

UNIVERSIDADE ESTADUAL DE MARINGÁ  
CENTRO DE TECNOLOGIA  
DEPARTAMENTO DE INFORMÁTICA

EDUARDO GERMANO PERISSATTO

**IDENTIFICAÇÃO DE ANOMALIAS ARQUITETURAIS EM  
PROJETOS DE LINHA DE PRODUTO DE SOFTWARE**

MARINGÁ  
2017

EDUARDO GERMANO PERISSATTO

## IDENTIFICAÇÃO DE ANOMALIAS ARQUITETURAIS EM PROJETOS DE LINHA DE PRODUTO DE SOFTWARE

Trabalho de Conclusão de Curso apresentado  
ao Departamento de Informática da  
Universidade Estadual de Maringá, como  
requisito parcial para a conclusão do curso de  
Bacharelado em Informática

Orientadora: Prof<sup>a</sup>. Dr<sup>a</sup>. Thelma Elita Colanzi  
Lopes

MARINGÁ  
2017

## RESUMO

Na abordagem de Linha de Produto de Software (LPS) é desenvolvida uma família de produtos para um dado domínio. Um dos principais artefatos gerados é a Arquitetura de Linha de Produto (ALP), que contém o projeto comum a todos os produtos deriváveis da LPS. Para que uma ALP tenha bom custo-benefício é necessário que os componentes da mesma estejam modularizados, o que não é algo trivial de se obter, devido a diversos fatores – possivelmente conflitantes – a serem considerados. Considerando este problema de otimização de projetos de ALP, foi proposta a *Multi-Objective Approach for Product-Line Architecture* (MOA4PLA), cuja automatização se deu por meio da ferramenta OPLA-Tool, que gera novas soluções a partir de uma solução inicial dada (o projeto original da ALP). No entanto, essas soluções possuem o inconveniente de apresentar anomalias arquiteturais, que são problemas em nível de arquitetura de software que afetam negativamente sua qualidade. Este trabalho consistiu de uma análise de diversas ALPs geradas pela OPLA-Tool, visando encontrar anomalias arquiteturais. Os resultados foram avaliados para se verificar a influência que a OPLA-Tool exerceu sobre os tipos de anomalias presentes nas soluções geradas, em contraste com os tipos de anomalias previamente existentes nas soluções originais. Por fim, foram propostas diretrizes para se detectar e remover os tipos de anomalias encontrados.

**Palavras-chave:** Linha de Produto de Software, Arquitetura de Linha de Produto, MOA4PLA, OPLA-Tool, anomalias arquiteturais.

## ABSTRACT

In the Software Product Line (SPL) approach is developed a family of products for a given domain. One of the main artifacts generated is the Product Line Architecture (PLA), which contains the common design to all SPL derivable products. In order for an PLA to be cost-effective it is necessary for the components of the PLA to be modularized, which is not trivial to obtain due to several – possibly conflicting – factors to consider. Considering this problem of optimization of PLA projects, it was proposed the Multi-Objective Approach for Product-Line Architecture (MOA4PLA), whose automation was done through the OPLA-Tool, which generates new solutions from a given initial solution (the original PLA design). However, these solutions have the drawback of presenting architectural smells, which are problems at the software architecture level that negatively affect their quality. This work consisted of an analysis of several PLAs, aiming to find architectural smells. The results were evaluated to verify the influence of OPLA-Tool on the types of anomalies present in the generated solutions, in contrast to the types of anomalies previously existing in the original solutions. Finally, guidelines were proposed to detect and remove the types of anomalies found.

**Keywords:** Software Product Line, Product Line Architecture, MOA4PLA, OPLA-Tool, architectural smells.

## LISTA DE FIGURAS

Figura 1: Atividades essenciais de LPS (adaptada de SEI, 2016) .....	13
Figura 2: Exemplo de problema multiobjetivo com duas métricas (COLANZI, 2014) .....	15
Figura 3: Processo da MOA4PLA (adaptada de COLANZI, 2014).....	16
Figura 4: Módulos da OPLA-Tool (COLANZI, 2014) .....	19
Figura 5: Algoritmo de detecção de Link Overload (GARCIA, 2014) .....	23
Figura 6: Algoritmo de detecção de Concern Overload (GARCIA, 2014) .....	23
Figura 7: Exemplos de ocorrência da anomalia Connector Envy (GARCIA, 2014).....	26
Figura 8: Exemplo de ocorrência da anomalia Scattered Parasitic Functionality (GARCIA, 2014) .....	26
Figura 9: Exemplo de ocorrência da anomalia Ambiguous Interfaces (GARCIA, 2014).....	29
Figura 10: Exemplo de ocorrência da anomalia Extraneous Adjacent Connector (GARCIA, 2014) .....	30
Figura 11: Exemplo de ocorrência da anomalia Duplicated Code (KERIEVSKY, 2004) .....	31
Figura 12: Exemplo de solução para o Duplicated Code indicado na Figura 11 (KERIEVSKY, 2004) .....	31
Figura 13: Exemplo de ocorrência de Link Overload e Concern Overload na ALP AGM original (CHOMA NETO, 2017).....	40
Figura 14: Exemplo de ocorrência de Large Class na ALP AGM original (CHOMA NETO, 2017) .....	41
Figura 15: Exemplos de ocorrência de Unused Interface na ALP AGM original (CHOMA NETO, 2017).....	41
Figura 16: Exemplo de ocorrência de Unused Brick na ALP AGM com melhor COE (CHOMA NETO, 2017) .....	42
Figura 17: Exemplo de ocorrência de Connector Envy na ALP Banking original (CHOMA NETO, 2017).....	42
Figura 18: Exemplo de ocorrência de Dependency Cycle na ALP Banking Original (CHOMA NETO, 2017).....	43
Figura 19: Exemplo de ocorrência de Sloppy Delegation na ALP AGM com melhor CM (DIAS, 2016).....	50
Figura 20: Quantidade de ocorrências de anomalias por LPS .....	60

## LISTA DE TABELAS

Tabela 1: Catálogo de anomalias arquiteturais consideradas para este trabalho (adaptada de JOHANSSON; OIZUMI, 2016) .....	22
Tabela 2: Quantidade de elementos arquiteturais nas versões originais da AGM, Banking e MM (adaptada de SANTOS et al., 2017) .....	37
Tabela 3: Quantidade de instâncias de cada LPS presente em cada grupo.....	38
Tabela 4: Thresholds de Link Overload (LO) e Concern Overload (CO) para cada LPS .....	39
Tabela 5: Anomalias na ALP AGM original .....	43
Tabela 6: Anomalias na ALP AGM com Padrão de Projeto Strategy .....	44
Tabela 7: Anomalias na ALP AGM com melhor COE .....	44
Tabela 8: Anomalias na ALP AGM com melhor ACCLASS .....	45
Tabela 9: Anomalias na ALP Banking original .....	45
Tabela 10: Anomalias na ALP Banking com melhor ACCLASS .....	46
Tabela 11: Anomalias na ALP Banking com melhor COE e melhor ED .....	46
Tabela 12: Anomalias na ALP MM original .....	46
Tabela 13: Anomalias na ALP MM com Padrão de Projeto Strategy .....	47
Tabela 14: Anomalias na ALP MM com melhor COE .....	47
Tabela 15: Anomalias na ALP MM com melhor ACCLASS e melhor ED .....	47
Tabela 16: Anomalias na ALP Banking original .....	48
Tabela 17: Anomalias na ALP Banking gerada pela OPLA-Tool.....	48
Tabela 18: Anomalias na ALP MM original .....	48
Tabela 19: Anomalias na ALP MM com melhor DC .....	49
Tabela 20: Anomalias na ALP MM com melhor ED .....	49
Tabela 21: Anomalias na ALP MM com melhor Eleg .....	50
Tabela 22: Anomalias na ALP AGM original .....	51
Tabela 23: Anomalias na ALP AGM com melhor ED e melhor FM .....	52
Tabela 24: Anomalias na ALP AGM com melhor CM .....	53

Tabela 25: Anomalias na ALP MM original .....	53
Tabela 26: Anomalias na ALP MM com melhor ED .....	54
Tabela 27: Anomalias na ALP MM com melhor CM .....	55
Tabela 28: Anomalias na ALP MM com melhor FM.....	56
Tabela 29: Quantidade de anomalias na LPS AGM .....	56
Tabela 30: Quantidade de anomalias na LPS Banking.....	57
Tabela 31: Quantidade de anomalias na LPS MM .....	58

## **LISTA DE ABREVIATURAS E SIGLAS**

AGM	Arcade Game Maker
ALP	Arquitetura de Linha de Produto
Bank	Banking
CO	Concern Overload
LO	Link Overload
LPS	Linha de Produto de Software
MM	Mobile Media
MOA4PLA	Multi-Objective Approach for Product-Line Architecture
MOEA	Multi-Objective Evolutionary Algorithm
OPLA-Tool	Optimization for Product Line Architecture Tool
SBSE	Search Based Software Engineering (Engenharia de Software Baseada em Busca)

## SUMÁRIO

1 Introdução .....	10
2 Otimização de Arquitetura de Linha de Produto .....	12
2.1 Linha de Produto de Software.....	12
2.2 Arquitetura de Linha de Produto.....	13
2.3 Engenharia de Software Baseada em Busca .....	14
2.4 MOA4PLA.....	15
2.5 OPLA-Tool .....	19
2.6 Considerações finais .....	20
3 Anomalias arquiteturais .....	21
3.1 Link Overload .....	21
3.2 Concern Overload .....	22
3.3 Unused Interface .....	24
3.4 Unused Brick .....	24
3.5 Duplicate Component Functionality .....	24
3.6 Sloppy Delegation.....	24
3.7 Dependency Cycle .....	25
3.8 Connector Envy .....	25
3.9 Scattered Parasitic Functionality.....	26
3.10 Large Class.....	27
3.11 Primitive Obsession .....	27
3.12 Lazy Class.....	27
3.13 Long Method.....	28
3.14 Alternative Classes with Different Interfaces .....	28
3.15 Connector Chain Overload .....	28
3.16 Brick Functionality Overload .....	29

3.17 Ambiguous Interfaces .....	29
3.18 Extraneous Adjacent Connector .....	30
3.19 Duplicated Code.....	30
3.20 Oddball Solution .....	32
3.21 Switch Statements.....	32
3.22 Conditional Complexity.....	32
3.23 Combinatorial Explosion .....	32
3.24 Unplanned and Uncontrolled Solution.....	33
3.25 Indecent Exposure.....	33
3.26 Considerações finais .....	33
<b>4 Metodologia do trabalho .....</b>	<b>35</b>
4.1 LPSs analisadas.....	35
4.2 Thresholds utilizados .....	38
4.3 Considerações finais .....	39
<b>5 Análise de anomalias arquiteturais .....</b>	<b>40</b>
5.1 Arquiteturas do Grupo 1 .....	40
5.2 Arquiteturas do Grupo 2 .....	48
5.3 Arquiteturas do Grupo 3 .....	50
5.4 Avaliação dos resultados da análise.....	56
5.5 Proposta de diretrizes para evitar a ocorrência de anomalias .....	60
<b>6 Conclusão.....</b>	<b>63</b>
<b>REFERÊNCIAS .....</b>	<b>64</b>
<b>ANEXOS .....</b>	<b>66</b>
<b>ANEXO A .....</b>	<b>67</b>

## 1 Introdução

Linha de Produto de Software (LPS) é uma abordagem de Engenharia de Software em que se desenvolve uma família de produtos para um determinado domínio, reutilizando os artefatos gerados (LINDEN *et al.*, 2007). Um dos principais artefatos é a Arquitetura de Linha de Produto (ALP), que contém o projeto comum a todos os produtos deriváveis da LPS.

Para se obter bom custo-benefício de uma ALP, é importante que os componentes definidos por ela estejam modularizados (MARTENS *et al.*, 2010), uma vez que ela influencia todas as demais etapas do ciclo de vida de uma LPS (TAYLOR *et al.*, 2010).

No entanto, obter essa modularização é uma tarefa complexa e não trivial, além de possibilitar mais de uma solução viável, porque há diferentes fatores a serem considerados, possivelmente conflitantes entre si, como a coesão e o acoplamento (COLANZI, 2014).

A Engenharia de Software Baseada em Busca (SBSE) oferece uma forma de tratar problemas difíceis de Engenharia de Software como problemas de busca (HARMAN *et al.*, 2009). Um desses problemas é a otimização de projetos de ALP, que pode ser tratada como um problema de otimização multiobjetivo, devido aos diversos fatores que esse problema envolve, que consistem dos objetivos de otimização.

Para automatizar o processo de otimização de projetos de ALP, Colanzi (2014) propôs uma abordagem chamada *Multi-Objective Approach for Product-Line Architecture* (MOA4PLA), que gera diversas soluções a partir de uma solução inicial dada, considerando os diferentes objetivos a serem otimizados. Dentro desse contexto, uma solução é uma alternativa de projeto de ALP encontrada pela abordagem MOA4PLA. A abordagem foi implementada por meio da ferramenta OPLA-Tool (FÉDERLE, 2014) (FÉDERLE *et al.*, 2015), que se baseia em um *framework* de otimização multiobjetivo que apresenta diversos algoritmos evolutivos.

Algumas soluções geradas pela OPLA-Tool possuem o inconveniente de apresentar anomalias arquiteturais. Anomalias arquiteturais são decisões de projeto de arquitetura comumente usadas que afetam negativamente a qualidade do sistema (GARCIA, 2014). Tais anomalias ocorrem porque a ferramenta não foi preparada para evitá-las. Desse modo, é importante investigar quais classes de anomalias arquiteturais costumam estar presentes nas soluções geradas pela OPLA-Tool e propor formas de evitar que tais anomalias sejam inseridas nas soluções durante o processo de otimização.

Sendo assim, este trabalho objetiva investigar as classes de anomalias arquiteturais presentes nas soluções geradas pela ferramenta OPLA-Tool. Como objetivos específicos têm-se: priorizar os tipos de anomalias arquiteturais a serem investigados a partir do catálogo de anomalias disponível na literatura; identificar anomalias recorrentes nas soluções geradas pela OPLA-Tool; e propor diretrizes para se evitar a ocorrência de tais anomalias nas arquiteturas geradas pela ferramenta.

Alternativas de projeto de ALP encontradas em trabalhos anteriores foram analisadas neste trabalho em busca de anomalias. Após a análise identificou-se quais as anomalias mais frequentes e propôs-se diretrizes para evitá-las.

Este trabalho está dividido em capítulos. No Capítulo 2 são introduzidos os conceitos básicos abordados no trabalho. O Capítulo 3 apresenta as anomalias arquiteturais levantadas na literatura. O Capítulo 4 explica a metodologia do estudo realizado. O Capítulo 5 apresenta os resultados da análise. Finalmente, o Capítulo 6 apresenta a conclusão do trabalho.

## 2 Otimização de Arquitetura de Linha de Produto

Neste capítulo são explicados os conceitos básicos abordados no trabalho. A Seção 2.1 apresenta o conceito de Linha de Produto de Software. A Seção 2.2 define a Arquitetura de Linha de Produto. A Seção 2.3 introduz a Engenharia de Software Baseada em Busca. A abordagem MOA4PLA é apresentada na Seção 2.4. Por fim, a ferramenta OPLA-Tool é abordada na Seção 2.5.

### 2.1 Linha de Produto de Software

Linha de Produto de Software (LPS) é uma abordagem de Engenharia de Software que tem sido utilizada na academia e na indústria, e que se caracteriza por desenvolver uma família de produtos de software voltada para um determinado domínio, reutilizando os artefatos dos produtos de software. Esta abordagem visa a redução dos custos e do tempo de desenvolvimento, além da melhoria da qualidade do produto e da facilidade de sua manutenção (LINDEN *et al.*, 2007).

As características (do inglês *features*) de um sistema de software são aspectos relevantes e visíveis ao usuário. Na abordagem de LPS, o núcleo de artefatos reutilizáveis desenvolvido considera as características obrigatórias, variáveis e opcionais. As características obrigatórias são as funcionalidades que devem estar presentes em todos os produtos da LPS. As características variáveis são as que diferem entre os produtos da LPS. As características opcionais são as que podem estar presentes ou ausentes em determinado produto da LPS (KANG *et al.*, 1990).

