade de citações por artigo

Artigo	Número de Citações	Técnica Associada
An overview of AspectJ	4278	AspectJ
Adding trace matching with free variables to AspectJ	538	Tracematch
Theme: An Approach for Aspect-Oriented Analysis and Design	413	Theme/UML
A UML-based aspect-oriented design notation for AspectJ	269	JPDD
On the use of Clone Detection for Identify Crosscutting Concern Code	223	Clone Detection
Aspect Mining Using Event Traces	199	Execution Patterns
Stateful Aspects in JAsCo	100	JAsCo
Automated Aspect Recommendation through Clustering-Based Fan-in Analysis	43	CBFA
Aspect Recommendation for Evolving Software	26	Concern Peers
Aspect Mining Using Method Call Tree	15	Method Call Tree
Mining Crosscutting Concerns through Random Walks	15	Random Walks
Identifying Cross-Cutting Concerns Using Software Repository	12	History-Based
AspectQuery: A Method for Identification of Crosscutting Concerns in the Requirement Phase	4	AspectQuery
AspectOCL: using aspects to ease maintenance of evolving constraint specification	4	AspectOCL

Fonte: o autor

3.2 UMA REVISÃO DO ESTADO DA ARTE QUANTO A ESPECIFICAÇÃO DE SELEÇÃO DE POCT

Dentre as 16 técnicas apresentadas na revisão sistemática foram selecionadas para análise do estado da arte 5 técnicas usadas para a especificação de seleção de POCT. A escolha das 5 notações se deu a partir da sua popularidade em termos de quantidade de citações, exceto para a notação AspectOCL que foi selecionada devido a técnica usar a OCL para especificar restrições transversais. A Tabela 2 apresenta a quantidade de citações das técnicas apresentadas na revisão sistemática, o qual escolhemos as 4 notações mais populares além da AspectOCL. Para melhor descrever as 5 técnicas selecionadas, esta dissertação faz uso do mesmo exemplo apresentado no capítulo 2 quanto a especificação relacionada ao desenvolvimento de um sistema embarcado de tempo-real de uma cadeira de rodas automatizada. O objetivo é mostrar os diversos aspectos das abordagens através da aplicação de um único exemplo. A seguir, apresentamos a especificação de seleção de POCT para cada técnica selecionada.

3.2.1 AspectJ

A primeira técnica é a AspectJ [Kiczales *et al.* 2001]. AspectJ é uma linguagem de programação que faz extensão da linguagem Java para definir um mecanismo dinâmico de intercepção transversal na execução de um programa. Esse mecanismo é suportado por construções denominadas pontos de junção (em inglês, *join points*) que representam os POCT. AspectJ é a base de muitos trabalhos que visam lidar com características transversais e especificar seleções de POCT. A AspectJ dá suporte para dois tipos de especificação de características transversais. O primeiro torna possível definir uma implementação adicional para executar em certos pontos na execução do programa. (Chama-se isso de mecanismo de corte transversal dinâmico). O segundo tipo permite definir novas operações em tipos existentes (Chama-se isso de corte transversal estático porque afeta a assinatura de tipo estático do programa). Esta dissertação apresenta apenas o corte transversal dinâmico da AspectJ devido a natureza comportamental do POCT relacionado ao controle de tarefas periódicas para o exemplo da cadeira de rodas automatizada. (i.e., não há uma mudança estrutural necessária e sim comportamental.)

A AspectJ é baseada em um conjunto de construções. Os pontos de junção (Em inglês *join points*) são pontos bem definidos na execução de um programa; os pontos de corte (Em inglês *pointcuts*) são um meio de se referir a coleções de pontos de junção; os conselhos (Em inglês *advices*) são construções semelhantes aos métodos usados para definir comportamento adicional em pontos de junção; os aspectos (Em inglês, *aspects*) são unidades de implementação transversal modular, composta de pontos de corte, conselhos e declarações comuns de membros Java. A Figura 18 apresenta um exemplo de uma aplicação da AspectJ na especificação da seleção de POCT associada ao RNF de temporização, i.e., *Timing*, apresentada no capítulo 2 referentes ao exemplo da cadeiras de rodas automatizada. Esse RNF possui um aspecto de controle de tarefas periódicas (*PeriodicTiming*), que especifica a seleção de POCT que tem como responsabilidade o controle da frequência das tarefas periódicas.

Nas linhas 2-3 é definido o pointcut pcFreqCtrl que se refere ao joinpoint *call(* *SchedulableResource.run(..))*. O elemento sintático *!within* significa que o joinpoint (i.e, o método a ser executado) não está dentro do aspecto, mas sim fora. É uma garantia de evitar referência cíclica. A linha 4 especifica que a implementação do tratamento da característica transversal deve ser executada após a chamada ao pointcut pcFreqCtrl.

Embora a AspectJ seja eficiente no tratamento de POCT, não é possível fazer uma seleção de POCT de uma forma desacoplada do próprio aspecto, i.e., não é possível selecionar apenas os POCT, mas sim trabalhar dentro de uma estrutura de código que a partir da combinação de POCT especificados executam o tratamento das características transversais. Isso seria mais uma limitação de não poder selecionar os POCT e sim acioná-los em tempo de execução. Uma Outro limitação encontrada é quanto a sua semântica que não é trivial, pois é necessário um esforço adicional para aprender a sintaxe, os símbolos e as regras para o uso correto da AspectJ.

Figura 18 – Exemplo de AspectJ

```
1 aspect PeriodicTiming {
2     pointcut pcFreqCtrl(): call(* *SchedulableResource.run(..)) && !
          within(PeriodicTiming);
3
4     after(): pcFreqCtrl() {
5         //Crosscutting Concerns Implementation
6     }
7 }
```

Fonte: o autor

3.2.2 Tracematch

A segunda técnica selecionada é a *Tracematch (TM)* [Allan *et al.* 2005]. A TM é uma abordagem textual que implementa expressões regulares sobre rastros de execução de programas baseados em um alfabeto de símbolos. Uma vez que haja um evento que faça a expressão regular corresponder ao rastro de execução, ele deve acionar o módulo de implementação do tratamento do POCT a ser executado. A TM foi totalmente projetada e implementada como uma extensão contínua da AspectJ. O uso da AspectJ permite que o engenheiro encapsule o tratamento das características transversais que é executado quando o programa em tempo de execução atende critérios específicos (i.e., seleção de POCT). A TM estende essa funcionalidade da AspectJ de uma forma que o histórico de execução do programa (Em inglês, *trace*) pode ser usado na especificação dos pontos onde o tratamento das características transversais deve ser executado. Um TM define um padrão e um bloco de código a ser executado quando o histórico de execução atual corresponde a esse padrão. Cada tracematch consiste em três partes: a declaração de um ou mais símbolos, um padrão envolvendo esses símbolos, e um pedaço de código a ser executado. Uma correspondência ocorre quando um sufixo do histórico de execução do programa atual corresponde a uma palavra na expressão regular especificada. A Figura 19 apresenta um exemplo

de uma aplicação da Tracematch na especificação da seleção de POCT associada ao RNF de temporização (i.e., *Timing*) apresentada no capítulo 2 referente ao exemplo da cadeiras de rodas automatizada. Esse RNF possui um aspecto de controle de tarefas periódicas (*PeriodicTiming*), que especifica a seleção de POCT que tem como responsabilidade o controle da frequência das tarefas periódicas.

A linha 1 é o cabeçalho tracematch. Em seguida, nas linhas 2-3, é definido um símbolo. O símbolo pcFreqCtrl corresponde a eventos de saída em join points que correspondem ao pointcut *call(* *SchedulableResource.run(..))*. A expressão regular na linha 4 especifica que o tratamento da característica transversal é acionado em rastreamentos que terminam com uma chamada para pcFreqCtrl. Finalmente, a linha 6 se refere a implementação do tratamento da característica transversal a ser executado. Ao combinar um padrão com o histórico de execução, quaisquer eventos no trace que não são declarados como símbolos no tracematch são ignorados, e apenas eventos declarados como símbolos podem acionar a comparação. Portanto, este tracematch corresponde a qualquer saída de uma chamada para pcFreqCtrl que é o sym referente a especificação de POCT referentes a característica transversal de controle de frequencia de temporização de um sistema embarcado de tempo-real (nesse exemplo, para cadeira de rodas automatizada).

Embora a Tracematch inove no tratamento de variáveis livres em padrões de rastreamento, seu uso não é trivial, semelhante a AspectJ, que precisa de um esforço adicional significativo para entender suas propriedades físicas (sintaxe) e aplicá-las adequadamente na especificação de POCT. Além disso, não é possível fazer uma seleção de POCT de uma forma desacoplada do próprio tracematch, i.e., não é possível selecionar apenas os POCT, mas sim trabalhar dentro de uma estrutura de código que a partir da combinação de POCT especificados executam o tratamento das características transversais. Isso seria mais uma limitação de não poder selecionar os POCT e sim acioná-los em tempo de execução.

Figura 19 – Exemplo de Tracematch

```
1  tracematch(){
2   sym pcFreqCtrl after: call(* *SchedulableResource.run(..))
3
4   pcFreqCtrl
5   {
6    //Crosscutting Concerns Implementation
7   }
8  }
```

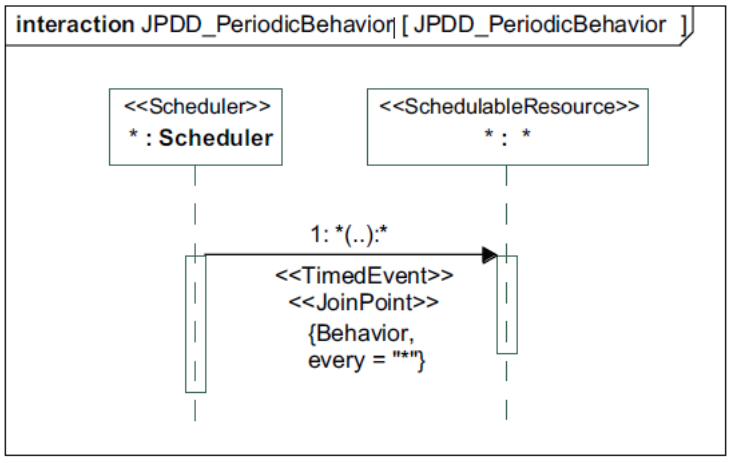
Fonte: o autor

3.2.3 Join Point Designation Diagram

A terceira técnica a ser apresentada é a *Join Point Designation Diagram* (JPDD) [Stein *et al.* 2006]. A JPDD é uma abordagem de modelagem para visualizar seleções de POCT com base na semântica da UML. Os projetistas criam diagramas dos eventos do sistema, símbolos de objetos e arestas de fluxo de objetos através da adoção de símbolos de ação que representam um fluxo de execução para atingir um determinado POCT. Da mesma forma que a UML, a JPDD é usada para especificar, construir, documentar e visualizar graficamente artefatos de software de uma maneira independente da linguagem de programação. Além disso, a JPDD faz uso dos símbolos amplamente conhecidos da UML, porém não de forma tradicional, o que pode não facilitar sua compreensão.

A Figura 20 apresenta um exemplo de uma aplicação da JPDD na especificação da seleção de POCT associada ao RNF de temporização (i.e., *Timing*) apresentada no capítulo 2 referente ao exemplo da cadeiras de rodas automatizada. Esse RNF possui um aspecto de controle de tarefas periódicas (*PeriodicTiming*), que especifica a seleção de POCT que tem como responsabilidade o controle da frequência das tarefas periódicas.

Figura 20 – Exemplo de JPDD



Fonte: [Wehrmeister 2009]

No lado esquerdo do diagrama de sequência temos uma *lifeline* que expressa o objeto escalonador (i.e., *Scheduler*). O Estereótipo <<JoinPoint>> expressa o ponto ou pontos que a especificação deseja selecionar, i.e., todos os métodos anotados com o estereótipo <<TimeEvent>> de qualquer nome, de todas as classes anotadas com o estereótipo <<SchedulableResource>>. O símbolo {Behavior, every = "*"} se refere a uma restrição de seleção de todos os comportamentos dos métodos na UML.

Embora a JPDD permita especificar seleções de POCT de maneira uniforme e independente da linguagem de programação, e desacoplada do AOSD, o uso dos símbolos da UML na JPDD não é de fácil compreensão pois difere bastante do uso tradicional dos mesmos elementos. Além disso, os engenheiros precisam ter uma grande experiência em AOSD para projetar os JPDDs adequadamente no contexto de requisitos funcionais e não-funcionais.

3.2.4 AspectOCL

A quarta técnica a ser apresentada é a AspectOCL [Khan *et al.* 2019]. AspectOCL é uma extensão da Linguagem de Restrições a Objetos (OCL) [Cabot e Gogolla 2012] que traz benefícios para especificação de seleção de restrições transversais.

As restrições desempenham um papel importante no desenvolvimento de software embarcado. Os sistemas embarcados comumente exibem comportamentos de restrições transversais em artefatos de software. O AOSD é uma abordagem bem estabelecida para lidar com características transversais e tem sido usada com sucesso para linguagens de programação, porém não trata as restrições transversais no desenvolvimento de software embarcado. A presença de restrições transversais torna difícil manter as restrições definidas nos modelos de sistemas industriais de grande escala. A AspectOCL permite a especificação dos aspectos e a entrelaça nas restrições da OCL fornecendo uma sintaxe abstrata e concreta da linguagem. Segundo [Khan *et al.* 2019], a AspectOCL reduz o esforço de manutenção no desenvolvimento de software em até 55%, de acordo com o estudo de caso apresentado pelos autores.

A Figura 21 apresenta um exemplo de uma aplicação da AspectOCL na especificação da seleção de POCT associada ao RNF de temporização (i.e., *Timing*) apresentada no capítulo 2 referente ao exemplo da cadeiras de rodas automatizada. Esse RNF possui um aspecto de controle de tarefas periódicas (*PeriodicTiming*), que especifica a seleção de POCT que tem como responsabilidade o controle da frequência das tarefas periódicas.

Nas linhas 1-4 é definido o mapeamento dos atributos a serem usados no aspecto. Na linha 8 se importa esses atributos, no caso desse exemplo, se tem apenas a classe SchedulableResource. Na linha 10 é criado o pointcut pcFreqCtrl que faz referência a seleção de POCT expressa nas linhas 13-18 que traz todos os todos os métodos anotados com o estereótipo <<TimeEvent>> de qualquer nome, de todas as classes anotadas com o estereótipo <<SchedulableResource>>, i.e., todos os pontos de ocorrências de características transversais do requisito não-funcional de temporização usados no exemplo da cadeira de rodas automatizada.

Figura 21 – Exemplo de AspectOCL

```
1  mapping mapTiminingAttributes
2     {
3         let T: {SchedulableResource}
4     }
5
6     aspect PeriodicTiming
7     {
8         import_mapping mapPeriodicTiming
9
10        pointcut pcFreqCtrl
11        context T :
12        inv:
13           self.allOwnedElements()
14        ->select(s | s.getAppliedStereotype('SchedulableResource'))
15        ->select(o | o.getOperations())
16        ->select(t | t.getAppliedStereotype('TimedEvent'))
17        ->select(m | m.getMethods())
18        ->select(b | b.getBehavior())
19
20    }
```

Fonte: o autor

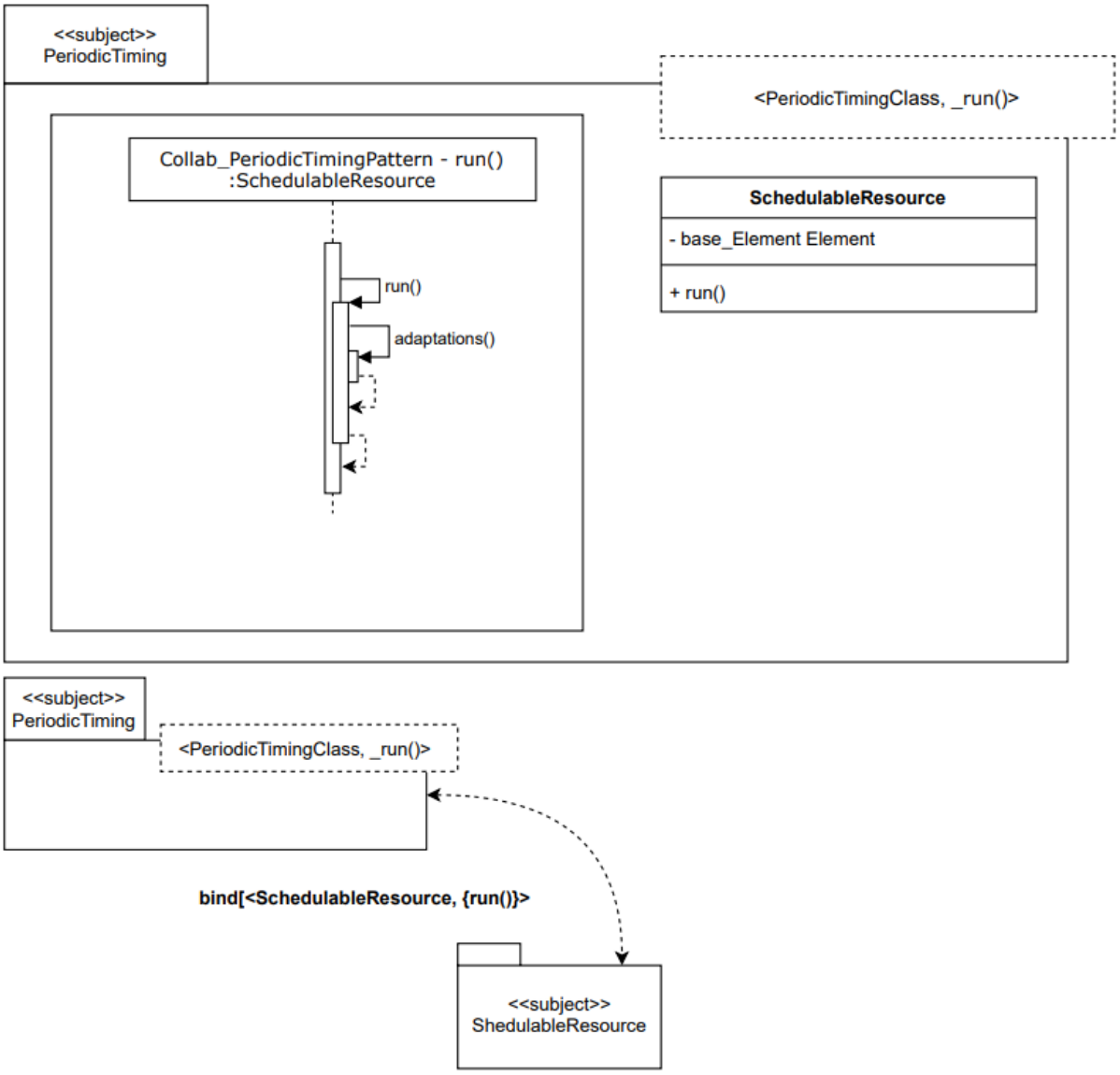
Apesar de ter sido apresentado que o uso do AspectOCL pode reduzir o número de restrições nos modelos, existem algumas limitações apresentadas pelos autores: (i) a especificação da restrição é dissociada dos elementos da modelagem, (ii) outra limitação é a falta de semântica formal, (iii) existe também a limitação na criação de aspectos compostos e (iv) a linguagem não possui avaliação empírica.

3.2.5 Theme/UML

A quinta e última técnica a ser apresentada se chama Theme/UML [Clarke 2002, Baniassad e Clarke 2004]. A Theme/UML é uma abordagem baseada em UML que suporta a separação simétrica de características transversais em vez de separação assimétrica. A última é suportada na maioria das abordagens do AOSD. Fragmentos de comportamento e/ou estruturas que representam uma característica transversal são encapsulados em um único tema. A integração entre temas é definida como uma ligação que indica quais elementos de um tema são afetados por elementos de outros temas. De alguma forma, isso é semelhante à especificação dos pontos de junção e a relação entre elementos funcionais afetados por aspectos [Kiczales *et al.* 2001]. No entanto, a especificação de como um tema afeta outros temas não é adequada devido à falta de escalabilidade, ou seja, em um sistema com uma grande quantidade de características transversais, a especificação do relacionamento de vínculo para expressar a integração desses temas dificulta na compreensão, manutenção e na evolução do modelo. A Theme/UML faz a

visualização dos relacionamentos entre os comportamentos em um documento de requisitos, identificando e isolando pontos de ocorrências de características transversais nos requisitos e modelando esses POCT usando uma notação visual. A Theme/UML fornece suporte para o desenvolvimento orientado a aspectos em dois níveis. No nível de requisitos, onde fornece visualizações do texto de especificação de requisitos e no nível de projetos, onde permite a um engenheiro modelar recursos e aspectos de um sistema e especificar como eles devem ser combinados.

Figura 22 – Exemplo de Theme/UML



Fonte: o autor

A Figura 22 apresenta um exemplo de uma aplicação da Theme/UML na especificação da seleção de POCT associada ao RNF de temporização (i.e., *Timing*) apresentada no capítulo 2 referente ao exemplo da cadeiras de rodas automatizada. Esse RNF possui um aspecto de

controle de tarefas periódicas (*PeriodicTiming*), que especifica a seleção de POCT que tem como responsabilidade o controle da frequência das tarefas periódicas.

No diagrama é possivel identificar uma *lifeline* que está dentro de um tema, i.e., um *subject* chamado *PeriodicTiming*. Esse tema especifica que após a chamada do método run(..) deve-se chamar as adaptações necessárias. A especificação de seleção de POCT se encontra mais abaixo, onde o tema *PeriodicTiming* é o produto da ligação entre a localização e seu tema transversal *bind{<SchedulableResource, run()>*);

No entanto, embora a Theme/UML faça uso da UML que é amplamente aceita na indústria, não há semântica precisa quando se trata da especificação de seleçao de POCT, uma vez que a UML foi criada para suportar o paradigma Orientado a Objetos, que é conhecido por apresentar problemas de modularidade no tratamento de questões transversais, i.e., o resultado é uma especificação bastante imprecisa das seleções de POCT.

3.3 DISCUSSÃO

De acordo com o que foi apresentado neste capítulo, o número de problemas em aberto ou deficiências encontradas nas técnicas apresentadas são altos.

As notações textuais (AspectJ, Tracematch e AspectOCL) apresentaram problemas comuns referente a não possibilidade de desacoplamento da especificação de seleção de POCT do AOSD, i.e., para especificar a seleção de POCT de forma textual é necessária a codificação e acoplamento de uma estrutura orientada a aspectos. Outro problema comum as notações textuais é quanto a dependência ao tempo de execução (Em inglês, *runtime*) de um programa para conseguir identificar os POCT, i.e., as notações textuais não especificam a seleção de POCT, apenas identificam em tempo de execução onde existem pontos de ocorrências de características transversais. Esse tipo de problema dificulta a modularização e o reuso gerando duplicidade de código e gerando mais elementos sintáticos na especificação.

Quanto as notações visuais (JPDD e Theme/UML), os problemas em comum encontrados são referentes a quantidade de elementos implícitos no modelo. Por mais que o propósito de uma notação visual seja em aumentar o nível de abstração de um conceito computacional, não significa que a especificação seja insuficiente quanto a eficácia cognitiva se duas propriedades de percepção, i.e., o fato de deixar elementos implícitos impactam de forma direta no entendimento do modelo visual, dependendo também de outros fatores, e.g., o uso de símbolos similares, o uso de símbolos que possuem mais de uma semântica, entre outros. Outro problema em comum

das notações visuais é quanto ao uso dos símbolos da UML divergir bastante do uso tradicional, gerando uma dificuldade na compreensão dos elementos, além do mais a UML foi criada para dar suporte ao paradigma de orientação a objetos.

Outras deficiências foram encontradas para ambas as abordagens (visual e textual), como a falta de formalismo representado pela fraca representação semântica, a falta de conhecimento do AOSD, a representação sintática complexa (a maioria das técnicas possui complexidade sintática semelhante), a falta de avaliação empírica entre outras apresentadas neste capítulo.

4 ESPECIFICAÇÃO DE PONTO DE OCORRENCIAS DE CARACTERÍSTICAS TRANSVERSAIS

Este capítulo tem por objetivo apresentar o desenvolvimento da técnica de especificação de seleção de Ponto de Ocorrências de Características Transversais (POCT), fundamentando-se em 5 princípios do framework PoN. Um dos objetivos dessa dissertação é propor uma nova técnica para especificar os POCT de uma forma mais compreensível que as técnicas usadas atualmente. A técnica proposta, chamada de Join Point Specification Diagram (JSD), se baseia fortemente em cinco princípios da Física das Notações (Clareza Semiótica, Discernimento de Percepção, Transparência Semântica, Expressividade Visual e Economia Gráfica), além de ter uma característica de permitir a expressão de forma textual e gráfica, baseada no princípio de Codificação Dupla que preza pelo uso de textos para complementar elementos gráficos [Moody 2009].

As seções a seguir discutem como usar a JSD para especificar a seleção de POCT inerentes a requisitos não-funcionais e funcionais no desenvolvimento de software para sistemas embarcados. Será apresentado um modelo de especificação de seleção de POCT através da JSD. Os POCT são relacionados ao RNF de temporização (i.e., *Timing*) apresentado no capítulo 2 referente ao exemplo da cadeiras de rodas automatizada. Esse RNF possui um aspecto de controle de tarefas periódicas (*PeriodicTiming*), que especifica a seleção de POCT que tem como responsabilidade o controle da frequência das tarefas periódicas. Esse RFN é usado como um aspecto no diagrama de visão geral de aspectos transversais (ACOD), proposta na abordagem AMoDE-RT apresentada também no capítulo 2. Sendo assim, demonstraremos os ganhos do uso da técnica JSD e a contribuição ao estado da arte quanto a solução das lacunas encontradas nas demais técnicas apresentadas.

4.1 SELECIONANDO PONTOS DE OCORRÊNCIAS DE CARACTERÍSTICAS TRANSVERSAIS (POCT) ATRAVÉS DA JSD

Embora na AMoDE-RT os aspectos sejam uma parte importante da especificação de requisitos não-funcionais, igualmente importante é a especificação de quais elementos do modelo são afetados pelas adaptações dos aspectos, i.e., os POCT. Originalmente, a AMoDE-RT usa a JPDD para seleção de POCT com base em três tipo de diagramas: (i) diagrama de fluxo de controle; (ii) diagrama de fluxo de dados; e (iii) diagrama de estado. O primeiro diagrama (i)

permite a seleção de POCT com base no fluxo de controle de execução representado em diagramas de sequência ou atividades, e.g., um JPDD seleciona ações realizadas no comportamento de um determinado método “a”, que é chamado dentro do comportamento de um método “b”. O segundo diagrama (ii) permite a seleção de POCT com base em dados passados de um método para outro, por exemplo, um JPDD seleciona um comportamento de um método que recebeu uma string começando com o caractere “s” como parâmetro. O último diagrama (iii) permite a seleção de POCT com base em seu estado explícito descrito em um diagrama de estado, e.g., um JPDD seleciona todos os objetos que estão em um “estado A”.

A seleção de POCT pode ser realizada de uma forma estática ou dinâmica, dentro do contexto estrutural ou comportamental do requisito não-funcional. Embora a JPDD possa especificar a seleção de POCT através da modelagem visual, existe um problema referente ao aumento de complexidade gerada por esses diagramas de acordo com as deficiências apresentadas no Capítulo 3, e.g., a Figura 20 apresenta uma especificação comportamental de seleção de POCT, usando a JPDD. Ao analisar o diagrama apresentado na Figura 20, é difícil inferir o que realmente significa essa especificação: temos uma lifeline de um objeto chamado Scheduler, com um estereótipo <<Scheduler>> que pode induzir um engenheiro iniciante a inferir que o elemento se trata de uma interface devido ao símbolo << >>. Além disso, temos um retângulo na horizontal que representa comportamento enviando uma mensagem para outro objeto que possui o estereótipo <<SchedulableResource>>, sem nome, e com 3 símbolos léxicos (*:*). Abaixo do símbolo de mensagem (i.e., a seta) tem-se dois estereótipos (<< TimedEvent>>) e uma restrição representada entre chaves ({Behavior, every = "*"}).

Um engenheiro júnior pode entender que essa especificação se refere a um diagrama de sequência, de um objeto mandando uma mensagem para outro objeto, sem reposta e com símbolos que são usados para representar interfaces na UML (além de propriedades de uma classe), i.e., fica difícil inferir que a especificação pretende fazer uma seleção de POCT. O fato da especificação de seleção de POCT ser bastante diferente do uso tradicional da UML é uma das deficiências encontradas na JPDD que implica uma dificuldade na compreensão da especificação.

Pelo fato de naturalmente o modelo visual ter informações implícitas, isso acaba afetando no entendimento e compreensão do diagrama, i.e., fica difícil inferir o que realmente significa o modelo devido a pouca eficácia do uso das propriedades perceptivas (sintaxe) dos elementos gráficos usados no modelo.

Devido a isso, a presente dissertação propõe uma técnica chamada de Join Point

Specification Diagram (JSD), que atenda cinco princípios da Física das Notações (Clareza Semiótica, Discernimento de Percepção, Transparência Semântica, Expressividade Visual e Economia Gráfica) através da avaliação da compreensão de forma indireta pela avaliação da eficácia cognitiva e das propriedades de percepção da notação. Além disso, essa técnica propõe usar expressões de forma textual e gráfica, baseada no princípio de Codificação Dupla. De acordo com a técnica de codificação dupla, o uso de símbolos textuais e símbolos gráficos juntos para transmitir uma informação é mais eficaz que o uso separado de um dos dois, contanto que o texto não seja para diferenciar um elemento gráfico de outro elemento, mas sim para completar o significado da especificação. Quando a informação é apresentada de forma verbal (símbolos textuais) e de forma visual (símbolos gráficos), representações dessas informações são arquivadas em lugares separados na memória e a conexão entre as mesmas são fortalecidas. [Moody 2009]

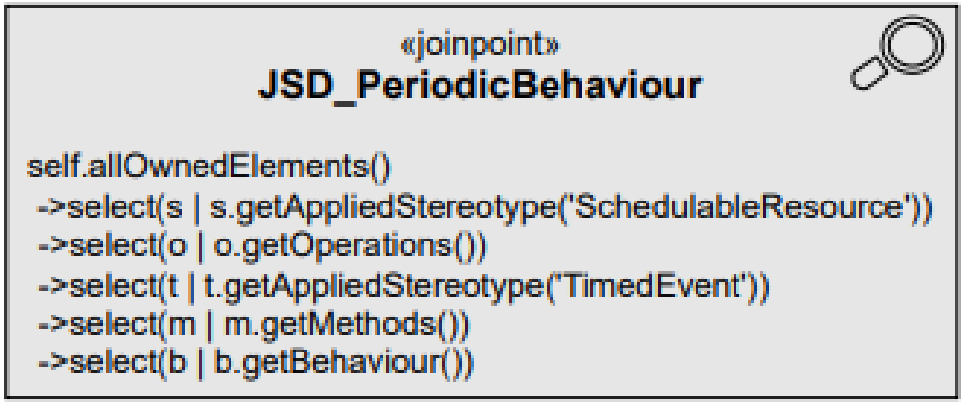
A Join Point Specification Diagram (JSD) é uma técnica de modelagem visual que possibilita o uso de elementos gráficos junto com elementos textuais que expressam de forma explícita os critérios de busca do modelo para a seleção dos POCT devido a algumas deficiências encontradas nos trabalhos da literatura, quanto ao fato de não existir uma semântica precisa nas notações visuais quando se trata de especificação de seleção de POCT, como também o fato do uso de símbolos da UML diferir bastante do uso tradicional.

A JSD usa a OCL [Cabot e Gogolla 2012] para descrever os critérios de seleção dos POCT no modelo, visando aplicar elementos textuais que enfatizem melhor o que a especificação estabelece. A Figura 23 apresenta um modelo usando a JSD de como é a especificação de seleção de POCT de uma forma explícita usando a linguagem OCL para expressar os critérios de seleção dos elementos do modelo UML.

A JSD apresentada nas Figura 23 atende os cinco princípios do PoN usados na presente dissertação. É possível verificar que o princípio da *Clareza Semiótica* é atendido devido a não haver redundância de símbolos, como também não haver sobrecarga de símbolos (e.g., apenas um retângulo). Isso também é perceptível através da análise da expressão lógica em OCL no qual apresenta a característica de um sistema notacional onde cada símbolo, seja ele léxico ou sintático apresenta um e apenas um significado associado.

O princípio do *Discernimento de Percepção* também é atendido pois o distanciamento visual entre os elementos sintáticos se apresenta de forma que para a maioria dos elementos sintáticos usados existem outros elementos diferentes (e.g., o estereótipo <<joinpoint>>, sendo único dentro da especificação), apesar de existirem elementos sintáticos iguais descrevendo a

Figura 23 – JSD para seleção de elementos comportamentais



Fonte: o autor

seleção de elementos na UML (e.g., *select(..)*).

Outro princípio atendido da JSD é o princípio da *Transparência Semântica* onde a técnica diminui a possibilidade de criação de elementos perversos, i.e., elementos que sugerem um significado contrário a sua semântica, como também permite a criação de elementos semanticamente imediatos (e.g., símbolo de lupa significando uma busca por pontos de ocorrências de características transversais).

A *Expressividade Visual* também é mais um princípio atendido na JSD devido ao uso de mais de uma variável de comunicação, e.g., o uso de cor para distinguir especificações estruturais (branco) e comportamentais (cinza); o uso da forma (e.g., retângulo); o tamanho, que aumenta ou diminui de acordo com o nível de complexidade da expressão de seleção de POCT; e a textura com linhas mais grossas de uma coloração intensa quando se trata de seleção de elementos passivos na UML (Classes ou Objetos).

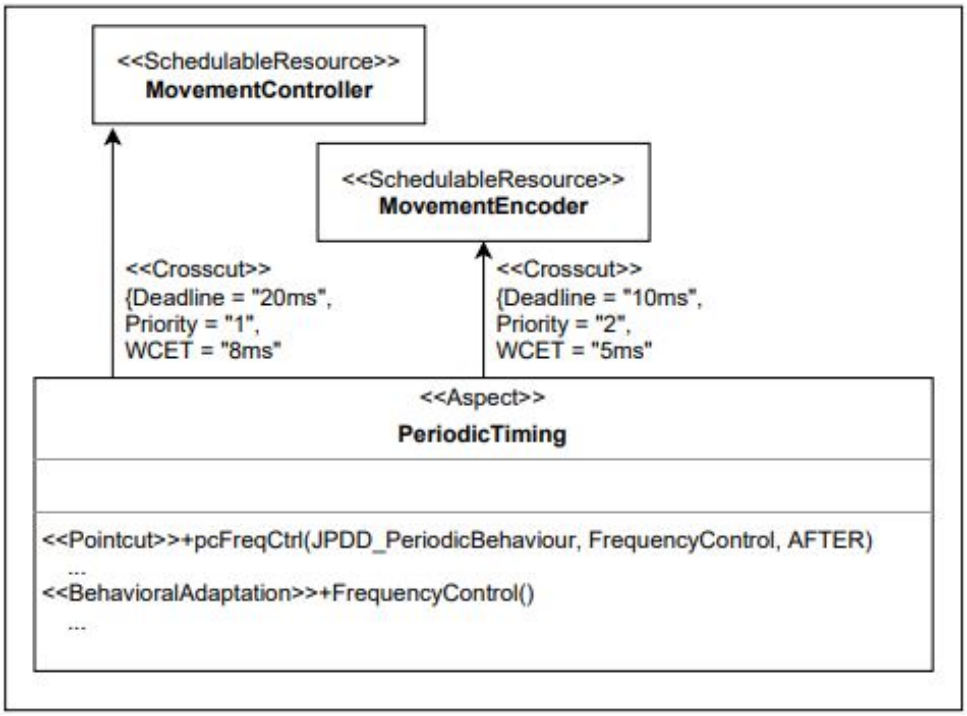
O princípio de *Economia Gráfica* também é atendido devido ao baixo número de elementos gráficos (e.g., um retângulo e uma lupa) e pelo fato de fazer uso do princípio de Codificação Dupla para completar e expressar os critérios de seleção através da OCL que, por mais que possa ser um elemento sintático por analogia, é uma expressão textual e não mais um elemento gráfico criado que poderia afetar um limite cognitivo de compreensão de acordo com o número de elementos criados e categorias visuais que podem ser efetivamente reconhecidas. [Moody 2009].

4.2 DIAGRAMA DE VISÃO GERAL DE ASPECTOS TRANSVERSAIS

WEHRMEISTER (2009), propôs o chamado *Aspects Crosscutting Overview Diagram* - (ACOD), que tem por objetivo representar aspectos dentro da UML pois ela não fornece nenhum elemento de metamodelo ou construção gráfica para representar aspectos. ACOD é baseado nos conceitos apresentados por Stein et al. (2002) e Schauerhuber et al. (2006), e mostra aspectos afetando ou não elementos funcionais. Existem duas versões do ACOD com diferentes níveis de detalhe: (i) visão geral do ACOD apresenta todos os aspectos que afetam as classes sem descrever detalhes sobre as informações dos aspectos; e (ii) ACOD detalhado descreve todos os aspectos especificados no modelo UML, juntamente com seus *advices* e *joinpoints*, e todas as classes que recebem novas informações de aspectos. ACOD detalhado é a principal fonte de informação para especificação de aspectos. A Figura 25 mostra os aspectos que são representados como classes decoradas com o estereótipo <<Aspect>>.

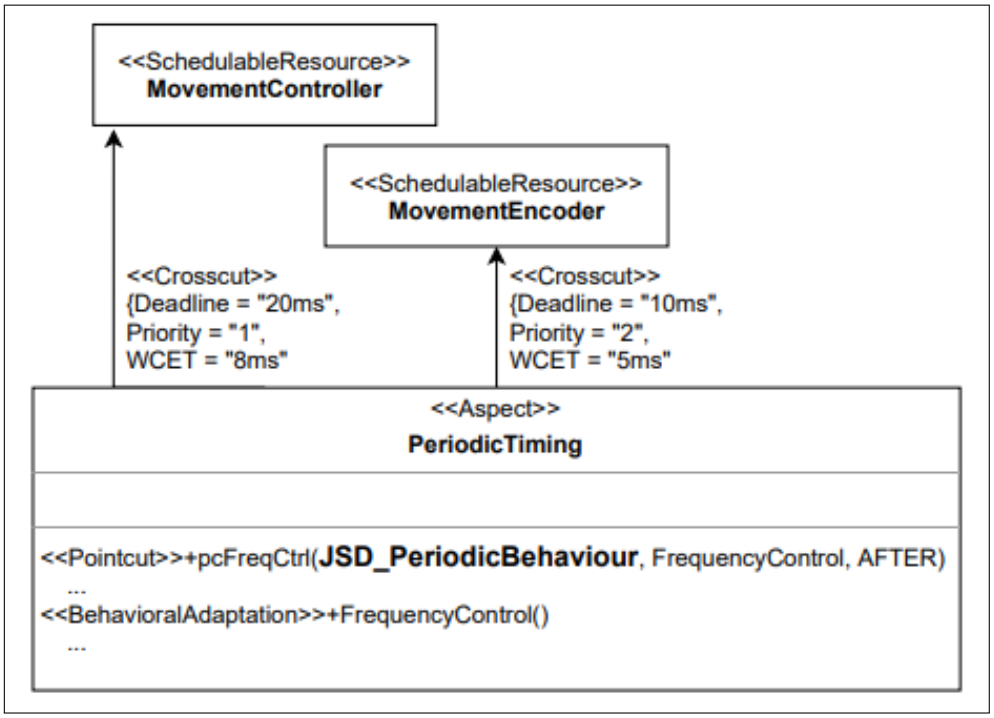
Para aplicação da JSD dentro do ACOD que por sua vez é usado na AMoDE-RT basta referenciar a chamada do próprio join point que está dentro do diagrama JSD. A Figura 25 apresenta um exemplo da mudança que é feita no ACOD, e.g., ao invés do pointcut do aspecto *PeriodicTiming* fazer referência ao diagrama da JPDD como é apresentado na Figura 24, será feita a referência ao POCT JSD_PeriodicBehavior que é um join point que faz parte do JSD comportamental explicado na Figura 23. Dessa forma, a referência a um JPDD é substituída pela referência a um dos *join points* especificados dentro de um JSD.

Figura 24 – Especificação de Aspectos usando a JPDD



Fonte: o autor

Figura 25 – Especificação de Aspectos usando a JSD



Fonte: o autor

4.3 DISCUSSÃO

O objetivo principal desse Capítulo foi apresentar a técnica proposta, a JSD, e também como é feita a especificação de seleção de POCT através da JPDD na AMoDE-RT. A ideia da criação de uma técnica de especificação de seleção de POCT mais simples para entender, se originou a partir da aplicação da JPDD na AMoDE-RT. A possibilidade de uma compreensão ambígua dos modelos de seleção de POCT através da JPDD foi a hipótese original do presente trabalho.

Sendo assim, esse trabalho propõe trocar os pontos onde são aplicados a JPDD para seleção de POCT e aplicar a JSD. O fato de a JSD ter sido criada sob a ótica da Física das Notações e tendo como princípio importante a aplicação da codificação dupla (i.e., uso da OCL para especificação dos critérios de seleção) permite uma integração sutil com o ACOD devido a compatibilidade com a UML. Outro ponto positivo quanto a troca da JPDD pela JSD é que, na JPDD, a especificação de seleção de POCT era de um para o diagrama, assim, para cada POCT era necessário um diagrama diferente. Por outro lado, na JSD é dada a liberdade de agrupar a especificação de seleção de POCT da forma que o engenheiro desejar, pois, existe a possibilidade de criar um diagrama para um ou mais POCT. Isso permite não só o agrupamento de POCT como também sua classificação com o objetivo de facilitar a compreensão. Conforme os resultados analisados no próximo capítulo, é possível verificar outras características positivas do uso da JSD.

5 MODELO DE QUALIDADE BASEADA NO PON

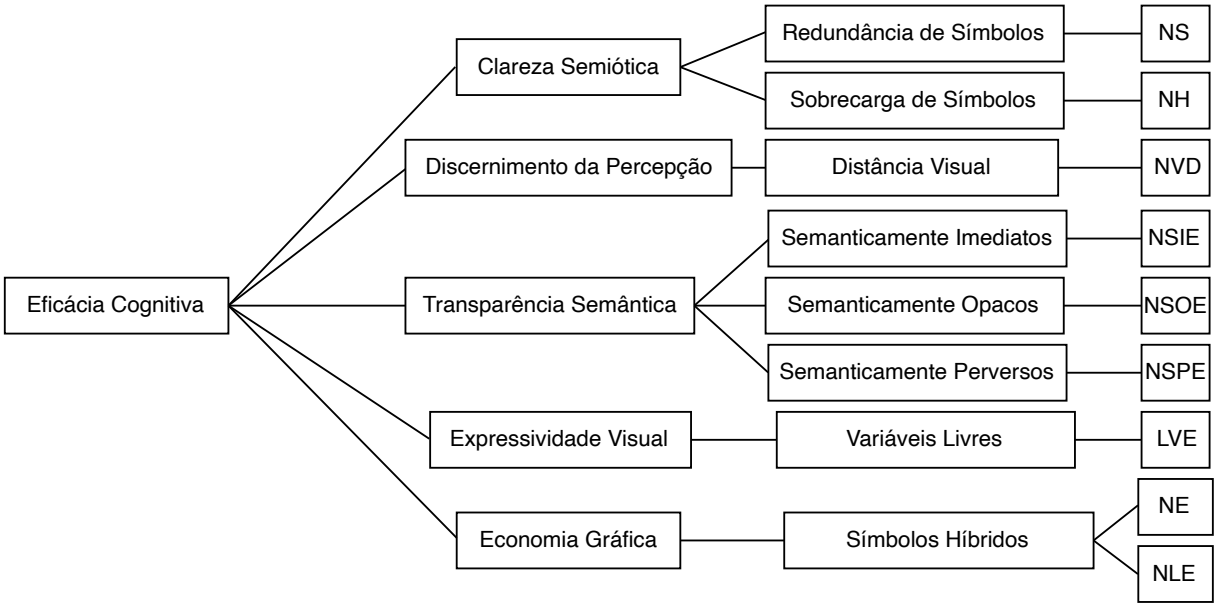
Atualmente, pode-se encontrar na literatura de engenharia de software várias métricas e modelos de qualidade que podem ser usados para extrair informações quantitativas de artefatos de software.

No entanto, até onde sabemos, não existe um modelo de avaliação ou de qualidade criado a partir da perspectiva das premissas do framework PoN que possibilita quantificar as propriedades perceptivas das notações gráficas e textuais, possibilitando avaliar os seus efeitos cognitivos.

Dessa forma, além de contribuir com a criação de uma nova técnica de especificação de seleção de POCT para sistemas embarcados, este trabalho propõe um novo modelo de qualidade baseado no framework PoN, contribuindo também quanto a avaliação da compreensão da especificação de seleção de POCT. Um conjunto de métricas denominado *Metrics Suite for PoN* (MS4PoN) foi criado.

O objetivo deste conjunto de métricas é quantificar as propriedades perceptivas de notações gráficas e textuais na especificação de seleções de POCT. As métricas do MS4PoN foram combinadas em um modelo de qualidade apresentado na Figura 26.

Figura 26 – Modelo de qualidade para avaliar as propriedades cognitivas



Fonte: o autor

Esse modelo de qualidade faz uso de 5 princípios da teoria associada ao framework PoN. O MS4PoN quantifica os elementos, símbolos e a sintaxe usados na aplicação de uma notação gráfica ou textual para a especificação da seleção de POCT. As métricas são:

- **Number of elements (NE):** essa métrica contabiliza o número de elementos contidos no modelo de seleção de POCT, independentemente da sua especificação ser em formato textual ou gráfico. Quanto maior o NE, mais complexo será a especificação

- **Number of Logical Expressions (NLE):** essa métrica é definida como a quantidade de expressões lógicas agnósticas[1] contidas na especificação da seleção de POCT. As expressões lógicas indicam os critérios de seleção implícitos ou explícitos para a seleção dos POCT. Nos experimentos, os valores NLE são iguais para cada um dos POCT, independente da notação que foi aplicada. O objetivo é mensurar a expressão lógica que representa o critério de seleção que deve ser especificado usando uma notação. O NLE permite gerar valores relativos viabilizando análises comparativas entre notações e fornecendo resultados mais precisos e homogêneos.

- **Number of Visual Distance (NVD):** essa métrica quantifica o número de elementos diferentes que são especificados na seleção de POCT. Em geral, quanto maior o número de elementos diferentes disponíveis na notação, maior é o NVD e, consequentemente, a especificação de seleção de POCT é compreendida com maior precisão.

- **Number of Semantically Immediate Elements (NSIE):** essa métrica determina o número total de elementos semanticamente imediatos especificados na seleção de POCT. Um elemento é semanticamente imediato se for fácil inferir seu significado a partir de sua aparência ou mnemônico. Quanto maior o NSIE, mais fácil será de entender a especificação do POCT em uma determinada notação. Um elemento ou símbolo semanticamente imediato deve ter em sua natureza uma associação com: (i) *similaridades funcionais*, por exemplo, o método elemento “before()” no AspectJ usa um termo que expressa o conceito de fazer algo antes (a palavra “before”) trazendo uma similaridade funcional; ou (ii) *metáforas*, por exemplo, um desenho de um boneco pode representar uma pessoa nos diagramas de casos de uso da UML; ou (iii) quaisquer *associações culturais*, por exemplo, um grande “X” em um diagrama de sequência da UML pode significar uma comunicação fechada ou interrompida.

[1] i.e., expressões lógicas abstratas e independentes da notação ou modelo de seleção de POCT

- **Number of Semantically Opaque Elements (NSOE):** essa métrica contabiliza o número de elementos semanticamente opacos especificados na seleção de POCT. Um elemento é semanticamente opaco (ou convencionado) se houver uma convenção puramente arbitrária entre a sua aparência e o seu significado. Quanto maior for o NSOE, mais elementos neutros ou convencionados são usados na especificação. Por exemplo o uso excessivo de retângulos nos diagramas da notação gráfica Theme/UML corresponde a um uso excessivo de elementos convencionados que, por sua vez, são puramente arbitrários quanto aos seus significados. NSOE representa o ponto zero na escala entre o NSIE e o NSPE, pois não aumenta a percepção cognitiva, mas também não sugere erroneamente um significado diferente de sua aparência, i.e., é um elemento convencionado que deve ser aprendido [Moody 2009].

- **Number of Semantically Perverse Elements (NSPE):** essa métrica quantifica o número de elementos semanticamente perversos especificados na seleção de POCT. Um elemento pode ser semanticamente perverso (ou ter falso mnemônico), quando o seu significado é oposto à sua aparência. Quanto maior o NSPE, pior será o entendimento da especificação. Um elemento ou símbolo semanticamente perverso possui em sua natureza algumas propriedades léxicas associadas a mnemônicos falsos que podem sugerir um significado totalmente diferente de sua aparência para um leitor iniciante. Algumas convenções da UML apresentam essa propriedade indesejável, e.g., o símbolo “merge” de pacotes da UML [Moody 2009].

- **Level of Expressiveness (LVE):** essa métrica determina o número de variáveis visuais usadas na notação para especificar a seleção de POCT. A LVE contabiliza a quantidade usada de variáveis de comunicação em notações visuais, e.g., forma, textura, cor etc. Conforme [Moody 2009], os valores de LVE variam de zero a oito. Portanto, quanto maior for o LVE, mais a notação aproveita a vantagem de ser uma notação gráfica. Quanto menor o LVE, mais próximo de uma notação textual é a notação visual.

- **Number of Synographs (NS):** essa métrica é definida como o número total de elementos sinógrafos especificados na seleção de POCT. Elementos sinógrafos aparecem quando há uma redundância no uso de símbolos ou elementos para representar a mesma construção semântica, e.g., interfaces na UML podem ser representadas de duas formas diferentes. Quanto maior o NS, pior é a compreensão da especificação. Uma notação gráfica satisfaz

os requisitos de um sistema de notação quando se fornece uma correspondência individual entre símbolos e seus conceitos associados [Moody 2009]. O elemento sinógrafo é uma anomalia que ocorre quando não existe essa correspondência individual, levando à redundância de símbolos e diminuindo a compreensão da especificação ou notação.

- **Number of Homographs (NH):** essa métrica determina o número total de elementos homógrafos na especificação da seleção de POCT. Um homógrafo ocorre quando há vários elementos semânticos distintos que podem ser representadas pelos mesmos símbolos ou elementos. Quanto maior for o NH, pior é para entender a especificação. Esse é o pior tipo de anomalia em uma notação, seja ela gráfica ou textual, pois leva à ambiguidade e a potenciais erros de interpretação [Moody 2009]. Por exemplo, Theme/UML evidencia essa anomalia pois a mesma convenção gráfica (e.g., retângulo) é usada para representar vários elementos semanticamente distintos: *subject*, *class*, *lifelines*, etc.

As métricas são combinadas em um modelo de qualidade para avaliar os cinco princípios definidos no framework PoN. O princípio da clareza semiótica (*Semiotic Clarity*) é definida pela redundância (*Symbol Redundancy*) e sobrecarga (*Symbol Overload*) de símbolos. O primeiro é impactado pelo número de sinógrafos (NS) encontrados na especificação de seleções de POCT, enquanto o último pelo número de homógrafos (NH). O princípio do discernimento da percepção (*Perceptual Discriminability*) é definido pela distância visual que, por sua vez, é diretamente afetado pelo número de elementos diferentes usados (NVD). O princípio da transparência semiótica (*Semiotic Transparency*) é influenciado pelo número de elementos considerados semanticamente imediatos (NSIE), semanticamente opacos (NSOE) e semanticamente perversos (NSPE). Idealmente, uma notação deve apresentar o maior número possível de elementos semanticamente imediatos e zero elementos semanticamente perversos. O princípio da expressividade visual (*Visual Expressiveness*) é influenciado pela quantidade de variáveis livres (*Free Variables*) que é diretamente afetada pelo número de canais de comunicação visual (LVE). Uma notação sem variáveis de comunicação visuais é chamada de notação não visual (ou textual), enquanto uma notação que usa todas as variáveis de comunicação visuais é considerada visualmente saturada [Moody 2009]. Por fim, o princípio da economia gráfica (*Graphic Economy*) é afetado pela quantidade de símbolos híbridos (*Hybrid Symbols*) que são medidos pelo NE e NLE. Uma notação aumenta sua complexidade proporcionalmente ao aumento da quantidade de elementos que são especificados por expressão lógica.

Os demais princípios do PoN não foram considerados relevantes para o objetivo e escopo desta dissertação, que é a especificação de seleção de POCT. O princípio de *Gerenciamento de Complexidade* é apenas aplicado quando o sistema não faz uso de modularidade devido a integração heterogênea ou outros motivos, o que não é o caso do contexto de seleção de POCT para sistemas embarcados. O princípio de *Integração Cognitiva* não é relevante para a análise apresentada neste trabalho pois é aplicado apenas quando múltiplos diagramas são usados para representar um sistema, o que não é o caso da seleção de POCT. Por fim, os princípios do *Ajuste Cognitivo* e da *Codificação Dupla* não foram abordados pois considerou-se que nenhuma das notações avaliadas contempla as características associadas a estes princípios. O primeiro sugere usar dois dialetos diferentes nas notações visuais, um para usuários experientes e outro para iniciantes. O segundo sugere usar textos como algo redundante dentro da notação para reforçar o significado dos elementos visuais.

5.1 EXEMPLO DE APLICAÇÃO DE MODELO DE QUALIDADE PROPOSTO

Para ilustrar como essas métricas são calculadas sobre as notações gráficas ou textuais, será apresentado a seguir a aplicação das métricas no mesmo exemplo que apresentamos nos capítulos anteriores referentes ao requisito não-funcional de temporização, onde é necessário especificar a seleção de POCT referente ao controle de tarefas periódicas em um sistema embarcado de tempo-real de uma cadeira de rodas automatizada. Este POCT será nomeado como *JP_PeriodicBehaviour* que representa um POCT agnóstico que foi nomeado dessa forma com o objetivo de não correlacionar o POCT com nenhuma outra técnica.

Destaca-se que objetivo do MS4PoN (e deste exemplo especificamente) não é a comparação das notações (gráficas ou textuais), mas sim a avaliação da compreensão da especificação da seleção de POCT, de forma indireta, através da avaliação dos efeitos cognitivos associados com as propriedades de percepção das notações medidas através das métricas propostas.

Posto isto, a seguir será apresentado como essas métricas foram calculadas em cada especificação para cada técnica selecionada na análise do estado da arte, i.e., para cada métrica será apresentado como foram quantificados os valores da aplicação da MS4PoN. A Tabela 3 apresenta as métricas calculadas para as 6 especificações relacionadas ao *JP_PeriodicBehaviour*. Como mencionado, o NLE mede o critério de seleção independente da notação que deve ser especificada usando as 6 técnicas.

Tabela 3 – Valores das Métricas MS4PoN para JPExclusiveSetCalls

Notação	NE	NLE	NVD	NSIE	NSOE	NSPE	LVE	NS	NH
JPDD	10	7	5	0	5	1	1	0	0
AspectJ	7	7	7	1	7	0	0	0	0
Theme/UML	21	7	8	0	12	1	2	0	8
Tracematch	6	7	6	1	6	0	0	0	0
AspectOCL	23	7	19	1	19	0	0	0	0
JSD	16	5	13	2	13	0	3	0	0

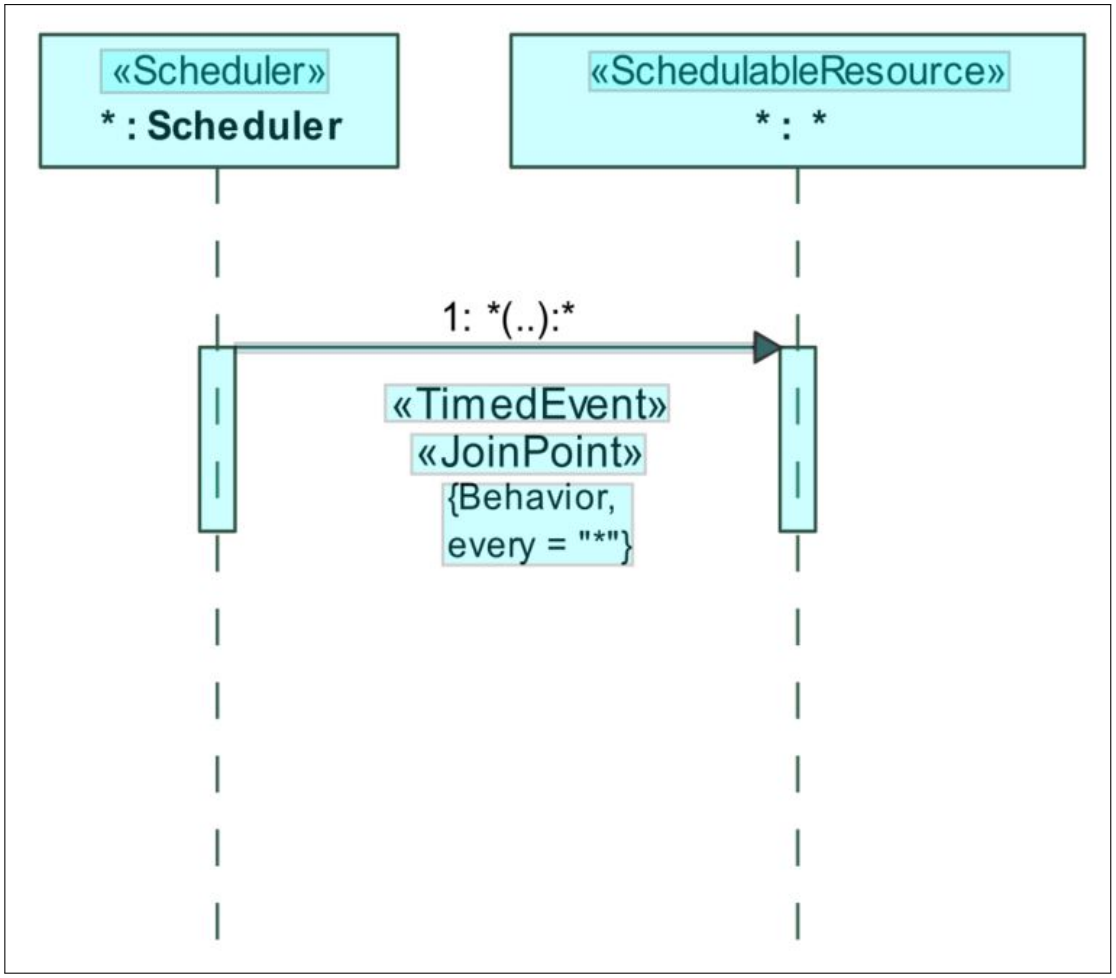
Fonte: o autor

Assim, para o *JP_PeriodicBehaviour*, esse critério é "`(selecionar elementos igual a ação de envio de mensagem) E (tipo igual a Behavior) E (todos os métodos E qualquer nome E anotado com o estereótipo <<TimedEvent>>) E (todas as classes E qualquer nome E anotado com o estereótipo <<SchedulableResource>>)`". Portanto, o NLE é o mesmo para as três especificações e, sendo assim, é possível fazer uma combinação do NLE com as outras métricas para gerar valores relativos, i.e., taxa calculadas com o valor de outras métricas dividido por NLE.

Na JPDD, o NE indica a quantidade dos elementos visíveis usados na especificação, por exemplo, a Figura 27 apresenta em azul os elementos sintáticos contabilizados, i.e., duas *lifelines*, dois retângulos verticais definidos pela UML como símbolo para comportamento, uma linha de conexão que significa uma ação de envio de mensagem, 4 estereótipos como elementos de texto (<<Scheduler>>, <<SchedulableResource>>, <<TimedEvent>>, <<JoinPoint>>) e uma restrição como elemento de texto entre chaves ({Behavior, every="*"}). Os outros símbolos, por exemplo "$*, ::, (..)$", não são reconhecidos como elementos, pois os mesmos são atributos do próprio elemento, por exemplo, o "*" significa o nome da classe na *lifeline*; outro exemplo seriam os "1: * (..): *" que indicam símbolos léxicos usados no modelo, i.e., não são elementos em si como o símbolo que representa o relacionamento de conexão com uma seta que simboliza o envio da mensagem.

Na AspectJ o NE considera os elementos textuais usados para descrever o aspecto em si: o próprio aspecto (linha 1) o pointcut (linha 2); a chamada do método "call" (que é um POCT implícito); a classe *SchedulableResource*; o método *run(..)*; os símbolos *!within* e *after()*. A Figura 28 apresenta em azul os elementos sintáticos contabilizados para AspectJ. Outra constatação quanto as linguagens textais é referente a Tracematch, onde o NE se mostrou com a menor quantidade de elementos sintáticos apesar da técnica estender os conceitos da AspectJ. A Figura 30 apresenta em azul os elementos contabilizados para Tracematch.

Figura 27 – JPDD: número de elementos sintáticos (NE)



Fonte: o autor

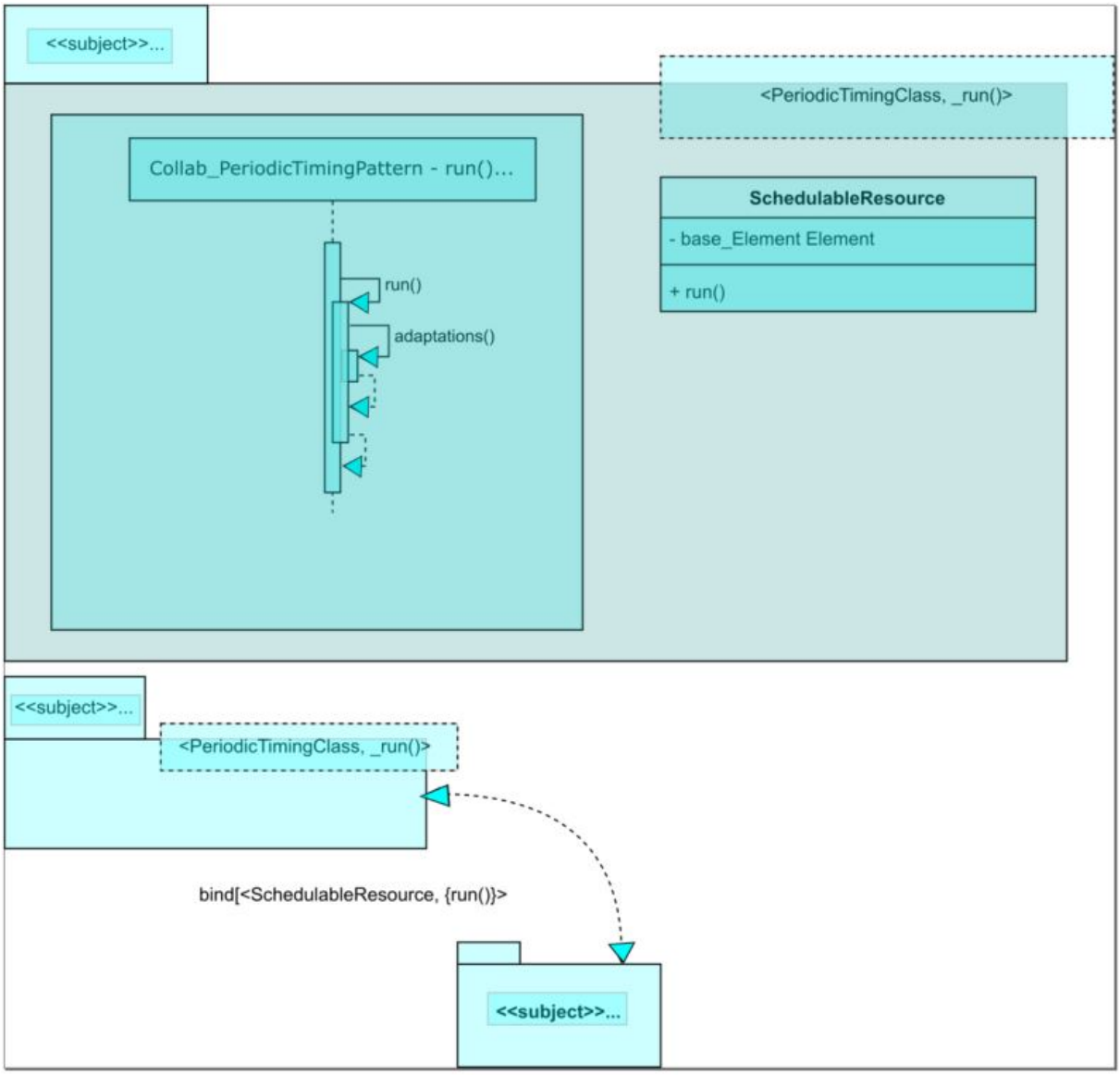
Figura 28 – AspectJ: número de elementos sintáticos (NE)

```
1  aspect PeriodicTiming {
2      pointcut pcFreqCtrl(): call(* *SchedulableResource.run(..)) && !
           within(PeriodicTiming);
3
4      after(): pcFreqCtrl() {
5          //Crosscutting Concerns Implementation
6      }
7  }
```

Fonte: o autor

A AspectOCL também apresentou um NE relativamente alto conforme Figura 31. Por sua vez, a especificação através da Theme/UML (Figura 29) apresentou um número de elementos sintáticos (NE) surpreendentemente alto na especificação de seleção de POCT. Todos os retângulos que significam classes, atributos, métodos, *subjects*, *lifelines*, comportamentos e relacionamentos de conexões que enviam símbolos de mensagens foram contabilizados.