As variabilidades são fundamentais em uma LPS, porque são o que diferencia uma arquitetura de software tradicional de uma Arquitetura de Linha de Produto (ALP). Elas obrigam, assim, a se pensar em todos os produtos deriváveis da ALP. Os artefatos que indicam as alternativas de projeto para se resolver os pontos de variação são chamados de variantes (LINDEN *et al.*, 2007).

As atividades essenciais no desenvolvimento da LPS são o desenvolvimento do núcleo de artefatos, o desenvolvimento do produto e o gerenciamento, estando todas as atividades relacionadas entre si, conforme ilustrado na Figura 1. O desenvolvimento do núcleo de artefatos envolve a engenharia de domínio, um processo em que são definidos as variabilidades e o escopo da LPS. Já o desenvolvimento do produto envolve a engenharia de

aplicação – na qual são definidos os requisitos, a arquitetura, os componentes e os testes do produto específico, além de relacionados estes itens aos artefatos do domínio e vinculadas as variabilidades conforme a necessidade da aplicação (SEI, 2016). O gerenciamento ocorre paralelamente, garantindo o processo e a evolução dos produtos da LPS.

Figura 1: Atividades essenciais de LPS (adaptada de SEI, 2016)



## 2.2 Arquitetura de Linha de Produto

De acordo com Schmid e Verlage (2002), um bom projeto de arquitetura é fundamental para que o projeto seja bem-sucedido, uma vez que a arquitetura orienta o desenvolvimento do software para que ele atenda aos seus propósitos.

Dentro do contexto de LPS, a ALP, gerada pela engenharia de domínio, é um dos principais artefatos, por representar a estrutura central do projeto da LPS. A ALP contém o projeto comum a todos os produtos deriváveis da LPS, ou seja, abstrai todas as possíveis arquiteturas de produtos que podem ser gerados pela LPS. Ela inclui, portanto, as similaridades e as variabilidades entre os produtos (LINDEN *et al.*, 2007). As variabilidades são importantes por permitirem o reuso na LPS.

Para se conseguir o melhor custo-benefício de uma ALP, de acordo com Martens *et al.* (2010), é necessária uma boa modularização dos componentes por ela definidos. Obter essa modularização não é algo trivial por diversas razões, tais como: a dificuldade de se balancear coesão e acoplamento; a distribuição das características dos produtos da LPS pelos componentes; e os pontos de variação entre os produtos.

É fundamental que se avalie a ALP durante o ciclo de vida da LPS devido à influência sobre as demais etapas (TAYLOR *et al.*, 2010). No entanto, esta é uma atividade complexa e para a qual há mais de uma solução factível. Tendo isso em vista, o projeto de ALP pode ser modelado como um problema de otimização multiobjetivo, porque envolve mais de um objetivo a ser otimizado e, em geral, tais objetivos são conflitantes entre si.

### **2.3 Engenharia de Software Baseada em Busca**

Uma das formas de se abordar o problema da otimização multiobjetivo de um projeto de ALP é utilizar a Engenharia de Software Baseada em Busca (do inglês *Search Based Software Engineering* – SBSE). A SBSE é um campo da Engenharia de Software que aplica técnicas computacionais baseadas em busca ao contexto dos problemas de Engenharia de Software, ou seja, os problemas de Engenharia de Software são abordados como problemas de busca (HARMAN *et al.*, 2009).

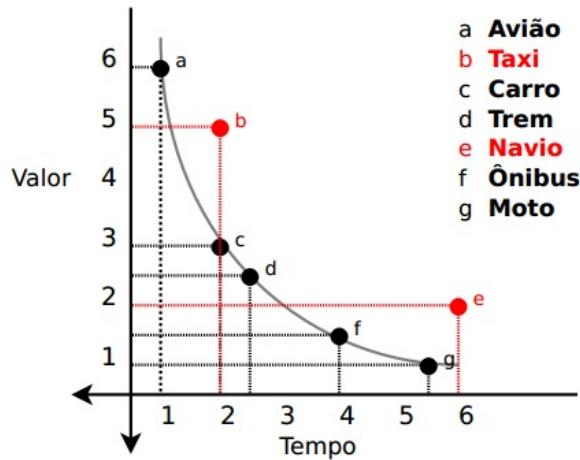
As técnicas de otimização baseadas em busca em geral estão associadas à geração das possíveis soluções para determinado problema e à uma função de avaliação da qualidade dessas soluções (HARMAN *et al.*, 2009). No entanto, quando se trata de otimização multiobjetivo há mais de um objetivo a ser otimizado, e os objetivos podem ser conflitantes. Consequentemente há mais de uma função de avaliação, e com isso pode haver mais de uma solução boa. Essa situação demanda um algoritmo multiobjetivo, que é capaz de utilizar mais de uma função de avaliação para realizar a otimização paralela de objetivos diferentes. Tendo isso em consideração, o ideal é que as soluções encontradas representem os diversos interesses e compromissos entre eles (COELLO *et al.*, 2007).

Na avaliação de aptidão das soluções encontradas por meio da SBSE, pode-se aplicar o conceito de dominância de Pareto (PARETO, 1927), que permite comparar as soluções com base em todos os objetivos do problema. Considerando um problema de minimização, o conceito de dominância de Pareto diz que uma solução A é dominada por uma solução B quando todos os valores de avaliação de A são maiores ou iguais aos valores de B, desde que haja pelo menos um valor de A maior que o correspondente em B. Assim é possível comparar as soluções considerando todos os objetivos desejados, de modo que as soluções não dominadas são as melhores para um determinado problema.

A Figura 2 ilustra o conceito de dominância de Pareto aplicado ao contexto de meios de transporte, onde os objetivos a serem minimizados são o tempo e o valor. De modo geral, os meios de transporte mais rápidos são mais caros, e os mais lentos são mais baratos. No

entanto, o táxi é uma solução dominada pelo carro, porque gasta exatamente o mesmo tempo, mas com um custo maior. Similarmente, o navio é dominado pelo ônibus e pela moto, pois é mais lento e também mais caro que ambos. A curva formada pelas soluções não dominadas (avião, carro, trem, ônibus e moto) forma uma aproximação à fronteira de Pareto.

Figura 2: Exemplo de problema multiobjetivo com duas métricas (COLANZI, 2014)



Uma das técnicas aplicadas na SBSE é a utilização de algoritmos genéticos. Tais algoritmos são ditos evolutivos, isto é, se baseiam na teoria da evolução, segundo a qual são priorizadas a sobrevivência e a reprodução dos indivíduos mais aptos de uma dada população. Os novos indivíduos, gerados pela reprodução entre indivíduos existentes, podem sofrer mutações, introduzindo novas características que podem melhorar ou piorar a aptidão dos mesmos (COELLO *et al.*, 2007).

Na próxima seção apresenta-se uma abordagem de SBSE para a otimização de projetos de ALP.

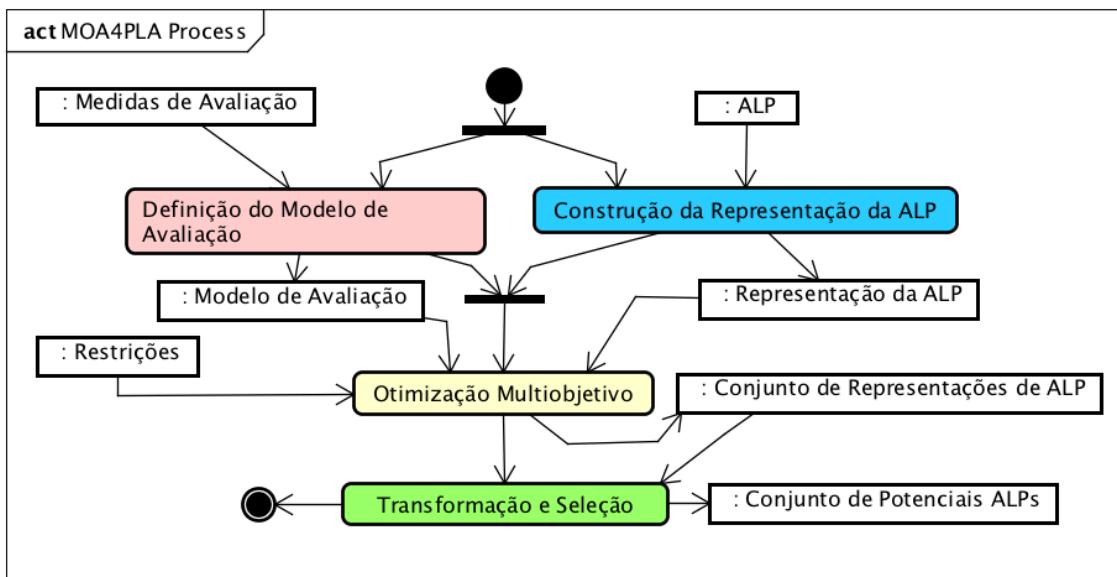
## 2.4 MOA4PLA

Colanzi (2014) propôs uma abordagem para automatizar a obtenção de melhores projetos de ALP, a qual utiliza algoritmos evolutivos multiobjetivos para avaliar e melhorar projetos de ALP. A abordagem, chamada *Multi-Objective Approach for Product-Line Architecture Design* (MOA4PLA), gera um conjunto de possíveis soluções a partir de uma solução inicial dada. Essas soluções apresentam o melhor balanceamento entre os objetivos avaliados, em termos de princípios básicos de projeto, modularização de características e extensibilidade da

LPS. A seguir, apresentam-se os principais conceitos da MOA4PLA, de acordo com a descrição apresentada por sua autora.

O processo executado pela MOA4PLA, ilustrado na Figura 3, possui quatro atividades principais: Construção da Representação da ALP; Definição do Modelo de Avaliação; Otimização Multiobjetivo; e Transformação e Seleção.

Figura 3: Processo da MOA4PLA (adaptada de COLANZI, 2014)



A Construção da Representação da ALP visa instanciar uma solução inicial a partir do projeto de ALP fornecido como entrada, para servir de ponto inicial para o processo evolutivo que é executado em etapa posterior.

A Definição do Modelo de Avaliação é a atividade em que são definidas as métricas que correspondem aos objetivos a serem otimizados de forma a atender as necessidades do projeto. O uso de diferentes tipos de métricas auxilia na análise do melhor balanceamento entre os objetivos, pois elas podem ser conflitantes.

A Otimização Multiobjetivo é a atividade em que o projeto de ALP dado como entrada na primeira atividade sofre as otimizações propriamente ditas, por meio da aplicação de operadores de busca, especificamente de operadores de cruzamento e de mutação. As soluções de ALP encontradas durante esta atividade são avaliadas de acordo com o modelo de avaliação definido na atividade anterior, e como saída é gerado um conjunto das melhores soluções encontradas.

A Transformação e Seleção é a última atividade da MOA4PLA. Esta atividade envolve receber as soluções geradas como saída da atividade anterior e convertê-las em um

formato legível para o usuário. Após isso, o usuário deve selecionar qual das arquiteturas geradas será adotada para a LPS, de acordo com a priorização de certas métricas ou com o balanceamento entre os objetivos considerados.

Em qualquer algoritmo de busca que se utilize, é necessário representar adequadamente o projeto de ALP, devido ao impacto que isso causa na implementação de todos os estágios do algoritmo. A representação usada pela MOA4PLA consiste de elementos arquiteturais que podem ser componentes, interfaces, operações de interfaces e relacionamentos entre esses elementos. Os elementos arquiteturais são relacionados às características por meio de estereótipos, e podem ser obrigatórios – comuns a toda a LPS – ou variáveis – específicos de alguns produtos. Os elementos variáveis estão relacionados aos pontos de variação e suas respectivas variantes.

Na atividade da Otimização Multiobjetivo são aplicados operadores de busca ao projeto de ALP. A MOA4PLA utiliza cinco operadores de busca convencionais e um operador de busca dirigido a características. Os operadores de busca convencionais utilizados são: *Move Method* (move um método de uma classe para outra), *Move Attribute* (move um atributo de uma classe para outra), *Add Class* (move um atributo ou método de uma classe para uma nova classe), *Move Operation* (move uma operação de uma interface para outra) e *Add Component* (move uma operação de uma interface de um componente para uma nova interface de um novo componente). O operador de busca dirigido a características é chamado de *Feature-Driven Operator*, e tenta modularizar uma característica que está entrelaçada com outras, em um componente aleatório.

O modelo de avaliação original da MOA4PLA era composto por quatro funções de avaliação: CM(pla), FM(pla), Extt(pla) e Eleg(pla) (COLANZI, 2014). Em trabalho posterior, Verdecia *et al.* (2016) refinaram e estenderam este modelo, de modo a torná-lo mais completo e sensível às mudanças realizadas no projeto de ALP durante o processo de busca. Sendo assim, o novo modelo de avaliação possui 17 funções de avaliação, descritas brevemente a seguir.

- **FM(pla):** função que avalia o grau de modularização de uma ALP em termos de suas características.
- **CM(pla):** função que avalia princípios básicos de projeto, como coesão, acoplamento e tamanho.
- **DC(pla):** função que mede a difusão de características, através da quantidade de elementos arquiteturais associados com as características da LPS.

- **EC(pla):** função que mede a interação da característica, através da quantidade de características com as quais a característica avaliada compartilha ao menos um elemento arquitetural.
- **LCC(pla):** função que mede a falta de coesão baseada em características, através da quantidade de características avaliadas por cada componente da ALP.
- **ACOMP(pla):** função que mede o acoplamento de componentes da ALP através da soma das dependências de entrada e de saída de cada componente.
- **ACLASS(pla):** função que soma a quantidade de elementos arquiteturais que dependem de cada classe do projeto à quantidade de elementos que cada classe do projeto depende.
- **COE(pla):** função que mede a coesão da ALP, através do número de relacionamentos internos entre classes e interfaces de um componente.
- **CS(pla):** função que mede o tamanho de um componente em termos de suas operações que são exigidas por outros serviços ou componentes.
- **SD(pla):** função que mede a similaridade de um projeto de ALP.
- **CV(pla):** função que mede o acoplamento de variabilidades considerando as dependências entre pontos de variabilidade da ALP.
- **Ext(pla):** função que avalia a extensibilidade dos pacotes de uma ALP.
- **TAM(pla):** função que mede a quantidade média de operações por interface da ALP.
- **RCC(pla):** função que mede o acoplamento de componentes através da quantidade de relacionamentos entre interfaces da ALP.
- **Eleg(pla):** função que mede a elegância de um projeto de ALP orientado a objetos.
- **SV(pla):** função que mede a variabilidade de estruturas da ALP.
- **TV(pla):** função que mede a variabilidade total de um projeto de ALP.

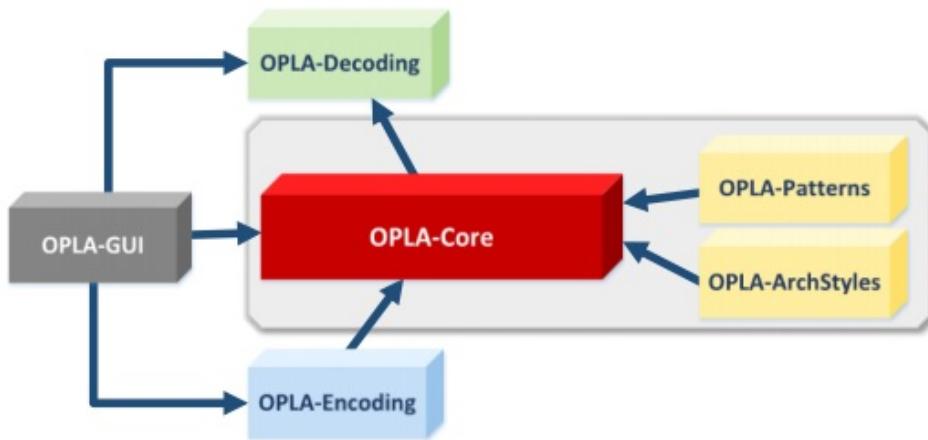
Todas as 17 funções de avaliação foram projetadas para serem minimizadas durante o processo de busca, o que significa que os algoritmos empregados devem buscar a minimização de cada função usada como objetivo. Como é inviável a otimização simultânea de todas as funções, cabe ao arquiteto escolher um subconjunto de funções a serem otimizadas de acordo com suas necessidades (VERDECIA *et al.*, 2016).