Figura 29 – Theme/UML: número de elementos sintáticos (NE)



Fonte: o autor

Figura 30 – Tracematch: número de elementos sintáticos (NE)

```
1  tracematch(){
2    sym pcFreqCtrl after: call(* *SchedulableResource.run(..))
3
4    pcFreqCtrl
5    {
6      //Crosscutting Concerns Implementation
7    }
8  }
```

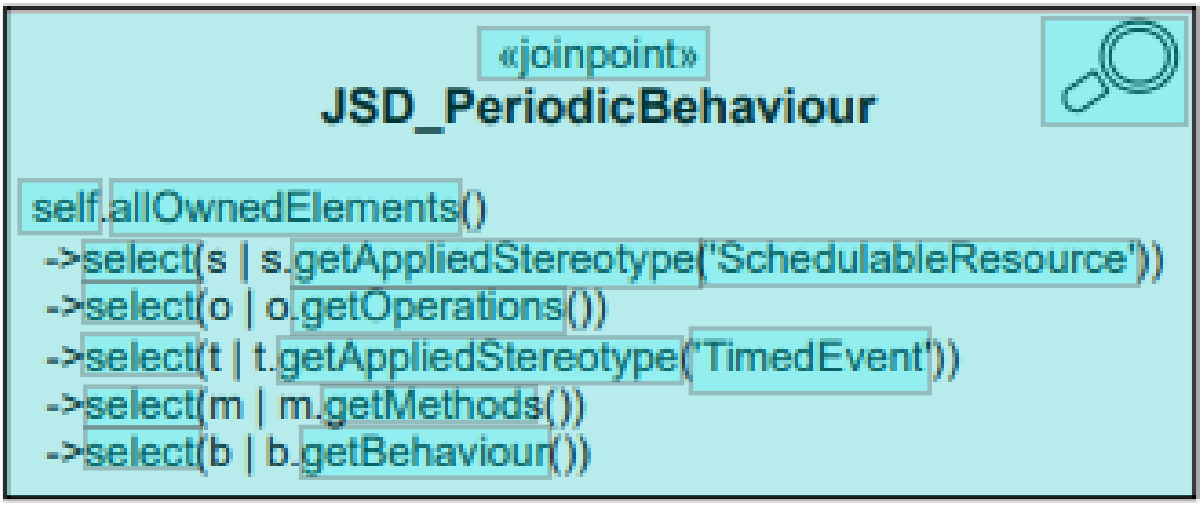
Fonte: o autor

Figura 31 – AspectOCL: número de elementos sintáticos (NE)

```
1  mapping mapPeriodicTiming
2      {
3          let T: {SchedulableResource}
4      }
5
6      aspect PeriodicTiming
7      {
8          import_mapping mapPeriodicTiming
9
10         pointcut pcFreqCtrl
11         context T :
12         inv:
13             self.allOwnedElements()
14         ->select(s | s.getAppliedStereotype('SchedulableResource'))
15         ->select(o | o.getOperations())
16         ->select(t | t.getAppliedStereotype('TimedEvent'))
17         ->select(m | m.getMethods())
18         ->select(b | b.getBehavior())
19
20     }
```

Fonte: o autor

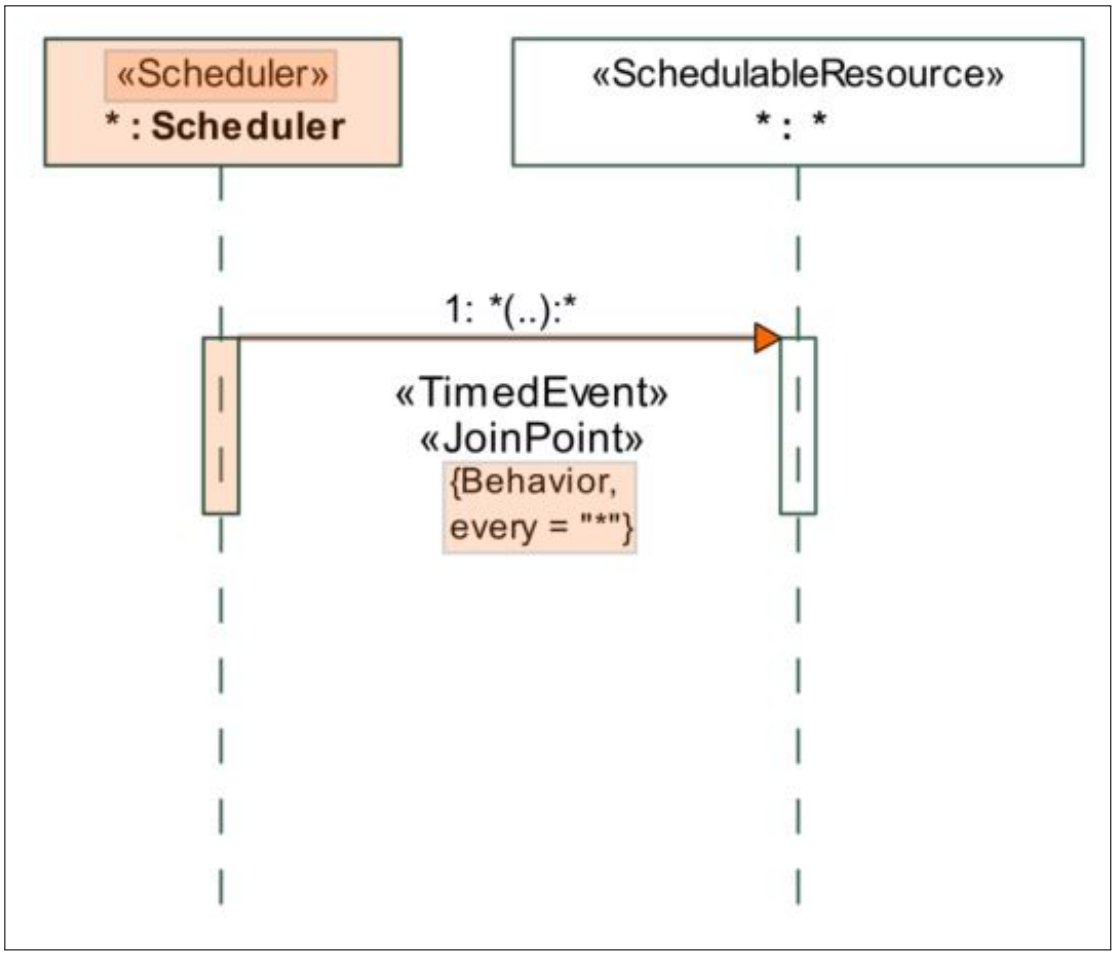
Figura 32 – JSD: número de elementos sintáticos (NE)



Fonte: o autor

Apesar da AspectJ ter apresentado um elemento sintático (NE) a mais que a Tracematch, as duas técnicas obtiveram valores ideais de elementos visualmente distantes (NVD), i.e., as notações textuais apresentaram um valor de elementos sintáticos visualmente distantes (NVD) iguais ao número de elementos sintáticos (NE) (para cada elemento sintático usado temos 1 e apenas 1 diferente), isso é o ideal almejado numa especificação de acordo com o princípio do Discernimento de Percepção [Moody 2009]. As Figuras 34, 36, 37 apresentam em laranja os elementos sintáticos visualmente distantes (NVD) das técnicas AspectJ, Tracematch e AspectOCL respectivamente. Quanto a JSD, apesar da técnica ter apresentado um número de elementos sintáticos (NE) alto, a quantidade de número de elementos visualmente distantes (NVD) também foi alta. O número de elementos visualmente distantes (NVD) é relevante sabendo que para cada elemento usado na especificação tem-se um elemento diferente, e isso ajuda no entendimento da especificação. A Figura 38 apresenta em laranja os elementos sintáticos visualmente distantes (NVD) na aplicação da JSD. Embora a JPDD tenha apresentado um valor baixo de elementos sintáticos (NE), em contrapartida apresentou um valor também baixo de elementos sintáticos visualmente distantes (NVD), o que implica que ou existem elementos implícitos ou bastante similares. Isso é muito ruim para entendimento da especificação. A Figura 33 apresenta em laranja os elementos visualmente distantes contabilizados para JPDD. Por sua vez, a Theme/UML também apresentou um baixo número de elementos visualmente distantes (NVD), i.e., também compartilha da mesma situação da JPDD de que ou existem elementos implícitos na especificação ou os elementos são bastante similares. A Figura 35 apresenta em laranja os elementos sintáticos visualmente distantes (NVD) da especificação de seleção de POCT através da Theme/UML.

Figura 33 – JPDD: número de elementos sintáticos visualmente distantes (NVD)



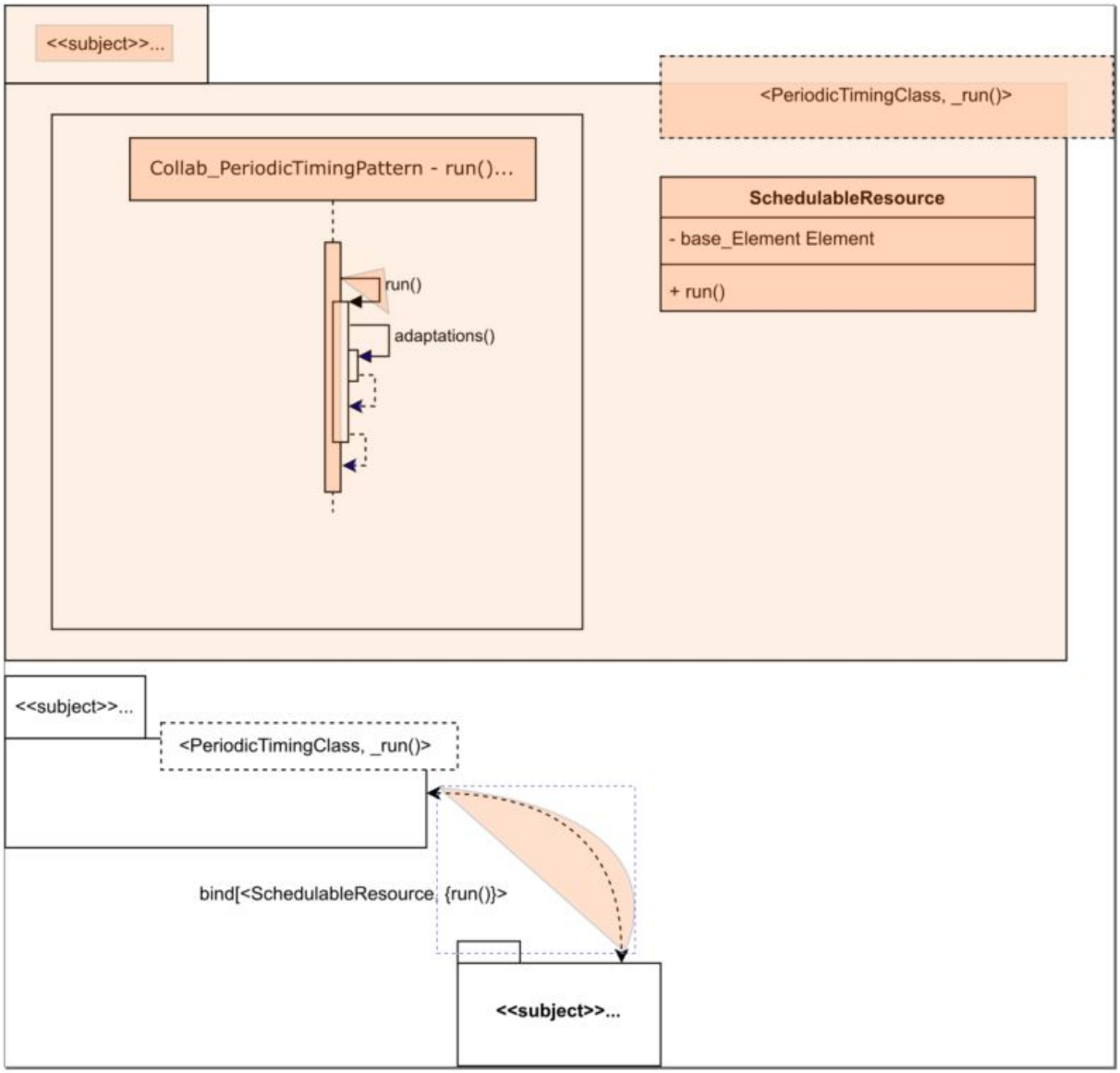
Fonte: o autor

Figura 34 – AspectJ: número de elementos sintáticos visualmente distantes (NVD)

```
1  aspect PeriodicTiming {
2      pointcut pcFreqCtrl(): call(* *SchedulableResource.run(..)) && !
           within(PeriodicTiming);
3
4      after(): pcFreqCtrl() {
5          //Crosscutting Concerns Implementation
6      }
7  }
```

Fonte: o autor

Figura 35 – Theme/UML: número de elementos sintáticos visualmente distantes (NVD)



Fonte: o autor

Figura 36 – Tracematch: número de elementos sintáticos visualmente distantes (NVD)

```
1  tracematch(){
2    sym pcFreqCtrl after: call(* *SchedulableResource.run(..))
3
4    pcFreqCtrl
5    {
6      //Crosscutting Concerns Implementation
7    }
8  }
```

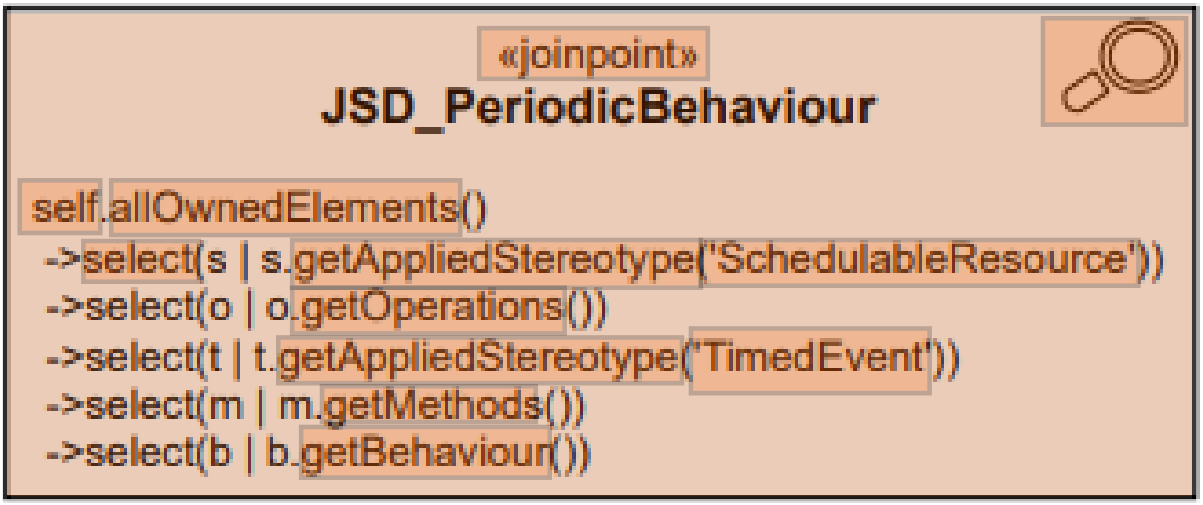
Fonte: o autor

Figura 37 – AspectOCL: número de elementos sintáticos visualmente distantes (NVD)

```
mapping mapPeriodicTiming
    {
        let T: {SchedulableResource}
    }

    aspect PeriodicTiming
    {
        import_mapping mapPeriodicTiming

        pointcut pcFreqCtrl
        context T :
        inv:
            self.allOwnedElements()
        ->select(s | s.getAppliedStereotype('SchedulableResource'))
        ->select(o | o.getOperations())
        ->select(t | t.getAppliedStereotype('TimedEvent'))
        ->select(m | m.getMethods())
        ->select(b | b.getBehavior())

    }
```

Fonte: o autor

Figura 38 – JSD: número de elementos sintáticos visualmente distantes (NVD)



Fonte: o autor

Quanto ao número de elementos semanticamente imediatos (NSIE), a técnica que apresentou o maior número foi a JSD. A Figura 42 apresenta em verde os elementos semanticamente imediatos contabilizados para a JSD. Em contrapartida, as outras notações visuais (JPDD e Theme/UMl) não fizeram uso de elementos semanticamente imediatos (NSIE), obtendo 0 elementos nessa métrica. A AspectOCL, Tracematch e AspectJ apresentaram apenas 1 elemento semanticamente imediato. As Figuras 39, 40 e 41 apresentam os elementos contabilizados para as notações textuais.

Quanto ao número de elementos semanticamente opacos (NSOE), i.e., convencionados arbitrariamente, apenas uma técnica (Theme/UML) obteve valores diferentes do número de elementos visualmente distantes (NVD). A Theme/UML obteve um valor de 12 elementos semanticamente opacos (NSOE). A Figura 43 apresenta em roxo os elementos contabilizados. As demais técnicas apresentaram o mesmo valor para o número de elementos semanticamente opacos (NSOE) e o número de elementos visualmente distantes (NVD). A diferença é que na Theme/UML existem elementos que são praticamente idênticos em suas sintaxes (i.e., símbolos usados) mas com semânticas distintas, e.g., Na especificação usando a Theme/UML, o símbolo que contorna a *lifeline* é o mesmo símbolo para a classe *ShedulableResource*, i.e., um retângulo, e não só isso, o mesmo símbolo é similar a todos os outros símbolos usados, por isso, temos um número de elementos opacos (NSOE) (i.e., convencionados arbitrariamente), mas que não são elementos diferentes entre si, i.e., não são visualmente distantes (NVD).

Em se tratando de elementos semanticamente perversos (NSPE), as notações textuais se saíram muito bem, apresentando zero elementos perversos em suas especificações de seleção de POCT. Não existem elementos sintáticos usados na AspectJ, Tracematch e AspectOCL que possam sugerir um significado diferente de sua sintaxe, i.e., um engenheiro júnior com um certo conhecimento da notação textual não teria problema em inferir o significado de sua sintaxe. Em contrapartida, as notações visuais apresentaram 1 elemento perverso por técnica. As Figuras 44, 45 apresentam em vermelho os elementos semanticamente perversos contabilizados. Tanto a JPDD quanto a Theme/UML possuem o mesmo elemento sintático que pode sugerir um significado diferente da sua sintaxe, nesse caso, tem-se o símbolo que é usado na UML tanto para estereótipo quantos para designar uma interface, como também uma propriedade, e.g., um engenheiro júnior poderia entender de forma equivocada que o símbolo usado para estereotipar a classe da *lifeline* especificada na Figura 44 seria uma interface. Poderia acontecer o mesmo equívoco no entendimento da Theme/UML apresentada na Figura 45.

Figura 39 – AspectJ: número de elementos semanticamente imediatos (NSIE)

```
aspect PeriodicTiming {
    pointcut pcFreqCtrl(): call(* *SchedulableResource.run(..)) && !
        within(PeriodicTiming);

    after(): pcFreqCtrl() {
        //Crosscutting Concerns Implementation
    }
}
```

Fonte: o autor

Figura 40 – Tracematch: número de elementos semanticamente imediatos (NSIE)

```
tracematch(){
 sym pcFreqCtrl after: call(* *SchedulableResource.run(..))

 pcFreqCtrl
 {
  //Crosscutting Concerns Implementation
 }
}
```

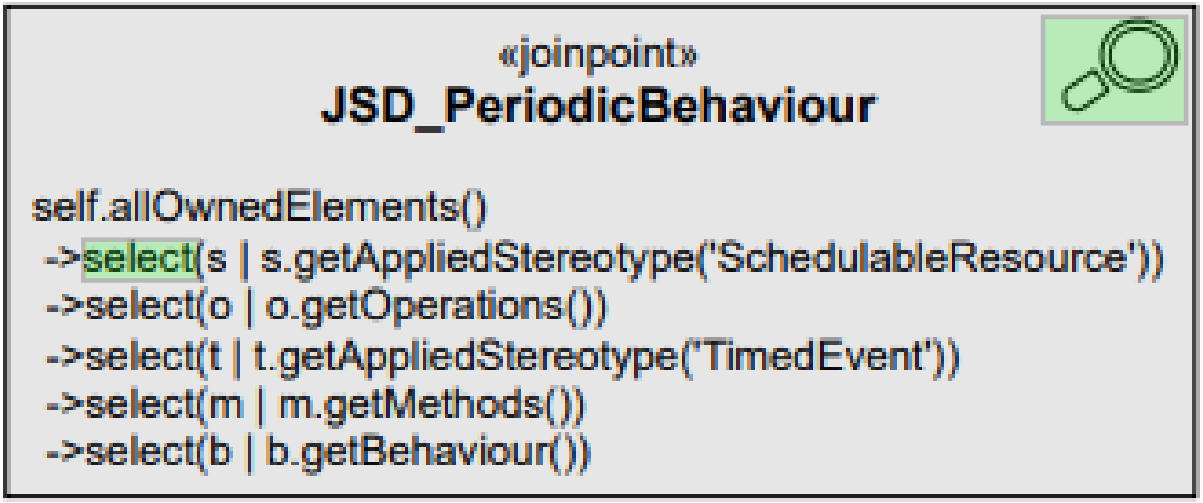
Fonte: o autor

Figura 41 – AspectOCL: número de elementos semanticamente imediatos (NSIE)

```
mapping mapPeriodicTiming
    {
        let T: {SchedulableResource}
    }

    aspect PeriodicTiming
    {
        import_mapping mapPeriodicTiming

        pointcut pcFreqCtrl
        context T :
        inv:
            self.allOwnedElements()
        ->select(s | s.getAppliedStereotype('SchedulableResource'))
        ->select(o | o.getOperations())
        ->select(t | t.getAppliedStereotype('TimedEvent'))
        ->select(m | m.getMethods())
        ->select(b | b.getBehavior())

    }
```

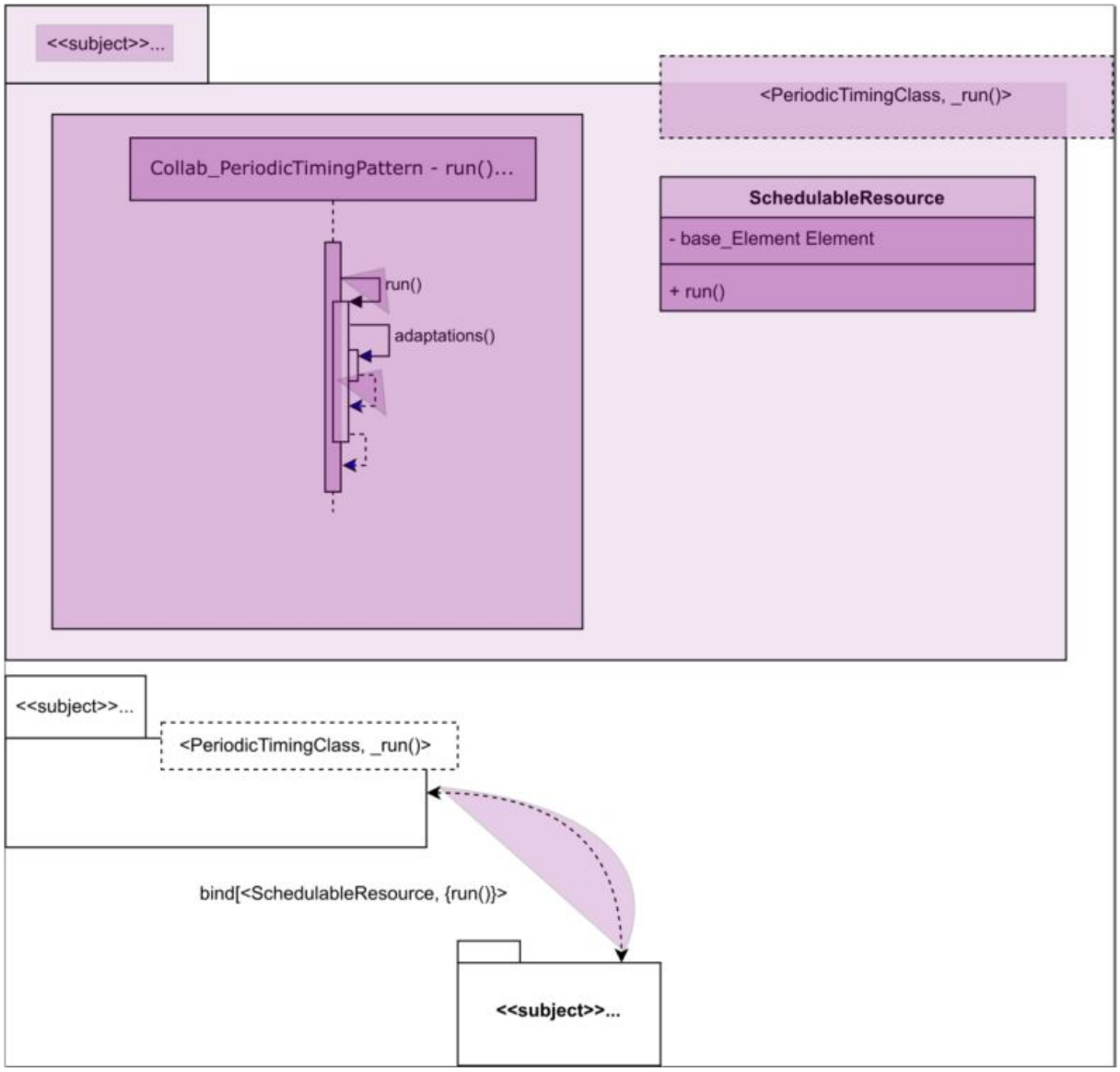
Fonte: o autor

Figura 42 – JSD: número de elementos semanticamente imediatos (NSIE)



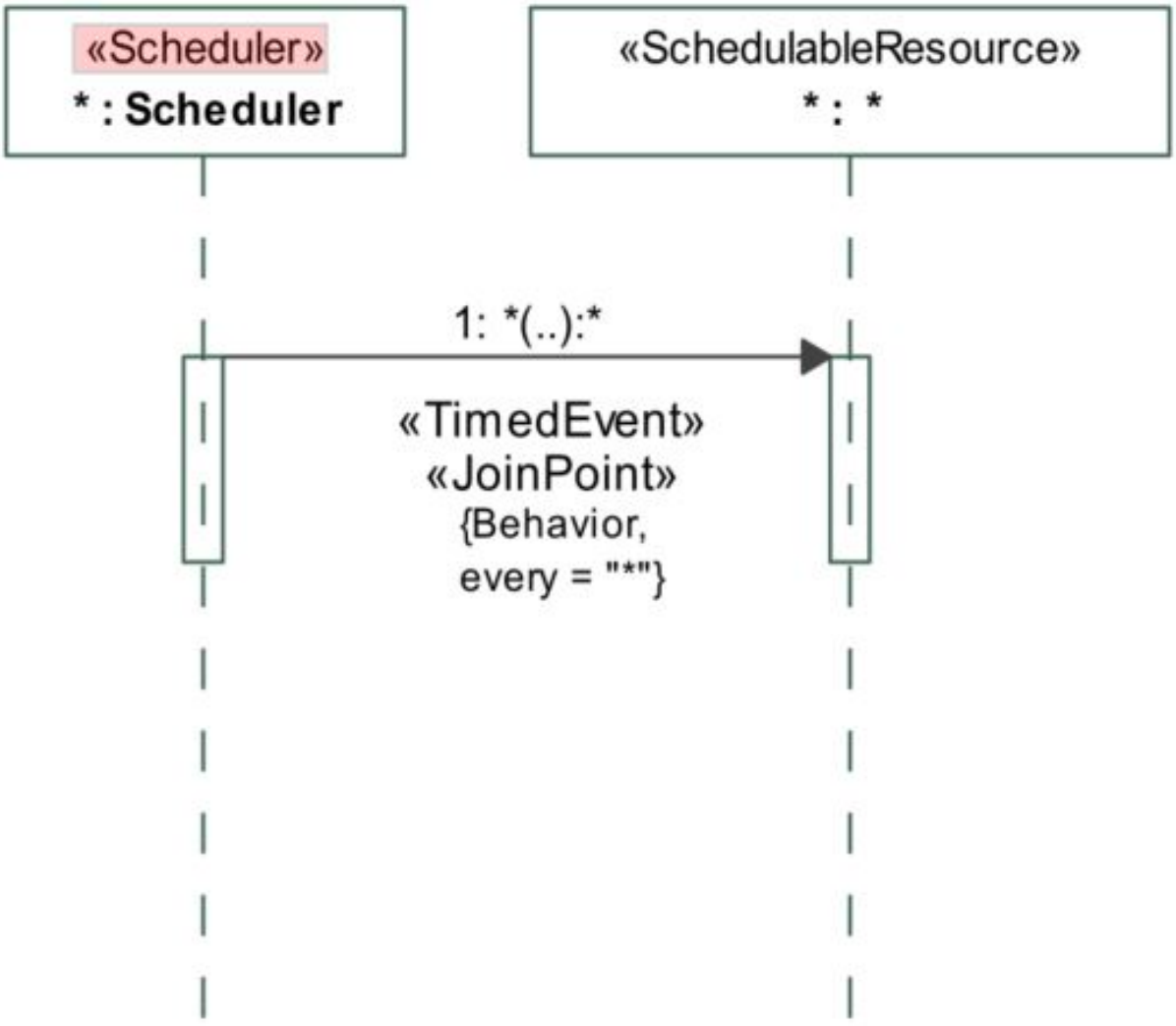
Fonte: o autor

Figura 43 – Theme/UML: número de elementos semanticamente opacos (NSOE)



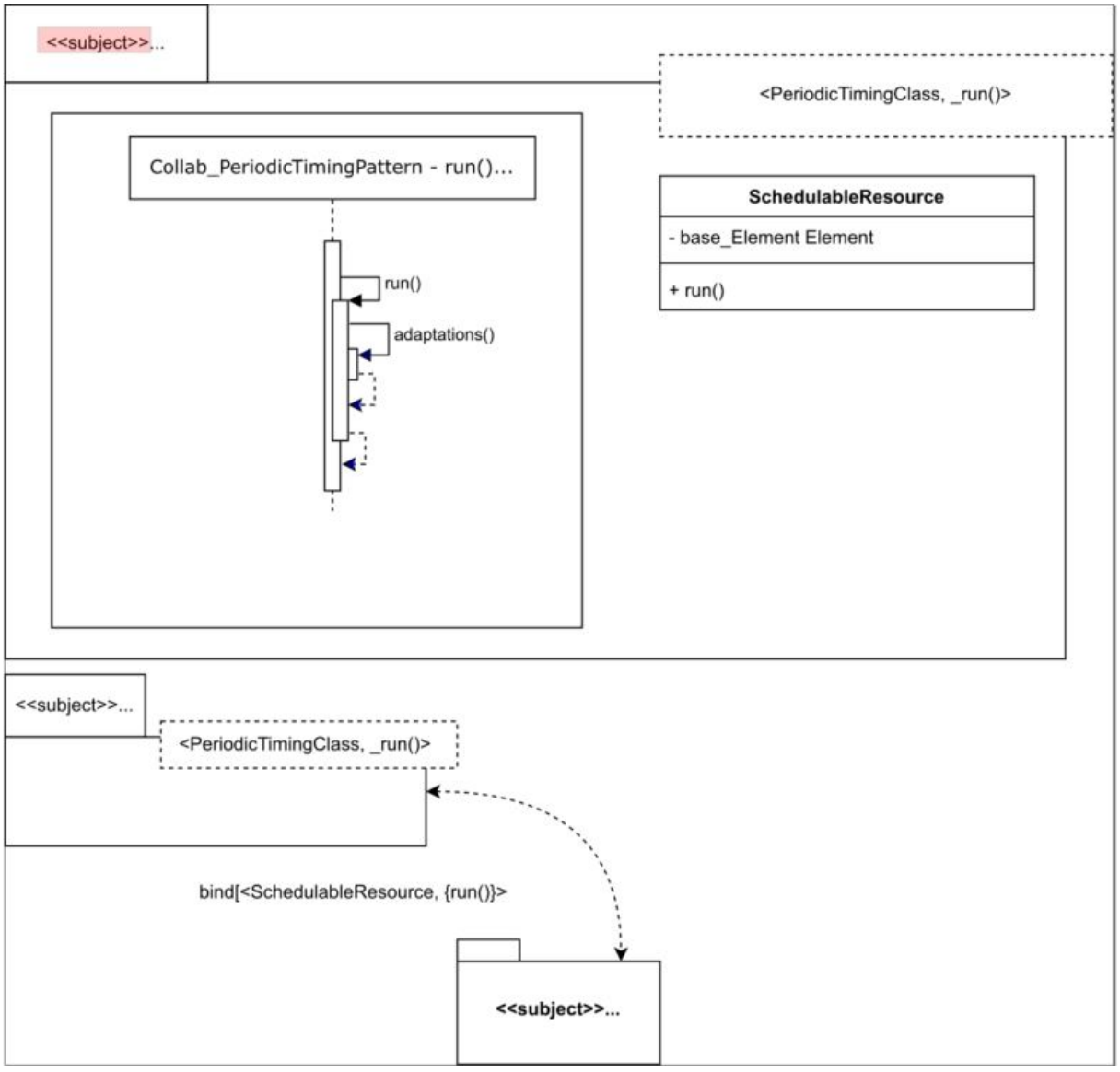
Fonte: o autor

Figura 44 – JPDD: número de elementos semanticamente perversos (NSPE)



Fonte: o autor

Figura 45 – Theme/UML: número de elementos semanticamente perversos (NSPE)

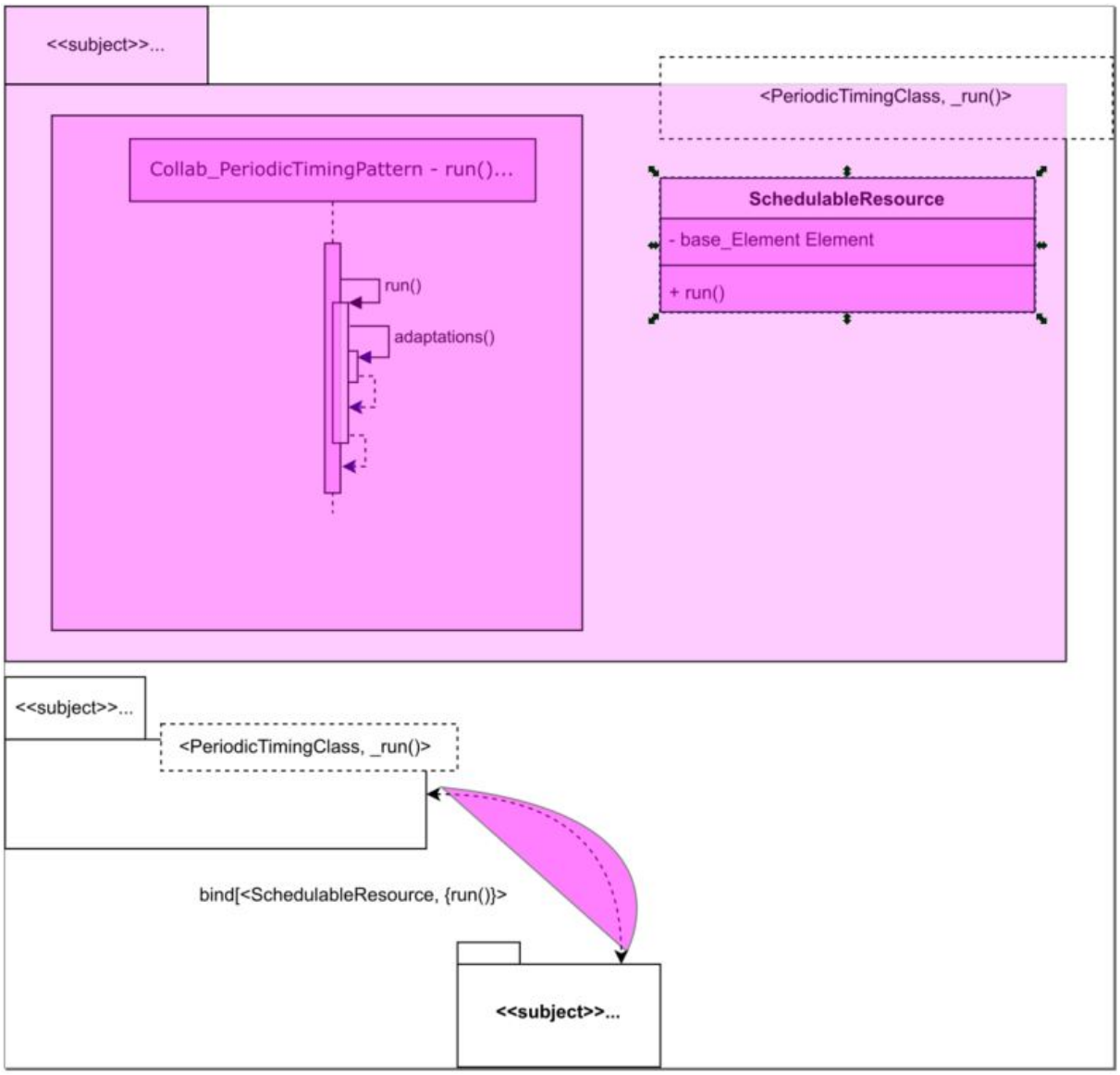


Fonte: o autor

Para as duas últimas métricas, a primeira referente ao número de elementos sinógrafos (NS) e a última referente ao número de elementos homógrafos (NH), apenas uma técnica apresentou elementos com uma dessas anomalias. A Theme/UML apresentou um alto valor para o número de elementos homógrafos (NH) devido à sobrecarga de retângulos usados na especificação, causando uma anomalia relacionada ao uso da mesma convenção gráfica para expressar vários elementos distintos. A Figura 46 apresenta em rosa os elementos contabilizados.