A próxima seção apresenta uma implementação da abordagem MOA4PLA.

## 2.5 OPLA-Tool

A ferramenta OPLA-Tool (*Optimization for PLA Tool*) foi proposta por Colanzi (2014) e desenvolvida por Féderle (2014) para implementar a abordagem MOA4PLA. A Figura 4 ilustra os módulos da ferramenta e suas interdependências.

Figura 4: Módulos da OPLA-Tool (COLANZI, 2014)



O módulo OPLA-GUI cuida da interface com o usuário, permitindo escolher o algoritmo utilizado na otimização, as funções objetivo, os operadores de busca, a ALP a ser utilizada como entrada e apresenta os resultados após o processamento. O módulo OPLA-Encoding faz a codificação da entrada, em formato XMI, para uma representação no formato utilizado pelo *core* da ferramenta. O módulo OPLA-Core realiza o processo evolutivo da ALP, de acordo com o algoritmo e os parâmetros informados. O módulo OPLA-Decoding decodifica as soluções geradas como saída, da representação interna usada pelo *core* para o formato XMI, de modo que sejam legíveis para o usuário. Os módulos OPLA-Patterns e OPLA-Arch-Styles contêm operadores de busca utilizados durante a otimização, que aplicam padrões de projetos e estilos arquiteturais, respectivamente.

O módulo OPLA-Core se baseia no jMetal (DURILLO; NEBRO, 2011), um *framework* voltado para otimização multiobjetivo que implementa diversos algoritmos evolutivos, como o algoritmo genético gGA e os MOEAs (*Multi-Objective Evolutionary Algorithms*) NSGA-II (Deb et al., 2002) e PAES (KNOWLES; CORNE, 2000).

## 2.6 Considerações finais

A abordagem MOA4PLA gera uma série de soluções de projeto de ALP a partir de uma solução inicial dada (o projeto original da ALP), usando algoritmos evolutivos. No entanto possui o inconveniente de não verificar a existência de anomalias arquiteturais nas soluções geradas.

O próximo capítulo apresenta o conceito de anomalias arquiteturais, bem como um catálogo de anomalias arquiteturais levantadas na literatura.

### 3 Anomalias arquiteturais

Anomalias arquiteturais – ou problemas arquiteturais (do inglês *Architectural Smells* ou *Architectural Bad Smells* (GARCIA, 2014), ou ainda *Bad Smells* (FOWLER, 1999)) – são problemas decorrentes de decisões de projeto de arquitetura, intencionais ou não, que afetam negativamente a qualidade do sistema (GARCIA, 2014).

Entre os aspectos que são influenciados negativamente com a ocorrência das anomalias arquiteturais estão a facilidade de entendimento do sistema, a testabilidade, a extensibilidade, o reuso e a manutenibilidade. Tais aspectos normalmente podem ser verificados em detalhes de implementação, como classes, atributos, métodos e pacotes (FOWLER, 1999).

Johansson e Oizumi (2016) fizeram um levantamento de uma série de anomalias arquiteturais conhecidas, descritas por Garcia (2014), Fowler (1999), Kerievsky (2004) e Parnas (1974). Esse levantamento foi utilizado como base para este trabalho, onde as anomalias arquiteturais foram classificadas entre as que foram e as que não foram analisadas. O critério de classificação adotado foi a demanda de consulta ao código-fonte. Anomalias que, por alguma razão, demandam consulta a código-fonte para serem identificadas, não foram analisadas neste trabalho.

A seguir são apresentados os conceitos das 25 anomalias arquiteturais levantadas, conforme o catálogo da Tabela 1.

#### 3.1 Link Overload

*Link Overload* é uma anomalia arquitetural onde um componente possui excessiva dependência de outros componentes, prejudicando assim a separação das funcionalidades e o isolamento das mudanças. Essas dependências podem se manifestar como *links* de entrada, de saída, ou ambos (GARCIA, 2014).

A definição do limite aceitável (*threshold*) de *links* proposto por Garcia (2014) é baseada na média somada a um desvio padrão, e é feita separadamente para cada direcionalidade (entrada, saída ou ambos), conforme detalhado no algoritmo da Figura 5.

No algoritmo de detecção de *Link Overload*, as linhas 1 a 3 inicializam variáveis. Nas linhas 4 a 6 é feita a contagem de dependências (*links*) presentes em cada componente, para cada direcionalidade (entrada, saída ou ambos). Da linha 7 a 10 é realizado o cálculo do

*threshold*. Por fim, nas linhas de 11 a 14 são identificados os componentes cuja quantidade de dependências ultrapassa o *threshold* calculado.

Tabela 1: Catálogo de anomalias arquiteturais consideradas para este trabalho (adaptada de JOHANSSON; OIZUMI, 2016)

Classificação	Anomalia arquitetural
Anomalias analisadas	Link Overload (GARCIA, 2014) Concern Overload (GARCIA, 2014) Unused Interface (GARCIA, 2014) Unused Brick (GARCIA, 2014) Duplicate Component Functionality (GARCIA, 2014) Sloppy Delegation (GARCIA, 2014) Dependency Cycle (GARCIA, 2014) Connector Envy (GARCIA, 2014) Scattered Parasitic Functionality (GARCIA, 2014) Large Class (FOWLER, 1999) Primitive Obsession (FOWLER, 1999)
Anomalias não analisadas	Lazy Class (FOWLER, 1999) Long Method (FOWLER, 1999) Alternative Classes with Different Interfaces (FOWLER, 1999) Connector Chain Overload (GARCIA, 2014) Brick Functionality Overload (GARCIA, 2014) Ambiguous Interfaces (GARCIA, 2014) Extraneous Adjacent Connector (GARCIA, 2014) Duplicated Code (KERIEVSKY, 2004) Oddball Solution (KERIEVSKY, 2004) Switch Statements (KERIEVSKY, 2004) Conditional Complexity (KERIEVSKY, 2004) Combinatorial Explosion (KERIEVSKY, 2004) Unplanned and Uncontrolled Solution (KERIEVSKY, 2004) Indecent Exposure (PARNAS, 1974)

### 3.2 Concern Overload

Esta anomalia é verificada quando um componente possui um número excessivo de interesses (do inglês *concerns*) (GARCIA, 2014), quebrando o princípio da responsabilidade única.

De forma semelhante ao algoritmo de detecção de *Link Overload*, o algoritmo de detecção de *Concern Overload* se baseia na média somada ao desvio padrão, conforme detalhado no algoritmo da Figura 6.

No algoritmo de detecção de *Concern Overload*, as linhas 1 e 2 inicializam variáveis. Nas linhas 3 a 7 é feita a contagem de interesses (*concerns*) presentes em cada componente. Da linha 8 a 10 é realizado o cálculo do *threshold*. Por fim, nas linhas de 11 a 13 são identificados os componentes cuja quantidade de interesses ultrapassa o *threshold* calculado.

Figura 5: Algoritmo de detecção de *Link Overload* (GARCIA, 2014)

---

**Algorithm 3: detectLO**


---

**Input:**  $B$ : a set of bricks,  $L$ : links between brinks  
**Output:**  $smells$  : a set of Link Overload instances

- 1  $smells \leftarrow \emptyset$
- 2  $numLinks \leftarrow$  initialize map as empty
- 3  $directionality \leftarrow \{"in", "out", "both"\}$
- 4 **for**  $b \in B$  **do**
  - 5   **for**  $d \in directionality$  **do**
    - 6      $numLinks[(b, d)] \leftarrow numLinks[(b, d)] + getNumLinks(b, d, L)$
  - 7   **for**  $d \in directionality$  **do**
    - 8      $meanLinks[d] \leftarrow computeMean(numLinks, d, B)$
    - 9      $stdDevLinks[d] \leftarrow computeStandardDeviation(numLinks, d, B)$
    - 10     $th_{lo}[d] \leftarrow meanLinks[d] + stdDevLinks[d]$
  - 11   **for**  $b \in B$  **do**
    - 12     **for**  $d \in directionality$  **do**
      - 13       **if**  $getNumLinks(b, d, L) > th_{lo}[d]$  **then**
        - 14          $smells \leftarrow smells \cup \{(b, d)\}$

---

Figura 6: Algoritmo de detecção de *Concern Overload* (GARCIA, 2014)

---

**Algorithm 2: detectCO**


---

**Input:**  $B$ : a set of bricks,  $T$ : a set of system concerns,  $th_{zb}$ : threshold for concern relevance  
**Output:**  $smells$  : a set of Concern Overload instances

- 1  $smells \leftarrow \emptyset$
- 2  $brickConcernCounts \leftarrow$  initialize all brick concern counts to 0
- 3 **for**  $b \in B$  **do**
  - 4    $T_b \leftarrow getConcernsOfBrick(b)$
  - 5   **for**  $z \in T_b$  **do**
    - 6     **if**  $P(z | b) > th_{zb}$  **then**
      - 7        $brickConcernCounts[b] = brickConcernCounts[b] + 1$
  - 8    $meanBrickConcerns \leftarrow computeMean(brickConcernCounts)$
  - 9    $stdDevOfBrickConcerns \leftarrow computeStandardDeviation(brickConcernCounts)$
  - 10    $th_t \leftarrow meanBrickConcerns + stdDevOfBrickConcerns$
  - 11   **for**  $b \in B$  **do**
    - 12     **if**  $brickConcernCounts[b] > th_t$  **then**
      - 13        $smells \leftarrow smells \cup \{b\}$

---

### **3.3 Unused Interface**

Uma interface de componente é dita não utilizada quando não está ligada a nenhum componente, adicionando, portanto, uma complexidade desnecessária ao sistema e dificultando sua manutenção (GARCIA, 2014). Normalmente isto é causado por alterações de escopo ou refatorações mal feitas.

### **3.4 Unused Brick**

A anomalia arquitetural *Unused Brick* ocorre quando todas as interfaces de um componente apresentam o problema *Unused Interface*, ou seja, quando nenhuma de suas interfaces possui ligação a outro componente. Isto adiciona uma complexidade desnecessária e dificulta a manutenção do sistema de software (GARCIA, 2014).

### **3.5 Duplicate Component Functionality**

Anomalia arquitetural encontrada quando um componente duplica as funcionalidades de outro existente no sistema. A duplicação de funcionalidade aumenta a complexidade do sistema, pois ao se efetuar mudanças em uma instância da funcionalidade é necessário replicar a mudança na outra instância. Deixar de fazer isso pode gerar erros no sistema (GARCIA, 2014). Além disso, em geral uma duplicação de funcionalidade de componente envolve também duplicação de código.

### **3.6 Sloppy Delegation**

*Sloppy Delegation* ocorre quando um componente delega uma quantidade muito pequena de funcionalidades a outros componentes, que ele próprio poderia realizar. A autossuficiência na realização de uma funcionalidade pode ser verificada quando todos os dados necessários a ela fazem parte do estado desse componente (GARCIA, 2014).

Um exemplo dessa anomalia ocorre quando um componente que armazena dados tais como a velocidade atual de uma aeronave, nível de combustível e altitude passa esses dados para outro componente que apenas calcula o consumo de combustível da aeronave (GARCIA,

2014). Nesse caso, o cálculo de consumo poderia ser realizado pelo próprio componente que contém os dados necessários.

### 3.7 Dependency Cycle

*Dependency Cycle* representa o problema de dependência cíclica, em que um conjunto de componentes se liga de forma que um dependa do outro, formando uma cadeia circular, e prejudicando a manutenção dos mesmos. Desse modo, mudanças em um desses componentes possivelmente afetam todos os outros componentes do ciclo (GARCIA, 2014).

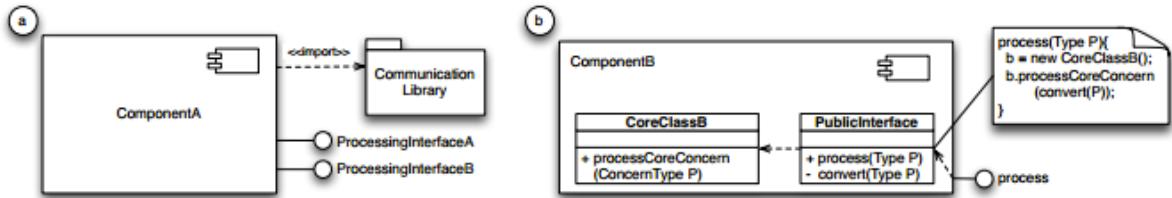
Esta anomalia pode ser detectada ao identificar componentes fortemente conectados. Um componente fortemente conectado pode ser representado como um grafo onde cada vértice é alcançável a partir de qualquer outro. Desse modo, qualquer algoritmo que detecte componentes fortemente conectados pode ser usado para identificar ciclos de dependência (GARCIA, 2014).

### 3.8 Connector Envy

*Connector Envy* ocorre quando um componente apresenta uma funcionalidade com elevado nível de interação, que deveria ser delegada a um conector, prejudicando assim o entendimento da funcionalidade e o reuso. Os conectores oferecem serviços de comunicação, coordenação, conversão e facilitação. A comunicação se refere à transferência de dados entre elementos arquiteturais (exemplo: mensagens e retornos computacionais). A coordenação cuida da transferência de controle entre elementos arquiteturais (exemplo: execução de *threads*). A conversão se responsabiliza pela tradução de diferentes serviços de interação entre os elementos arquiteturais (exemplo: tradução de formatos de dados, de tipos ou de protocolos). A facilitação envolve a mediação, otimização e simplificação da interação (exemplo: balanceamento de carga, monitoramento e tolerância a erros). Os componentes que utilizam largamente a funcionalidade de pelo menos uma dessas categorias sofrem do problema *Connector Envy* (GARCIA, 2014).

A Figura 7 apresenta dois exemplos de ocorrência de *Connector Envy*. Em (a) o problema envolve a implementação de serviços de comunicação e facilitação dentro do *ComponentA*. Em (b) o problema envolve um serviço de conversão.

Figura 7: Exemplos de ocorrência da anomalia *Connector Envy* (GARCIA, 2014)

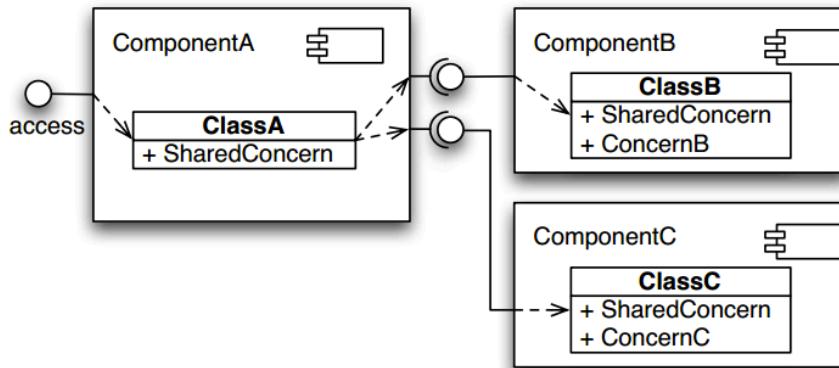


### 3.9 Scattered Parasitic Functionality

Este problema ocorre quando vários componentes são responsáveis por realizar o mesmo interesse de alto nível e, além disso, alguns desses componentes são responsáveis por interesses transversais. Isso viola o princípio da separação de interesses de duas maneiras. Em primeiro lugar, ao dispersar um único interesse em diversos componentes. Em segundo lugar, pelo menos um componente aborda interesses transversais, além de seus próprios interesses. Ou seja, o interesse disperso “contamina” o componente com um interesse transversal, como se fosse um parasita (GARCIA, 2014).

A Figura 8 ilustra um exemplo de ocorrência do problema *Scattered Parasitic Functionality*, onde o interesse de alto nível *SharedConcern* está disperso pelos três componentes, ao mesmo tempo em que o ComponentA e o ComponentB são também responsáveis por interesses próprios.

Figura 8: Exemplo de ocorrência da anomalia *Scattered Parasitic Functionality* (GARCIA, 2014)



### **3.10 Large Class**

A anomalia *Large Class* é verificada quando uma classe possui quantidade excessiva de atributos ou métodos, indicando normalmente que a mesma possui um excesso de responsabilidades. A resolução desse problema se dá pela extração de classes e subclasses para dividir as responsabilidades da classe atual (FOWLER, 1999).

É comum que uma classe com esse tipo de anomalia apresente redundância de código, por exemplo, com métodos que possuem código em comum. Nesse caso, uma boa solução é a extração de métodos, em que novos métodos são criados para encapsular o código anteriormente redundante (FOWLER, 1999).

### **3.11 Primitive Obsession**

Esta anomalia ocorre quando um código manipula uma quantidade excessiva de variáveis de tipos primitivos, por exemplo: booleanos, strings, números e datas, que são tipos primitivos em muitas linguagens. A ocorrência de *Primitive Obsession* indica um mau uso da orientação a objetos (FOWLER, 1999).

Há várias formas de se resolver esse problema, de acordo com o contexto em que ocorre, mas de forma geral a solução envolve o uso de classes e objetos. No caso de grupo de valores que deveriam estar juntos, a extração de classe é a solução recomendada. Quando um método recebe uma lista de parâmetros de tipos primitivos, pode-se substituí-la por um parâmetro que seja um objeto (FOWLER, 1999).

### **3.12 Lazy Class**

*Lazy Class* trata-se de uma anomalia em que uma classe não faz o suficiente para justificar sua existência, devendo, portanto, ser eliminada. Costuma ocorrer quando uma classe tinha funcionalidade que justificava sua existência, mas foi reduzida após refatoração no sistema, e com isso deixou de ser necessária. Outra situação comum para a ocorrência desse problema é quando classes são criadas por conta de modificações que foram planejadas, mas ainda não implementadas (FOWLER, 1999).

Este tipo de anomalia arquitetural não foi analisado por ser inviável, sem acesso ao código-fonte, avaliar se uma classe possui funcionalidade suficiente para justificar sua existência.

### **3.13 Long Method**

Indica o problema de métodos muitos longos nas classes, que possivelmente compartilham parte da lógica, gerando duplicação de código e dificultando seu entendimento. A solução normalmente envolve a extração de métodos, onde um método longo é dividido em diversos métodos menores (FOWLER, 1999).

Não foi possível avaliar a ocorrência deste tipo de anomalia arquitetural por exigir acesso ao código fonte, de modo a identificar a existência de métodos muito extensos.

### **3.14 Alternative Classes with Different Interfaces**

*Alternative Classes with Different Interfaces* é uma anomalia que ocorre quando duas classes muito parecidas possuem interfaces diferentes. Uma das soluções é renomear métodos que possuem assinaturas diferentes, mas que realizam a mesma coisa. Outra solução é mover métodos para as classes até que os protocolos sejam idênticos. Se for necessário mover código de forma redundante para conseguir isso, é possível extrair uma superclasse (FOWLER, 1999).

Esta anomalia não foi analisada por ser inviável, sem acesso ao código-fonte, verificar a semelhança entre classes com interfaces diferentes.

### **3.15 Connector Chain Overload**

*Connector Chain Overload* ocorre quando um longo conjunto de conectores ligados entre si possui um número excessivo de tipos de conectores. Isso pode resultar no uso simultâneo de tipos de conectores incompatíveis, e no fornecimento de serviços de interação excessivamente complexos, que podem ser substituídos por um mecanismo mais simples. Por exemplo, um conjunto de conectores ligados que oferece serviços de autenticação, autorização, criptografia, transmissão, acesso de dados e distribuição é uma instância do problema *Connector Chain Overload* (GARCIA, 2014).

Este tipo de anomalia não foi analisado por ser inviável, sem acesso ao código-fonte, encontrar locais onde haja cadeias de conectores ligados entre si.

### 3.16 Brick Functionality Overload

*Brick Functionality Overload* ocorre quando um componente realiza uma quantidade excessiva de funcionalidades. Isso indica uma modularização inapropriada em um sistema de software, que viola os princípios da separação de interesses e do isolamento de mudanças. O *threshold* que indica o número excessivo de operações é definido pelo arquiteto de software (GARCIA, 2014).

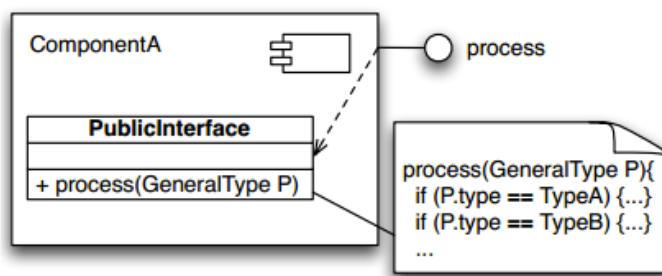
Esta anomalia arquitetural não foi avaliada neste trabalho por exigir acesso ao código fonte, de modo a encontrar um grande número de funcionalidades dentro de um mesmo componente. Isto não pode ser identificado apenas pela quantidade de métodos presentes, já que envolve a maneira como os métodos são definidos e seu comportamento.

### 3.17 Ambiguous Interfaces

Interfaces ambíguas são interfaces que oferecem um único ponto de entrada geral para um componente ou conector, contendo um único parâmetro, e disparam diferentes tipos de operações internas com base no conteúdo deste parâmetro. Costumam ocorrer em sistemas baseados em eventos onde as interações não estão modeladas explicitamente e vários componentes trocam mensagens através de um evento compartilhado.

A Figura 9 mostra um exemplo deste tipo de anomalia arquitetural, em que o *ComponentA* possui uma interface ambígua chamada *PublicInterface*, que disponibiliza um método que recebe como parâmetro um tipo genérico *GeneralType*. Internamente, o código do método verifica o tipo específico do parâmetro recebido, para então disparar ações distintas correspondentes a este tipo.

Figura 9: Exemplo de ocorrência da anomalia *Ambiguous Interfaces* (GARCIA, 2014)

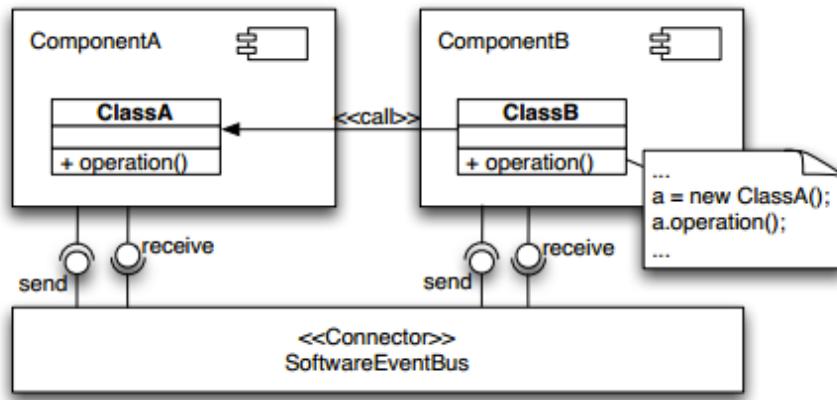


Este tipo de anomalia arquitetural não foi avaliado neste trabalho por exigir acesso ao código fonte, de forma que sejam identificadas as operações distintas que são lançadas por uma interface ambígua, com base no tipo do parâmetro recebido.

### 3.18 Extraneous Adjacent Connector

Indica o problema em que dois conectores de tipos diferentes são usados para ligar um par de componentes (GARCIA, 2014). Este problema pode ocorrer com quaisquer tipos de conectores. A Figura 10 ilustra um exemplo em que o *ComponentA* e o *ComponentB* se comunicam através de dois tipos distintos de conectores, onde um deles se baseia em eventos e o outro em invocação direta de método do destinatário.

Figura 10: Exemplo de ocorrência da anomalia *Extraneous Adjacent Connector* (GARCIA, 2014)



Este tipo de anomalia arquitetural não foi avaliado neste trabalho, pois seria preciso verificar no código fonte os tipos de conectores existentes, e a forma de interação que eles usam para se comunicar com os componentes.

### 3.19 Duplicated Code

*Duplicated Code* é um problema frequente, em que há código duplicado explicitamente – código idêntico – ou implicitamente – através da repetição de estruturas ou processamentos aparentemente diferentes, mas idênticos em sua essência (KERIEVSKY, 2004).

A Figura 11 mostra um exemplo de código duplicado nos construtores de uma classe denominada *Loan*, onde há repetição de instanciação de diversos atributos em cada um dos três construtores. Para este caso, uma refatoração simples poderia ser fazer com que os dois

primeiros construtores invocassem o terceiro, aproveitando assim a instanciação da maioria dos atributos, conforme ilustrado na Figura 12.

Este tipo de anomalia arquitetural não foi avaliado neste trabalho por exigir acesso ao código fonte, de forma que sejam identificados os trechos de código duplicado.

Figura 11: Exemplo de ocorrência da anomalia *Duplicated Code* (KERIEVSKY, 2004)

```
public class Loan {
    ...
    public Loan(float notional, float outstanding, int rating, Date expiry) {
        this.strategy = new TermROC();
        this.notional = notional;
        this.outstanding = outstanding;
        this.rating = rating;
        this.expiry = expiry;
    }
    public Loan(float notional, float outstanding, int rating, Date expiry, Date maturity) {
        this.strategy = new RevolvingTermROC();
        this.notional = notional;
        this.outstanding = outstanding;
        this.rating = rating;
        this.expiry = expiry;
        this.maturity = maturity;
    }
    public Loan(CapitalStrategy strategy, float notional, float outstanding,
               int rating, Date expiry, Date maturity) {
        this.strategy = strategy;
        this.notional = notional;
        this.outstanding = outstanding;
        this.rating = rating;
        this.expiry = expiry;
        this.maturity = maturity;
    }
}
```

Figura 12: Exemplo de solução para o *Duplicated Code* indicado na Figura 11 (KERIEVSKY, 2004)

```
public class Loan {
    ...
    public Loan(float notional, float outstanding, int rating, Date expiry) {
        this(new TermROC(), notional, outstanding, rating, expiry, null);
    }
    public Loan(float notional, float outstanding, int rating, Date expiry, Date maturity) {
        this(new RevolvingTermROC(), notional, outstanding, rating, expiry, maturity);
    }
    public Loan(CapitalStrategy strategy, float notional, float outstanding,
               int rating, Date expiry, Date maturity) {
        this.strategy = strategy;
        this.notional = notional;
        this.outstanding = outstanding;
        this.rating = rating;
        this.expiry = expiry;
        this.maturity = maturity;
    }
}
```

### **3.20 Oddball Solution**

O problema *Oddball Solution* ocorre quando um mesmo problema é solucionado de maneiras diferentes ao longo do mesmo sistema, havendo assim uma solução não usual ou inconsistente. A ocorrência desse problema indica duplicação de código (KERIEVSKY, 2004).

Esta anomalia arquitetural não foi avaliada neste trabalho por exigir acesso ao código fonte das arquiteturas analisadas, de modo a identificar as soluções diferentes para o mesmo problema. Como o código fonte não está acessível, por se tratar de arquiteturas geradas pela OPLA-Tool, não é possível avaliar a ocorrência desse tipo de anomalia.

### **3.21 Switch Statements**

Representa a ocorrência de longas cadeias de instruções condicionais, normalmente dentro de métodos de classes, para servir de validação. Normalmente não são nocivas, mas podem se tornar caso sejam muito rígidas ou complexas, dificultando sua manutenção e inteligibilidade (KERIEVSKY, 2004).

Este tipo de anomalia arquitetural não foi avaliado neste trabalho por exigir acesso ao código fonte, de modo a identificar as cadeias de instruções condicionais.

### **3.22 Conditional Complexity**

*Conditional Complexity* indica a complexidade de estruturas condicionais presentes no sistema. Tais estruturas costumam ser simples e fáceis de entender inicialmente, mas se tornam extensas e complexas com o tempo, conforme são feitas modificações no sistema (KERIEVSKY, 2004).

Esta anomalia arquitetural não foi avaliada neste trabalho por exigir acesso ao código fonte, de forma a encontrar tais estruturas condicionais.

### **3.23 Combinatorial Explosion**

*Combinatorial Explosion* envolve uma forma particular de duplicação de código em que diversos trechos de código realizam o mesmo tipo de ação, porém com diferentes tipos ou quantidades de dados. Pode ser exemplificada com uma situação onde determinada classe

possui muitos métodos para realização de consultas a banco de dados, em que cada método possui uma quantidade de condições e dados muito específicos. Isso gera uma explosão combinatória na quantidade de métodos necessários para atender todas as circunstâncias desejadas (KERIEVSKY, 2004).

Esta anomalia arquitetural não foi possível de ser avaliada neste trabalho por exigir acesso ao código fonte, de modo a identificar os métodos que realizam o mesmo tipo de ação, mas com dados e/ou condições diferentes.

### **3.24 Unplanned and Uncontrolled Solution**

*Unplanned and Uncontrolled Solution* ocorre quando uma mudança é realizada rapidamente no sistema, sem gastar-se o devido tempo tentando simplificar a funcionalidade ou melhor acomodá-la. Isso faz com que o código e os dados que implementam essa funcionalidade se tornem espalhados entre diversas classes, e implica em gerar muitas importações e dependências de atributos de outros componentes (KERIEVSKY, 2004).

Este tipo de anomalia arquitetural não foi analisado por exigir acesso ao código-fonte, de forma a encontrar código e dados relacionados a determinada funcionalidade espalhados entre diversas classes.

### **3.25 Indecent Exposure**

*Indecent Exposure* denota o problema da exposição pública de métodos e classes que não deveriam estar visíveis ao usuário, violando o encapsulamento. Assim, os clientes possuem conhecimento de código que não lhes é importante, ou que é apenas indiretamente importante, aumentando a complexidade do projeto (PARNAS, 1974).

Esta anomalia arquitetural não foi avaliada neste trabalho, porque seria necessário analisar o código fonte para decidir que métodos e classes são relevantes para se expor aos usuários.

### **3.26 Considerações finais**

Neste capítulo foram apresentados o conceito de anomalia arquitetural e a descrição de 25 tipos de anomalias arquiteturais obtidos na literatura, dos quais 11 foram selecionados para serem analisados neste trabalho, de modo a identificar sua ocorrência em projetos de ALP.

O próximo capítulo apresenta a metodologia do trabalho, descrevendo as LPSS analisadas e os *thresholds* utilizados.

## 4 Metodologia do trabalho

Os materiais utilizados no desenvolvimento do trabalho foram: um computador, um processador de texto e soluções de projeto de ALP geradas pela OPLA-Tool, obtidas de trabalhos anteriores (CHOMA NETO, 2017; LAVEZZO, 2017; DIAS, 2016). Os métodos utilizados foram a revisão de literatura sobre anomalias arquiteturais; análise das ALPs obtidas, visando encontrar as anomalias estudadas; e proposição de diretrizes para evitar a ocorrência dessas anomalias.

Visando alcançar o objetivo proposto, neste trabalho foram analisadas diversas instâncias de ALPs referentes às LPSs *Arcade Game Maker* (SEI, 2016), *Banking* (GOMAA, 2011) e *Mobile Media* (YOUNG, 2005), em busca de anomalias arquiteturais. Cada instância é um exemplar de ALP, seja o original proposto pelos respectivos autores ou um projeto alternativo gerado pela OPLA-Tool a partir do original. As figuras das instâncias analisadas encontram-se no Anexo A.

Este capítulo está dividido em seções conforme segue. A Seção 4.1 descreve brevemente as LPSs analisadas. A Seção 4.2 apresenta os *thresholds* utilizados na análise. Por fim, a Seção 4.3 apresenta as considerações finais do capítulo.