Por outro lado, a JSD obteve um número de elementos homógrafos (NH) igual a zero, uma vez que os elementos utilizados na especificação têm uma correspondência de um para um entre o símbolo e sua semântica e o número de elementos semanticamente perversos (NSPE) também igual a zero. A JPDD também obteve um NH igual a zero, entretando possui a maior quantidade de NSPE de todas as técnicas apresentadas. A JPDD também apresentou uma baixa expressividade visual (LVE), sendo muito próxima a uma linguagem textual. Percebe-se que a JPDD não faz uso dos benefícios de ser uma linguagem gráfica. A JSD por sua vez apresentou uma média significativa quanto a sua expressividade visual (LVE), pois utiliza de uma notação visual que faz uso de mais de uma variável de expressão, e.g., o uso da forma, cor, tamanho e textura. A JSD também apresentou o maior número de NSIE, i.e., elementos semanticamente imediatos, devido a apresentar elementos sintáticos mnemônicos na especificação de seleção de POCT (e.g., lupa, "select").

Figura 46 – Theme/UML: número de homógrafos (NH)



Fonte: o autor

5.2 DISCUSSÃO

Os critérios mencionados para calcular as métricas do MS4PoN permitem aplicar as métricas propostas para avaliar as especificações da seleção dos POCT, independentemente de usar uma notação gráfica ou textual. Portanto, o elemento de uma notação que pode ser visto em uma especificação gráfica também é um elemento que pode ser visto em uma especificação textual. A diferença está na forma de como o elemento é expresso e apresentado. Uma linguagem, seja ela gráfica ou textual, é composta de elementos léxicos, sintáticos e semânticos. Através desses elementos é possível expressar uma especificação de um requisito funcional ou não-funcional como também de restrições entre outras características do desenvolvimento de software.

Portanto, é possível aplicar diretamente o modelo de qualidade apresentado neste capítulo, como também suas métricas (MS4PoN) em especificações que podem ser processadas (e.g. linguagens) independentemente de serem notações textuais ou gráficas. A questão aqui é a aplicação do modelo de qualidade no conceito computacional associado ao elemento sintático, i.e, em um projeto de software embarcado, tanto o código (notação textual), quanto o modelo visual (notação gráfica) são especificações criadas pelo engenheiro que podem ser usadas como entrada para alguma ferramenta, e.g. um programa escrito em AspectJ pode ser processado por um compilador, enquanto um modelo UML pode ser um meta-modelo processado por uma ferramenta da geração de código. Em outras palavras, o código-fonte e os modelos visuais são artefatos que são criados com o objetivo de facilitar o processo de desenvolvimento do software embarcado.

Posto isto, é possível uma comparação por analogia dos elementos sintáticos usados nas notações, tanto gráficas quanto textuais, pois elas são artefatos que podem ser processados, sendo que o primeiro é apresentado visualmente e o segundo apresentado textualmente. Por isso, é válida essa analogia. Enquanto a Engenharia de Software tem desenvolvido métodos maduros para avaliar a semântica (propriedades lógicas) dos modelos visuais, existe uma deficiência quanto aos métodos equivalentes para sintaxe (propriedades perceptuais). Sendo assim, uma das contribuições desta dissertação é avaliar por analogia quanto a compreensão da especificação de seleção de POCT de forma indireta através da avaliação das propriedades perceptuais (sintaxe) dos elementos das especificações seja pelo uso de notações textuais ou gráficas.

6 EXPERIMENTOS E RESULTADOS

Esta seção discute os experimentos empíricos realizados para avaliar a especificação da seleção de POCT usando três técnicas gráficas (JPDD, Theme/UML e JSD) e três técnicas textuais (AspectJ, Tracematch e AspectOCL). As cinco notações usadas (além da técnica JSD, que é a contribuição principal deste trabalho) foram escolhidas devido a grande quantidade de citações de trabalhos relacionados e por serem amplamente conhecidas dentro do contexto do AOSD [Filman *et al.* 2004]. O objetivo é avaliar os efeitos cognitivos e as propriedades de percepção dessas notações, empregando o modelo de qualidade proposto através do cálculo das métricas MS4PoN descritas na seção anterior.

A fonte de dados para essas experiências inclui 16 POCT comumente encontrados no desenvolvimento de software para sistemas ciber-físicos e sistemas embarcados de tempo-real. Esses POCT foram criados pelos próprios autores, em três estudos de caso realizados em trabalhos anteriores [Wehrmeister *et al.* 2013, Wehrmeister *et al.* 2014]: (i) o sistema de controle de movimento de um veículo aéreo não tripulado; (ii) o sistema de controle de um sistema industrial de empacotamento; e (iii) o sistema de controle de movimento de uma cadeira de rodas automatizada. O anexo A contêm informações de cenários usados nos experimentos. Os experimentos realizados neste trabalho consistem em criar três especificações para cada uma das dezesseis POCT usando as notações JPDD, AspectJ, Theme/UML, Tracematch, AspectOCL e JSD, totalizando 96 especificações distintas.

Em seguida, as métricas da MS4PoN foram calculadas sobre cada especificação distinta para todas as seleções de POCT. Embora essas notações usadas permitam uma especificação completa baseada no AOSD, os experimentos e seus resultados consideram apenas a parte das notações relacionadas a especificação das seleções de POCT.

A Tabela 4 mostra os valores das métricas do MS4PoN calculadas a partir das 96 especificações de seleção de POCT. Os valores representam a soma dos valores individuais de cada métrica para cada especificação, com exceção do valor referente a expressividade visual (LVE), que é calculado como um valor médio. A LVE representa a expressividade visual da notação de acordo com o framework PoN [Moody 2009], cujo valor deve variar de zero a oito. A soma da métricas LVE de todas as especificações não representa adequadamente o conceito de expressividade visual porque indicaria frequentemente uma notação saturada (i.e., valor maior que 8) conforme o número de especificações aumenta.

Tabela 4 – Métricas para todas as especificações de seleção de POCT

Notação	NE	NLE	NVD	NSIE	NSOE	NSPE	LVE	NS	NH
JPDD	97	79	60	0	60	23	1,312	0	16
AspectJ	208	79	208	29	208	0	0	0	0
Theme/UML	381	79	53	0	102	16	2	15	158
Tracematch	147	79	147	52	147	0	0	0	3
AspectOCL	243	79	209	42	238	0	0	0	0
JSD	228	79	191	42	191	0	3.25	0	0

Fonte: o autor

As métricas da MS4PoN quantificam os recursos de uma notação usados para especificação de seleção de POCT. No entanto, para melhorar a análise e a comparação dos resultados, foram criadas taxas (i.e., valores relativos) através da associação entre as métricas. A Tabela 5 mostra os valores das métricas divididos pelo número de elementos sintáticos (NE) ou pelo número de expressões lógicas (NLE) ou ambos. As taxas permitem quantificar melhor a eficácia cognitiva de uma notação sob a perspectiva do framework PoN. O objetivo é identificar, para cada métrica, como a propriedade de percepção é distribuída pelos elementos descritos nas especificações.

A taxa referente ao número de elementos visualmente distantes por número de elementos sintáticos (NVD/NE) indica quantos elementos diferentes aparecem para cada elemento usado na especificação. Para essa taxa a notação textual AspectJ apresentou um valor mais alto em comparação com as notações gráficas JPDD e Theme/UML. Isso significa que a linguagem AspectJ emprega mais elementos diferentes para cada elemento usado na especificação de seleção de POCT. O mesmo raciocínio é válido para a relação NVD/NLE que considera o número de elementos diferentes por expressão lógica. Quanto maior a distância visual por expressão lógica, melhor o entendimento da especificação. No entanto, é importante destacar que um bom valor para essas proporções mencionadas deve ser próximo a um (1), i.e., a notação possui apenas um elemento distinto por expressão lógica. Valores menores que 1 indicam que existem elementos implícitos na especificação e valores maiores que 1 indicam que a notação é saturada. Ambas as situações dificultam de alguma forma o entendimento da especificação.

As taxas relacionadas ao número de elementos semanticamente imediatos (NSIE) indicam quantos elementos semanticamente imediatos são usados por elemento na especificação ou por expressão lógica. Essas proporções identificam quão fácil é inferir o significado correto dos elementos descritos na especificação pela aparência dos elementos e símbolos. A proporção número de elementos semanticamente imediatos por número de elementos sintáticos (NSIE/NE) para a Tracematch é um pouco mais alta do que nas outras técnicas textuais, ou seja, a Tracematch

Tabela 5 – Taxas das métricas MS4PoN

Taxas	AspectJ	JPDD	Theme/UML	Tracematch	AspectOCL	JSD	Valores Ideais*
NE/NLE	2.632	1.227	4.822	1.860	3.075	2.886	1
NVD/NE	1.000	0.618	0.139	1	0.860	0.837	[1,2)
NVD/NLE	2.632	0.759	0.670	1.860	2.645	2.417	[1,2)
NSIE/NE	0.139	0.000	0.000	0.353	0.172	0.184	>0
NSIE/NLE	0.367	0.000	0.000	0.658	0.531	0.531	>0
NSOE/NE	1.000	0.618	0.267	1	0.979	0.837	0
NSOE/NLE	2.632	0.759	1.291	1.860	3.012	2.417	0
NSPE/NE	0.000	0.237	0.041	0	0	0	0
NSPE/NLE	0.000	0.291	0.202	0	0	0	0
LVE	0.000	1.312	2.000	0	0	3.25	8
NS/NE	0.000	0.000	0.039	0	0	0	0
NS/NLE	0.000	0.000	0.189	0	0	0	0
NH/NE	0.000	0.164	0.414	0.020	0	0	0
NH/NLE	0.000	0.202	2.000	0.037	0	0	0

* A notação ideal não deve ter nenhum homógrafos, sinógrafos, ou elementos perversos ou opacos que são elementos convencionados arbitrariamente. Por outro lado, quanto maior o número de elementos semanticamente imediatos, melhor a compreensão.

Fonte: o autor

possui mais elementos (na especificação) que apresentam entendimento semântico implícito que a AspectJ e a AspectOCL. Por outro lado, as técnicas textuais se apresentaram com uma proporção mais alta que a JPDD e a Theme/UML, visto que essas duas técnicas gráficas não possuem nenhum elemento semanticamente imediato em suas especificações. Isso pode ter ocorrido devido ao fato de que JPDD e Theme/UML serem extensões da UML. De acordo com Moody, a maioria das notações que estendem a UML não foi projetada considerando as propriedades perceptivas e cognitivas que uma notação visual deve ter. No entanto, a proporção referente ao número de elementos semanticamente imediatos pelo número de expressões lógicas (NSIE/NLE) para a JSD se apresentou com um valor mais alto comparado com a JPDD, Theme/UML, AspectJ e AspectOCL. Isso é resultado do desenvolvimento da JSD a partir das premissas da Física das Notações (e.g., o uso de uma lupa no modelo representando um símbolo cultural de busca por elemento criando assim um mnemônico cultural.)

Posto isto, com base nos resultados das métricas apresentadas, é possível verificar através da quantificação dos dados que a JSD se apresentou melhor que as outras técnicas quanto a eficácia cognitiva embora o número de elementos sintáticos por expressão lógica (NE/NLE) tenha apresentado um valor de 2.886, sendo esse valor maior que o ideal de um (1) elemento por expressão lógica. Por outro lado, a JPDD obteve um valor de (1.227) elementos sintáticos por expressão lógica (NE/NLE). Apesar de ser um valor mais próximo do ideal, a JPDD não contempla todos os elementos necessários de acordo com o critério de seleção (i.e., expressão lógica, NLE).

Isso se aplica para todas as outras técnicas. Por exemplo, a Theme/UML apresentou o maior número de NE/NLE, i.e., a Theme/UML para cada critério de seleção usa 4.822 elementos sintáticos na especificação. Isso demonstra uma especificação saturada, e em se tratando da Theme/UML é ainda pior, pois sua avaliação apresentou muitos elementos homógrafos (NH) para especificar as seleções dos POCT. Os homógrafos são anomalias que levam à ambiguidade e potencializam o erro nas interpretações dos modelos [Moody 2009].

No entanto, para concluir qual técnica ou notação especifica melhor a seleção de POCT (i.e., a compreensão de forma indireta através das avaliação de propriedades perceptivas) é necessário entender cada métrica, uma por uma. i.e., o fato da JPDD ter apresentado uma taxa mais próxima do ideal referente ao número de elementos sintáticos pelo número de expressões lógicas (NE/NLE), não quer dizer que esses elementos sintáticos tenham um distanciamento visual ideal (i.e., são diferentes entre eles), também não indica se na especificação não existem elementos implícitos que dificultaria a compreensão, e.g., a taxa referente ao número de elementos visualmente distantes por número de elementos sintáticos (NVD/NE) que representa qual a distância visual para cada elemento sintático, mostra que que JPDD possui um valor de 0.618, o que significa que para cada elemento da especificação, 0.618 são diferentes entre si, e no entanto para JSD o valor apresentado da métrica foi de 0.837, quer dizer que para a JSD, temos 0.837 elementos diferentes para cada elemento sintático usado na especificação, o que é melhor que a JPDD e mais próximo do ideal que é de ser maior que 1 ou menor ou igual a 2.

Todas as linguagens textuais (AspectJ, Tracematch e AspectOCL) apresentaram valores mais próximos ao ideal para a taxa referente ao número de elementos visualmente distantes por número de elementos sintáticos (NVD/NE). Outro ponto negativo para as notações gráficas é o fato da Theme/UML além de ter apresentado o maior valor referente a taxa de número de elementos sintáticos por número de expressões lógicas (NE/NLE), apresentou também o menor valor para a taxa relacionada ao número de elementos visualmente distantes por número de elementos sintáticos (NVD/NE), i.e., possui 4.822 elementos sintáticos por critério de seleção e 0.139 elementos diferentes por elemento sintático, o que significa que os elementos usados ou são iguais ou bastante similares. Em se tratando da taxa referente ao número de elementos visualmente distantes por número de expressões lógicas (NVD/NLE), é possível identificar que a JSD ficou mais próxima do ideal do que a JPDD, tendo em vista que o valor ideal para essa métrica é que seja maior que 1 e menor ou igual a 2. A JPDD apresentou um resultado de 0.759 elementos sintáticos diferentes por expressão lógica, i.e., não atingiu a primeira condição

bit-circuit de ser maior que um, por outro lado a JSD apresentou um resultado de 2.417 elementos sintáticos diferentes por expressão lógica o que demonstra um melhor distanciamento visual devido ao valor ter sido maior que 1, no entanto passou um pouco da segunda condição de não ter sido menor ou igual a 2. Um resultado positivo referente as notações textuais se apresentam nos valores apresentados pela Tracematch quanto a taxa NVD/NLE. A Tracematch apresentou um valor de 1.860 elementos diferentes por expressao lógica, i.e., a distância visual entre os elementos na Tracematch é melhor que nas outras técnicas avaliadas pois apresenta um valor dentro dos valores ideais.)

Um número relevante de elementos semanticamente perversos (NSPE) nas notações gráficas avaliadas chama a atenção. Na JPDD e particularmente Theme/UML, a especificação da seleção de POCT inclui elementos ou símbolos cuja natureza possui um falso mnemônico. A JPDD apresenta um valor de 0.237 elementos perversos por elemento sintático (NSPE/NE), i.e., para cada elemento sintático usado na especificação 0.237 possui um falso mnemônico associado. Igualmente na Theme/UML que apresentou um valor um pouco menor de 0.041. Quanto a taxa referente ao número de elementos perversos por expressão lógica (NSPE/NLE), esse valor aumenta para as notações visuais, e isso é ruim, e.g., a JPDD apresentou um valor de 0.291 elementos perversos por expressão lógica. Isso pode induzir um engenheiro iniciante a inferir um significado diferente do que se apresenta na especificação. Por outro lado, a JSD apresentou zero elementos perversos em suas especificações, não só a JSD como as demais notações textuais (AspectJ, Tracematch e AspectOCL).

Além disso, o número de elementos semanticamente imediatos (NSIE/NE) se apresentou melhor na JSD que na JPDD, i.e., enquanto a JPDD faz uso de zero elementos que possuem um mnemônico que sugere o seu significado, a JSD faz uso de 0.184 elementos semanticamente imediatos por elemento usado (NSIE/NE) como também faz uso de 0.531 elementos semanticamente imediatos por expressão lógica (NSIE/NE). Vale ressaltar que a Tracematch mais uma vez se apresentou melhor nessa taxa que as demais técnicas inclusive a JSD. No entanto, isso é mais um indício que as técnicas textuais se apresentam mais eficazes quanto as suas propriedades perceptivas (sintaxe), logicamente que tendo essa comparação por analogia, o que não impacta na contribuição desse trabalho que é referente a criação de uma técnica visual que seja melhor em sua compreensão que a JPDD usada na AMoDE-RT.

Outro ponto relevante para o entendimento dos resultados é quanto ao número de elementos opacos (NSOE), vale ressaltar que essa métrica tem um valor neutro ou convencional

pois define uma relação arbitrária entre a aparência e o significado do elemento sintático. Moody afirma que a transparência semântica não possui um estado binário, mas um estado contínuo, no qual um elemento semanticamente opaco representa o ponto zero da escala entre o elemento semanticamente imediato e o perverso. Dessa forma, quanto menos elementos opacos (NSOE) forem usados na especificação mais elementos imediatos (NSIE) ou perversos (NSPE) teremos, e.g., a JPDD possui um valor de 0.759 elementos opacos por expressão lógica (NSOE/NLE), no entanto possui um valor de 0.291 de elementos perversos por expressão lógica (NSPE/NLE) em sua especificação. De outra forma, a JSD possui um valor de 2.417 de NSOE/NLE, mas, com zero elementos perversos (NSPE/NLE) e um valor relevante de 0.531 de elementos imediatos (NSIE/NLE). Vale ressaltar que quanto maior o número de NSIE, melhor a compreensão.

Outro resultado interessante foi quanto ao nível de expressividade (LVE) das notações avaliadas. As notações gráficas por exemplo apresentaram um LVE muito baixo, próximo a de uma notação textual, pois elas apresentam poucas variáveis de comunicação visual, indicando que por exemplo a JPDD ou Theme/UML não se beneficiam das vantagens esperadas de uma linguagem visual. Por outro lado, a JSD se apresentou com um valor significativo de LVE se aproximando mais de uma notação visual que as outras notações visuais avaliadas. De acordo com Moody, uma notação visual ideal deveria fazer uso das 8 variáveis de comunicação (e.g., forma, tamanho, cor, claridade, orientação, textura, posição horizontal e posição vertical). Moody também afirma que a maioria das notações visuais usadas na engenharia de software são pobres quanto a sua expressividade visual e não podem ser chamadas se quer de notação pois ferem princípios básicos de um sistema notacional e princípios básicos de comunicação visual, sendo baseadas em opiniões e conjecturas sem nenhuma base lógico-científica.

Além disso, a JSD também se beneficia por não apresentar nenhum elemento sinógrafo (NS) ou homógrafo (NH) que são anomalias relacionadas a clareza semiótica. A primeira indica elementos redundantes numa especificação e a outra indica uma sobrecarga de símbolos, podendo gerar confusão no entendimento da especificação de seleção de POCT, e consequentemente gerar ambiguidades e baixa eficácia cognitivas de suas propriedades de percepção. Portanto, conclui-se que a JSD se beneficia melhor quanto as vantagens esperadas de uma notação visual para a especificação de seleção de POCT que as técnicas avaliadas. Assim, pode-se afirmar que a JSD é uma técnica que contribui e inova quanto ao uso apropriado dos benefícios de uma linguagem visual, como também ao uso apropriado dos benefícios de uma linguagem textual através da aplicação do princípio da codificação dupla [Moody 2009].

Ao analisar as taxas apresentadas na Tabela 5, percebe-se que a JSD é mais eficaz em relação a compreensão das especificações de seleção de POCT em comparação com a JPDD, Theme/UML, AspectJ, AspectOCL e Tracematch. Apesar da Tracematch ter se apresentado com proporções um pouco melhores quanto ao distanciamento visual (NVD/NE e NVD/NLE) e quanto ao uso de elementos sintáticos semanticamente imediatos (NSIE/NE, NSIE/NLE), a especificação apresentou também uma proporção de elementos homógrafos o que implica na sobrecarga de símbolos que podem gerar ambiguidades na compreensão da especificação. No entanto, como mencionado neste trabalho, a avaliação das técnicas se faz por analogia tendo em vista que estamos comparando uma especificação através de uma técnica textual com uma especificação através de uma técnica gráfica.

Isto posto, ao analisarmos as taxas, conclui-se que a JSD é mais eficaz que as técnicas visuais avaliadas, i.e., JPDD e Theme/UML, e não só isso, mas, por analogia, mais eficaz cognitivamente que as técnicas textuais, através da avaliação das propriedades de percepção (i.e., sintaxe) das notações. Esses resultados empíricos indicam que a especificação de seleções de POCT através da JPDD e Theme/UML apresentam pouco discernimento na eficácia cognitiva das representações visuais, visto que tanto a JPDD como a Theme/UML ficaram aquém do esperado, sendo piores em termos de eficácia cognitiva que as linguagens textuais. Portanto, essa análise realizada sob a perspectiva do framework PoN aponta para o entendimento de que o uso de notações gráficas para a especificação da seleção de POCT é um processo inconsciente, pois as especificações são de alguma maneira difíceis de se entender quando os engenheiros não tem experiência suficiente na notação gráfica usada, sendo necessária a criação de uma nova técnica sob a ótica da Física das Notações, o qual foi criada na presente dissertação e se mostrou melhor que todas as técnicas avaliadas quanto a compreensão de forma indireta através da avaliação da eficácia cognitiva e das propriedades de percepção das notações.

6.1 LIMITAÇÕES

Embora os resultados obtidos forneçam o suporte adequado para as conclusões deste trabalho, alguns fatores podem ter influenciado a avaliação realizada, como é habitual em qualquer estudo empírico. Os fatores de ameaças à validade interna da avaliação podem surgir do fato de que os resultados da eficácia cognitiva podem ser influenciados pela experiência de quem lê as especificações, no caso, os autores deste trabalho. Entretanto, consideramos o risco desse fato afetar os resultados obtidos como baixo, uma vez que o modelo de qualidade proposto e os

resultados experimentais são baseados na perspectiva do framework PoN. Esse risco também é atenuado, porque as POCT usadas no experimento são de estudos de caso do mundo real, e não criados artificialmente.

Os resultados alcançados podem ter relação direta com a qualidade das especificações criadas. A falta de experiência em lidar com características transversais (tanto na especificação quanto na implementação do software embarcado), e também no uso das notações visuais, pode levar a criação de especificações com baixa qualidade, o que impacta na sua compreensão. O risco de resultados obtidos não representarem a realidade é baixo pois os autores possuem bastante experiência nos temas mencionados. Além disso, o framework PoN e as métricas MS4PoN avaliam as propriedades físicas perceptivas (i.e., sintaxe) das notações e não sua semântica, o que nos permite ter confiança nos resultados alcançados.

Outra ameaça à validade pode estar relacionada a generalidade dos resultados, uma vez que estão limitados aos três casos de estudos do domínio de sistemas embarcados. O modelo de qualidade proposto e as métricas MS4PoN são genéricas e podem ser usadas para avaliar outras notações visuais e textuais. No entanto, dentro do escopo da especificação de seleção de POCT, a variedade de tipos das ocorrências usadas nos experimentos nos permite acreditar que resultados semelhantes serão obtidos em outros domínios de aplicação.

Além dessas ameaças a validade, uma outra ameaça é quanto a validade da Física das Notações (PoN) e do conjunto de métricas criado neste trabalho (MS4PoN). Poderia haver questionamentos se existiria evidências empíricas quanto a validade de ambos. No entanto essa questão é resolvida tendo em vista que o PoN foi analisado em diversos trabalhos, conforme indicado em [Linden e Hadar 2019]. Com relação à validade do conjunto de métricas MS4PoN, a mesma ameaça a validade poderia ser apontada para outras métricas de engenharia de software, por exemplo, quando C&K foi discutido pela primeira vez no artigo OOPSLA'91. Da mesma forma que C&K, o MS4PoN também foi construído em uma base teórica sólida; neste caso sob a ótica do PoN. Sendo assim, identificamos esse risco como baixo devido ao MS4PoN fornecer meios para quantificar as propriedades sintáticas de uma notação (visual e textual) com base no PoN, que, por sua vez, tem sido amplamente utilizada até agora para avaliações qualitativas [Linden e Hadar 2019].

Por fim, uma outra ameaça a validade poderia estar relacionada a diferença de expressividade das notações, ou a falta de clareza em relação as técnicas terem a mesma expressividade, e.g., pode-se entender que Theme/UML é mais complexa porque permite expressar mais. No

entanto, este não é o caso, pois o objetivo principal deste trabalho é avaliar como a seleção de POCT (ou seja, os critérios de seleção com base em expressões lógicas) é especificada usando notações visuais e textuais distintas. Em outras palavras, é analisada apenas a parte relativa à seleção de POCT, não a notação inteira nem todos os recursos que eles fornecem para lidar com as características transversais. Portanto, embora as notações analisadas tenham diferentes níveis de expressividade, isso não afeta os resultados apresentados, uma vez que analisamos exatamente a mesma parte de cada notação.

6.2 DISCUSSÃO

Antes de mais nada, é necessário destacar que as métricas do MS4PoN não avaliam a cognição humana diretamente. No entanto, elas fazem isso indiretamente para o entendimento da especificação de seleção de Pontos de Ocorrências de Características Transversais (POCT) e não para sua criação. Por exemplo, quanto maior a notação NSPE, maior é o esforço para compreender o significado correto de um elemento, cuja semântica é o oposto de sua aparência. Assim, afirmamos que especificações que apresentam NSPE/NE maior que zero são intuitivamente mais difíceis de entender, exigindo um maior esforço cognitivo.

Posto isto, como o PoN visa a sintaxe das notações visuais (não a semântica), acreditamos que seus princípios podem ser aplicados às linguagens textuais no que se refere ao entendimento da especificação. Portanto, criamos algumas analogias relacionadas aos princípios do PoN (por meio da suíte de métricas MS4PoN) para aplicá-las às notações textuais. Por exemplo, uma linguagem textual pode apresentar as palavras-chave e símbolos que são semanticamente imediatos, por exemplo, "before()"na AspectJ. Sendo assim, ao analisarmos os resultados percebe-se que a JSD apresenta proporções melhores que as outras técnicas quanto a avaliação de suas propriedades perceptivas, sendo os resultados apresentados de forma empírica onde os indícios apontam que a JSD tem uma melhor eficácia cognitiva quanto a suas propriedades de percepção, i.e., sintaxe.

As implicações práticas dos resultados relatados dizem respeito à compreensibilidade das especificações de seleção de POCT. Quanto mais complexo se torna o software embarcado, maior é a quantidade de POCT. Assim, é essencial que uma especificação possa lidar com os pontos onde se devem tratar as características transversais. Portanto, escolher uma notação adequada para especificar a seleção de POCT é fundamental para evitar (ou atenuar) os problemas associados a mal-entendidos, especialmente na fase de implementação.

Por fim, vale ressaltar que o modelo de qualidade criado e conjunto de métricas MS4PoN fornecem meios de reunir evidências científicas em vez de conjecturas e opiniões subjetivas para avaliar uma notação, tendo em vista que a Física das Notações (PoN) foi usada majoritariamente para análise qualitativa e, no entanto a presente dissertação propõe algo mais científico e menos subjetivo. Até onde sabemos, este é o primeiro trabalho que propõe um conjunto de métricas quantitativas com base no PoN.

7 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho contribui para a melhoria de qualidade da especificação de seleção de Pontos de Ocorrências de Características Transversais (POCT) através da proposição de uma nova técnica para especificar seleções de POCT em software embarcado. Para tal, esta dissertação avalia a compreensão da especificação de seleções de POCT de forma indireta, através da avaliação das propriedades de percepção de notações gráficas e textuais para seis técnicas distintas (AspectJ, JPDD, Theme/UML, Tracematch, AspectOCL e JSD). Para isso, foram propostos um modelo de qualidade e um conjunto de métricas baseados nas premissas do framework conceitual *Physics of Notations* (PoN) [Moody 2009].

Os resultados experimentais mostraram evidências empíricas de que a especificação da seleção de POCT através da JSD é melhor que a JPDD pois apresenta um melhor discernimento na eficácia cognitiva das representações visuais de acordo com a ótica das Físicas das Notações (PoN). Por exemplo, um número alto de elementos semanticamente perversos (NSPE) foram encontrados nas notações gráficas avaliadas. Na JPDD e particularmente Theme/UML, a especificação da seleção de POCT inclui elementos ou símbolos cuja natureza possui um falso mnemônico. A JPDD apresenta um valor de 0.237 elementos perversos por elemento sintático (NSPE/NE), i.e., para cada elemento sintático usado na especificação 0.237 possui um falso mnemônico associado. A Theme/UML se apresentou com o mesmo comportamento, porém com um valor um pouco menor de 0.041. Quanto a proporção NSPE/NLE, esse valor também é alto para as notações visuais, e isso é ruim, e.g., a JPDD apresentou um valor de 0.291 elementos perversos por expressão lógica. Isso pode induzir um engenheiro iniciante a inferir um significado diferente do que se apresenta na especificação.

Por outro lado, a JSD apresentou zero elementos perversos em suas especificações, além disso, o número de elementos semanticamente imediatos (NSIE/NE) se apresentou melhor na JSD que na JPDD, i.e., enquanto a JPDD faz uso de zero elementos que possuem um mnemônico que sugere o seu significado, a JSD faz uso de 0.184 elementos semanticamente imediatos por elemento sintático (NSIE/NE), como também faz uso de 0.531 elementos semanticamente imediatos por expressão lógica (NSIE/NLE).

A JSD também se beneficia por não apresentar nenhum elemento sinógrafo (NS) ou homógrafo (NH) que são anomalias relacionadas a clareza semiótica. A primeira indica elementos redundantes numa especificação e a outra indica uma sobrecarga de símbolos, podendo gerar

confusão no entendimento da especificação de seleção de POCT, e consequentemente gerar ambiguidades e baixa eficácia cognitiva.

Portanto, conclui-se que a JSD se beneficia melhor quanto as vantagens esperadas de uma notação visual para a especificação de seleção de POCT que as técnicas avaliadas. Assim, pode-se afirmar que a JSD é uma técnica que contribui e inova quanto ao uso apropriado dos benefícios de uma linguagem visual, como também ao uso apropriado dos benefícios de uma linguagem textual através da aplicação do principio de codificação dupla [Moody 2009]. Sendo esta a maior contribuição desta dissertação pois o objetivo deste trabalhou que se originou a partir da hipótese de compreensão ambígua da JPDD dentro da AMoDE-RT, é de substituir a JPDD pela JSD na especificação de seleção de POCT para software embarcado.

Além da contribuição principal deste trabalho de criação de uma nova técnica de especificação de seleção de POCT, esta dissertação contribuiu quanto a originalidade e inovação na criação de um novo modelo de qualidade baseado no framework Física das Notações (PoN), como também na criação de um conjunto de métricas denominado *Metrics Suite for PoN (MS4PoN)* combinadas no modelo de qualidade proposto. Até onde sabemos, este trabalho é o primeiro a avaliar notações distintas para seleção de POCT seguindo as premissas do framework da Física das Notações (PoN). Outro ponto é que, nenhum outro estudo empírico apresentado no estado da arte da engenharia de software analisa as propriedades perceptivas de notações relacionadas ao Desenvolvimento de Software Orientado a Aspectos (AOSD). Essa contribuição gerou uma publicação de um artigo no X Simpósio Brasileiro de Engenharia de Sistemas Computacionais (SBESC) [Oliveira e Wehrmeister 2020].

Como trabalho futuro, planeja-se avaliar outras notações gráficas e textuais para especificar a seleção de POCT mais complexas. (e.g., sistemas de manufatura inteligente, sistemas de medição inteligente de energia, aplicações na indústria 4.0). Além disso, planeja-se aumentar o escopo para além do software embarcado e avaliar as propriedades perceptivas de seleção de POCT no desenvolvimento de software geral. Outro objetivo futuro é aplicar a MS4PoN em outros contextos que não seja especificação de seleção de POCT e assim gerar mais trabalhos empíricos de avaliação de propriedades perceptivas (sintaxe) de uma forma sistemática onde a aplicação pode englobar não só os POCT, mas também outros contextos onde podem ocorrer características transversais como Padrões de Projeto de Software (Em inglês, Design Patterns) entre outros.

Por fim, planeja-se se aprofundar quanto ao estudo da eficácia cognitiva e propriedades de percepção em notações visuais da engenharia de software fazendo comparações com a cartografia, almejando evoluir tanto a técnica criada quanto o modelo de qualidade e métricas (MS4PoN) dentro do contexto de Engenharia de Sistemas.

REFERÊNCIAS

ACREŢOAIE, Vlad; STÖRRLE, Harald; STRÜBER, Daniel. VMTL: a language for end-user model transformation. **Software & Systems Modeling**, Springer, v. 17, n. 4, p. 1139–1167, 2018.

ALLAN, Chris *et al.* Adding trace matching with free variables to aspectj. *In*: . *[S.l.: s.n.]*, 2005. v. 40, p. 345–364.

BANIASSAD, Elisa; CLARKE, Siobhan. Theme: An approach for aspect-oriented analysis and design. *In*: IEEE. **26th Int. Conf. on Software Engineering**. *[S.l.]*, 2004. p. 158–167.

BOWERS, Kate M.; FREDERICKS, Erik M.; HARIRI, Reihaneh H.; H. C. Cheng, Betty. Providentia: Using search-based heuristics to optimize satisficement and competing concerns between functional and non-functional objectives in self-adaptive systems. **Journal of Systems and Software**, v. 162, p. 110497, 2020. ISSN 0164-1212. Disponível em: https://www.sciencedirect.com/science/article/pii/S0164121219302717.

CABOT, Jordi; GOGOLLA, Martin. Object constraint language (ocl): A definitive guide. *In*: . *[S.l.: s.n.]*, 2012. v. 7320, p. 58–90.

CLARKE, Siobhán. Extending standard uml with model composition semantics. **Science of Computer Programming**, Elsevier, v. 44, n. 1, p. 71–100, 2002.

CLARKE, Siobhàn; BANIASSAD, Elisa. **Aspect-oriented analysis and design**. *[S.l.]*: Addison-Wesley Professional, 2005.

DEVRIES, B.; CHENG, B. Goal-based modeling and analysis of non-functional requirements. *In*: **2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)**. *[S.l.: s.n.]*, 2019. p. 261–271.

ELRAD, Tzilla *et al.* Aspect-oriented modeling: Bridging the gap between implementation and design. *In*: BATORY, Don *et al.* (Ed.). **Generative Programming and Component Engineering**. *[S.l.]*: Springer, 2002. p. 189–201. ISBN 978-3-540-45821-0.

FILMAN, Robert; ELRAD, Tzilla; CLARKE, Siobhán; AKsIT, Mehmet (Ed.). **Aspect-Oriented Software Development**. First. *[S.l.]*: Addison-Wesley Professional, 2004. ISBN 0321219767.

Freitas, Edison Pignaton *et al.* DERAF: A high-level aspects framework for distributed embedded real-time systems design. *In*: **Early Aspects: Current Challenges and Future Directions**. Berlin, Heidelberg: Springer, 2007. p. 55–74. Doi: 10.1007/978-3-540-76811-1_4.

HANNOUSSE, A. A development process for component software with crosscutting and dynamic features. *In*: **2019 International Conference on Theoretical and Applicative Aspects of Computer Science (ICTAACS)**. *[S.l.: s.n.]*, 2019. v. 1, p. 1–8.

JONES, David; GREGOR, Shirley. The anatomy of a design theory. **Journal of the Association for Information Systems**, v. 8, n. 5, p. 1, 2007.

KHAN, Muhammad Uzair *et al.* AspectOCL: using aspects to ease maintenance of evolving constraint specification. **Empirical Software Engineering**, Springer, v. 24, n. 4, p. 2674–2724, 2019.

KICZALES ERIK HILSDALE, Jim Hugunin Mik Kersten Jeffrey Palm William G. Griswold Gregor. An overview of aspectj. *In*: **Proceedings of European Conference on Object-Oriented Programming (ECOOP)**. *[S.l.: s.n.]*, 2001.

KICZALES, Gregor *et al.* Aspect-oriented programming. *In*: **Proc. of European Conference on Object-Oriented Programming (ECOOP)**. Springer-Verlag. LNCS 1241: *[s.n.]*, 1997.

KICZALES, Gregor *et al.* An overview of aspectj. *In*: SPRINGER. **European Conference on Object-Oriented Programming**. *[S.l.]*, 2001. p. 327–354.

KIENZLE, Jörg; ABED, Wisam Al; KLEIN, Jacques. Aspect-oriented multi-view modeling. *In*: **Proc. International Conference on Aspect-Oriented Software Development**. *[S.l.]*: ACM, 2009. p. 87–98. ISBN 9781605584423.

KITCHENHAM, Barbara. **Procedures for Performing Systematic Reviews**. 2004.

LINDEN, D. van der; HADAR, I. A systematic literature review of applications of the physics of notations. **IEEE Transactions on Software Engineering**, IEEE Computer Society, Los Alamitos, CA, USA, v. 45, n. 08, p. 736–759, aug 2019. ISSN 1939-3520.

MONGIOVI, Misael; PAPPALARDO, Giuseppe; TRAMONTANA, Emiliano. Specifying and identifying widely used crosscutting concerns. **Knowledge-Based Systems**, v. 126, p. 20 – 32, 2017. ISSN 0950-7051.

MOODY, Daniel L. The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering. **IEEE Transactions on Software Engineering**, v. 35, n. 6, p. 756–779, 2009.

Object Management Group (OMG). **Unified Modeling Language version 2.5.1**. 2017. Https://www.omg.org/spec/UML/2.5.1/.

Object Management Group (OMG). **UML Profile for Modeling and Analysis of Real-Time Embedded systems (MARTE), version 1.2**. 2018. Https://www.omg.org/spec/MARTE/.

OLIVEIRA, Rodrigo de; WEHRMEISTER, Marco Aurélio. Avaliando a especificação das ocorrências das características transversais em software embarcado. *In*: **Anais Estendidos do X Simpósio Brasileiro de Engenharia de Sistemas Computacionais**. Porto Alegre, RS, Brasil: SBC, 2020. p. 9–16. ISSN 0000-0000. Disponível em: https://sol.sbc.org.br/index.php/sbesc_estendido/article/view/13085.

ROQUE, Alexandre dos Santos; POHREN, Daniel; FREITAS, Edison Pignaton; PEREIRA, Carlos Eduardo. An approach to address safety as non-functional requirements in distributed vehicular control systems. **Journal of Control, Automation and Electrical Systems**, Springer, v. 30, n. 5, p. 700–715, 2019.

SCHMIDT, Douglas C. Model-driven engineering. **Computer-IEEE Computer Society-**, Citeseer, v. 39, n. 2, p. 25, 2006.

STEIN, Dominik *et al.* Join point designation diagrams: a graphical representation of join point selections. **Int. Journal of Software Engineering and Knowledge Engineering**, World Scientific, v. 16, n. 03, p. 317–346, 2006.

WEHRMEISTER, Marco Aurélio. **An Aspect-Oriented Model-Driven Engineering Approach for Distributed Embedded Real-Time Systems**. 2009. Tese (Doutorado) — Universidade Federal do Rio Grande do Sul, 2009.

WEHRMEISTER, Marco Aurélio; de Freitas, Edison Pignaton; BINOTTO, Alécio Pedro Delazari; PEREIRA, Carlos Eduardo. Combining aspects and object-orientation in model-driven engineering for distributed industrial mechatronics systems. **Mechatronics**, v. 24, n. 7, p. 844 – 865, 2014. ISSN 0957-4158. Doi: 10.1016/j.mechatronics.2013.12.008.

Wehrmeister, M. A.; Pereira, C. E.; Rammig, F. J. Aspect-oriented model-driven engineering for embedded systems applied to automation systems. **IEEE Transactions on Industrial Informatics**, v. 9, n. 4, p. 2373–2386, 2013. Doi: 10.1109/TII.2013.2240308.

WIERDA, G. **A Serious Introduction to the Archimate (r) Enterprise Architecture Modeling Language**. *[S.l.]*: R&a, 2014.

WIMMER, Manuel *et al.* A survey on uml-based aspect-oriented design modeling. **ACM Computing Surveys**, ACM, New York, NY, USA, v. 43, n. 4, p. 28:1–28:33, out. 2011. ISSN 0360-0300. Disponível em: http://doi.acm.org/10.1145/1978802.1978807.

YANG, SU; WEI-DONG, ZHONG. Aspect-oriented modeling in concurrent system. *In*: IEEE. **IEEE Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)**. *[S.l.]*, 2019. p. 836–840.

YOUNAS, Muhammad; JAWAWI, Dayang NA; GHANI, Imran; SHAH, Muhammad Arif. Extraction of non-functional requirement using semantic similarity distance. **Neural Computing and Applications**, Springer, v. 32, n. 11, p. 7383–7397, 2020.

YU, Yijun; LEITE, Julio CSP; MYLOPOULOS, John. From goals to aspects: discovering aspects from requirements goal models. *In*: IEEE. **Proceedings. 12th IEEE International Requirements Engineering Conference, 2004.** *[S.l.]*, 2004. p. 38–47.

ZHOU, Z.; ZHI, Q.; MORISAKI, S.; YAMAMOTO, S. An evaluation of quantitative non-functional requirements assurance using archimate. **IEEE Access**, v. 8, p. 72395–72410, 2020. ISSN 2169-3536.

ZUBCOFF, Jose; GARRIGóS, Irene; CASTELEYN, Sven; MAZóN, Jose-Norberto; AGUILAR, Jose-Alfonso; GOMARIZ-CASTILLO, Francisco. Evaluating different i*-based approaches for selecting functional requirements while balancing and optimizing non-functional requirements: A controlled experiment. **Information and Software Technology**, v. 106, p. 68–84, 2019. ISSN 0950-5849. Disponível em: https://www.sciencedirect.com/science/article/pii/S0950584917300770.

ANEXO A – EXEMPLOS DE ESPECIFICAÇÕES DE SELEÇÕES DE POCT PARA SOFTWARE EMBARCADO

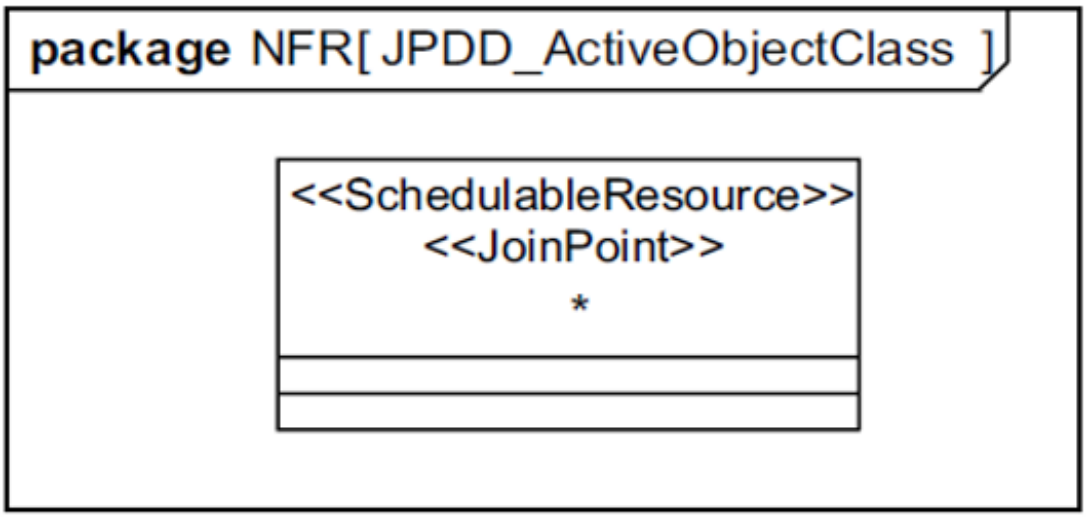
Este capítulo apresenta 12 exemplos das especificações de 2 POCT utilizados nos experimentos, representando, respectivamente um caso de POCT simples e outro complexo. As especificações são descritas em AspectJ, Tracematch, AspectOCL, JPDD, Theme/UML, e JSD.

Figura 47 – AspectJ para seleção de objetos ativos

```
1  aspect TiminingAttributes {
2
3      pointcut pcActClass(): execution(SchedulableResource.new(..)) && !
           within(TiminingAttributes);
4
5      ActiveObjectClass activeObjectClass = new ActiveObjectClass();
6
7      before(): pcActClass() {
8          activeObjectClass.selectActiveObjectClass();
9      }
10 }
11
12 class ActiveObjectClass {
13
14     public void selectActiveObjectClass() {
15
16         try {
17             URLClassLoader classLoader = new URLClassLoader(
18                 new URL[] {
19                     new File("SchedulableResource.jar").toURI().toURL()
20                 },
21                 ClassLoader.getSystemClassLoader());
22
23             Class << ? >> myImplementation = classLoader.loadClass("
                   SchedulableResource.class");
24
25         } catch (MalformedURLException | ClassNotFoundException e) {
26
27         }
28     }
29 }
```

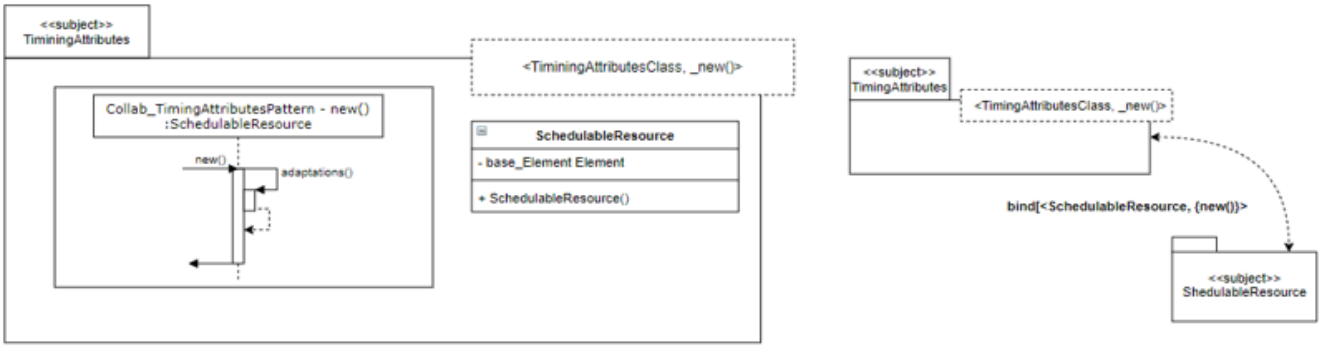
Fonte: o autor

Figura 48 – JPDD para seleção de objetos ativos



Fonte: o autor

Figura 49 – Theme/UML para seleção de objetos ativos



Fonte: o autor

Figura 50 – Tracematch para seleção de objetos ativos

```
1  tracematch() {
2      sym pcActClass before: execution(SchedulableResource.new(..));
3
4      pcActClass {
5          try {
6
7              URLClassLoader classLoader = new URLClassLoader(
8                  new URL[] {
9                      new File("SchedulableResource.jar").toURI().toURL
                           ()
10                 },
11                 ClassLoader.getSystemClassLoader()
12             );
13             Class << ? > myImplementation = classLoader.loadClass("
                   SchedulableResource.class");
14
15         } catch (MalformedURLException | ClassNotFoundException e) {
16
17         }
18
19         //Crosscutting Concerns Implementation
20     }
21 }
```

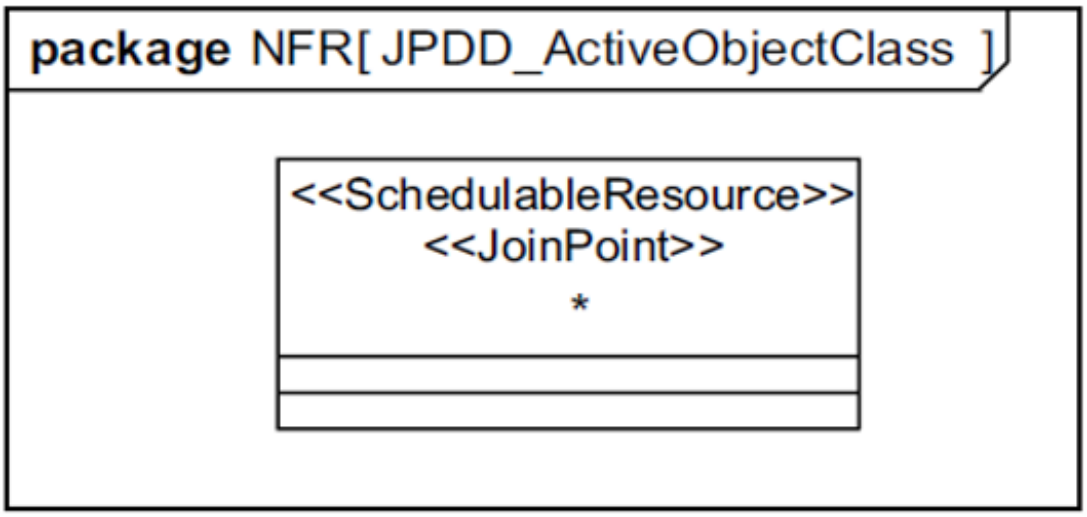
Fonte: o autor

Figura 51 – AspectOCL para seleção de objetos ativos

```
1  mapping mapTiminingAttributes
2     {
3          let T: {SchedulableResource}
4     }
5
6     aspect TiminingAttributes
7     {
8          import_mapping mapTiminingAttributes
9
10         pointcut pctActClass
11         context T :
12         inv:
13         self.allOwnedElements()
14         ->select(s | s.getAppliedStereotype("SchedulableResource"))
15    }
```

Fonte: o autor

Figura 52 – JSD para seleção de objetos ativos



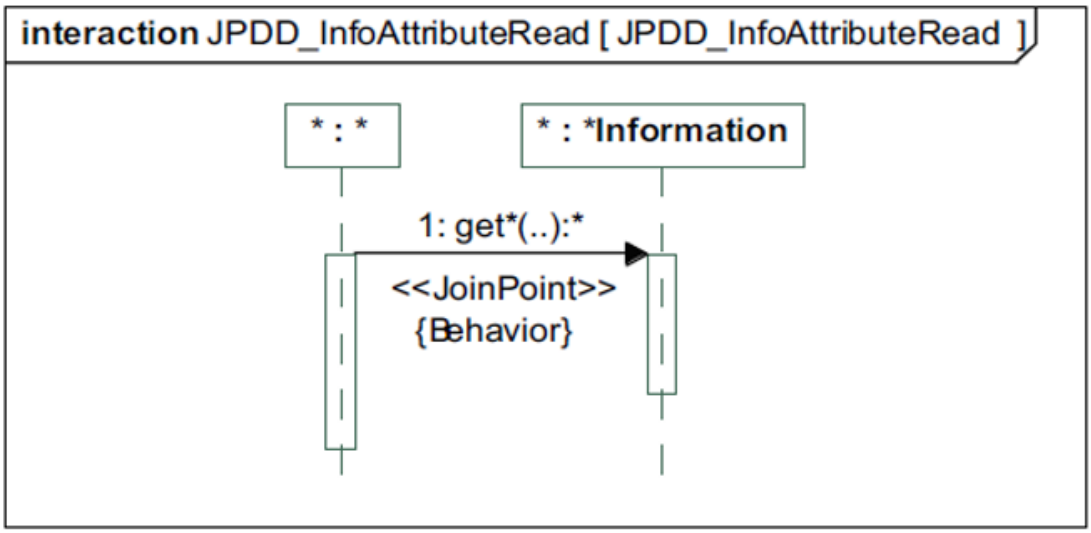
Fonte: o autor

Figura 53 – AspectJ seleção de mensagens cujo nome começa com “get"

```
1  aspect DataFreshness {
2      pointcut pcReadAttrValue(): call(* *Information.get(..)) && !within
           (DataFreshness);
3
4      before(): pcReadAttrValue() {
5          //Crosscutting Concerns Implementation
6      }
7  }
```

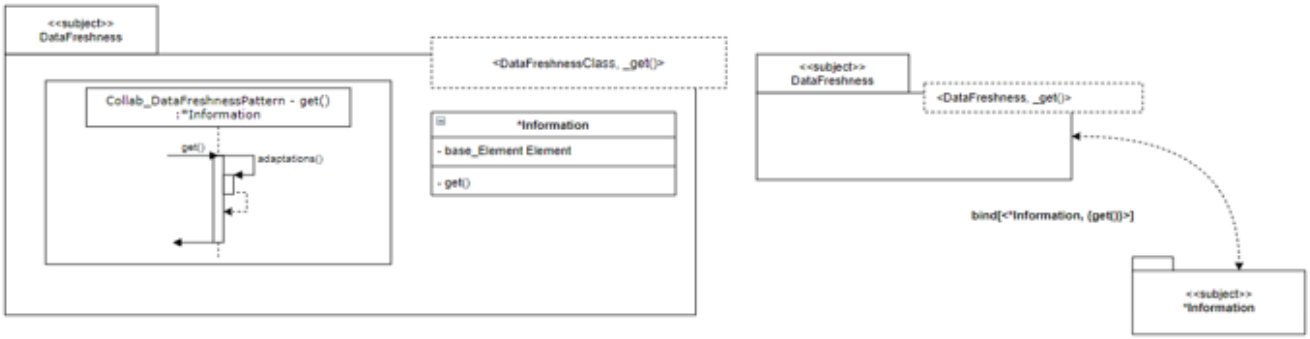
Fonte: o autor

Figura 54 – JPDD seleção de mensagens cujo nome começa com “get"



Fonte: o autor

Figura 55 – Theme/UML seleção de mensagens cujo nome começa com “get"



Fonte: o autor

Figura 56 – Tracematch seleção de mensagens cujo nome começa com “get"

```
1  tracematch(){
2   sym pcReadAttrValue before: call(* *Information.get(..))
3
4   pcReadAttrValue
5   {
6    //Crosscutting Concerns Implementation
7   }
8  }
```

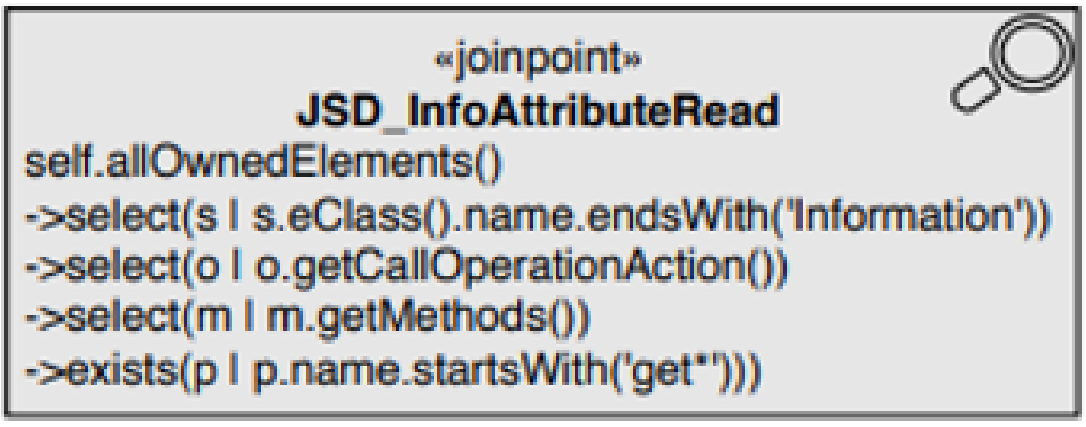
Fonte: o autor

Figura 57 – AspectOCL seleção de mensagens cujo nome começa com “get"

```
1  mapping mapTiminingAttributes
2     {
3          let T: {SchedulableResource}
4     }
5
6     aspect TiminingAttributes
7     {
8          import_mapping mapTiminingAttributes
9
10         pointcut pcReadAttrValue
11         context T :
12         inv:
13           self.allOwnedElements()
14         ->select(n | n.eClass().name.endsWith('Information'))
15         ->select(o | o.getOperations())
16         ->select(m | m.getMethods()
17         ->exists(p | p.name.startsWith('get')))
18    }
```

Fonte: o autor

Figura 58 – JSD seleção de mensagens cujo nome começa com “get"



Fonte: o autor

ANEXO B – MAPEAMENTO SISTEMÁTICO DE LITERATURA

Specifying and Identifying Points That May Have Crosscutting Concerns: A Systematic Mapping

Rodrigo S. C. de Oliveira[1], **Marco A. Wehrmeister**[2], **Douglas P. B. Renaux**[3]

[1] DAINF – Universidade Tecnológica Federal do Paraná (UTFPR)

{rodrigoo}@alunos.utfpr.edu.br, {wehrmeister}@utfpr.edu.br,
{douglasrenaux}@utfpr.edu.br

Abstract. *[Context] Comprehensibility is one of the major challenges when it comes to the identification and specification of Crosscutting concerns. Although there are many works in literature that propose a new approach to identify and specify either graphically or textually crosscutting concerns, there is a lack of documentation and experiments in those studies that prevents the empiricism of those methods and techniques. [Objective] to identify, collect and evaluate the main techniques and methods used in the identification and specification of crosscutting concerns, with the aim to provide a synthesis of the challenges and problems of the related works contributing with the subject of the creation of a technique to specify points that may have crosscutting concerns. [Method] the systematic mapping was conducted using 4 electronic databases. We have performed a manual search following a plan with the aim to classify the relevant studies. [Results] we have found that most of the researches seek to create an abstract way of the specification of crosscutting concern that could be understood independent of the programming language used. Also, we have discovered that the current approaches of the identification of Crosscutting concerns have little or none acceptance in the industry. [Conclusion] based on our results we can arrive at the conclusion that the identification and specification of crosscutting concern is a very explored subject in scientific literature but with a low precision in the critical analysis results and with a tendency to subjectivity. Besides the techniques created, there was also an interest from researchers to find the appropriated means to designate the sets of join points. We have found that this interest in this field of research generated different techniques that comes with most of the techniques created to specify such crosscutting concerns.*

1. Introduction

The identification and specification of crosscutting concerns is a process that tries to locate and recognize points in the code that may have crosscutting concerns without using any aspect-oriented programming language. It can be used in legacy software systems or in a new software system that will be developed. The approach that is being used for the legacy system is known as Aspect Mining and the approach for future system development is known as Early Identification. Aspect Mining is concerned with the development of tools that aims to search and identify points that are crosscutting concerns in a legacy software system. On the other hand, Early Identification is concerned with the development of tools that aims to specify point that may have crosscutting concerns in a future software implementation, hence it tries to identify points in a prior stage of software development process, for example in the requirements phase.

The identification and specification of crosscutting concerns can grow in complexity according to the business rules and/or the nonfunctional requirements of the system, and to support that, it could be more efficient if we have a good technique or language to identify and specify the crosscutting concerns in a concise way. This subject is still being explored by researchers around the globe in the context of Aspect-oriented Programming (AOP) [Gregor Kiczales 1997] and Aspect-oriented Modeling (AOM) [Elrad et al. 2002], and there are many works with the aim to identify and specify crosscutting concerns. The majority of the works in literature tends to be divided into two types of specification, the graphical and textual. The specification of crosscutting concerns through graphical mode it's an area of research that has been very explored through the years. One of the main approaches created to identify and specify points that mat have crosscutting concerns through graphical mode is the Join Point Designation Diagram (JPDD). [Dominik Stein 2006] JPDD is a graphical notation that has the responsibility to express the join point selection (Also called pointcuts) independent of the aspect-oriented system or language. The graphical notation uses the Unified Modeling Language (UML). [OMG 2003]. Despite the fact that UML is very accepted in the industry when it comes to identification and specification of crosscutting concerns its semantics are not completely formalized due to the fact that it was created in the context of Object Oriented Programming, thus resulting in a quite imprecise specification with difficult maintenance. [Muhammad Uzair Khan 2015]

Originally developed by Stein et all 2009, JPDD is a visual means specification of join point selection that has been used in some works with the purpose of code generation. [Wehrmeister 2009]. It is based on UML, but as UML does not use a fully formalized semantics, it leads to ambiguity in the constraints producing an imprecise artifact in auto code generation. The model adopts action symbols to represent system events that need to come to pass in order to reach a join point. In Figure 1 that was taken from [Stein and Hanenberg 2011] you can see an example related to the specification of a join point selection that aims to ensure the safe usage of iterators.

Figure 1 above represents a join point selection using a JPDD. Basically, the JPDD selects all method invocations (?jp) of a method called "next" which are invoked on a target instance of type "Iterator". That target instance of type "Iterator" must originate from a method call to a method named "iterator" which must be invoked on a target instance of type "DataSource". Furthermore, there must be an invocation of a method named "update" (addressed to the same instance as the invocation of method "iterator") which occurs in between the two method calls (of method "iterator" and of method "next"). Apart from the selected method invocation (?jp), the JPDD exposes the object (?o) which is returned by the invocation of method "update".

Therefore, one of the issues mentioned in the literature is that the visual specification of join point selection through graphical mode generates artifacts of difficult comprehension and maintenance [Muhammad Uzair Khan 2015], hence, this kind of representation indeed has an impact on the comprehensibility of identification and specification of points that may have crosscutting concerns. In this work, we present a Systematic Mapping (SM) that identifies the main techniques and methods used in the identification and specification of crosscutting concerns and also identifies the problems and open points of the related works.

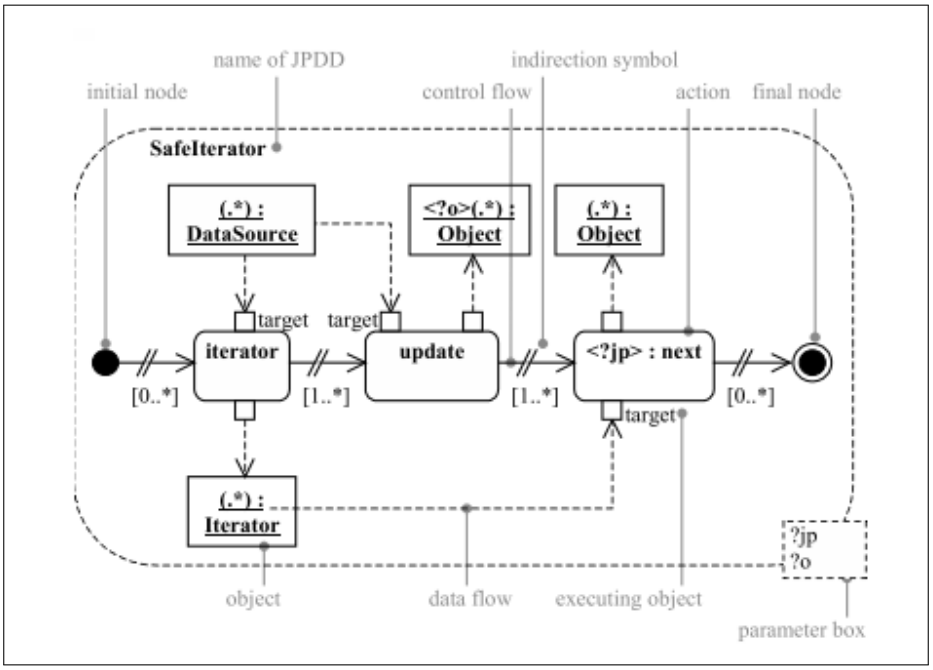


Figure 1. JPDD Model

We also propose a Taxonomy from the studies found, following all the steps of Identification and Specification of Crosscutting Concerns.

A mapping study presented in this paper investigates the following issues:

(i) The methods used to identify and specify point crosscutting concerns
(ii) The algorithms used
(iii) The challenges and problems in each method
(iv) The solution for each challenge and problems

The remainder of this paper is structured as follows. Section 2 presents related research. Section 3 discusses the research method applied to perform the mapping study. Results are presented in Section 4. Section 5 discusses the results, their implications, and limitations. Finally, Section 6 concludes the paper and presents directions for future work

2. Background

Crosscutting concerns are the concerns which affect or crosscut other concerns. Basically, they are conceptual pieces of functionality that have their implementation tangled and/or scattered among various modules in a software system. Code scattering refers to the code which is spread across modules of the software system and code tangling refers to the code which is mixed with other code. Example of crosscutting concerns is Logging, Persistence Caching, Memory Management, Synchronization among others. The consequence of having crosscutting concerns in the code is that it leads the program to be difficult to maintain due to the hard comprehension of what is implemented. To mitigate this problem, there was created a paradigm called Aspect-oriented Programming [Gregor Kiczales 1997] with the aim of encapsulating the crosscutting concern in a modular unit of code and separate the core functionality of a software system from concerns that

have a more system-wide behaviour, in that way increasing its modularity through code isolation and code reuse.

Inside the context of Aspect-oriented programming paradigm, there is the concept of the so-called join points. Join points has a degree of importance in the application of the Model Driven Software Engineering paradigm. [Markus Völter 2013]. They are points in the base program that has the responsibility to inform the program (through interception) in the control flow where the crosscutting code unit shall take effect. The set of join points that has the responsibility to advise the program and intercept the base functionality are known as join point selection (also called pointcuts). For instance, let's consider the pointcut implemented in the code below. This code was taken from [Gregor Kiczales 1997] and had a little change in the naming conventions. It uses AspectJ that it is a Java language programming extension to specify crosscutting concerns as aspects. Basically, this code unit implements a simple error logging that says that if any public method defined inside the package com.landis.endpoints logs any errors, it throws back to its responsible caller:

Listing 1. AspectJ example

```
aspect ErrorLogging {
  Log log = new Log();

  pointcut publicCalls():
        receptions(public * com.landis.endpoints.*.*(..));

  after() throwing (Error e): publicCalls() {
    log.write(e);
  }
}
```

The Listing 1 is just a simple example of pointcuts, the purpose is only to understand what are join points in the context of crosscutting concerns. Under the circumstances of modern software systems, the expression of join point selection can grow in complexity if it involves the selection of points that refer to multiple system events in the execution of a program, consequently, express such points that may have crosscutting concerns have been a subject of extensive research in the domain of identification and specification of crosscutting concerns and it is believed that they have better comprehensibility than their conventionally codified equivalents. The current problem in the identification and specification of crosscutting concerns using graphical or textual mode is that for both there is a tendency to create complex artifacts, difficult to read and maintain as the software system grows in complexity and this is the reality of modern technology projects.

To address this problem it is necessary to improve the current techniques through experiments with the aim to create a proper common benchmark and also filter what is better in each technique or method and presumably create a new technique or method that does the identification and specification of points that may have a crosscutting concerns

in a concise, clear, modularized way what could be understood simply by the software developers and engineers.

2.1. Related Work

Mapping studies are secondary studies that have the objective to identify and classify the content related to the subject of the research that could point subtopics where more primary studies are needed [Kitchenham 2004]. The systematic mapping usually starts from a search string that must be applied in the electronic databases choose. The search string presented in Table 1 was applied in the respective database: *IEEE Xplore, Scopus, ACM Digital Library and Springer*. The result of the search was a total of 563 studies. After eliminating duplication's and applying the selection criteria, 104 studies were returned.