### 4.1 LPSs analisadas

A *Arcade Game Maker* (SEI, 2016), também conhecida pela sigla AGM, é uma LPS que abrange os seguintes jogos de *arcade*: *Bowling*, *Brickles* e *Pong*. Seu modelo de características prevê as seguintes características:

- Play;
- Persistence;
- Exception Handling;
- Logging;
- Action;
- Rule;
- Configuration;
- Service;
- Save;
- Bowling;

- Pong;
- Brickles;
- Movement;
- Collision;
- Pause;
- Ranking.

A *Banking* (GOMAA, 2011), também abreviada como Bank, é uma LPS para um sistema bancário que consiste de um servidor e vários clientes caixas eletrônicos. Seu modelo de características prevê as seguintes características:

- user\_interaction;
- io;
- max\_pin\_attempts;
- output;
- external\_putput\_device;
- state\_dependent\_control;
- statement;
- maintenance;
- deposit;
- external\_io\_device;
- greeting;
- external\_user;
- german;
- spanish;
- french;
- english.

A *Mobile Media* (YOUNG, 2005), também conhecida pela sigla MM, é uma LPS para o gerenciamento (criação, exclusão, visualização, execução e envio) de diversos tipos de mídias em dispositivos móveis, como música, fotos e vídeos. Seu modelo de características prevê as seguintes características:

- AlbumManagement;
- CreateDelete;
- ViewPlayMedia;
- SMSTransfer;

- LabelMedia;
- LinkMedia;
- Favourites;
- MM;
- Video;
- Music;
- Photo;
- CopyMedia;
- Media;
- GUI.

O porte de cada LPS pode ser verificado através da quantidade de elementos arquiteturais em sua ALP, conforme detalhado na Tabela 2. Nessa tabela, componentes são tratados como sinônimo de pacotes.

Tabela 2: Quantidade de elementos arquiteturais nas versões originais da AGM, Banking e MM  
(adaptada de SANTOS *et al.*, 2017)

ALP	Componentes	Interfaces	Classes	Características obrigatórias	Características variáveis
AGM	9	14	30	6	5
Banking	4	5	25	13	3
MM	8	15	14	7	7

A análise de anomalias arquiteturais foi realizada em 24 instâncias de ALPs obtidas em trabalhos anteriores. Essas instâncias foram divididos em conjuntos, em dois níveis: em um primeiro nível separou-se as instâncias de acordo com o trabalho de origem das mesmas, formando três grupos: o Grupo 1 (instâncias obtidas do trabalho de Choma Neto (2017)), o Grupo 2 (instâncias obtidas do trabalho de Lavezzo (2017)) e o Grupo 3 (instâncias obtidas do trabalho de Dias (2016)); em um segundo nível separou-se por LPS, formando os grupos AGM, Banking e MM, todos contendo a ALP original e algumas geradas pela ferramenta. A Tabela 3 apresenta a quantidade de instâncias de cada LPS presente em cada grupo.

A análise de ocorrência das anomalias arquiteturais se deu por meio de inspeção visual em cada instância de ALP, de forma a identificar as anomalias descritas no Capítulo 3. Na próxima seção são apresentados os *thresholds* utilizados nessa análise.

Tabela 3: Quantidade de instâncias de cada LPS presente em cada grupo

LPS	Grupo 1	Grupo 2	Grupo 3	Total
AGM	4	-	3	7
Banking	3	2	-	5
MM	4	4	4	12
<b>Total</b>	<b>11</b>	<b>6</b>	<b>7</b>	<b>24</b>

## 4.2 Thresholds utilizados

Cada anomalia arquitetural avaliada neste trabalho precisou ter um *threshold* definido para identificar sua ocorrência. No entanto, em alguns casos não foi possível estabelecer um critério exato para a detecção, demandando, portanto, uma avaliação interpretativa. Abaixo seguem os *thresholds* utilizados para cada anomalia analisada deste trabalho.

- *Link Overload*: seguiu-se a fórmula indicada por Garcia (2014), descrita na Seção 3.1, com o *threshold* calculado para cada direcionalidade (considerou-se bidirecionais os *links* sem direção definida e fez-se a contagem em cada classe e interface). Devido à formula resultar em valores fracionários, os valores foram arredondados para cima. A Tabela 4 apresenta os valores obtidos para cada LPS.
- *Concern Overload*: seguiu-se a fórmula indicada por Garcia (2014), descrita na Seção 3.2, e tomou-se como sinônimo de *concern* as características expostas na Seção 4.1. Foram contadas as características associadas ao nome da classe ou interface. Devido à formula resultar em valores fracionários, os valores foram arredondados para cima. A Tabela 4 apresenta os valores obtidos para cada LPS.
- *Unused Interface*: foram consideradas para esta anomalia as interfaces e classes isoladas, ou seja, que não possuíam nenhum relacionamento.
- *Unused Brick*: foram considerados para esta anomalia os pacotes onde nenhuma interface ou classe possuía relacionamentos.
- *Duplicate Component Functionality*: foi realizada uma avaliação interpretativa, visando encontrar componentes que duplicavam alguma funcionalidade.
- *Sloppy Delegation*: foi realizada uma avaliação interpretativa, visando encontrar classes ou interfaces que delegavam uma quantidade muito pequena de funcionalidades a outros componentes.

- *Dependency Cycle*: considerou-se os ciclos de dependência entre classes e/ou interfaces, ou seja, classes e/ou interfaces com ligação mútua.
- *Connector Envy*: foi realizada uma avaliação interpretativa, visando encontrar componentes (classes ou interfaces) que apresentassem alguma funcionalidade com elevado nível de interação.
- *Scattered Parasitic Functionality*: considerou-se os pacotes que tratam de uma característica (expostas na Seção 4.1), em que a mesma também é tratada em algum pacote adjacente, em paralelo com suas próprias características.
- *Large Class*: considerou-se tanto classes quanto interfaces. Foram consideradas *large classes* as que possuíam mais de 10 atributos, ou mais de 10 métodos, ou mais de 15 itens somando atributos e métodos.
- *Primitive Obsession*: foi realizada uma avaliação interpretativa, visando encontrar classes ou interfaces que apresentassem métodos que recebem muitos parâmetros de tipos primitivos, exceto nos casos onde esses parâmetros fossem considerados coerentes com o contexto.

Tabela 4: *Thresholds* de *Link Overload* (LO) e *Concern Overload* (CO) para cada LPS

<b>Threshold</b>	<b>AGM</b>	<b>Banking</b>	<b>MM</b>
LO (entrada)	3	2	3
LO (saída)	3	2	3 (Grupo 1) / 4 (Grupo 2)
LO (bidirecional)	1	3	0
CO	2	2	2

### 4.3 Considerações finais

Neste capítulo foram apresentados a metodologia do trabalho, as LPSs analisadas, a divisão dos conjuntos de ALPs e os *thresholds* utilizados na análise.

O próximo capítulo apresenta os resultados detalhados da análise, separados por grupos (conforme a Tabela 3), bem como faz uma avaliação sobre os resultados dessa análise, apontando a quantidade de anomalias encontradas e propõe diretrizes para evitar a ocorrência das mesmas.

## 5 Análise de anomalias arquiteturais

A análise das instâncias de ALPs consideradas – cujas figuras encontram-se no Anexo A – resultou na identificação de uma série de anomalias arquiteturais. Os resultados encontram-se detalhados nas tabelas das seções seguintes, onde a coluna *Local* segue o padrão *pacote.nome\_da\_classe\_ou\_interface* e a coluna *Anomalia* indica o nome da anomalia arquitetural encontrada. Além disso, foi ilustrada uma ocorrência de cada tipo de anomalia encontrada na análise.

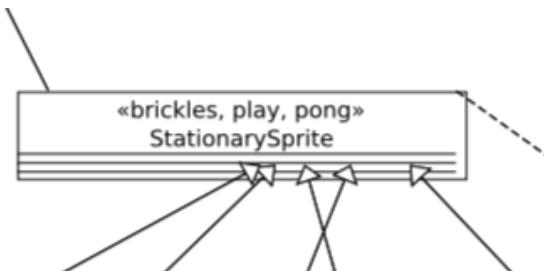
Este capítulo está dividido em seções conforme segue. As Seções 5.1 a 5.3 detalham as anomalias arquiteturais encontradas nos Grupos 1 a 3, respectivamente. A Seção 5.4 avalia os resultados da análise, agrupando os resultados obtidos pelo tipo de anomalia. Por fim, a Seção 5.5 propõe diretrizes para evitar a ocorrência das anomalias encontradas.

### 5.1 Arquiteturas do Grupo 1

No Grupo 1 (instâncias obtidas do trabalho de Choma Neto (2017)) foram avaliadas 4 instâncias da LPS AGM, 3 instâncias da LPS Banking e 4 instâncias da LPS MM. As Figuras 13 a 19 ilustram ocorrências das anomalias encontradas neste grupo.

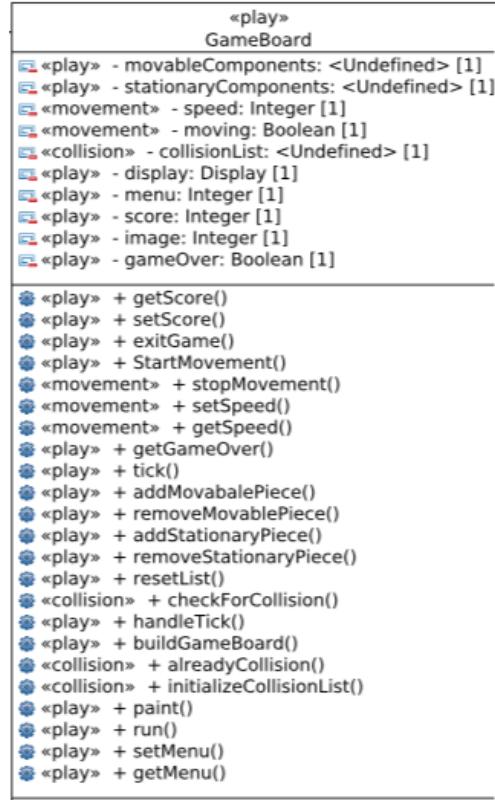
A Figura 13 ilustra um exemplo tanto de *Link Overload* quanto de *Concern Overload*. A classe *StationarySprite* apresenta 5 links de entrada, excedendo o *threshold* de 3 para esta LPS. Além disso, trata de 3 interesses, excedendo o *threshold* de 2.

Figura 13: Exemplo de ocorrência de *Link Overload* e *Concern Overload* na ALP AGM original (CHOMA NETO, 2017)



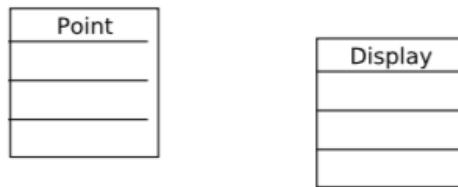
A Figura 14 ilustra um exemplo de *Large Class*. A classe *GameBoard* possui 23 métodos, excedendo o *threshold* de 10.

Figura 14: Exemplo de ocorrência de *Large Class* na ALP AGM original (CHOMA NETO, 2017)



A Figura 15 ilustra dois exemplos de ocorrência de *Unused Interface*. Foi considerada esta anomalia para classes também, conforme definido na Seção 3.2. As classes *Point* e *Display* não possuem ligação com nenhuma outra classe ou interface da ALP.

Figura 15: Exemplos de ocorrência de *Unused Interface* na ALP AGM original (CHOMA NETO, 2017)



A Figura 16 ilustra um exemplo de *Unused Brick*. No pacote *PlayGameGUI* nenhuma das interfaces ou classes (no caso, apenas a classe *PlayGameGUI*) possui links a outros componentes. Desse modo, *PlayGameGUI* é um pacote isolado no projeto.

A Figura 17 ilustra um exemplo de *Connector Envy*. A classe de negócios *DebitCard* apresenta métodos de CRUD (*create*, *read* e *delete*), que estariam mais adequados se delegados a um conector.

Figura 16: Exemplo de ocorrência de *Unused Brick* na ALP AGM com melhor COE (CHOMA NETO, 2017)

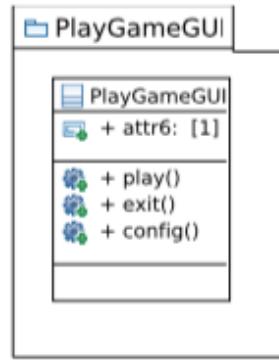
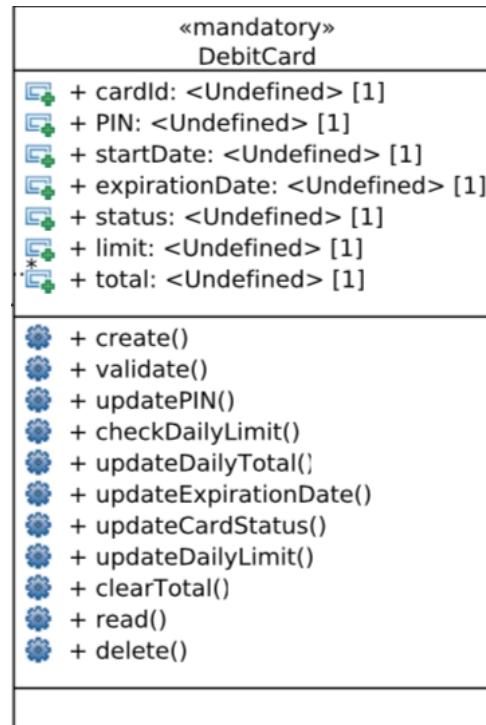
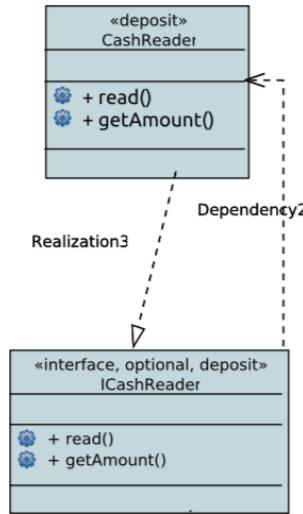


Figura 17: Exemplo de ocorrência de *Connector Envy* na ALP Banking original (CHOMA NETO, 2017)



A Figura 18 ilustra um exemplo de Dependency Cycle. A classe *CashReader* e a interface *ICashReader* estão ligadas entre si, formando uma dependência cíclica.

Figura 18: Exemplo de ocorrência de *Dependency Cycle* na ALP Banking Original (CHOMA NETO, 2017)



A seguir, as Tabelas 5 a 15 apresentam as listas de todas as anomalias encontradas no Grupo 1.

Tabela 5: Anomalias na ALP AGM original

Local	Anomalia
GameGUI.GameGUI	Link Overload
GameCtrl.GameCtrl	Link Overload
GameBoardCtrl.StationarySprite	Link Overload
GameBoardCtrl.Player	Link Overload
GameBoardCtrl.GameBoard	Link Overload
GameMgr.IGameMgt	Concern Overload
GameBoardCtrl.StationarySprite	Concern Overload
GameBoardCtrl.Point	Unused Interface
GameBoardCtrl.Display	Unused Interface
GameBoardCtrl.GameBoard	Large Class
GameBoardCtrl.Player	Large Class
GameBoardMgr.IGameBoardMgt	Large Class

Tabela 6: Anomalias na ALP AGM com Padrão de Projeto *Strategy*

<b>Local</b>	<b>Anomalia</b>
GameBoardCtrl.GameBoard	Link Overload
GameBoardCtrl.Player	Link Overload
GameCtrl.GameCtrl	Link Overload
InitializeMgr.InitializationMgt	Link Overload
Package85591Ctrl.StationarySprite	Link Overload
Package85591Ctrl.StationarySprite	Concern Overload
GameMgr.IGameMgt	Concern Overload
GameBoardCtrl.GameBoard	Large Class
GameBoardCtrl.Player	Large Class
GameBoardMgr.IGameBoardMgt	Large Class

Tabela 7: Anomalias na ALP AGM com melhor COE

<b>Local</b>	<b>Anomalia</b>
Package20560Ctr.StationarySprite	Link Overload
GameBoardCtrl.GameBoard	Link Overload
GameBoardCtrl.Player	Link Overload
GameCtrl.GameCtrl	Link Overload
Package20560Ctrl.StationarySprite	Concern Overload
GameMgr.IGameMgt	Concern Overload
GameBoardCtrl.GameBoard	Large Class
GameBoardCtrl.Player	Large Class
GameBoardMgr.IGameBoardMgt	Large Class
GameCtrl.IPlayPong	Unused Interface
GameCtrl.IPlayBrickles	Unused Interface
GameCtrl.IPlayBowling	Unused Interface
PlayGameGui.PlayGameGUI	Unused Interface
AnimationLoopMgr.IAnimationLoopMgr	Unused Interface
GameMgr.IGameMgt	Unused Interface
PlayGameGUI	Unused Brick

Tabela 8: Anomalias na ALP AGM com melhor ACLASS

<b>Local</b>	<b>Anomalia</b>
Package23385Ctr.GameBoard	Link Overload
Package23858Ctr.StationarySprite	Link Overload
GameBoardCtrl.Player	Link Overload
GameCtrl.GameCtrl	Link Overload
Package23385Ctr.GameBoard	Large Class
Package24023Mgr.IGameBoardMgt	Large Class
GameBoardCtrl.Player	Large Class
GameMgr.IGameMgt	Concern Overload
Package23858Ctr.StationarySprite	Concern Overload
AnimationLoopMgr.IAnimationLoopMgt	Unused Interface
GameMgr.IGameMgt	Unused Interface
PlayGameGui.PlayGameGUI	Unused Interface
GameCtrl.IPlayPong	Unused Interface
GameCtrl.IPlayBrickles	Unused Interface
GameCtrl.IPlayBowling	Unused Interface
PlayGameGUI	Unused Brick

Tabela 9: Anomalias na ALP Banking original

<b>Local</b>	<b>Anomalia</b>
BankingServiceServer1.Account	Link Overload
ATMClient1.ATMControl	Link Overload
DisplayPromptsClient1.DisplayPrompts	Link Overload
ATMClient1.ATMTransactior	Link Overload
ATMClient1.ATMControl	Concern Overload
CardServiceServer2.DebitCard	Large Class
CardServiceServer2.DebitCard	Connector Envy

Tabela 10: Anomalias na ALP Banking com melhor ACLASS

<b>Local</b>	<b>Anomalia</b>
Package79980.Interface39888	Link Overload
BankingServiceServer1.Account	Link Overload
DisplayPromptsClient1.DisplayPrompts	Link Overload
ATMClient1.ATMControl	Link Overload
ATMClient1.ATMTransactior	Link Overload
ATMClient1.ATMControl	Concern Overload
CardServiceServer2.DebitCard	Large Class
CardServiceServer2.DebitCard	Connector Envy
ATMClient1.ATMCard	Unused Interface
ATMClient1.ATMCash	Unused Interface

Tabela 11: Anomalias na ALP Banking com melhor COE e melhor ED

<b>Local</b>	<b>Anomalia</b>
Package86.Interface40	Link Overload
Package46.Interface20	Link Overload
Package130.Interface65	Link Overload
ATMClient1.ATMControl	Link Overload
ATMClient1.ATMTransactior	Link Overload
ATMClient1.ATMControl	Concern Overload
CardServiceServer2.DebitCard	Large Class
CardServiceServer2.DebitCard	Connector Envy
ATMClient1.ATMCard	Unused Interface
ATMClient1.ATMCash	Unused Interface

Tabela 12: Anomalias na ALP MM original

<b>Local</b>	<b>Anomalia</b>
MediaGUI.MediaGUI	Link Overload
MediaMgr.Media	Link Overload
AlbumCtrl.AlbumCtrl	Link Overload
MediaMgr.MediaMgr	Large Class
MediaMgr.IMediaMgt	Large Class

Tabela 13: Anomalias na ALP MM com Padrão de Projeto *Strategy*

<b>Local</b>	<b>Anomalia</b>
AlbumCtrl.AlbumCtrl	Link Overload
MediaGUI.MediaGUI	Link Overload
MediaMgr.Media	Link Overload
MediaCtrl.MediaCtrl	Link Overload
MediaMgr.MediaMgr	Large Class
MediaMgr.IMediaMgt	Large Class

Tabela 14: Anomalias na ALP MM com melhor COE

<b>Local</b>	<b>Anomalia</b>
AlbumCtrl.AlbumCtrl	Link Overload
AlbumGUI.AlbumGUI	Link Overload
MediaGUI.MediaGUI	Link Overload
MediaMgr.Media	Link Overload
MediaCtrl.MediaCtrl	Link Overload
MediaMgr.MediaMgr	Large Class
MediaMgr.IMediaMgt	Large Class
MediaCtrl.IManageMedia	Large Class

Tabela 15: Anomalias na ALP MM com melhor ACLASS e melhor ED

<b>Local</b>	<b>Anomalia</b>
MediaGUI.MediaGUI	Link Overload
Package10Mgr.Media	Link Overload
Package10Mgr.Class1	Link Overload
MediaCtrl.MediaCtrl	Link Overload
MediaMgr.MediaMgr	Link Overload
UserMgr.User	Link Overload
MediaMgr.MediaMgr	Large Class
MediaMgr.IMediaMgt	Large Class
MediaCtrl.IManageMedia	Large Class

## 5.2 Arquiteturas do Grupo 2

No Grupo 2 (instâncias obtidas do trabalho de Lavezzo (2017)) foram avaliadas 2 instâncias da LPS Banking e 4 instâncias da LPS MM. Todos os tipos de anomalias encontrados neste grupo foram verificados também no Grupo 1, sendo assim as figuras de exemplos para este grupo foram suprimidas. As Tabelas 16 a 21 apresentam as listas de todas as anomalias encontradas neste grupo.

Tabela 16: Anomalias na ALP Banking original

Local	Anomalia
BankingServiceServer1.Account	Link Overload
ATMClient1.ATMControl	Link Overload
DisplayPromptsClient1.DisplayPrompts	Link Overload
ATMClient1.ATMTransactior	Link Overload
ATMClient1.ATMControl	Concern Overload
CardServiceServer2.DebitCard	Large Class
CardServiceServer2.DebitCard	Connector Envy

Tabela 17: Anomalias na ALP Banking gerada pela OPLA-Tool

Local	Anomalia
BankingServiceServer1.Account	Link Overload
ATMClient1.ATMTransactior	Link Overload
ATMClient1.ATMControl	Link Overload
CardServiceServer2.DebitCard	Link Overload
CardServiceServer2.DebitCard	Connector Envy
CashReader + ICashReader	Dependency Cycle

Tabela 18: Anomalias na ALP MM original

Local	Anomalia
MediaGUI.MediaGUI	Link Overload
MediaMgr.Media	Link Overload
AlbumCtrl.AlbumCtrl	Link Overload
MediaMgr.MediaMgr	Large Class
MediaMgr.IMediaMgr	Large Class

Tabela 19: Anomalias na ALP MM com melhor DC

<b>Local</b>	<b>Anomalia</b>
MediaCtrl.MediaCtrl	Link Overload
MediaGUI.MediaGUI	Link Overload
MediaMgr.MediaMgr	Link Overload
Package1371Mgr.Media	Link Overload
AlbumMgr.Album	Link Overload
AlbumMgr.Class522	Link Overload
MediaCtrl.IManageMedia	Large Class
Package1770Ctrl.IAddMediaAlbum	Unused Interface
Interface2080	Unused Interface
Interface1896	Unused Interface
Interface1681	Unused Interface
Interface2319	Unused Interface
Package420Mgr.Interface382	Unused Interface
Package290.Mgr.Interface260	Unused Interface
Package1143Mgr.Interface1076	Unused Interface
Package339Mgr.Interface303	Unused Interface
Package420Mgr	Unused Brick

Tabela 20: Anomalias na ALP MM com melhor ED

<b>Local</b>	<b>Anomalia</b>
MediaMgr.MediaMgr	Link Overload
MediaGUI.MediaGUI	Link Overload
MediaCtrl.MediaCtrl	Link Overload
MediaMgr.Interface1004	Unused Interface
Interface647	Unused Interface
Package339Mgr.Interface303	Unused Interface
Package290Mgr.Interface260	Unused Interface
Package1143.Interface1076	Unused Interface
Package1770Ctr.Interface1681	Unused Interface
Interface2080	Unused Interface
Interface2319	Unused Interface
Interface1896	Unused Interface
IAddMediaAlbum	Unused Interface
Package420Mgr.Interface382	Unused Interface
Package420Mgr	Unused Brick

Tabela 21: Anomalias na ALP MM com melhor Eleg

Local	Anomalia
MediaMgr.MediaMgr	Link Overload
MediaGUI.MediaGUI	Link Overload
MediaCtrl.MediaCtrl	Link Overload
Package6665Ctr.Interface6605	Unused Interface
Interface6473	Unused Interface
Interface6166	Unused Interface
Interface6272	Unused Interface
Package5657Mgr.Interface5331	Unused Interface
Package5834Mgr.Interface5492	Unused Interface
Package 5631Mgr.Interface5308	Unused Interface
Package5540Mgr.Interface5220	Unused Interface
Package5834Mgr	Unused Brick

### 5.3 Arquiteturas do Grupo 3

No Grupo 3 (instâncias obtidas do trabalho de Dias (2016)) foram avaliadas 3 instâncias da LPS AGM e 4 instâncias da LPS MM.

A Figura 19 ilustra um exemplo de *Sloppy Delegation*. A classe *AnimationLoopMgr* delega apenas o método *stopAnimation* para a *Class1749*, o que é muito pouco e poderia ser realizado pela própria classe *AnimationLoopMgr*.

Figura 19: Exemplo de ocorrência de *Sloppy Delegation* na ALP AGM com melhor CM (DIAS, 2016)

Abaixo, as Tabelas 22 a 28 apresentam as listas de todas as anomalias encontradas neste grupo.

Tabela 22: Anomalias na ALP AGM original

<b>Local</b>	<b>Anomalia</b>
GameBoardCtrl.Point	Unused Interface
GameBoardCtrl.Display	Unused Interface
GameBoardCtrl.GameBoard	Large Class
GameBoardCtrl.Player	Large Class
GameBoardMgr.IGameBoardMgt	Large Class
GameBoardCtrl.StationarySprite	Link Overload
GameCtrl.GameCtrl	Link Overload
GameGUI.GameGUI	Link Overload
GameMgr.IGameMgt	Concern Overload
GameBoardCtrl.StationarySprite	Concern Overload

Tabela 23: Anomalias na ALP AGM com melhor ED e melhor FM

<b>Local</b>	<b>Anomalia</b>
Package31542Mgr.Interface20868	Unused Interface
Package31185Ctrl.Interface20618	Unused Interface
Package31685Mgr.Interface20958	Unused Interface
Package31259Ctrl.Interface20671	Unused Interface
PlayGameGUI.PlayGameGUI	Unused Interface
Package31121Ctrl.Interface20571	Unused Interface
Package31346Mgr.Interface20733	Unused Interface
Package31962Mgr.Interface21132	Unused Interface
Package30994Ctrl.Interface20783	Unused Interface
Package30994Ctrl.Interface20480	Unused Interface
Package30994Ctrl.InstallGame	Unused Interface
GameBoardCtrl.Player	Large Class
Package30757Ctrl.GameBoard	Large Class
GameBoardMgr.IGameBoardMgt	Large Class
Package31542Mgr	Unused Brick
Package31185Ctrl	Unused Brick
Package31685Mgr	Unused Brick
Package31259Ctrl	Unused Brick
PlayGameGUI	Unused Brick
Package31121Ctrl	Unused Brick
Package31346Mgr	Unused Brick
Package30940Mgr.InitializationMgt	Link Overload
Package31093Ctrl.StationarySprite	Link Overload
GameCtrl.GameCtrl	Link Overload
Package30757Ctrl.GameBoard	Link Overload
Package31093Ctrl.StationarySprite	Concern Overload

Tabela 24: Anomalias na ALP AGM com melhor CM

Local	Anomalia
Package7534Ctrl.Interface4811	Unused Interface
Package8552Mgr.Interface5486	Unused Interface
Package7436Ctrl.Interface4742	Unused Interface
Package8085Ctrl.Interface5191	Unused Interface
Package8460Mgr.Interface5434	Unused Interface
PlayGameGUI.PlayGameGUI	Unused Interface
Package8001Mgr.Interface5129	Unused Interface
Package8259Mgr.Interface5310	Unused Interface
AnimationLoopMgr.IAnimationLoopMgt	Unused Interface
Package7898Mgr.Interface5071	Unused Interface
Package7384Ctrl.Interface4706	Unused Interface
Package7795Ctrl.GameBoard	Large Class
Package7534Ctrl	Unused Brick
Package8552Mgr	Unused Brick
Package7436Ctrl	Unused Brick
Package8085Ctrl	Unused Brick
Package8460Mgr	Unused Brick
PlayGameGUI	Unused Brick
Package8001Mgr	Unused Brick
Package8259Mgr	Unused Brick
Package7950Ctrl.StationarySprite	Link Overload
GameCtrl.GameCtrl	Link Overload
Package7795Ctrl.GameBoard	Link Overload
Package7950Ctrl.StationarySprite	Concern Overload
AnimationLoopMgr.AnimationLoopMgr	Sloppy Delegation

Tabela 25: Anomalias na ALP MM original

Local	Anomalia
MediaMgr.MediaMgr	Large Class
MediaMgr.IMediaMgt	Large Class
MediaMgr.Media	Link Overload
MediaGui.MediaGUI	Link Overload
AlbumCtrl.AlbumCtrl	Link Overload
UserMgr.User	Sloppy Delegation

Tabela 26: Anomalias na ALP MM com melhor ED

<b>Local</b>	<b>Anomalia</b>
MediaGUI.IMediaGUI	Unused Interface
Package66Mgr.Interface59	Unused Interface
Package304Mgr.Interface278	Unused Interface
Package439Mgr.Interface400	Unused Interface
MediaMgr.Interface470	Unused Interface
UserMgr.Interface332	Unused Interface
Package402Mgr.Interface369	Unused Interface
Package778Mgr.Interface728	Unused Interface
Package240Ctrl.Interface219	Unused Interface
Package240Ctrl.IManageAlbum	Unused Interface
Package240Ctrl.IAddMediaAlbum	Unused Interface
Package679Mgr.Media	Link Overload
MediaGui.MediaGUI	Link Overload
MediaCtrl.MediaCtrl	Link Overload
MediaMgr.MediaMgr	Link Overload
MediaCtrl.IManageMedia	Large Class
Package402Mgr	Unused Brick
Package778Mgr	Unused Brick

Tabela 27: Anomalias na ALP MM com melhor CM

<b>Local</b>	<b>Anomalia</b>
Package4141Mgr.Interface3956	Unused Interface
Package4141Mgr.Interface4366	Unused Interface
Package4141Mgr.IAlbumMgt	Unused Interface
Package4141Mgr.Interface4167	Unused Interface
Package4141Mgr.Interface4013	Unused Interface
Package4141Mgr.Interface39..	Unused Interface
Package2750Mgr.Interface26..	Unused Interface
Package2611Mgr.Interface2501	Unused Interface
Package2853Mgr.Interface2724	Unused Interface
Package4606Ctrl.Interface4457	Unused Interface
Package4606Ctrl.IAddMediaAlbum	Unused Interface
Package4673Mgr.Interface4523	Unused Interface
Package2924Mgr.Interface2792	Unused Interface
Package4786Mgr.Interface4635	Unused Interface
Package3112Mgr.Interface2972	Unused Interface
AlbumMgr	Unused Brick
Package4673Mgr	Unused Brick
Package2924Mgr	Unused Brick
Package4786Mgr	Unused Brick
Package3112Mgr	Unused Brick
Package2750Mgr.Media	Link Overload
MediaGUI.MediaGUI	Link Overload
MediaMgr.MediaMgr	Link Overload
MediaCtrl.MediaCtrl	Link Overload

Tabela 28: Anomalias na ALP MM com melhor FM

Local	Anomalia
Package33467Mgr.Interface32446	Unused Interface
Package33498Mgr.Interface32476	Unused Interface
Package34270Ctrl.Interface33246	Unused Interface
UserMgr.Interface22093	Unused Interface
Package34195Mgr.Interface33169	Unused Interface
Package34195Mgr.IMediaMgt	Unused Interface
Package34195Mgr.Interface33842	Unused Interface
Package33280Mgr.Interface32272	Unused Interface
Package33132Mgr.Interface32139	Unused Interface
Package33017Mgr.Interface32035	Unused Interface
Package33467Mgr	Unused Brick
Package33498Mgr	Unused Brick

#### 5.4 Avaliação dos resultados da análise

Para avaliar os resultados da análise das anomalias encontradas, inicialmente os resultados foram agrupados pela quantidade de ocorrências de cada tipo de anomalia, em cada instância de ALP. As Tabelas 29 a 31 apresentam estas quantidades, em cada LPS analisada.