Among these studies, there was found some interesting secondary studies. In [McFadden and Mitropoulos 2013] the authors examine the aspect mining literature in order to consolidate a list of existing quality measures that validates the effectiveness used across different aspect mining techniques. They conclude that there is a critical lack of standard benchmarks for aspect mining, making it difficult to evaluate the techniques through empirical validation saying that this lack of proper benchmarks impacts the objective comparison and empirical validation of aspect mining research. In [Khan and Jaffar-ur-Rehman 2005] the authors create a survey that deals with the open issues and observations of the Aspect-Oriented Requirements Engineering (AORE) models where each AORE model was analyzed to see what level of systematic activities of separation of concerns they perform. They have identified some deficiencies in AORE models which needed to be addressed. In [Ali and Kasirun 2008] the authors review the existing approaches on the effectiveness of the techniques responsible to identify crosscutting concerns in a semi-automated way with the aim to solve the issue of human intervention in that process. Based on their studies they conclude that on all existing methods and techniques for crosscutting concerns identification, none of them can do the identification automatically. In [Marçal et al. 2016a] the authors proposed a systematic literature review (SLR) arguing that there are few works that provide analysis, synthesis, and documentation of the aspect mining literature. They conclude that most techniques for identification and specification of crosscutting concerns provide a poor evaluation in their results indicating a high number of false-positives. Also, they argue that the lack of experiments prevents establishing common benchmarks for aspect mining. In [Kellens et al. 2007a] the authors presented a survey that provides a comparison of the current aspect mining tools and techniques focusing only on automated techniques that mine implemented code for candidate aspects. The main contribution of the paper was an initial comparative framework and also a taxonomy for distinguishing aspect mining techniques. In [Ali et al. 2010] the authors provide a systematic review that aimed to identify, analyze and report the evidence published in the literature that compares the AOP approaches with Non-AOP approaches. They have identified 22 studies that have empirical evidence of the AOP to Non-AOP comparison and they conclude that AOP is likely to have a positive effect on performance, code size, modularity, and evolution. Among these secondaries studies, there was one that attracts our attention. [Durelli et al. 2013] conduct a pooled systematic review of 62 studies to describe characteristics of each mining technique for crosscutting concern, generated precise indicators of validity, and extended a taxonomy adapted from [Kellens et al. 2007a].

The fields of the studies analyzed were very homogeneous, and most of them were classified as direct to software engineering research (90%). Others where in the field of computer science (10%). Based on the results from this investigation, we can say that there is an extensive literature focused in identification and specification of crosscutting concerns but most of them with a poor evaluation of their results accompanied by lack of documentation and lack of experiments that prevents the creation of a common benchmark resulting in no empirical validation. [Marçal et al. 2016a] This motivated us to conduct this study.

Table 1
Keywords of the Search String

Areas	Keywords
Join Point	”join point selection”, ”joinpoint selection”, ”join point designation diagram”
Crosscutting Concern	”identifying crosscutting concerns”, ”specifying crosscutting concerns”
String de Busca	(”join point selection” OR ”joinpoint selection” OR ”join point designation diagram” OR ”identifying crosscutting concerns” OR ”specifying crosscutting concerns”) AND (”systematic mapping” OR ”SLR” OR ”review” OR ”survey”) OR ”join point selection” OR ”joinpoint selection” OR ”join point designation diagram” OR ”identifying crosscutting concerns” OR ”specifying crosscutting concerns”

3. Mapping Study

The purpose of the mapping is to develop a classification process with the aim to organize the categories of the research. The search string method for the systematic mapping presented in this article was developed based on the guidelines suggested by [Kitchenham 2004]. The mapping process involves three main phases: Review Planning, Conduct of Review, and Publication of Results.

3.1. Research Questions

This Systematic Mapping (SM) was conducted with the aim to identify the main techniques and methods used in the specification of points that may have crosscutting concerns, as well as to identify the relevant points or benefits and gaps of related works. Based on this objective, 3 research questions were defined:

- **RQ1:** When have the studies been published?
- **RQ2:** What are the methods and techniques used to identify and specify crosscutting concerns?
- **RQ3:** What are the challenges and problems in each method and technique?

3.1.1. Search Strategy

This section describes the strategy used to conduct a review of the studies in order to answer the research questions (RQs) raised in the first phase of the mapping. The strategy is described as follows:

3.2. Terms and Keywords

After analyzing meaningful words within the search context, it was found that the most appropriate and comprehensive keywords and terms to be used as a search string would be ”crosscutting concerns”, ”join point selections”, ”aspect mining”, ”early identification”.

3.2.1. Electronic Databases

The research process was carried out from 4 electronic databases. The process prioritizes the main sources used in the area of Computer Science that meet the main criteria related to the publication of articles such as quality in publications, a large number of journals, conferences, and regularity in their updates.

The application of the search string in the selected sources returned a total of 563 publications. Table 2 shows the sources considered for research and the total number of articles found in the process:

Table 2
Total de Papers found in the Electronic Databases

Source	Site	No
Springer	http://wwww.springerlink.com	223
Scopus	http://wwww.scopus.com	58
ACM Digital Library	http://wwww.dl.acm.org	27
IEEE Xplorer	http://wwww.ieeexplore.ieee.org	255

3.3. Selection Process

The selection criteria are fundamental to guarantee quality in the results obtained in an SM. The selection criteria are organized in Inclusion Criterion (IC) and Exclusion Criteria (EC). Table 3 presents these criteria:

Table 3
Selection Criteria

Inclusion Criterion (IC)	The study must be within the Computer Science context, describing the approaches, methods or techniques for identifying and specifying crosscutting concerns.
Exclusion Criterion (EC)	The study is not written in English. The study does not have an abstract. The study is just published as an abstract. The study is a short paper, white papers, theses, and papers that were not peer-reviewed The study is not in the scope of Computer Science, although it uses the word ”crosscutting”

The selection process was divided into five stages, as shown in Figure 2.

In the 1st stage duplicated studies were eliminated, resulting in 445 publications (reducing approximately 21%).In the 2nd stage, the selection criteria (inclusion and exclusion criteria) were applied for title, abstract and keywords, leading to 132 studies (reducing approximately 70%).In the 3rd stage, the selection criteria were applied considering the full text, resulting in a set of 104 studies (reducing approximately 21%). In summary, a total of 459 studies were excluded, a reduction of approximately 81.5% of the total selected. As a final result, we got those 104 studies to be analyzed in our systematic mapping.

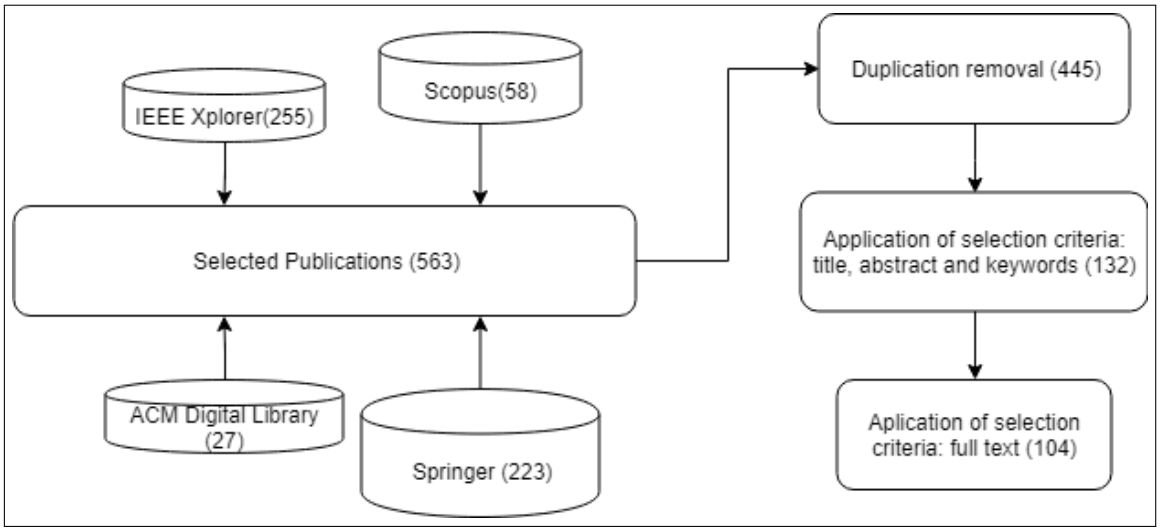


Figure 2. Selection mapping process

3.4. Classification and Data Extraction

The scope of classification and extraction of data used in this mapping is intended to answer the research questions defined in the process. The publications were classified

into categories of different facets: Techniques and Methods (RQ2), Reported Challenges and Problems (RQ3). Also, we have presented a taxonomy in Figure 3 that was taken from [Durelli et al. 2013]. The taxonomy was just changed in its format and also we just extended adding a new flow for Early Identification of Crosscutting Concerns.

4. Results and Discussion of the Mapping

In this section, we present the results of our systematic mapping based on the scope of the classification and data extraction to answer the RQs.

4.1. RQ1 - Frequency of Publication

In order to offer a general view of the efforts in the area of Identification and Specification of Crosscutting Concerns, a distribution of the total 563 selected studies over the years is shown in Figure 2. As this figure suggests, the research in the context of identifying and specify crosscutting concerns is not recent, the distribution of the papers in this chart suggest that in 1997 there was an interest in this subject and has been moderately increasing within 2002–2005, with a significant increase in 2007 and 2009. As the figure present, it can be seen that 2009 was the peak of interesting in this subject having an intermittent interesting within 2010-2015 and declines within 2016-2018.

Regarding publication vehicles, we noticed that there was a preference for conferences as the main communication channel, having approximately 70% (394 studies) of publications in conferences. The rest were articles published in Journals representing 30% (168 studies)

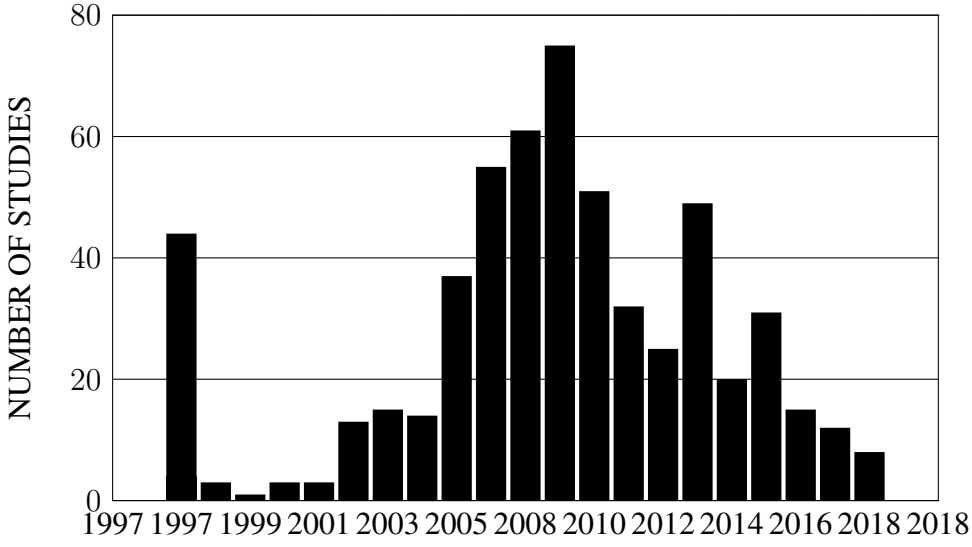


Fig. 2. Distribution of the selected studies over the years.

4.2. RQ2 - Techniques and Methods

We have identified several methods and notations in scientific literature used to identify and specify points that may have crosscutting concerns. The difference between those methods is that one uses diagrammatic means to specify the data constraints and other uses textual means for it. For example, Tracematches that was created by [Allan et al.

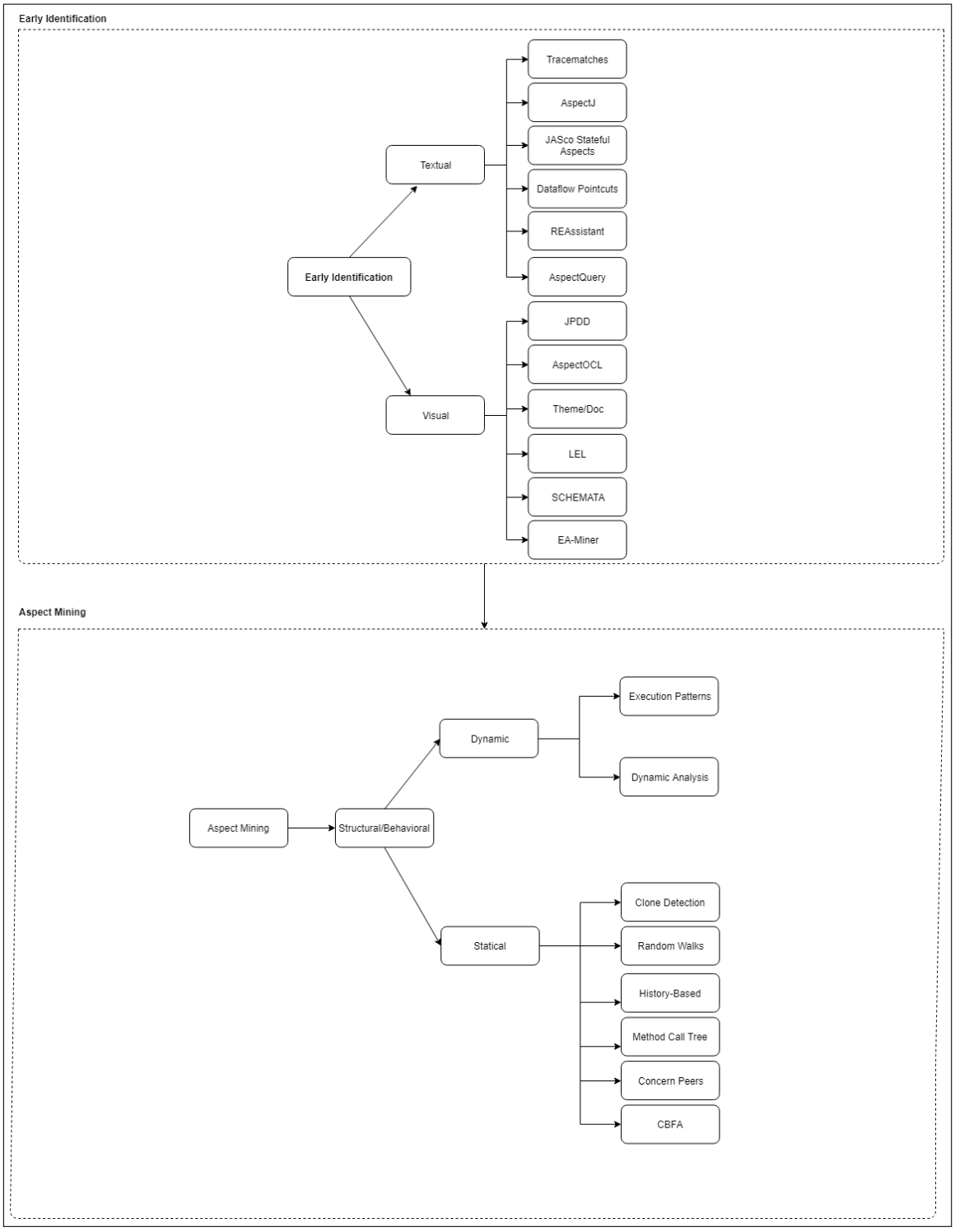


Figure 3. Crosscutting Concerns Specification Taxonomy

2005] and used in the work of [Stein and Hanenberg 2011] is a textual approach that implements regular expressions over execution traces of programs based on an alphabet of symbols, once there is an event that makes the regular expression match the execution trace, it should trigger the modular unit of crosscutting implementation to be executed. Although Tracematches innovate in its treatment of free variables in trace patterns, the use of it is a nontrivial approach. Also, there are a number of pragmatic issues that must be addressed as memory usage and performance optimizations. [Allan et al. 2005]

Another quite empirical method used to specify points in the execution program is called AspectJ [Gregor Kiczales 2001]. AspectJ is also a textual means represented as a language extension to Java that make possible to define additional implementation to the such called join points in the execution of the program. This is known in AspectJ as a dynamic crosscutting mechanism. The dynamic crosscutting mechanism in AspectJ is based on a small set of constructs such as join points, that are the points in the execution program that can have or already have a crosscutting concerns; pointcuts, that are the collection of those join points (Also called join point selections); advice, that are constructs used to define additional implementation at join points, and aspects, that are basically modular units of crosscutting implementation. Although AspectJ has been accepted as the basis of an empirical assessment of aspect-oriented programming and also being supported to important IDEs, the language has some limitations in the compiler when it uses javac instead of generating class file directly, when it requires access to all the source code for the system and when it performs a full recompilation whenever any parts of the user program changes.

In JAsCo [Vanderperren et al. 2005] we also found an aspect-oriented extension for Java that allows the specification of points for a clean modularization of crosscutting concerns. The language introduces a new entity called *Aspect Beans* that is an extended version of the proper Java Bean used in many Java frameworks allowing decoupled describing of crosscutting concerns. JAsCo is very easy to understand due to its declarative specification, however, there is a limitation that the JAsCo stateful aspects can only specify regular protocols. Protocols that require non-regular language cannot be represented.

In DataFlow Pointcut [Masuhara and Kawauchi 2003] the researchers proposes a new kind of pointcut namely dflow to be implemented using AspectJ. The technique identifies join points based on the dataflow of values. The pointcut allows the software engineer to write declarative aspects where the origins of data matter. Although dataflow can be used in many situations and concepts like design patterns or security problems there is the necessity to have a formal framework of dflow pointcut for the completeness of the semantics.

In REAssistant created by [Rago et al. 2015], the authors proposed an automated tool called REAssistant that is able to extract semantic information from textual use cases and reveal crosscutting concerns candidates. The tool REAssistant can perform a linguistic analysis of the uses cases and apply searching rules to identify crosscutting concerns. However, there are no case studies presented that prove its applicability.

In AspectQuery created by [He and Tu 2013], the authors proposed a technique based on goal model for identifying crosscutting concerns. Despite the fact that AspectQuery can support efficiently for identifying crosscutting concerns in a requirements

phase there is a limitation in the transformation of goal model into an XML file that is implemented by manually which hinders the full-automation support for identifying crosscutting concerns in the requirements phase.

In Join Point Designation Diagrams (JPDDs), created by [Dominik Stein 2006] and used by [Wehrmeister 2009] we can see a visual means to specify join point selections. It is an existing modeling approach to visualize join point selections based on the conceptual model of message sending. It is based on UML to create diagrams of the system events, object symbols and object flow edges through the adoption of action symbols that represents the flow to reach a join point (Identifiers, indirection symbols). It also uses regular expressions. Although JPDD has the capability to express the same matters in the same uniform and programming language-independent way, so the developers don't need to learn the exact syntax and semantics of particular programming language, there is still need an additional expense that developers need to learn and understand the symbols of JPDD and how to design it properly in the context of functional and non-functional requirements.

In AspectOCL created by [Muhammad Uzair Khan 2015], the authors proposed an extension of the Object Constraint Language (OCL) [Cabot and Gogolla 2012] that brings benefits in the specification of crosscutting constraints as aspects. Basically, it does the specification of the aspects and woven it in OCL constraints. Despite the fact that it was presented that the use of AspectOCL can reduce the number of constraints in the models, there are some limitations presented by the authors, one is that the constraint specification is decoupled from the modeling elements, another limitation is the lack of formal semantics, there is also the limitation in the creation of composite aspects and the last limitation is that the language does not have empirical evaluation.

In Theme/Doc created by [Baniassad and Clarke 2004] the authors presented a semi-automated technique to identify a point that may have crosscutting concerns and model aspectual requirements and checking design decisions in the context of the requirements. Despite the fact that Theme/Doc is very effective in helping to identify aspects in requirements specification, there is a limitation that the approach is only applicable for structured requirements documents. [Ali and Kasirun 2008].

In [Antonelli et al. 2015] the authors described an approach that proposes to identify crosscutting concerns through the Language Extended Lexicon (LEL), analyzing the symbols most frequently in the requirements when describing the behaviors of other symbols. Despite the fact that the approach the authors proposed can perform the identification of crosscutting concerns very early avoiding reworking of the artifacts and redundancy information there is a limitation that the approach presented does not aim to propose a composition language as some other approach does.

In [Mongiovì et al. 2017] the authors proposed an approach to identify crosscutting concerns based on the definition of schemata and their automatic identification in software systems. The approach analyses the implementation of classes and their relationships and looks for specific user-defined schemata. Despite the fact that the proposed method has been shown to be a valuable tool in identifying crosscutting concerns there are some limitations that impact on its applicability as the necessity of human intervention, problems in scalability issues, the accuracy of the ground truth and limitation in the

experimental results hence lack empiricism.

In EA-Miner created by [Sampaio et al. 2005], the authors proposed a tool that provides automated support for identifying and separate aspectual and non-aspectual concerns as well as their relationships at the requirements level. The tool utilizes natural language processing techniques and it can work with viewpoints, use cases, starting its analysis on the colloquial language from documents. Although the tool is considered by the authors to be very effective there is a strong dependency on the requirements engineer and its intervention in the modeling process. Hence we cannot say that the approach is automated but semi-automated.

We also have found several techniques in the scientific literature that is related to the Aspect Mining area. In Execution Patterns create by [Breu and Krinke 2004a], the authors describe the first dynamic program analysis approach that mines aspects based on program traces. The technique investigates the traces generated by the run-time behavior of the software and searches for recurring execution patterns. Despite the fact that the approach proposed focuses on occurrences in different called context, unlike other approaches that mines for specific static user-defined aspects, there is a limitation that since the technique observes run-time behavior it uses resolved dynamic binding which can lead to false positives.

In Dynamic Analysis created by [Ceccato 2008], the authors proposed a technique that uses the formal concept analysis (FCA) algorithm [Wille 2005] to discover possible crosscutting concerns. Although the technique is based on the use of FCA it has a limitation to report only portions of actual crosscutting concerns, due to the fact that it relies on a limited set of execution scenarios.

In Clone Detection created by [Bruntink et al. 2005], the authors propose a technique that receives as input the Abstract Syntax Tree and creates an output that is the number of code parts which are considered to be the clone of other parts. Despite the fact that Clone detection techniques are promising in the respect of automation there is a limitation that in general the clone class selection algorithm only considers those clone class that can add recall at a given point during the selection. Clone class which does not add recall do not influence the average precision.

In Random Walks created by [Zhang and Jacobsen 2012], the authors propose a technique that creates a trajectory of successive random steps performing a random walk on the coupling graphs extracted from the code creating a list of crosscutting concerns candidates. Although the technique calculates precisely the probabilistic values under crosscutting concerns identification there is some misclassification the authors had in their experiments as syntactically scatter and widely references.

In History-Based created by [Mulder and Zaidman 2010], the authors propose a technique that aims to identify crosscutting concerns analyzing the history of the source code repository using source code version control tools. Despite the fact that there was a contribution through two case studies and also there was an evaluation approach to assess the merits of the technique there are many limitations as a reduction in overall precision and also the authors cannot be 100% sure that the crosscutting concerns information they have is good enough to be conclusive about its technique applicability.

In Method Call tree created by [Qu and Liu 2007a], the authors propose a method

that has the responsibility to generate method call traces and search for a pattern to mining aspects. Aspect Mining using method call tree has promised in recognizing aspect candidates. However, the authors conclude that it will be necessary to conduct further case studies with programs that have a deep inheritance hierarchy in order to be more conclusive.

In Concern Peers created by [Nguyen et al. 2011], the authors propose a technique that identifies groups of code fragments that presumably share a crosscutting concern. The detection is based on their interactions, contexts, and similar method names. Despite the fact that the authors created an evaluation that showed that the technique achieves a level of accuracy in the recommendation of aspect candidates there is a limitation that the tool still needs to improve its precision to reduce wrong recommendations for aspect candidates.

In Clustering-Based Fan-in Analysis (CBFA) created by [Zhang et al. 2008], the authors propose a technique that says that if the same methods are being called often from different classes then there is a high probability of crosscutting and it should be clustered. Although experiments provide accurate recommendations for aspect mining there is a limitation in the use of fan-in, for instance, it uses a threshold to filter out methods with a low fan-in value which are not likely to be aspects impacting in the precision of the technique.

4.3. RQ3 - Reported Challenges and Problems

The challenges and problems found during the research were:

- **Method Accuracy** [Zhang et al. 2008] [Nguyen et al. 2011] [Mulder and Zaidman 2010] [Breu and Krinke 2004a] [Baniassad and Clarke 2004] [Muhammad Uzair Khan 2015] [Bruntink et al. 2005] [He and Tu 2013]: Most of the methods presented in this paper have a limitation in the precision of the technique generating uncertainty in the process of crosscutting concerns recommendation and resulting in inconsistencies in the data that can lead to false positives and vice versa.

- **Syntactical Scatter** [Zhang and Jacobsen 2012] [Bruntink et al. 2005] [Dominik Stein 2006] [Ceccato 2008] [Muhammad Uzair Khan 2015] [Masuhara and Kawauchi 2003]: In some notations it was found a limitation in the non-trivial use of variables, presenting a complex declarative own semantics that takes too much time to understand and create some misclassification in the experiments resulting in the increasing of the complexity in its applicability.

- **Human Intervention** [Sampaio et al. 2005] [Mongiovì et al. 2017] [Dominik Stein 2006] [He and Tu 2013]: Some techniques presented in this paper has a limitation to not be totally automated. For those techniques that are semi-automated, there is a strong dependency in the intervention of an engineer in the modeling process to manually map the relationship between the themes and requirements and also the engineer needs to do a walk-through in the whole requirements source document to identify the crosscutting concerns. This is very time consuming and costly to handle a large amount of those requirements sources and also to create the models is necessary an intervention of an expert to

validate the results provided by the tool.

- **Empirical Evaluation** [Muhammad Uzair Khan 2015] [Rago et al. 2015]: The application in Industry is one potential factor to the dissemination of the language. Some techniques have little or none empirical evaluation of its legibility, comprehension, and efficiency. This is one of the determining factors for the dissemination of language.

- **Technology Coupling** [Vanderperren et al. 2005] [Gregor Kiczales 2001] [Mulder and Zaidman 2010] [Ceccato 2008] [Antonelli et al. 2015]: For some techniques, a specific technology and structure format is needed to find the results in the process of identification and specification of crosscutting concerns. This is a limitation in the context of modern technology projects that needed to be integrated regardless of its platform or any technology dependency.

- **Comprehensibility:** [Dominik Stein 2006], [Baniassad and Clarke 2004] [Antonelli et al. 2015] [Allan et al. 2005]: In some visual notations, there is a problem related to the comprehension of the model concerned with the deterministic retrieval of points that may have crosscutting concerns. The authors try to facilitate the comprehension and development of join point selections but it still complexes to understand some diagrams and notations.

According to the mapping, the number of problems and challenges to be solved is quite high. Understanding the problem is necessary to propose a proper solution to overcome the challenge. The systematic mapping showed some solutions that could help to solve the limitations found in the search process.

For example, in the lack of formalism problem that is related to the poor semantic representation of some notations and techniques, it can be solved using an approach that is embedded in the context of the object-oriented system that has a strong semantic representation. [Mongiovì et al. 2017]. For the complex syntactical representation, the majority of techniques has similar syntactical complexity in its using producing artifact's difficult to understand either graphically or textually. An approach that could solve this issue is presented in [Antonelli et al. 2015], having an advantage of being very easy to apply it since it begins with a description of the application language in a very simple way produced by LEL (Lexical Extended Language) that is very rich in its syntactical representation. Another important challenge to mention is that some studies do very well in the identification and specification of points that may have crosscutting concerns but it lacks in a crucial factor to the dissemination of its technique, the acceptance in the industry. One work that shows a lot of promise in its application in industry is presented in [Muhammad Uzair Khan 2015], that is a creation of a language responsible to specify crosscutting constraints through OCL (Object Constraint Language) extension. Since OCL is very accepted in the industry, this limitation can be full-filled by this work.

Identify point that may have crosscutting concerns from early stages is not a trivial task, even because the identification of such concerns by hand is an error-prone and cumbersome effort. Hence, it is very time-consuming, many works mentioned in this mapping has this limitation, time-consuming, the programmer needs to check or validate manually

for potential crosscutting concerns. A possible solution for this limitation is treated by the authors of the creation of a tool called REAssistant Tool. [Rago et al. 2015]. This automated tool is able to extract semantic information from textual requirements and point properly the presumable points that may have crosscutting concerns. It does that performing a series of advanced NLP (Natural Language Processing), thus it performs a linguistic analysis of the use cases or features requirement, applying search rules to identify points that potentially may have crosscutting concerns. Another challenge that has been struggled among researches to find a suitable solution was the poor recognition of words in automated tools and the accuracy generated in the recommendation of points that may have crosscutting concerns.

In [Zhang et al. 2008] it was proposed an automated aspect mining approach called Clustered-Based Fan-in Analysis (CBFA) that could be a solution for the limitation of poor recognition. The technique adopts a lexical based approach to group methods associated with the same crosscutting concerns and suggests the potential points that may have crosscutting based on the cluster fan-in ranking metric. The authors also have presented experiments showing the improvements in the accuracy of their technique and also the significant evolution in the recognition of words and crosscutting mining coverage compared to the other state-of-the-art approaches. For the complexity in the comprehension of the model [Dominik Stein 2006], to the extent of our knowledge we didn't find any works in literature with a proposal solution via diagrammatic means that solves this issue. There are some textual notations that are a little less complex to understand but it still difficult to read and need prior knowledge in the domain of the language. [Allan et al. 2005], [He and Tu 2013].

4.4. Limitation

Several threats to validity were considered and the current review has some limitations. The first limitation is related to the selection of papers and data extraction, the steps for those were performed by just one of the authors, do it can create a subjectivity in the process. The second limitation is related to the use of a few online bibliographic databases that might have lead to the omission of relevant papers. The third limitation is the fact that this paper misses a fine-grained classification via a quality assessment. As with all reviews it was limited by the search terms used and the electronic databases included, some synonyms could be included on the search terms but it was not and certain it can impact on the searching results as well. However, the studies discussed in this mapping provide a snapshot of empirical research on outcomes and impacts of existing research on identifying and specify points that may have crosscutting concerns. We just considered studies indexed by the selected electronic databases. Another limitation is that we didn't do a snowballing. For that reason there is the possibility of some valuable studies may have been left out of our analysis.

5. Conclusion

In this study, we have applied systematic mapping method to analyze the scientific literature related to the identification and specification of points that may have crosscutting concerns with the aim to identify, collect and evaluate the main methods and techniques used in the identification and specification of crosscutting concerns and also identify the challenges and problems giving an idea of shortcomings in existence evidence and how

they are being solved by the scientific community. This work presented a review of published literature on Identification and Specification of Crosscutting Concerns in order to identify the state of the art. In this systematic mapping, we have found 104 relevant studies in 4 electronic databases. The studies were extracted and synthesized with the objective to answer 4 research that was defined to investigate the following facets: (i) distribution of the selected publications over the years; (ii) main methods used to identify and specify points that may have crosscutting concerns; (iii) challenges and problems in each method; and (iv) how the challenges and problems were solved.

The main contribution of this mapping study is on providing a critical analysis of the selected studies associated to the identification and specification of points that may have crosscutting concerns with the aim to identify the open issues of this subject as also contributes in order to drive future research in this area. The results provide an overview on existing crosscutting identification and specification approaches to support the development of a new technique with the aim to specify point that may have crosscutting concerns in early phases before generating a meta-model or syntactical tree in the process of an automatic code generation of modern complex systems. Another research opportunity concerns in conduct further evaluation research investigating the feasibility of applying the proposed approaches of this systematic mapping.

References

Abufardeh, S. and Magel, K. (2009). Software internationalization: Crosscutting concerns across the development lifecycle. In *2009 International Conference on New Trends in Information and Service Science*, pages 447–450.

Al-Mansari, M. and Hanenberg, S. (2006). Path expression pointcuts: Abstracting over non-local object relationships in aspect-oriented languages. pages 81–96. cited By 2.

Ali, B. S. and Kasirun, Z. M. (2008). 3ci: A tool for crosscutting concern identification. In *2008 International Conference on Computational Intelligence for Modelling Control Automation*, pages 351–355.

Ali, B. S. and Kasirun, Z. M. (2008). A review on approaches for identifying crosscutting concerns. In *2008 International Conference on Advanced Computer Theory and Engineering*, pages 855–859.

Ali, M. S., Babar, M. A., Chen, L., and Stol, K.-J. (2010). A systematic review of comparative evidence of aspect-oriented programming. *Information and Software Technology*, 52(9):871 – 887.

Allan, C., Avgustinov, P., Christensen, A. S., Hendren, L., Kuzins, S., Lhoták, O., de Moor, O., Sereni, D., Sittampalam, G., and Tibble, J. (2005). Adding trace matching with free variables to aspectj. *SIGPLAN Not.*, 40(10):345–364.

Alves, V., Schwanninger, C., Clements, P., Rashid, A., Moreira, A., Araújo, J., Baniassad, E., and Tekinerdogan, B. (2008). Early aspects: Aspect-oriented requirements and architecture for product lines (ea@splc.08). In *2008 12th International Software Product Line Conference*, pages 382–382.

Amirat, A., Meslati, D., and Laskri, M. T. (2006). Elicitation of crosscutting aspects at the early phases of software development. In *2006 2nd International Conference on Information Communication Technologies*, volume 2, pages 3575–3576.

Anbalagan, P. and Xie, T. (2007). Automated inference of pointcuts in aspect-oriented refactoring. In *29th International Conference on Software Engineering (ICSE'07)*, pages 127–136.

Antonelli, L., Rossi, G., and do Prado Leite, J. C. S. (2010). Early identification of crosscutting concerns in the domain model guided by states. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 275–280, New York, NY, USA. ACM.

Antonelli, L., Rossi, G., Leite, J. C. S. d. P., and Araújo, J. (2015). Early identification of crosscutting concerns with the language extended lexicon. *Requirements Engineering*, 20(2):139–161.

Araujo, J., Whittle, J., and Kim, D.-K. (2004). Modeling and composing scenario-based requirements with aspects. In *Proceedings. 12th IEEE International Requirements Engineering Conference, 2004.*, pages 58–67.

Baniassad, E. and Clarke, S. (2004). Theme: An approach for aspect-oriented analysis and design. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE '04, pages 158–167, Washington, DC, USA. IEEE Computer Society.