Tabela 29: Quantidade de anomalias na LPS AGM

Anomalia	Grupo 1				Grupo 3			Total
	Original	Padrão de Projeto Strategy	Melhor COE	Melhor ACLASS	Original	Melhor ED e FM	Melhor CM	
Link Overload	5	5	4	4	3	4	3	<b>28</b>
Concern Overload	2	2	2	2	2	1	1	<b>12</b>
Unused Interface	2	0	6	6	2	11	11	<b>38</b>
Unused Brick	0	0	1	1	0	7	8	<b>17</b>
Large Class	3	3	3	3	3	3	1	<b>19</b>
Connector Envy	0	0	0	0	0	0	0	<b>0</b>
Dependency Cycle	0	0	0	0	0	0	0	<b>0</b>
Sloppy Delegation	0	0	0	0	0	0	1	<b>1</b>

Tabela 30: Quantidade de anomalias na LPS Banking

Anomalia	Grupo 1			Grupo 2		Total
	Original	Melhor ACLASS	Melhor COE e ED	Original	Gerada pela OPLA-Tool	
Link Overload	4	5	5	4	4	<b>22</b>
Concern Overload	1	1	1	1	0	<b>4</b>
Unused Interface	0	2	2	0	0	<b>4</b>
Unused Brick	0	0	0	0	0	<b>0</b>
Large Class	1	1	1	1	0	<b>4</b>
Connector Envy	1	1	1	1	1	<b>5</b>
Dependency Cycle	0	0	0	0	1	<b>1</b>
Sloppy Delegation	0	0	0	0	0	<b>0</b>

Tabela 31: Quantidade de anomalias na LPS MM

Anomalia	Grupo 1				Grupo 2				Grupo 3				Total
	Original	Padrão de Projeto Strategy	Melhor COE	Melhor ACLASS e ED	Original	Melhor DC	Melhor ED	Melhor Eleg	Original	Melhor ED	Melhor CM	Melhor FM	
Link Overload	3	4	5	6	3	6	3	3	3	4	4	0	<b>44</b>
Concern Overload	0	0	0	0	0	0	0	0	0	0	0	0	<b>0</b>
Unused Interface	0	0	0	0	0	9	11	8	0	11	15	10	<b>64</b>
Unused Brick	0	0	0	0	0	1	1	1	0	2	5	2	<b>12</b>
Large Class	2	2	3	3	2	1	0	0	2	1	0	0	<b>16</b>
Connector Envy	0	0	0	0	0	0	0	0	0	0	0	0	<b>0</b>
Dependency Cycle	0	0	0	0	0	0	0	0	0	0	0	0	<b>0</b>
Sloppy Delegation	0	0	0	0	0	0	0	0	1	0	0	0	<b>1</b>

Foram observados, ao todo, 8 tipos de anomalias: *Link Overload*, *Concern Overload*, *Unused Interface*, *Unused Brick*, *Large Class*, *Connector Envy*, *Dependency Cycle* e *Sloppy Delegation*.

O tipo de anomalia mais frequente nas ALPs originais foi o *Link Overload*. No caso das LPSs AGM e Banking houve pouca variação na quantidade deste tipo de anomalia entre as instâncias originais e as geradas pela ferramenta OPLA-Tool, oscilando entre um aumento de 1 e uma redução de 1. Apenas na LPS MM foi verificada uma grande variação, com tendência de aumento nos Grupos 1 e 2; já no Grupo 3, duas instâncias sofreram aumento de 1, enquanto uma instância sofreu redução de 3.

O segundo tipo de anomalia mais frequente nas ALPs originais foi o *Large Class*. Para todas as LPSs houve tendência de manutenção ou redução na quantidade deste tipo de anomalia, quando se compara as instâncias geradas pela OPLA-Tool às originais. A única exceção se deu na LPS MM, no Grupo 1, onde foi verificado um leve aumento.

A quantidade de ocorrências de *Concern Overload* nas ALPs originais sofreu manutenção ou leve redução em todos casos, não verificando nenhum aumento.

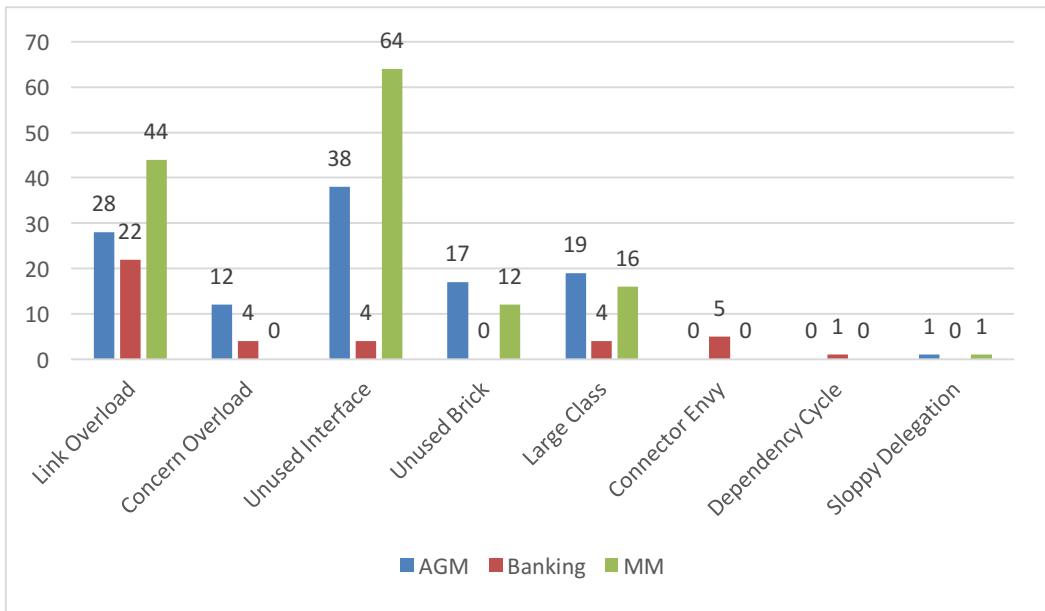
O tipo de anomalia encontrado com maior frequência nas soluções geradas pela OPLA-Tool, e que não estava presente nas instâncias originais, foi o *Unused Interface* – que gerou também o problema *Unused Brick* nos casos um pacote continha uma única interface, e esta era uma *Unused Interface*. Acredita-se que este problema seja oriundo de um erro de implementação da ferramenta, e não da abordagem MOA4PLA, ao criar pacotes contendo apenas uma interface, a qual não era ligada a nenhum outro componente. A inserção destes dois tipos de anomalias está sendo corrigida em um trabalho em andamento.

A Figura 20 agrupa em um gráfico a quantidade de ocorrências de anomalias por LPS. Observa-se que *Link Overload*, *Concern Overload*, *Unused Interface*, *Unused Brick* e *Large Class* são os tipos de anomalias que aparecem com frequência. Enquanto isso, *Connector Envy*, *Dependency Cycle* e *Sloppy Delegation* aparecem em casos isolados.

Quanto à inserção – nas soluções geradas pela OPLA-Tool – de outros tipos de anomalias além de *Unused Interface* e *Unused Brick*, a ocorrência se deu em quantidade desprezível, e pode ser considerada consequência do fator aleatório presente na abordagem MOA4PLA.

Sendo assim, conclui-se de forma geral que a OPLA-Tool influenciou em um leve aumento na quantidade de *Link Overload*, leve redução na quantidade de *Concern Overload* e de *Large Class*, e aumento expressivo de *Unused Interface* e *Unused Brick* (em correção), tendo influência desprezível para os demais tipos de anomalias.

Figura 20: Quantidade de ocorrências de anomalias por LPS



Isso se deve ao fato de que alguns operadores de busca presentes na abordagem MOA4PLA criam novas classes na ALP que está sendo otimizada, para mover atributos ou métodos das classes originais. Por um lado, isto implica no aumento da quantidade de *links* entre as classes existentes (aumentando as ocorrências de *Link Overload*). Por outro lado, isto distribui melhor os interesses tratados pelas classes (reduzindo as ocorrências de *Concern Overload*) e reduz o próprio tamanho da classe (reduzindo as ocorrências de *Large Class*).

### 5.5 Proposta de diretrizes para evitar a ocorrência de anomalias

A análise realizada em instâncias de projetos de ALP obtidas nos trabalhos de Choma Neto (2017), Lavezzo (2017) e Dias (2016) identificou a ocorrência de 8 tipos de anomalias arquiteturais. Com base nessa análise, foram propostas as seguintes diretrizes para evitar os tipos de anomalias encontrados, bem como sugestões de ações a serem realizadas quando elas forem detectadas:

- **Link Overload:** A fórmula definida em Garcia (2014) – e utilizada neste trabalho – se mostrou um bom ponto de partida para estabelecer o *threshold* de *links*. No entanto, este *threshold* foi muito rígido em casos de heranças de classes. Sendo assim, propõe-se que os *links* relacionados a herança sejam contabilizados separadamente, de modo que mesmo com a aplicação da mesma fórmula, obtenham um valor de *threshold* próprio. Quando é detectado *Link Overload* em alguma das classes de uma

determinada solução, uma possibilidade de prevenção dessa anomalia seria penalizar seu *fitness*.

- **Concern Overload:** A fórmula definida em Garcia (2014) – e utilizada neste trabalho – se mostrou eficiente em verificar o excesso de interesses tratados pelas classes. Sendo assim, sugere-se sua aplicação sem alterações. Quando é detectada uma classe com *Concern Overload*, pode-se sugerir a aplicação do *Feature-Driven Operator*, de modo a tentar modularizar as características daquela classe onde a anomalia está presente.
- **Unused Interface:** Este tipo de anomalia arquitetural já está sendo corrigido em um trabalho em andamento.
- **Unused Brick:** Este tipo de anomalia arquitetural já está sendo corrigido em um trabalho em andamento.
- **Large Class:** Propõe-se incluir na OPLA-Tool uma configuração para o *threshold* de quantidade de atributos e métodos em uma classe. Sugere-se adaptar a fórmula do *threshold* de *Link Overload* definida por Garcia (2014), para contar os atributos, os métodos e a soma de atributos e métodos presentes em cada classe. Quando é detectada uma Large Class, pode-se sugerir a aplicação de operadores que movem atributos ou métodos para outras classes (*Move Attribute*, *Move Method* e *Add Class*), de modo a reduzir o tamanho da classe original.
- **Connector Envy:** A detecção deste tipo de anomalia demanda uma avaliação interpretativa, já que depende do tipo de ação realizada por um método específico. Sendo assim, considera-se inviável propor um método para detectá-la. Porém, uma vez detectada, pode-se penalizar o *fitness* da solução avaliada, de modo que ela não seja priorizada pelo algoritmo.
- **Dependency Cycle:** De forma trivial, pode-se verificar a ocorrência de ciclos de dependência entre classes. Sugere-se que a solução tenha seu *fitness* penalizado, uma vez que a aplicação dos operadores de busca da MOA4PLA não garante a remoção da dependência cíclica. Outra possibilidade seria mostrar essa parte do projeto de ALP para o usuário palpitar se, dado o contexto, é apropriado esse ciclo de dependência.
- **Sloppy Delegation:** A detecção deste tipo de anomalia demanda uma avaliação interpretativa, já que depende do tipo de ação realizada por um método específico. Porém, uma vez detectada, sugere-se que a solução tenha seu *fitness* penalizado, já que

a aplicação dos operadores de busca não garantiria a coesão do método com a classe que possui os dados necessários a ele.

Essas diretrizes fornecem os elementos básicos para viabilizar a detecção e a remoção de anomalias nas soluções de projetos de ALP geradas pela OPLA-Tool, contudo é preciso aprofundar os estudos de como implementá-las e de sua efetividade.

## 6 Conclusão

Neste trabalho foram estudados temas de Engenharia de Software relacionados a Otimização de Arquitetura de Linha de Produto, com foco na abordagem MOA4PLA, na ferramenta OPLA-Tool e em anomalias arquiteturais listadas por Garcia (2014), Fowler (1999), Kerievsky (2004) e Parnas (1974). Em seguida, foi realizada uma análise sobre diversas instâncias de projeto de ALPs – referentes às LPSs AGM, Banking e MM – obtidas dos trabalhos de Choma Neto (2017), Lavezzo (2017) e Dias (2016), visando detectar as anomalias levantadas. Com base na análise dos resultados obtidos, verificou-se quais tipos de anomalias eram frequentes e quais eram raros nas instâncias originais, bem como os tipos de anomalias que a OPLA-Tool inseriu ou removeu nas instâncias por ela geradas. Por fim, foram propostas diretrizes para se detectar e remover os tipos de anomalias encontrados.

Este trabalho contribuiu ao identificar o cenário geral de anomalias arquiteturais nas instâncias de projeto de ALP originais e nas geradas pela OPLA-Tool. Desta forma foi possível identificar a influência que a ferramenta exerceu sobre as soluções geradas, em termos dos tipos de anomalias arquiteturais presentes. Identificou-se que, nas instâncias originais, os tipos de anomalias mais comuns foram *Link Overload*, *Large Class* e *Concern Overload*. Por outro lado, os tipos de anomalias mais frequentes nas soluções geradas pela OPLA-Tool foram *Unused Interface* e *Unused Brick*, ambos em correção em um trabalho em andamento. De forma geral, a ferramenta influenciou no aumento de ocorrências de *Link Overload* e na redução de ocorrências de *Concern Overload* e *Large Class*, sendo isto, provavelmente, resultado dos operadores de busca da MOA4PLA que movem atributos ou métodos de uma classe original para uma nova classe. Foram também dados os subsídios iniciais para se detectar e remover estes tipos de anomalias.

Como trabalhos futuros, sugere-se: um refinamento das diretrizes propostas na Seção 5.5, de forma a otimizar o processo de detecção e a remoção de anomalias arquiteturais; a implementação dessas diretrizes na OPLA-Tool; e a replicação deste estudo sobre outras LPSs.