Baniassad, E., Clements, P. C., Araujo, J., Moreira, A., Rashid, A., and Tekinerdogan, B. (2006). Discovering early aspects. *IEEE Software*, 23(1):61–70.

Bernardi, M. L. and Di Lucca, G. A. (2009). Analysing object type hierarchies to identify crosscutting concerns. In Lee, Y.-h., Kim, T.-h., Fang, W.-c., and Ślezak, D., editors, *Future Generation Information Technology*, pages 216–224, Berlin, Heidelberg. Springer Berlin Heidelberg.

Bernardi, M. L. and Di Lucca, G. A. (2011). Identifying the crosscutting among concerns by methods' calls analysis. In Kim, T.-h., Adeli, H., Kim, H.-k., Kang, H.-j., Kim, K. J., Kiumi, A., and Kang, B.-H., editors, *Software Engineering, Business Continuity, and Education*, pages 147–158, Berlin, Heidelberg. Springer Berlin Heidelberg.

Bernardi, M. L. and Lucca, G. A. D. (2009). Conan: A tool for the identification of crosscutting concerns in object oriented systems based on type hierarchy analysis. In *2009 16th Working Conference on Reverse Engineering*, pages 319–320.

Bernardi, M. L. and Lucca, G. A. D. (2010). The conan tool to identify crosscutting concerns in object oriented systems. In *2010 IEEE 18th International Conference on Program Comprehension*, pages 48–49.

Boukhtouta, A., Alhadidi, D., and Debbabi, M. (2009). A practical framework for the dataflow pointcut in aspectj. In *2009 International Conference on Availability, Reliability and Security*, pages 835–840.

Breu, S. and Krinke, J. (2004a). Aspect mining using event traces. In *Proceedings. 19th International Conference on Automated Software Engineering, 2004.*, pages 310–315.

Breu, S. and Krinke, J. (2004b). Aspect mining using event traces. In *Proceedings. 19th International Conference on Automated Software Engineering, 2004.*, pages 310–315.

Bruntink, M., van Deursen, A., Tourwe, T., and van Engelen, R. (2004a). An evaluation of clone detection techniques for crosscutting concerns. In *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.*, pages 200–209.

Bruntink, M., Van Deursen, A., Tourwé, T., and Van Engelen, R. (2004b). An evaluation of clone detection techniques for identifying crosscutting concerns. pages 200–209. cited By 66.

Bruntink, M., van Deursen, A., van Engelen, R., and Tourwe, T. (2005). On the use of clone detection for identifying crosscutting concern code. *IEEE Transactions on Software Engineering*, 31(10):804–818.

Cabot, J. and Gogolla, M. (2012). Object constraint language (ocl): A definitive guide. volume 7320, pages 58–90.

Canfora, G., Cerulo, L., and Di Penta, M. (2006). On the use of line co-change for identifying crosscutting concern code. pages 213–222. cited By 25.

Ceccato, M. (2008). Automatic support for the migration towards aspects. In *2008 12th European Conference on Software Maintenance and Reengineering(CSMR)*, volume 00, pages 298–301.

Ceccato, M., Marin, M., Mens, K., Moonen, L., Tonella, P., and Tourwe, T. (2005). A qualitative comparison of three aspect mining techniques. In *13th International Workshop on Program Comprehension (IWPC'05)*, pages 13–22.

Ceccato, M., Marin, M., Mens, K., Moonen, L., Tonella, P., and Tourwé, T. (2006). Applying and combining three different aspect mining techniques. *Software Quality Journal*, 14(3):209–231.

Chen, T. and He, C. (2013). A comparison of approaches to legacy system crosscutting concerns mining. In *2013 International Conference on Computer Sciences and Applications*, pages 813–816.

Chitchyan, R., Rashid, A., Rayson, P., and Waters, R. (2007). Semantics-based composition for aspect-oriented requirements engineering. In *Proceedings of the 6th International Conference on Aspect-oriented Software Development*, AOSD '07, pages 36–48, New York, NY, USA. ACM.

Cojocar, G. and Guran, A. (2017). On a top down aspect mining approach for monitoring crosscutting concerns identification. In *2017 IEEE 14th International Scientific Conference on Informatics*, pages 51–56.

Cojocar, G. S. and Czibula, G. (2008). On clustering based aspect mining. In *2008 4th International Conference on Intelligent Computer Communication and Processing*, pages 129–136.

Cojocar, G. S. and Serban, G. (2007). On evaluating aspect mining techniques. In *2007 IEEE International Conference on Intelligent Computer Communication and Processing*, pages 217–224.

Costa, H. A. X., Junior, P. A. P., de Camargo, V. V., and Penteado, R. A. D. (2009). Recovering class models stereotyped with crosscutting concerns. In *2009 16th Working Conference on Reverse Engineering*, pages 311–312.

d. S. Filho, J. A. N., Noya, R. C., and Breitman, K. K. (2009). Agentcross: A platform for the development of crosscutting concerns in multi-agent systems. In *2009 33rd Annual IEEE Software Engineering Workshop*, pages 25–32.

Dahi, F. and Bounour, N. (2012). Crosscutting concerns identification approach based on the sequence diagram analysis. In Abelló, A., Bellatreche, L., and Benatallah, B., editors, *Model and Data Engineering*, pages 141–152, Berlin, Heidelberg. Springer Berlin Heidelberg.

de Freitas, E. P., Wehrmeister, M. A., Silva, E. T., Carvalho, F. C., Pereira, C. E., and Wagner, F. R. (2007). Deraf: A high-level aspects framework for distributed embedded real-time systems design. In Moreira, A. and Grundy, J., editors, *Early Aspects: Current Challenges and Future Directions*, pages 55–74, Berlin, Heidelberg. Springer Berlin Heidelberg.

Dominik Stein, Stefan Hanenberg, R. U. (2006). Joinpoint designation diagrams:a graphical representation of joinpoint selections. *nternational Journal of Software Engineering and Knowledge Engineering*, 16(3):317–346.

Durelli, R. S., Santibáñez, D. S. M., Anquetil, N., Delamaro, M. E., and de Camargo, V. V. (2013). A systematic review on mining techniques for crosscutting concerns. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 1080–1087, New York, NY, USA. ACM.

Eaddy, M., Aho, A., and Murphy, G. C. (2007). Identifying, assigning, and quantifying crosscutting concerns. In *First International Workshop on Assessment of Contemporary Modularization Techniques (ACoM '07)*, pages 2–2.

Eaddy, M., Aho, A. V., Antoniol, G., and Guéhéneuc, Y. (2008). Cerberus: Tracing requirements to source code using information retrieval, dynamic analysis, and program analysis. In *2008 16th IEEE International Conference on Program Comprehension*, pages 53–62.

Elrad, T., Aldawud, O., and Bader, A. (2002). Aspect-oriented modeling: Bridging the gap between implementation and design. In *Proceedings of the 1st ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering*, GPCE '02, pages 189–201, London, UK, UK. Springer-Verlag.

Fu, Y., Ding, J., and Bording, P. (2009). An approach for modeling and analyzing crosscutting concerns. In *2009 IEEE/INFORMS International Conference on Service Operations, Logistics and Informatics*, pages 91–97.

Gregor Kiczales, John Lamping, A. M. C. M. C. V. L. J.-M. L. J. I. (1997). Aspect-oriented programming. In *Proceedings of European Conference on Object-Oriented Programming (ECOOP)*, Finland. Springer-Verlag LNCS 1241.

Gregor Kiczales, Erik Hilsdale, J. H. M. K. J. P. W. G. G. (2001). An overview of aspectj. In *Proceedings of European Conference on Object-Oriented Programming (ECOOP)*.

Griswold, W. G., Kato, Y., and Yuan, J. J. (1999). Aspect browser : Tool support for managing dispersed aspects.

Groher, I., Jackson, A., Schwanninger, C., and Völter, M. (2007). Models and aspects - handling crosscutting concerns in mdsd. In Südholt, M. and Consel, C., editors, *Object-Oriented Technology. ECOOP 2006 Workshop Reader*, pages 21–25, Berlin, Heidelberg. Springer Berlin Heidelberg.

Hamza, H. S. and Darwish, D. (2009). On the discovery of candidate aspects in software requirements. In *2009 Sixth International Conference on Information Technology: New Generations*, pages 819–824.

He, C. and Tu, C. (2013). Aspectquery: A method for identification of crosscutting concerns in the requirement phase. *IEICE Transactions on Information and Systems*, E96-D(4):897–905. cited By 2.

He, C. and Ye, S. (2015). A method for identification of crosscutting concerns based on goal model and two-state algorithm. In *2015 4th International Conference on Computer Science and Network Technology (ICCSNT)*, volume 01, pages 431–435.

Huang, J., Carminati, F., Betev, L., Zhu, J., and Lu, Y. (2012). Identifying composite crosscutting concerns with scatter-based graph clustering. *Wuhan University Journal of Natural Sciences*, 17(2):114–120.

Ishio, T., Date, H., Miyake, T., and Inoue, K. (2008). Mining coding patterns to detect crosscutting concerns in java programs. In *2008 15th Working Conference on Reverse Engineering*, pages 123–132.

Júnior, P. A. P., Mendes, W., de Camargo, V. V., Penteado, R. A. D., and Costa, H. A. X. (2012). Mining crosscutting concerns with comscid: A rule-based customizable mining tool. In *2012 XXXVIII Conferencia Latinoamericana En Informatica (CLEI)*, pages 1–9.

Kellens, A., Mens, K., and Tonella, P. (2007a). *A Survey of Automated Code-Level Aspect Mining Techniques*, pages 143–162. Springer Berlin Heidelberg, Berlin, Heidelberg.

Kellens, A., Mens, K., and Tonella, P. (2007b). *A Survey of Automated Code-Level Aspect Mining Techniques*, pages 143–162. Springer Berlin Heidelberg, Berlin, Heidelberg.

Khan, M. U., Arshad, N., Iqbal, M. Z., and Umar, H. (2015). Aspectocl: Extending ocl for crosscutting constraints. In Taentzer, G. and Bordeleau, F., editors, *Modelling Foundations and Applications*, pages 92–107, Cham. Springer International Publishing.

Khan, S. S. and Jaffar-ur-Rehman, M. (2005). A survey on early separation of concerns. In *12th Asia-Pacific Software Engineering Conference (APSEC'05)*, pages 7 pp.–.

Khan, S. S. and ur Rehman, M. J. (2005). A survey on early separation of concerns. In *12th Asia-Pacific Software Engineering Conference (APSEC'05)*, pages 7 pp.–.

Khatchadourian, R. T. and Rashid, A. (2008). Rejuvenate pointcut: A tool for pointcut expression recovery in evolving aspect-oriented software. In *2008 Eighth IEEE International Working Conference on Source Code Analysis and Manipulation*, pages 261–262.

Kitchenham, B. (2004). Procedures for performing systematic reviews.

Krinke, J. (2006). Mining control flow graphs for crosscutting concerns. In *2006 13th Working Conference on Reverse Engineering*, pages 334–342.

Krinke, J. (2008). Mining execution relations for crosscutting concerns. *IET Software*, 2(2):65–78.

Lengyel, L., Levendovszky, T., and Angyal, L. (2009). Identification of crosscutting constraints in metamodel-based model transformations. In *IEEE EUROCON 2009*, pages 359–364.

Marçal, I., Garcia, R. E., Eler, D. M., Olivete Junior, C., and Correia, R. C. M. (2016a). Techniques for the identification of crosscutting concerns: A systematic literature review. In Latifi, S., editor, *Information Technology: New Generations*, pages 569–579, Cham. Springer International Publishing.

Marçal, I., Garcia, R. E., Eler, D. M., Olivete Junior, C., and Correia, R. C. M. (2016b). Techniques for the identification of crosscutting concerns: A systematic literature review. In Latifi, S., editor, *Information Technology: New Generations*, pages 569–579, Cham. Springer International Publishing.

Marin, M., Moonen, L., and Deursen, A. V. (2006). A common framework for aspect mining based on crosscutting concern sorts. In *2006 13th Working Conference on Reverse Engineering*, pages 29–38.

Marin, M., Moonen, L., and van Deursen, A. (2007). Soquet: Query-based documentation of crosscutting concerns. In *29th International Conference on Software Engineering (ICSE'07)*, pages 758–761.

Marin, M., van Deursen, A., and Moonen, L. (2004). Identifying aspects using fan-in analysis. In *11th Working Conference on Reverse Engineering*, pages 132–141.

Marin, M., van Deursen, A., Moonen, L., and van der Rijst, R. (2009). An integrated crosscutting concern migration strategy and its semi-automated application to jhotdraw. *Automated Software Engineering*, 16(2):323–356.

Markus Völter, Thomas Stahl, J. B. A. H. S. H. (2013). *Model-driven software development: technology, engineering, management*. John Wiley Sons.

Masuhara, H. and Kawauchi, K. (2003). Dataflow Pointcut in Aspect-Oriented Programming. In Goos, G., Hartmanis, J., van Leeuwen, J., and Ohori, A., editors, *Programming Languages and Systems*, volume 2895, pages 105–121. Springer Berlin Heidelberg, Berlin, Heidelberg.

McFadden, R. R. and Mitropoulos, F. J. (2012). Aspect mining using model-based clustering. In *2012 Proceedings of IEEE Southeastcon*, pages 1–8.

McFadden, R. R. and Mitropoulos, F. J. (2013). Survey and analysis of quality measures used in aspect mining. In *2013 Proceedings of IEEE Southeastcon*, pages 1–8.

McFadden, R. R. and Mitropoulos, F. J. (2013). Survey of aspect mining case study software and benchmarks. In *2013 Proceedings of IEEE Southeastcon*, pages 1–5.

McFadden, R. R. and Mitropoulos, F. J. (2013). Survey of aspect mining case study software and benchmarks. In *2013 Proceedings of IEEE Southeastcon*, pages 1–5.

Mongiovì, M., Pappalardo, G., and Tramontana, E. (2017). Specifying and identifying widely used crosscutting concerns. *Knowledge-Based Systems*, 126:20 – 32.

Moreira, A. and Araújo, J. (2011). *The Need for Early Aspects*, pages 386–407. Springer Berlin Heidelberg, Berlin, Heidelberg.

Muhammad Uzair Khan, Numra Arshad, M. Z. I. H. U. (2015). An extension to ocl for specifying crosscutting constraints. In *Proceedings of European Conference on Modelling Foundations and Applications (ECMFA)*.

Mulder, F. and Zaidman, A. (2010). Identifying cross-cutting concerns using software repository mining. In *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*, IWPSE-EVOL '10, pages 23–32, New York, NY, USA. ACM.

Mussbacher, G., Amyot, D., and Weiss, M. (2007). *Visualizing Early Aspects with Use Case Maps*, pages 105–143. Springer Berlin Heidelberg, Berlin, Heidelberg.

Nguyen, T. T., Nguyen, H. V., Nguyen, H. A., and Nguyen, T. N. (2011). Aspect recommendation for evolving software. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 361–370, New York, NY, USA. ACM.

Nogueira Santos, F. J., Cappelli, C., Santoro, F., do Prado Leite, J. C. S., and Batista, T. V. (2012). Analysis of heuristics to identify crosscutting concerns in business process models. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, pages 1725–1726, New York, NY, USA. ACM.

NORA, B. (2006). A comparative classification of aspect mining approaches. *J. Computer Science*, 2(4):322–325.

Nunez-Varela, A. S., Perez-Gonzalez, H. G., Flores-Puente, Y. T., and Valdes-Souto, F. (2017). Finding core crosscutting concerns from object oriented systems using information retrieval. In *2017 5th International Conference in Software Engineering Research and Innovation (CONISOFT)*, pages 18–24.

OMG (2003). Unified modeling language specification.

Qiu, X. and Zhang, L. (2014). Specifying redundancy tactics as crosscutting concerns using aspect-oriented modeling. *Frontiers of Computer Science*, 8(6):977–995.

Qu, L. and Liu, D. (2007a). Aspect mining using method call tree. In *2007 International Conference on Multimedia and Ubiquitous Engineering (MUE'07)*, pages 407–412.

Qu, L. and Liu, D. (2007b). Aspect mining using method call tree. In *2007 International Conference on Multimedia and Ubiquitous Engineering (MUE'07)*, pages 407–412.

Qu, L., Yin, G., Yang, J., and Hou, X. (2011). Aspect mining using relative reduced concept lattice. In *2011 3rd International Conference on Computer Research and Development*, volume 2, pages 183–187.

Rago, A., Abait, E., Marcos, C., and Diaz-Pace, A. (2009). Early aspect identification from use cases using nlp and wsd techniques. In *Proceedings of the 15th Workshop on Early Aspects*, EA '09, pages 19–24, New York, NY, USA. ACM.

Rago, A., Marcos, C., and Diaz-Pace, J. (2015). Reassistant: A tool for identifying crosscutting concerns in textual requirements. volume 1554, pages 32–35. cited By 0.

Rago, A., Marcos, C., and Diaz-Pace, J. A. (2016). Assisting requirements analysts to find latent concerns with reassistant. *Automated Software Engineering*, 23(2):219–252.

Rashid, A. (2008). Aspect-oriented requirements engineering: An introduction. In *2008 16th IEEE International Requirements Engineering Conference*, pages 306–309.

Rashid, A., Moreira, A., and Araújo, J. (2003). Modularisation and composition of aspectual requirements. In *Proceedings of the 2Nd International Conference on Aspect-oriented Software Development*, AOSD '03, pages 11–20, New York, NY, USA. ACM.

Sampaio, A., Chitchyan, R., Rashid, A., and Rayson, P. (2005). Ea-miner: A tool for automating aspect-oriented requirements identification. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, ASE '05, pages 352–355, New York, NY, USA. ACM.

Shepherd, D., Pollock, L., and Tourwé, T. (2005). Using language clues to discover crosscutting concerns. *SIGSOFT Softw. Eng. Notes*, 30(4):1–6.

Stein, D. and Hanenberg, S. (2011). Comparison of a visual and a textual notation to express data constraints in aspect-oriented join point selections: A controlled experiment. In *2011 IEEE 19th International Conference on Program Comprehension*, pages 141–150.

Stein, D., Hanenberg, S., and Unland, R. (2004). Query models. In Baar, T., Strohmeier, A., Moreira, A., and Mellor, S. J., editors, *UML 2004 — The Unified Modeling Language. Modeling Languages and Applications*, pages 98–112, Berlin, Heidelberg. Springer Berlin Heidelberg.

Stein, D., Hanenberg, S., and Unland, R. (2005a). A graphical notation to specify model queries for mda transformations on uml models. In Aßmann, U., Aksit, M., and Rensink, A., editors, *Model Driven Architecture*, pages 77–92, Berlin, Heidelberg. Springer Berlin Heidelberg.

Stein, D., Hanenberg, S., and Unland, R. (2005b). Visualizing join point selections using interaction-based vs. state-based notations exemplified with help of business rules. pages 94–107. cited By 4.

Su, Y. and Pan, X. (2010). Recovering crosscutting concern from legacy software based on use-cases driven formal concept analysis. In *2010 International Conference on Computational Intelligence and Software Engineering*, pages 1–4.

Tourwe, T. and Mens, K. (2004). Mining aspectual views using formal concept analysis. In *Source Code Analysis and Manipulation, Fourth IEEE International Workshop on*, pages 97–106.

Trifu, M. (2008). Using dataflow information for concern identification in object-oriented software systems. In *2008 12th European Conference on Software Maintenance and Reengineering*, pages 193–202.

van den Berg, K., Conejero, J. M., and Hernández, J. (2007). *Analysis of Crosscutting in Early Software Development Phases Based on Traceability*, pages 73–104. Springer Berlin Heidelberg, Berlin, Heidelberg.

Vanderperren, W., Suvée, D., Cibrán, M. A., and De Fraine, B. (2005). Stateful aspects in jasco. In Gschwind, T., Aßmann, U., and Nierstrasz, O., editors, *Software Composition*, pages 167–181, Berlin, Heidelberg. Springer Berlin Heidelberg.

Wehrmeister, M. A. (2009). *An Aspect-Oriented Model-Driven Engineering Approach for Distributed Embedded Real-Time Systems*. PhD thesis, Universidade Federal do Rio Grande do Sul.

Wen, X., Yu, H., and Zheng, H. (2013). Aspect-oriented design method for embedded systems based on timed statecharts. *China Communications*, 10(9):33–42.

Wille, R. (2005). *Formal Concept Analysis as Mathematical Theory of Concepts and Concept Hierarchies*, pages 1–33. Springer Berlin Heidelberg, Berlin, Heidelberg.

Wu, P. and Lieberherr, K. (2005). Shadow programming: Reasoning about programs using lexical join point information. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3676 LNCS:141–156. cited By 2.

Zhang, C. and Jacobsen, H. (2012). Mining crosscutting concerns through random walks. *IEEE Transactions on Software Engineering*, 38(5):1123–1137.

Zhang, D., Guo, Y., and Chen, X. (2008). Automated aspect recommendation through clustering-based fan-in analysis. In *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, ASE '08, pages 278–287, Washington, DC, USA. IEEE Computer Society.

ANEXO C – ARTIGO EM CONFERÊNCIA: REQUISITO DE MESTRADO

DE OLIVEIRA, Rodrigo; WEHRMEISTER, Marco Aurélio. **Avaliando a Especificação das Ocorrências das Características Transversais em Software Embarcado.** In: ARTIGOS COMPLETOS - SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SISTEMAS COMPUTACIONAIS (SBESC), 10. , 2020, Evento Online. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2020 . p. 9-16.

Avaliando a Especificação das Ocorrências de Características Transversais em Software Embarcado

Rodrigo Souza Cavalcanti de Oliveira, Marco Aurelio Wehrmeister[a]
Universidade Tecnológica Federal do Paraná (UTFPR)
Av. Sete de Setembro, 3165 - CEP 80230-901 - Curitiba - PR - Brazil
rodrigoo@alunos.utfpr.edu.br, wehrmeister@utfpr.edu.br
[a]http://orcid.org/0000-0002-1415-5527

***Resumo*—A engenharia de software embarcado vem evoluindo rapidamente ao longo dos últimos anos, em especial para lidar com o aumento da complexidade associada a grande quantidade de requisitos funcionais e características transversais associadas aos requisitos não-funcionais. O processo de identificar e especificar a seleção de pontos onde as características transversais devem ser tratadas vem sendo amplamente discutido desde meados dos anos 90, especialmente no contexto da *Aspect-Oriented Software Development* (AOSD) e modelagem. No entanto, a compreensão de tais especificações é muitas vezes difícil, pois a sintaxe e a semântica das linguagens utilizadas, principalmente as linguagens visuais, não permitem intuir facilmente o significado dos símbolos usados. Tal situação produz artefatos difíceis de compreender e manter. Portanto, o processo de localizar, identificar, e especificar os pontos onde as características transversais devem ser tratadas é chave no projeto de software embarcado, pois pode levar a problemas no desenvolvimento e, principalmente, na integração dos componentes do software do sistema. Este trabalho apresenta uma avaliação empírica com o objetivo de avaliar a compreensão da especificação de seleção de POCT em sistemas embarcados de forma indireta através da avaliação dos efeitos cognitivos e das propriedades de percepção das notações. Foram avaliadas três notações, duas gráficas (JPDD e Theme/UML) e uma textual (AspectJ), que foram usadas para especificar implícita e explicitamente a seleção de POCT (para 15 pontos distintos) em três projetos de software embarcados diferentes que representam aplicações reais no contexto de sistemas de automação. Para tal, propõe-se um modelo de qualidade baseado no framework conceitual "Physics of Notation" (PON). Foram usadas nove métricas para quantificar as propriedades perceptuais na especificação da seleção de POCT, sendo que algumas métricas foram criadas e outras representam conceitos dentro dos princípios do PON. Os resultados dos experimentos mostram como cada uma das notações impacta na compreensão das especificações criadas. Foram encontradas evidências empíricas de que a especificação gráfica das seleções dos POCT tem pouca discriminação sobre a eficácia cognitiva das representações visuais (conforme o framework PON) e, portanto, os indícios apontam para a conclusão de que o uso de notações visuais para especificar a seleção de POCT é um processo inconsciente e sujeito a ambiguidades tanto na especificação dos artefatos quanto na compreensão e interpretação deles.**

***Index Terms*—software embarcado, características transversais, notação visual, propriedades cognitivas, métricas de software**

Este trabalho recebeu o apoio financeiro da Fundação Araucária do Paraná através dos convênios 337/2014 e 34/2019.

I. Introdução

O desenvolvimento de software embarcado geralmente inclui características funcionais e não-funcionais que estão vinculadas a um único serviço ou função. Nos paradigmas tradicionais de decomposição de software, e.g., paradigma de orientação a objetos e/ou procedural, tais responsabilidades são disseminadas por várias unidades de modularização, tanto nas especificações usando modelos visuais quanto na codificação do software embarcado. As características transversais afetam os requisitos funcionais e os não-funcionais levando a um entrelaçamento e espalhamento do código por vários módulos do software, afetando negativamente o princípio de responsabilidade única e também a construção de um código desacoplado e coeso. A literatura apresenta muitas abordagens com o objetivo de resolver esses problemas de modularização e também da quebra de proposições chave da engenharia de software, e.g., a premissa de responsabilidade única. Uma das técnica mais populares é o Desenvolvimento de Software Orientado a Aspectos (em inglês, *Aspect-oriented Software Development* - AOSD) [1]. A AOSD propõe tratar uma característica transversal ao requisito através de unidades de modularização denominadas aspectos. Os aspectos encapsulam toda a implementação responsável por interceptar o fluxo de execução do software e injetar o tratamento da característica transversal nele. Assim, os problemas associados ao entrelaçamento e espalhamento da característica transversal pelo software embarcado são atenuados.

A especificação da seleção de *Pontos de Ocorrência das Características Transversais* (POCT) indica onde, no código fonte, podem ou devem haver características transversais que são inerentes aos requisitos funcionais ou não-funcionais do sistema embarcado. Existem várias linguagens de programação e também notações visuais orientadas a aspectos que apresentam diversos meios linguísticos para identificação e especificação dessas ocorrências. As formas mais comuns de designar a seleção de POCT são baseadas em propriedades léxicas aplicadas em um contexto comportamental ou estrutural das características transversais [2].

Embora existam um número significativo de linguagens, seja

em forma gráfica ou textual, a compreensão dos elementos relacionados à AOSD geralmente requer um conhecimento profundo não só dos conceitos mas também da sintaxe e da semântica da linguagem. No entanto, existem muitas extensões de linguagens consolidadas, e.g., Java e UML, que empregam uma semântica diferente uma da outra para indicar POCT. Deste modo, é difícil entender a especificação de seleções de POCT escritas em uma linguagem adaptada ou em notações visuais que são geralmente adotadas nas abordagens de modelagem da AOSD [3]–[5]. Tal situação compromete o entendimento de "como" e "onde" as características transversais ocorrem e são tratadas. Destaca-se que projeto de sistemas embarcados de tempo real geralmente apresenta vários tipos distintos de características transversais inerentes não somente aos requisitos funcionais e não-funcionais como também as suas restrições [6], e.g., restrições de tempo, concorrência, monitoramento de recursos, segurança, confiabilidade e outros.

O estado da arte atual apresenta dois tipos de abordagens para especificar o tratamento das características transversais: as notações gráficas e as textuais. A especificação da seleção de POCT através da notação gráfica é uma área de pesquisa que tem sido bastante explorada nos últimos anos [7]–[9]. Por exemplo, o *Join Point Designation Diagram* (JPDD) [2] é uma técnica interessante que foi criada para identificar e especificar a seleção de POCT através da forma gráfica. O JPDD propõe uma extensão da UML [10] cujo objetivo é especificar as seleções de POCT de uma forma independente do sistema ou da linguagem orientada a aspectos. Outra abordagem a ser destacada é a *Theme/UML* [11], que também é baseada na UML mas que atende a separação simétrica das características transversais; a separação assimétrica é suportada pela maioria das abordagens AOSD. Os desenvolvedores especificam ligações (*bindings*) entre os temas para indicar onde a seleção de POCT deve ser inserida no fluxo de execução do software.

Contudo, mesmo sendo a UML amplamente aceita na indústria e na academia, ainda não existe uma semântica precisa quando se trata da especificação de seleções de POCT [12], [13], até mesmo porque a UML foi criada para atender ou dar suporte ao paradigma orientado a objetos. Consequentemente, embora existam abordagens que estendem a semântica da UML para suportar a AOSD [2], [11], [12], o resultado é uma especificação imprecisa e ambígua das seleções de POCT, devido a pouca discriminação sobre a eficácia cognitiva das representações visuais [14]. Isso aumenta de forma indireta a dificuldade no desenvolvimento, compreensão e manutenção dos artefatos gerados para sistemas embarcados.

Sendo assim, este artigo apresenta uma avaliação empírica que compara três notações - duas gráficas, JPDD e Theme/UML, e uma textual, AspectJ - para especificar implícita ou explicitamente a seleção de POCT no software embarcado. O objetivo é avaliar a compreensão da especificação das seleções de POCT em sistemas embarcados, seja ela gráfica ou textual, de forma indireta através da avaliação dos efeitos cognitivos e das propriedades de percepção das notações. Essa é a contribuição principal do trabalho.

Para isso, este trabalho propõe um modelo de qualidade baseado nas premissas conceituais do framewok *Physics of Notations* (PON) [14]. Tal modelo inclui nove métricas com o objetivo de quantificar propriedades físicas perceptivas (i.e. sintaxe) das notações usadas para especificar a seleção de POCT em software embarcado. Tal modelo de qualidade e suas métricas também são contribuições relevantes desse trabalho.

Para a avaliação, foram utilizados quinze POCT comumente encontradas no desenvolvimento de software de sistemas embarcados [6] e que foram especificados em três estudos de caso representando sistemas reais [12], [13]. Cada POCT foi especificado usando as três notações mencionadas. Os resultados iniciais mostram evidências empíricas de que a especificação gráfica das seleções de POCT tem pouca discriminação sobre a eficácia cognitiva das representações visuais e, portanto, os indícios apontam para a conclusão de que o uso de notações visuais para especificar a seleção de POCT é um processo inconsciente e sujeito a ambiguidades, tanto na especificação dos artefatos, quanto na sua compreensão e interpretação.

Esse artigo está organizado da seguinte forma. A Seção 2 apresenta uma visão geral sobre o framework PON. A Seção 3 discute os trabalhos relacionados e analisa os problemas em aberto. A Seção 4 apresenta o modelo de avaliação proposto para a especificação da seleção de POCT. A Seção 5 apresenta a avaliação empírica e analisa seus resultados, enquanto a Seção 6 discute as possíveis ameaças à validade dos resultados e da avaliação empírica. Finalmente, a Seção 7 apresenta as conclusões e indica direções para os trabalhos futuros.

II. A Física das Notações

O framework conceitual *Physics of Notations* (PON) [14] fornece uma base lógico-científica para o design das notações visuais na engenharia de software. Moody [14] afirma que as notações visuais são parte integrante da engenharia de software, mas os pesquisadores e designers de notações têm historicamente ignorado ou subestimado questões importantes quanto a eficácia cognitiva da representação visual.

Ele afirma que, embora vários trabalhos na literatura avaliem e comparam as notações, os detalhes quanto a sintaxe das representações visuais são raramente discutidos. Para preencher essa lacuna, o PON aborda as propriedades físicas e perceptivas das notações (sintaxe), em vez de suas propriedades lógicas (semânticas). As propriedades perceptivas são analisadas à luz da Teoria do Design [15] e envolve nove princípios. Este trabalho usou os 5 princípios mais relevantes para a especificação das seleções de POCT em sistemas embarcados. São eles:

- ***Clareza Semiótica*:** deve haver uma correspondência 1:1 entre construções semânticas e símbolos gráficos;
- ***Discriminabilidade Perceptiva*:** símbolos diferentes devem ser claramente distinguíveis um do outro;
- ***Transparência Semântica*:** a aparência das representações visuais deve sugerir seu significado;
- ***Expressividade Visual*:** o uso de todas as possibilidades e capacidades de variáveis visuais;
- ***Economia gráfica*:** o número de símbolos gráficos diferentes devem ser cognitivamente gerenciável.

Os demais princípios do PON não foram considerados relevantes para o objetivo e escopo deste trabalho. O princípio de *Gerenciamento de Complexidade* é apenas aplicado quando o sistema não faz uso de modularidade devido a integração heterogênea ou outros motivos, o que não é o caso do contexto de seleção de POCT para sistemas embarcados. O princípio de *Integração Cognitiva* não é relevante para a análise apresentada neste trabalho pois é aplicado apenas quando múltiplos diagramas são usados para representar um sistema, o que não é o caso da seleção de POCT. Por fim, o princípios do *Ajuste Cognitivo* e da *Codificação Dupla* não foram abordados pois considerou-se que nenhuma das notações avaliadas contempla as características associadas a estes princípios. O primeiro sugere usar dois dialetos diferentes nas notações visuais, um para usuários experientes e outro para iniciantes. O segundo sugere usar textos como algo redundante dentro da notação para reforçar o significado dos elementos visuais.

III. Trabalhos Relacionados

Vários trabalhos propuseram abordagens distintas para especificar a seleção de POCT. Por exemplo, AspectJ [16] é uma linguagem de programação textual que faz extensão da linguagem Java para definir um mecanismo dinâmico de intercepção transversal na execução de um programa. Esse mecanismo é suportado por construções denominadas pontos de junção que representam os POCT. AspectJ é a base de muitos trabalhos que visam lidar com características transversais e especificar seleções de POCT. Todavia, entender a sua semântica não é trivial, pois é necessário um esforço adicional para aprender a sintaxe, os símbolos e as regras para o uso correto da AspectJ.

A notação JPDD [2] é uma abordagem de modelagem para visualizar seleções de POCT com base na semântica da UML. Os engenheiros criam diagramas dos eventos do sistema, símbolos de objetos e arestas de fluxo de objetos através da adoção de símbolos de ação que representam um fluxo de execução para atingir um POCT. Embora a JPDD permita especificar seleções de POCT de maneira uniforme e independente da linguagem de programação, o uso dos símbolos da UML no JPDD não é de fácil compreensão pois difere bastante do uso tradicional dos mesmos elementos. Além disso, os engenheiros precisam ter uma grande experiência em AOSD para projetar os JPDDs adequadamente no contexto de requisitos funcionais e não funcionais.

A técnica Theme/UML [11], [17] é uma abordagem baseada em UML que suporta a separação simétrica de características transversais em vez de separação assimétrica. A última é suportada na maioria das abordagens de AOSD. Fragmentos de comportamento e/ou estruturas que representam uma característica transversal são encapsulados em um único tema. A integração entre temas é definida como uma ligação que indica quais elementos de um tema são afetados por elementos de outros temas. De alguma forma, isso é semelhante à especificação dos pontos de junção e a relação entre elementos funcionais afetados por aspectos [16]. No entanto, a especificação de como um tema afeta outros temas não é adequada devido à falta de escalabilidade, ou seja, em um sistema com uma grande quantidade de características transversais, a especificação do relacionamento de vínculo para expressar a integração desses temas dificulta na compreensão, manutenção e na evolução do modelo.

Além disso, o trabalho publicado em [18] examina a literatura sobre características transversais para reunir uma lista de métricas de qualidade existentes para avaliar a eficácia de diferentes técnicas. Aqueles autores concluíram que existe uma lacuna crítica de busca pelas melhores práticas quanto a especificação de seleções de POCT, dificultando a avaliação dos métodos e técnicas por meio da avaliação empírica. A falta de parâmetros de referência adequados afeta a comparação objetiva e a validação empírica das pesquisas no tema.

Em [19], um experimento controlado compara a notação visual (JPDD) e uma notação textual (Tracematch [20]) para especificar seleções de POCT em relação a capacidade das técnicas de facilitar a compreensão de expressões lógicas. Esse trabalho envolveu 35 alunos que realizaram dois tipos de tarefas de compreensão com base em 28 seleções de POCT criadas artificialmente. Cada participante realizou as mesmas tarefas. Segundo aqueles autores, os resultados do experimento indicaram que a JPDD é melhor em relação ao Tracematch na maioria dos casos. O presente trabalho tem semelhanças com [19] pois compara notações gráficas e textuais com o objetivo de quantificar a compreensão da especificação de seleções de POCT. No entanto, nosso trabalho é complementar e se concentra em avaliar as propriedades perceptivas das notações, e não a semântica das especificações de seleção de POCT.

Uma questão importante no entendimento de uma especificação de seleção de POCT é a falta de conhecimento sobre o paradigmas voltados ao encapsulamento das características transversais, e.g., AOSD, a partir dos quais o conceito de ocorrências de características transversais é derivado. Na AspectJ, as ocorrências de características transversais são pontos bem definidos na execução do programa, por exemplo, variáveis, chamadas de métodos, construtores e outros elementos comportamentais. Contudo, essa definição não é abrangente o suficiente para o nível de modelagem e existe uma complexidade associada quando se pretende modelar uma seleção de POCT. Por exemplo, nas linguagens de modelagem, é necessário unificar conceitos suportados em várias linguagens de programação em elementos de modelagem mais abstratos. Portanto, as linguagens de modelagem são mais ricas em termos de conceitos (i.e., elementos de modelagem) que podem servir como pontos de ocorrências de características transversais. Porém, a relação entre os níveis de abstração usados na modelagem e na execução do software é menos clara do que no caso das linguagens de programação.

Uma boa compreensão é uma meta qualitativa importante quando se pretende abstrair um conceito computacional em uma notação, seja ela gráfica ou textual. Portanto, considerando a grande ocorrência de características transversais no software embarcado, é essencial haverem mecanismos para avaliar a especificação de seleções de POCT. Tais mecanismos devem fornecer informações quantitativas para permitir uma avaliação mais precisa dos artefatos. Até onde sabemos, este

trabalho é o primeiro a avaliar notações distintas (i.e., notações gráficas, e.g., JPDD e Theme/UML, e também notações textuais, e.g., AspectJ) para seleção de POCT seguindo as premissas do framework PON. Além disso, nenhum outro estudo empírico apresentado no estado da arte da engenharia de software analisa as propriedades perceptivas de notações relacionadas a AOSD através de métricas quantitativas como as propostas neste trabalho.

IV. Métricas e Modelo de Qualidade para Avaliar as Propriedades Cognitivas

Atualmente, pode-se encontrar na literatura de engenharia de software várias métricas e modelos de qualidade que podem ser usados para extrair informações quantitativas de artefatos de software. No entanto, até onde sabemos, não existe um modelo de avaliação ou de qualidade criado a partir da perspectiva das premissas do framework PON que possibilita quantificar as propriedades perceptivas das notações gráficas e/ou textuais, possibilitando avaliar os seus efeitos cognitivos.

Assim, além de contribuir com uma avaliação da compreensão da especificação de seleção de POCT em sistemas embarcados, este trabalho propõem um novo modelo de qualidade baseado no framework PON. Um conjunto de métricas denominado *Metrics Suite for PON conceptual framework* (MS4PON) foi criado. O objetivo deste conjunto de métricas é quantificar as propriedades perceptivas de notações gráficas e textuais na especificação de seleções de POCT. As métricas do MS4PON foram combinadas em um modelo de qualidade apresentado na Figura 1. Esse modelo de qualidade faz uso de 5 princípios da teoria associada ao framework PON. O MS4PON quantifica os elementos, símbolos e a sintaxe usados na aplicação de uma notação gráfica ou textual para a especificação da seleção de POCT. As métricas são:

- **Number of elements (NE):** essa métrica contabiliza o número de elementos contidos no modelo de seleção de POCT, independentemente da sua especificação ser em formato textual ou gráfico. Quanto maior o NE, mais complexo será a especifição
- **Number of Logical Expressions (NLE):** essa métrica é definida como a quantidade de expressões lógicas agnósticas[1] contidas na especificação da seleção de POCT. As expressões lógicas indicam os critérios de seleção implícitos ou explícitos para a seleção dos POCT. Nos nossos experimentos, os valores NLE são iguais para cada um dos POCT, independente da notação que foi aplicada. O objetivo é mensurar a expressão lógica que representa o critério de seleção que deve ser especificado usando uma notação. O NLE permite gerar valores relativos viabilizando analises comparativas entre notações e fornecendo resultados mais precisos e homogêneos.
- **Number of Visual Distance (NVD):** essa métrica quantifica o número de elementos diferentes que são especificados na seleção de POCT. Em geral, quanto maior o número de elementos diferentes disponíveis na notação, maior é o NVD e, consequentemente, a especificação de seleção de POCT é compreendida com maior precisão.
- **Number of Semantically Immediate Elements (NSIE):** essa métrica determina o número total de elementos semanticamente imediatos especificados na seleção de POCT. Um elemento é semanticamente imediato se for fácil inferir seu significado a partir de sua aparência ou mnemônico. Quanto maior o NSIE, mais fácil será de entender a especificação do POCT em uma determinada notação. Um elemento ou símbolo semanticamente imediato deve ter em sua natureza uma associação com: (i) *similaridades funcionais*, por exemplo, o método elemento "before()" no AspectJ usa um termo que expressa o conceito de fazer algo antes (a palavra "before") trazendo uma similaridade funcional; ou (ii) *metáforas*, por exemplo, um desenho de um boneco pode representar uma pessoa nos diagramas de casos de uso da UML; ou (iii) quaisquer *associações culturais*, por exemplo, um grande "X" em um diagrama de sequência da UML pode significar uma comunicação fechada ou interrompida.
- **Number of Semantically Opaque Elements (NSOE):** essa métrica contabiliza o número de elementos semanticamente opacos especificados na seleção de POCT. Um elemento é semanticamente opaco (ou convencionado) se houver uma convenção puramente arbitrária entre a sua aparência e o seu significado. Quanto maior for o NSOE, mais elementos neutros ou convencionados são usados na especificação. Por exemplo o uso excessivo de retângulos nos diagramas da notação gráfica Theme/UML corresponde a um uso excessivo de elementos convencionados que, por sua vez, são puramente arbitrários quanto aos seus significados. NSOE representa o ponto zero na escala entre o NSIE e o NSPE, pois não aumenta a percepção cognitiva, mas também não sugere erroneamente um significado diferente de sua aparência, i.e., é um elemento convencionado que deve ser aprendido [14].
- **Number of Semantically Perverse Elements (NSPE):** essa métrica quantifica o número de elementos semanticamente perversos especificados na seleção de POCT. Um elemento pode ser semanticamente perverso (ou ter falso mnemônico), quando o seu significado é oposto à sua aparência. Quanto maior o NSPE, pior será o entendimento da especificação. Um elemento ou símbolo semanticamente perverso possui em sua natureza algumas

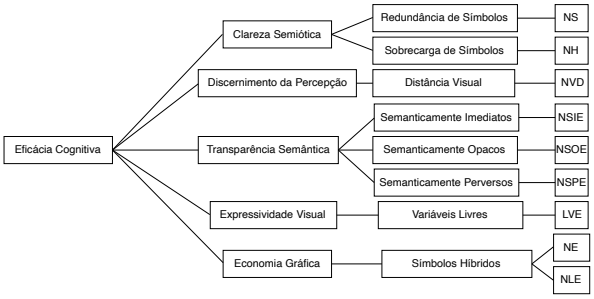


Figura 1. Modelo de qualidade para avaliar as propriedades cognitivas

[1] i.e., expressões lógicas abstratas e independentes da notação ou modelo de seleção de POCT

propriedades léxicas associadas a mnemônicos falsos que podem sugerir um significado totalmente diferente de sua aparência para um um leitor iniciante. Algumas convenções da UML apresentam essa propriedade indesejável, e.g., o símbolo "merge" de pacotes da UML [14].

- **Level of Expressiveness (LVE):** essa métrica determina o número de variáveis visuais usadas na notação para especificar a seleção de POCT. A LVE contabiliza a quantidade usada de variáveis de comunicação em notações visuais, e.g., forma, textura, cor, etc. Conforme [14], os valores de LVE variam de zero a oito. Portanto, quanto maior for o LVE, mais a notação aproveita a vantagem de ser uma notação gráfica. Quanto menor o LVE, mais próximo de uma notação textual é a notação visual.
- **Number of Synographs (NS):** essa métrica é definida como o número total de elementos sinográficos especificados na seleção de POCT. Elementos sinográficos aparecem quando há uma redundância no uso de símbolos ou elementos para representar a mesma construção semântica, e.g., interfaces na UML podem ser representadas de duas formas diferentes. Quanto maior o NS, pior é a compreensão da especificação. Uma notação gráfica satisfaz os requisitos de um sistema de notação quando se fornece uma correspondência individual entre símbolos e seus conceitos associados [14]. O elemento sinográfico é uma anomalia que ocorre quando não existe essa correspondência individual, levando à redundância de símbolos e diminuindo a compreensão da especificação ou notação.
- **Number of Homographs (NH):** essa métrica determina o número total de elementos homógrafos na especificação da seleção de POCT. Um homógrafo ocorre quando há vários elementos semânticos distintos que podem ser representadas pelos mesmos símbolos ou elementos. Quanto maior for o NH, pior é para entender a especificação. Esse é o pior tipo de anomalia em uma notação, seja ela gráfica ou textual, pois leva à ambiguidade e a potenciais erros de interpretação [14]. Por exemplo, Theme/UML evidencia essa anomalia pois a mesma convenção gráfica (e.g., retângulo) é usada para representar vários elementos semanticamente distintos: *subject*, *class*, *lifelines*, etc.

As métricas são combinadas em um modelo de qualidade para avaliar os cinco princípios definidos no framework PON. O princípio da clareza semiótica (*Semiotic Clarity*) é definida pela redundância (*Symbol Redundancy*) e sobrecarga (*Symbol Overload*) de símbolos. O primeiro é impactado pelo número de sinógraficos (NS) encontrados na especificação de seleções de POCT, enquanto o último pelo número de homógrafos (NH). O princípio do discernimento da percepção (*Perceptual Discriminability*) é definido pela distância visual que, por sua vez, é diretamente afetado pelo número de elementos diferentes usados (NVD). O princípio da transparência semiótica (*Semiotic Transparency*) é influenciado pelo número de elementos considerados semanticamente imediatos (NSIE), semanticamente opacos (NSOE) e semanticamente perversos (NSPE). Idealmente, uma notação deve apresentar o maior número possível de elementos semanticamente imediatos e zero elementos semanticamente perversos. O princípio da expressividade visual (*Visual Expressiveness*) é influenciado pela quantidade de variáveis livres (*Free Variables*) que é diretamente afetada pelo número de canais de comunicação visual (LVE). Uma notação sem variáveis de comunicação visuais é chamada de notação não visual (ou textual), enquanto uma notação que usa todas as variáveis de comunicação visuais é considerada visualmente saturada [14]. Por fim, o princípio da economia gráfica (*Graphic Economy*) é afetado pela quantidade de símbolos híbridos (*Hybrid Symbols*) que são medidos pelo NE e NLE. Uma notação aumenta sua complexidade proporcionalmente ao aumento da quantidade de elementos que são especificados por expressão lógica.

Para ilustrar como essas métricas são calculadas sobre as notações gráficas ou textuais, um exemplo simples de especificação da seleção de POCT é apresentado a seguir. Os POCT a serem selecionados são as ações de chamada para métodos cujos nomes começam com *set* (métodos *setter*) das classes cujos objetos são acessados simultaneamente por vários objetos ativos (i.e., threads/processos). Este POCT será nomeado como *JPExclusiveSetCalls*. Destaca-se que objetivo do MS4PON (e deste exemplo especificamente) não é a comparação das notações (gráficas ou textuais), mas sim a avaliação da compreensão da especificação da seleção de POCT, de forma indireta, através da avaliação dos efeitos cognitivos associados com as propriedades de percepção das notações medidas através das métricas propostas.

A Figura 2 apresenta o JPDD que especifica a seleção dos POCT mencionados. Este JPDD especifica a seleção de mensagens relacionadas aos métodos com nomes que começam com "set" e é enviado para objetos de classes que são anotadas com o estereótipo $<<MutualExclusionResource>>$ [21]. A Figura 3 mostra exatamente a mesma especificação da seleção de POCT porém descrita usando a notação Theme/UML. Adicionalmente, a Listagem 1 apresenta essa mesma seleção de POCT através da linguagem AspectJ que cria uma analogia dos elementos usados por meio de uma representação textual com a mesma intenção conceitual que foi expressa através das notações gráficas JPDD e Theme/UML.

A Tabela I apresenta as métricas calculadas para essas três especificações do *JPExclusiveSetCalls*. Como mencionado, o NLE mede o critério de seleção independente da notação que deve ser especificado usando as três notações. Assim, para o *JPExclusiveSetCalls*, esse critério é "`(selecione o elemento que é igual a ação`

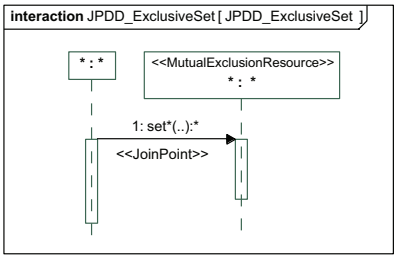


Figura 2. JPDD: seleção das mensagens cujo nome começa com "set"

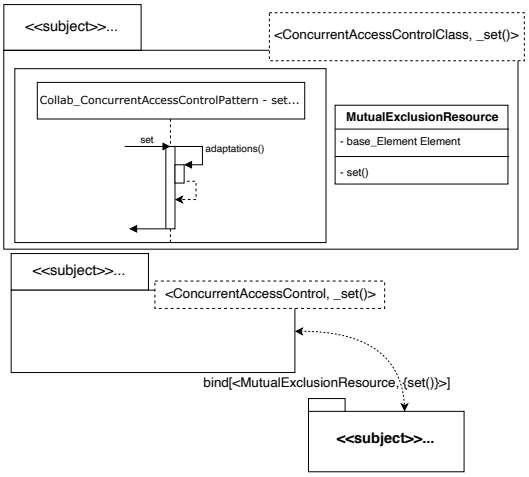


Figura 3. Theme/UML: seleção das mensagens cujo nome começa com "set"

Listagem 1
ASPECTJ: SELEÇÃO DAS CHAMADAS DE MÉTODOS "SET"

```
1 aspect ConcurrentAccessControl{
2   pointcut pcBeforeWrite() :
3     call(* MutualExclusionResource.set(..))
          && !within(ConcurrentAccessControl);
4   before(): pcBeforeWrite() {
5      //Crosscutting Concerns Implementation
6     }
7 }
```

Tabela I
VALORES DAS MÉTRICAS MS4PON PARA JPEXCLUSIVESETCALLS

Notation	NE	NLE	NVD	NSIE	NSOE	NSPE	LVE	NS	NH
JPDD	7	6	4	0	4	1	1	0	0
AspectJ	7	6	7	1	7	0	0	0	0
Theme/UML	24	6	3	0	6	1	2	1	10

de enviar mensagem) E (todos os métodos E nome que começam com 'set') E (todas as classes E qualquer nome E classe anotada com <<MutualExclusionResource>>)". Portanto, o NLE é o mesmo para as três especificações e, sendo assim, é possível fazer uma combinação do NLE com as outras métricas para gerar valores relativos, i.e., taxa calculadas com o valor de outras métricas dividido por NLE.

Na JPDD (Figura 2), o NE indica a quantidade dos elementos visíveis usados na especificação, por exemplo, a Figura 2 mostra duas *lifelines*, dois retângulos verticais definidos pela UML como símbolo para comportamento, uma linha de conexão que significa uma ação de envio de mensagem e dois estereótipos como elementos de texto (<<JoinPoint>>, <<MutualExclusionResouce>>). Os outros símbolos, por exemplo "∗, :;", não são reconhecidos como elementos, pois os mesmos são atributos do próprio elemento, por exemplo, o "*" significa o nome da classe na *lifeline*; outro exemplo seriam os "1: set * (..): *" que indicam símbolos léxicos usados no modelo, i.e., não são elementos em si como o símbolo que representa o relacionamento de conexão com uma seta que simboliza o envio da mensagem. Na AspectJ (Listagem 1), o NE considera os elementos textuais usados para descrever o aspecto em si: o pointcut (linha 2); a chamada do método "set" (que é um POCT implícito); a classe *MutualExclusionResource*; o método *set(..)*; os símbolos *!within* e *before()*.

Por sua vez, a notação usada na Theme/UML (Figura 3) apresentou um NE surpreendentemente grande na especificação de seleção de POCT. Todos os retângulos que significam classes, atributos, métodos, *subjects*, *lifelines*, comportamentos e relacionamentos de conexões que enviam símbolos de mensagens foram contabilizados. Além disso, a aplicação da Theme/UML, apresentou um alto valor para o NH devido à sobrecarga de retângulos usados na especificação, causando uma anomalia relacionada ao grande número de elementos homógrafos, i.e., o uso da mesma convenção gráfica para expressar vários elementos distintos. Por outro lado, o JPDD tem o NH igual a zero, uma vez que os elementos utilizados na especificação têm uma correspondência de um para um entre o símbolo e sua semântica. O mesmo ocorreu no uso da notação textual AspectJ onde o NH é zero pois nenhum elemento da linguagem textual viola os requisitos *sine qua non* para se denominar como um sistema notacional [14].

Os critérios mencionados para calcular as métricas do MS4PON demonstram que é possível aplicar as métricas propostas para avaliar as especificações da seleção dos POCT, independentemente de usar uma notação gráfica ou textual. Portanto, o elemento que pode ser visto em uma especificação gráfica também é um elemento que pode ser visto em uma especificação textual. A diferença está na forma de como o elemento é expresso e apresentado.

V. EXPERIMENTOS E RESULTADOS

Esta seção discute os experimentos empíricos realizados para avaliar a especificação da seleção de POCT usando duas notações gráficas (JPDD e Theme/UML) e uma notação textual (AspectJ). Essas três notações foram escolhida devido a grande quantidade de citações de trabalhos relacionados e por serem amplamente conhecidas dentro do contexto da AOSD [1]. O objetivo é avaliar os efeitos cognitivos e as propriedades de percepção dessas notações, empregando o modelo de qualidade proposto através do cálculo das métricas MS4PON descritas na seção anterior.

A fonte de dados para essas experiências inclui 15 POCT comumente encontrados no desenvolvimento de software para sistemas ciber-físicos e sistemas embarcados de tempo real [6]. Esses POCT foram criados pelos próprios autores, em três estudos de caso realizados em trabalhos anteriores [12], [13]: (i) o sistema de controle de movimento de um veículo aéreo não tripulado; (ii) o sistema de controle de um sistema industrial de empacotamento; e (iii) o sistema de controle de movimento de uma cadeira de rodas automatizada. Os experimentos realizados neste trabalho consistem em criar três especificações para cada uma das quinze POCT usando as notações JPDD, Theme/UML e AspectJ, totalizando 45 especificações distintas. Em seguida, as métricas do MS4PON foram calculadas sobre cada especificação distinta para todas as seleções de POCT. Embora essas notações usadas permitam

Tabela II
MÉTRICAS PARA TODAS AS ESPECIFICAÇÕES DE SELEÇÃO DE POCT

Notação	NE	NLE	NVD	NSIE	NSOE	NSPE	LVE	NS	NH
JPDD	87	72	55	0	55	22	1,333	0	16
AspectJ	201	72	201	28	201	0	0	0	0
Theme/UML	360	72	45	0	90	15	2	15	150

uma especificação completa baseada na AOSD, os experimentos e seus resultados consideram apenas a parte das notações relacionadas a especificação das seleções de POCT.

A Tabela II mostra os valores das métricas do MS4PON calculadas a partir das 45 especificações de seleção de POCT. Os valores representam a soma dos valores individuais de cada métrica para cada especificação, com exceção do valor da LVE, que é calculado como um valor médio. A LVE representa a expressividade visual da notação de acordo com o framework PON [14], cujo valor deve variar de zero a oito. A soma da métricas LVE de todas as especificações não representa adequadamente o conceito de expressividade visual porque indicaria frequentemente uma notação saturada (i.e., valor maior que 8) conforme o número de especificações aumenta.

As métricas da MS4PON quantificam os recursos de uma notação usados para especificação de seleção de POCT. No entanto, para melhorar a análise e a comparação dos resultados, foram criadas taxas (i.e., valores relativos) através da associação entre as métricas. A Tabela III mostra os valores das métricas divididos por NE ou NLE ou ambos. As taxas permitem quantificar melhor a eficácia cognitiva de uma notação sob a perspectiva do framework PON. O objetivo é identificar, para cada métrica, como a propriedade de percepção é distribuída pelos elementos descritos nas especificações.

A taxa NVD/NE indica quantos elementos diferentes aparecem para cada elemento usado na especificação. Para essa taxa a notação textual AspectJ apresentou um valor mais alto em comparação com as notações gráficas JPDD e Theme/UML. Isso significa que a linguagem AspectJ emprega mais elementos diferentes para cada elemento usado na especificação de seleção de POCT. O mesmo raciocínio é válido para a relação NVD/NLE que considera o número de elementos diferentes por expressão lógica. Quanto maior a distância visual por expressão lógica, melhor o entendimento da especificação. No entanto, é importante destacar que um bom valor para essas proporções mencionadas deve ser próximo a um (1), i.e., a notação possui apenas um elemento distinto por expressão lógica. Valores menores que 1 indicam que existem elementos implícitos na especificação e valores maiores que 1 indicam que a notação é saturada. Ambas as situações dificultam de alguma forma o entendimento da especificação.

As taxas relacionadas ao NSIE indicam quantos elementos semanticamente imediatos são usados por elemento na especificação ou por expressão lógica. Essas proporções identificam quão fácil é inferir o significado correto dos elementos descritos na especificação pela aparência dos elementos e símbolos. Por exemplo, a proporção NSIE/NE para o AspectJ é um pouco mais alta do que nas outras técnicas, ou seja, a AspectJ possui mais elementos (na especificação) que

Tabela III
TAXAS DAS MÉTRICAS MS4PON

Taxas	AspectJ	JPDD	Theme/UML	Valores Ideais*
NE/NLE	2.790	1.210	5.000	1
NVD/NE	1.000	0.632	0.125	[1,2)
NVD/NLE	2.792	0.764	0.625	[1,2)
NSIE/NE	0.139	0.000	0.000	>0
NSIE/NLE	0.389	0.000	0.000	>0
NSOE/NE	1.000	0.632	0.250	0
NSOE/NLE	2.792	0.764	1.250	0
NSPE/NE	0.000	0.250	0.040	0
NSPE/NLE	0.000	0.306	0.208	0
LVE	0.000	1.333	2.000	8
NS/NE	0.000	0.000	0.042	0
NS/NLE	0.000	0.000	0.208	0
NH/NE	0.000	0.184	0.417	0
NH/NLE	0.000	0.222	2.083	0

* A notação ideal não deve ter nenhum homógrafos, sinógrafos, ou elementos perversos ou opacos que são elementos convencionados arbitrariamente. Por outro lado, quanto maior o número de elementos semanticamente imediatos, melhor a compreensão.

apresentam entendimento semântico implícito que a JPDD e a Theme/UML. Isso pode ter ocorrido devido ao fato de que JPDD e Theme/UML são extensões da UML. De acordo com [14], a maioria das notações que estendem a UML não foi projetada considerando as propriedades perceptivas e cognitivas que uma notação visual deve ter.

Ao analisar as taxas apresentadas na Tabela III, percebe-se que AspectJ é mais eficaz em relação a compreensão das especificações de selecão de POCT em comparação com JPDD e Theme/UML. Esses resultados empíricos indicam que a especificação de seleções de POCT através de notações gráficas apresenta pouco discernimento na eficácia cognitiva das representações visuais. Portanto, essa análise realizada sob a perspectiva do framework PON aponta para o entendimento de que o uso de notações gráficas para a especificação da seleção de POCT é um processo inconsciente, pois as especificações são de alguma maneira difíceis de se entender quando os engenheiros de software não tem experiência suficiente na notação gráfica usada.

VI. LIMITAÇÕES

Embora os resultados obtidos forneçam o suporte adequado para as conclusões deste trabalho, alguns fatores podem ter influenciado a avaliação realizada, como é habitual em qualquer estudo empírico. Os fatores de ameaças à validade interna da avaliação podem surgir do fato de que os resultados da eficácia cognitiva podem ser influenciados pela experiência de quem lê as especificações, no caso, os autores deste trabalho. Entretanto, consideramos o risco desse fato afetar os resultados obtidos como baixo, uma vez que o modelo de qualidade proposto e os resultados experimentais são baseados na perspectiva do framework PON. Esse risco também é atenuado, porque as POCT usadas no experimento são de estudos de caso do mundo real, e não criados artificialmente.

Os resultados alcançados podem ter relação direta com a qualidade das especificações criadas. A falta de experiência em lidar com com características transversais (tanto na especificação quanto na implementação do software embarcado), e também a no uso das notações visuais, pode levar a

criação de especificações com baixa qualidade, o que impacta na sua compreensão. O risco de resultados obtidos não representarem a realidade é baixo pois os autores possuem bastante experiência nos tema mencionados. Além disso, o framework PON e as métricas MS4PON avaliam as propriedades físicas perceptivas (i.e. sintaxe) das notações e não sua semântica, o que nos permite ter confiança nos resultados alcançados.

Outra ameaça à validade pode estar relacionada a generalidade dos resultados, uma vez que estão limitados aos três casos de estudos do domínio de sistemas embarcados. O modelo de qualidade proposto e as métricas MS4PON são genéricas e podem ser usadas para avaliar outras notações visuais e textuais. No entanto, dentro do escopo da especificação de seleção de POCT, a variedade de tipos das ocorrências usadas nos experimentos nos permite acreditar que resultados semelhantes serão obtidos em outros domínios de aplicação.

VII. Conclusões e Trabalhos Futuros

Este trabalho avalia a compreensão da especificação de seleções de POCT de forma indireta, através da avaliação das propriedades de percepção de notações gráficas e textuais para três técnicas distintas (AspectJ, JPDD e Theme/UML). Para isso, foram propostos um modelo de qualidade e um conjunto de métricas baseados nas premissas do framework conceitual *Physics of Notations* (PON) [14].

Os resultados experimentais mostraram evidências empíricas de que a especificação gráfica da seleção de POCT apresenta baixo discernimento na eficácia cognitiva das representações visuais. As notações gráficas avaliadas nos experimentos apresentam um grande número de elementos homógrafos para especificar as seleções dos POCT. Os homógrafos são anomalias que levam à ambiguidade e potencializam o erro nas interpretações dos modelos [14].

Além disso, a distância visual das notações gráficas avaliadas ficou aquém do esperado. Os resultados mostram um valor máximo de 0,764 elementos diferentes por expressão lógica. Isso significa que a maioria dos elementos visuais são iguais ou semelhantes, afetando negativamente a compreensão e a cognição dos engenheiros. Por outro lado, a notação textual avaliada (AspectJ) mostra uma razão de 2,792, indicando que, para cada expressão lógica, existem mais de dois elementos diferentes, i.e., a distância visual entre os elementos na AspectJ é melhor que em JPDD e Theme/UML.

O grande número de elementos semanticamente perversos (NSPE) nas notações gráficas avaliadas chama a atenção. Na JPDD e particularmente Theme/UML, a especificação da seleção de POCT inclui elementos ou símbolos cuja natureza possui um falso mnemônico. Isso pode induzir um engenheiro de software iniciante a inferir um significado diferente do que se apresenta na especificação. Além disso, o nível de expressividade (LVE) das notações gráficas avaliadas se apresentou muito baixo, próximo a de uma notação textual, pois elas apresentam poucas variáveis de comunicação visual (e.g., forma, textura, cores, etc.). No entanto, os indícios apontam que a JPDD e a Theme/UML não se beneficiam das vantagens esperadas de uma linguagem visual. Moody [14] afirma que isso é semelhante ao que ocorre com a maioria das notações visuais usadas na engenharia de software.

Como trabalho futuro, planeja-se avaliar outras notações gráficas e textuais para especificar a seleção de POCT mais complexas. Além disso, considerando que as notações gráficas não melhoraram o entendimento da seleção de POCT, pretende-se criar em uma nova técnica para especificação de POCT através de um modelo que inclui os elementos textuais e gráficos (i.e., dual coding [14]), visando aumentar a eficácia cognitiva de suas propriedades de percepção.

Referências

[1] R. Filman, T. Elrad, S. Clarke, and M. Akşit, Eds., *Aspect-Oriented Software Development*, 1st ed. Addison-Wesley Professional, 2004.

[2] D. Stein *et al.*, "Join point designation diagrams: a graphical representation of join point selections," *Int. Journal of Software Engineering and Knowledge Engineering*, vol. 16, no. 03, pp. 317–346, 2006.

[3] T. Elrad *et al.*, "Aspect-oriented modeling: Bridging the gap between implementation and design," in *Generative Programming and Component Engineering*, D. Batory *et al.*, Eds. Springer, 2002, pp. 189–201.

[4] J. Kienzle, W. Al Abed, and J. Klein, "Aspect-oriented multi-view modeling," in *Proc. of AOSD'09*. ACM, 2009, pp. 87–98.

[5] S. Yang and Z. Wei-Dong, "Aspect-oriented modeling in concurrent system," in *IEEE Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*. IEEE, 2019, pp. 836–840.

[6] E. P. Freitas *et al.*, "DERAF: A high-level aspects framework for distributed embedded real-time systems design," in *Early Aspects: Current Challenges and Future Directions*. Berlin, Heidelberg: Springer, 2007, pp. 55–74, doi: 10.1007/978-3-540-76811-1_4.

[7] M. Mongiovì *et al.*, "Specifying and identifying widely used crosscutting concerns," *Knowledge-Based Systems*, vol. 126, pp. 20 – 32, 2017.

[8] V. Acreţoaie, H. Störrle, and D. Strüber, "VMTL: a language for end-user model transformation," *Software & Systems Modeling*, vol. 17, no. 4, pp. 1139–1167, 2018.

[9] M. U. Khan *et al.*, "AspectOCL: using aspects to ease maintenance of evolving constraint specification," *Empirical Software Engineering*, vol. 24, no. 4, pp. 2674–2724, 2019.

[10] Object Management Group (OMG), "Unified Modeling Language version 2.5.1," 2017, https://www.omg.org/spec/UML/2.5.1/.

[11] S. Clarke, "Extending standard uml with model composition semantics," *Science of Computer Programming*, vol. 44, no. 1, pp. 71–100, 2002.

[12] M. A. Wehrmeister, E. P. de Freitas, A. P. D. Binotto, and C. E. Pereira, "Combining aspects and object-orientation in model-driven engineering for distributed industrial mechatronics systems," *Mechatronics*, vol. 24, no. 7, pp. 844 – 865, 2014, doi: 10.1016/j.mechatronics.2013.12.008.

[13] M. A. Wehrmeister, C. E. Pereira, and F. J. Rammig, "Aspect-oriented model-driven engineering for embedded systems applied to automation systems," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 4, pp. 2373–2386, 2013, doi: 10.1109/TII.2013.2240308.

[14] D. Moody, "The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering," *IEEE Transactions on Software Engineering*, vol. 35, no. 6, pp. 756–779, 2009.

[15] D. Jones and S. Gregor, "The anatomy of a design theory," *Journal of the Association for Information Systems*, vol. 8, no. 5, p. 1, 2007.

[16] G. Kiczales *et al.*, "An overview of aspectj," in *European Conference on Object-Oriented Programming*. Springer, 2001, pp. 327–354.

[17] E. Baniassad and S. Clarke, "Theme: An approach for aspect-oriented analysis and design," in *26th Int. Conf. on Software Engineering*. IEEE, 2004, pp. 158–167.

[18] R. R. McFadden and F. J. Mitropoulos, "Survey of aspect mining case study software and benchmarks," in *2013 Proceedings of IEEE Southeastcon*, April 2013, pp. 1–5.

[19] D. Stein and S. Hanenberg, "Comparison of a visual and a textual notation to express data constraints in aspect-oriented join point selections: A controlled experiment," in *Int. Conf. on Program Comprehension*. IEEE, 2011, pp. 141–150.

[20] C. Allan *et al.*, "Adding trace matching with free variables to aspectj," *Sigplan Notices - SIGPLAN*, vol. 40, pp. 345–364, 10 2005.

[21] Object Management Group (OMG), "UML profile for Modeling and Analysis of Real-Time Embedded systems (MARTE), version 1.2," 2018, https://www.omg.org/spec/MARTE/.