## REFERÊNCIAS

- Choma Neto, J. *Uma abordagem memética para otimizar projeto de linha de produto de software*. Dissertação de Mestrado, Universidade Estadual de Maringá, 2017.
- Coello, C. A. C.; Lamont, G.; van Veldhuizen, D. *Evolutionary algorithms for solving multi-objective problems*. Genetic and Evolutionary Computation, 2nd ed. Berlin, Heidelberg: Springer, 2007.
- Colanzi, T. E. *Uma abordagem de otimização multiobjetivo para projeto arquitetural de linha de produto de software*. Tese de Doutorado, Universidade Federal do Paraná, 2014.
- Colanzi, T. E.; Vergilio, S. R.; Gimenes, I. M. S.; Oizumi, W. N. *A Search-Based Approach for Software Product Line Design*. In: The 18th International Software Product Line Conference (SPLC), Florence, Italy. 2014.
- Deb, K.; Pratap, A.; Agarwal, S.; Meyarivan, T. *A fast and elitist multiobjective genetic algorithm: NSGA-II*. IEEE Transactions on Evolutionary Computation, 6(2): 182-197, 2002.
- Dias, G. C. *Avaliação de projeto baseado em busca para arquitetura de linha de produto de software*. Trabalho de Conclusão de Curso, Universidade Estadual de Maringá, 2016.
- Durillo, J. J.; Nebro, A. J. *jMetal: A Java framework for multi-objective optimization*. Advances in Engineering Software, Volume 42 Issue 10, 2011.
- Féderle, E. L. *Uma ferramenta de apoio ao projeto arquitetural de linha de produto de software baseado em busca*. Dissertação de Mestrado, Universidade Federal do Paraná, 2014.
- Féderle, E.; Ferreira, T. N.; Colanzi, T. E.; Vergilio, S. R. *OPLA-Tool: A Support Tool for Search-Based Product Line Architecture Design*. In Proc. of the 19th International Conference on Software Product Line (SPLC '15). 370–373.
- Garcia, J. *A Unified Framework for Studying Architectural Decay of Software Systems*. Tese de Doutorado, Universidade do Sul da California, 2014.
- Gomaa, H. *Software Modeling and design: UML, use cases, patterns and software architectures*. Cambridge University Press, 2011.
- Fowler, M. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 1999.
- Harman, M.; Mansouri, S. A.; Zhang, Y. *Search based software engineering: A comprehensive analysis and review of trends techniques and applications*. Relatório Técnico TR-09-03, King's College London, 2009.
- Johansson, E. S.; Oizumi, W. N. *Avaliação de Projetos Arquiteturais Baseados em Busca*. Trabalho de Conclusão de Curso: Instituto Federal do Paraná, 2016.
- Kang, K. C.; Cohen, S. G.; Hess, J. A.; Novak, W. E.; Peterson, A.S. *Feature-oriented domain analysis FODA feasibility study*. Relatório Técnico, CMU/SEI-90-TR-21, Carnegie-Mellon University - Software Engineering Institute (SEI), 1990.
- Kerievsky, J. *Refactoring to Patterns*. Addison-Wesley Professional, 2004.

- Knowles, J. D.; Corne, D. W. *Approximating the nondominated front using the Pareto archived evolution strategy*. Evolutionary Computation, 8:149-172, 2000.
- Lavezzo, T. O. *Análise Qualitativa de Arquitetura de Linha de Produto de Software otimizada a partir da Ferramenta OPLA-Tool*. Trabalho de Conclusão de Curso, Universidade Estadual de Maringá, 2017.
- Linden, F. J. v. d.; Schmid, K.; Rommes, E. *Software product lines in action: The best industrial practice in product line engineering*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.
- Martens, A.; Koziolek, H.; Becker, S.; Reussner, R. Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. In: *Proceedings of the first joint WOSP/SIPEW International Conference on Performance Engineering (WOSP/SIPEW '10)*, New York, NY, USA: ACM, 2010, p. 105–116. Disponível em <http://doi.acm.org/10.1145/1712605.1712624>
- Pareto, V. *Manuel d'économie politique*. Paris: Ams Press, 1927.
- Santos, M. C. B.; Colanzi, T. E.; Amaral, A. M. M. M.; Oliveira Junior, E. Preliminary Study on the Correlation of Objective Functions to Optimize Product-Line Architectures. In *Proceedings of SBCARS 2017, Fortaleza, CE, Brazil, September 18–19, 2017*, 10 pages.
- Schmid, K.; Verlage, M. *The economic impact of product line adoption and evolution*. IEEE Software, v. 19, p. 50–57, 2002. Disponível em <http://dx.doi.org/10.1109/MS.2002.1020287>
- SEI Software Engineering Institute. Software Product Lines, 2016. <http://www.sei.cmu.edu/productlines/>.
- Taylor, R. N.; Medvidovic, N.; Dashofy, E. M. *Software Architecture: Foundations, Theory and Practice*. John Wiley & Sons, 2010.
- Verdecia, Y. D.; Colanzi, T. E.; Vergilio, S. R.; dos Santos, M. C. B. *An Enhanced Evaluation Model for Search-based Product Line Architecture Design*. In: XX Ibero-American Conference on Software Engineering (CIbSE 2017) - 39th International Conference on Software Engineering, 2017, Buenos Aires. Proceedings of the XX Ibero-American Conference on Software Engineering (CIbSE 2017), 2017.
- Young, T. *Using AspectJ to Build a Software Product Line for Mobile Devices*. Dissertação de Mestrado, University of British Columbia, 2005.

**ANEXOS**

## **ANEXO A – Instâncias de ALPs analisadas**

Este anexo contém as instâncias de projeto de ALP que foram analisadas durante o desenvolvimento deste trabalho. Esses projetos também estão disponíveis em: <https://github.com/eduardogermano/tcc-figuras-alp>.

Figura 21: AGM original (CHOMA NETO, 2017)

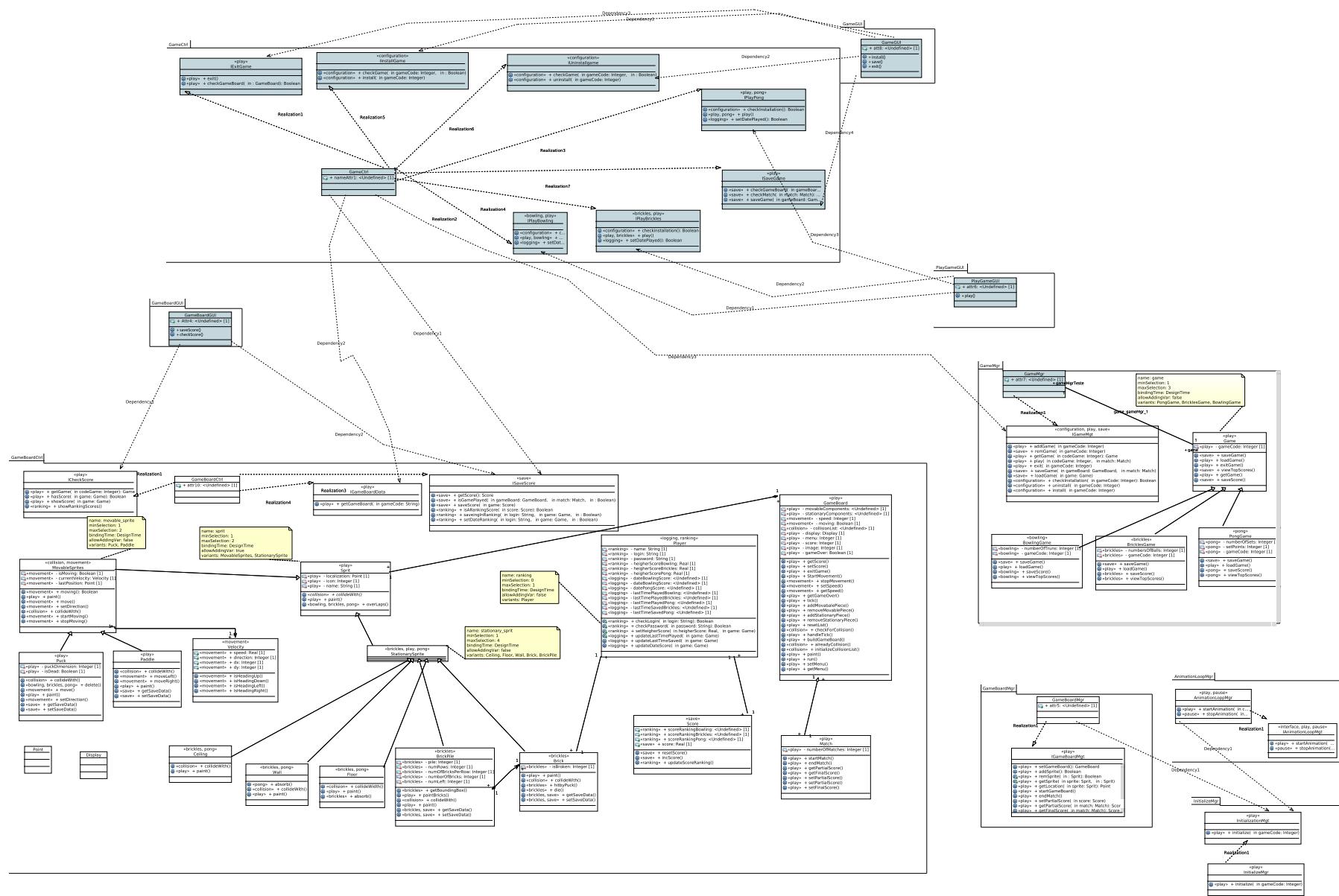


Figura 22: AGM com Padrão de Projeto Strategy (CHOMA NETO, 2017)



Figura 23: AGM com melhor COE (CHOMA NETO, 2017)

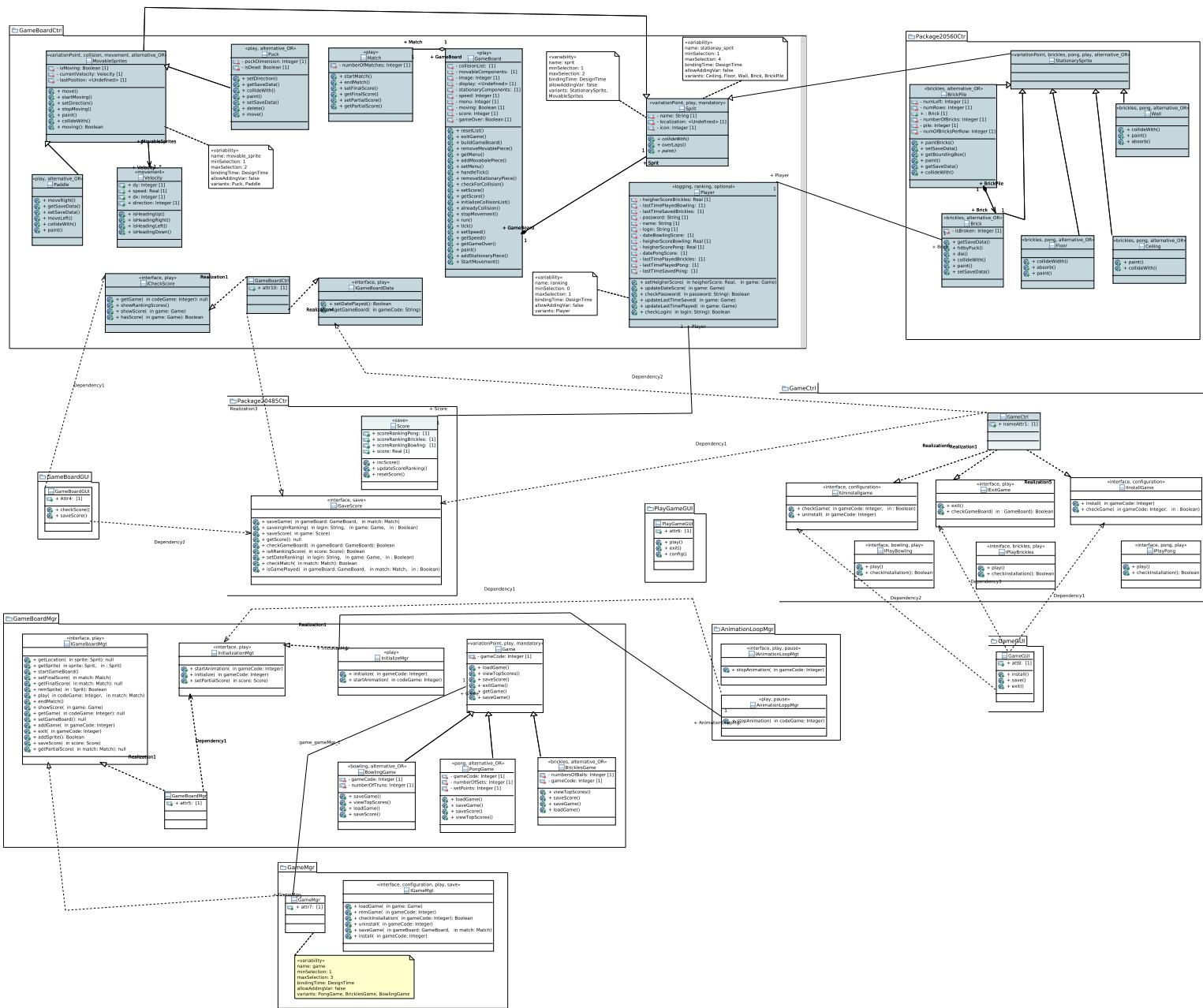


Figura 24: AGM com melhor ACLASS (CHOMA NETO, 2017)

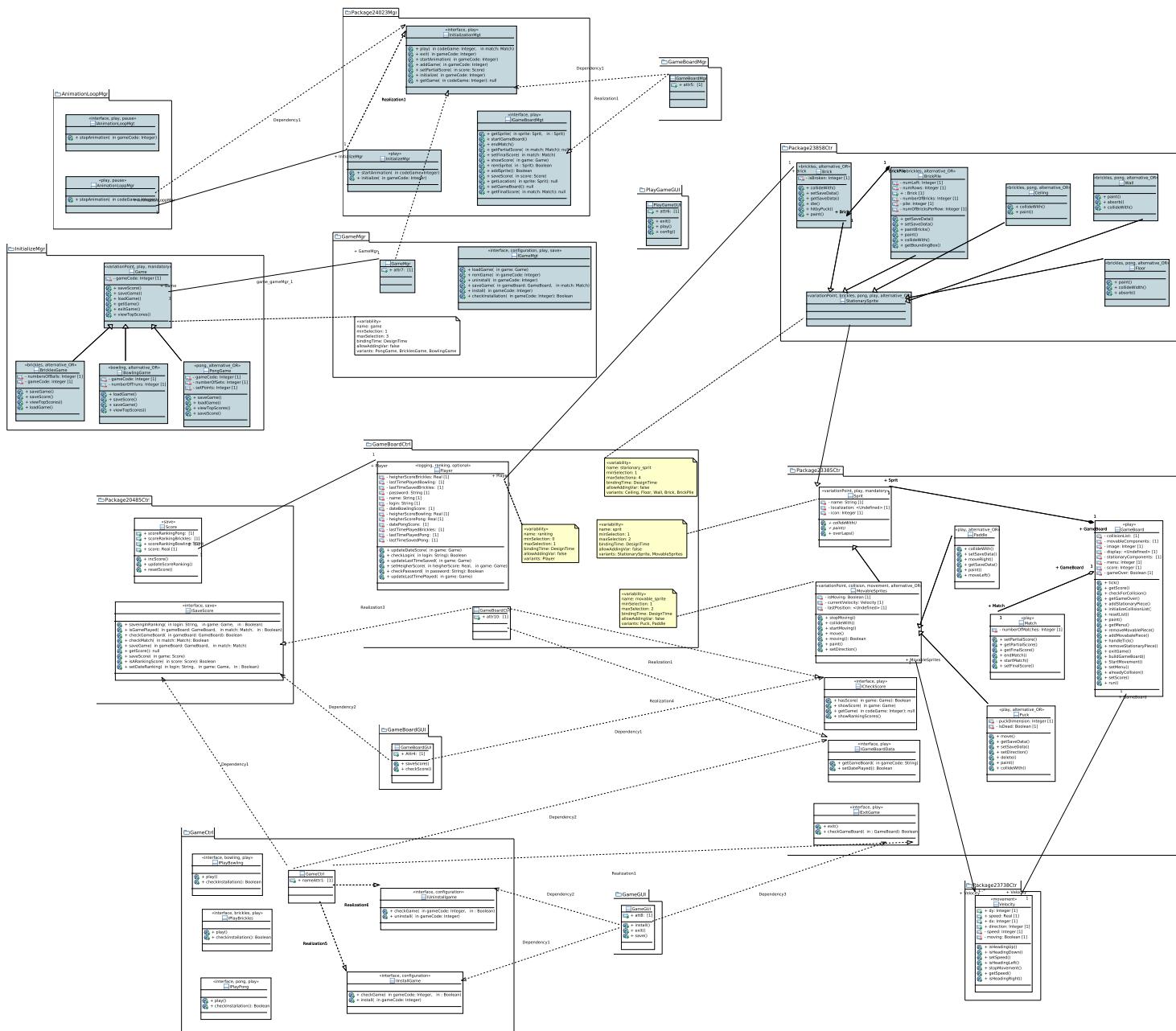


Figura 25: Banking original (CHOMA NETO, 2017)

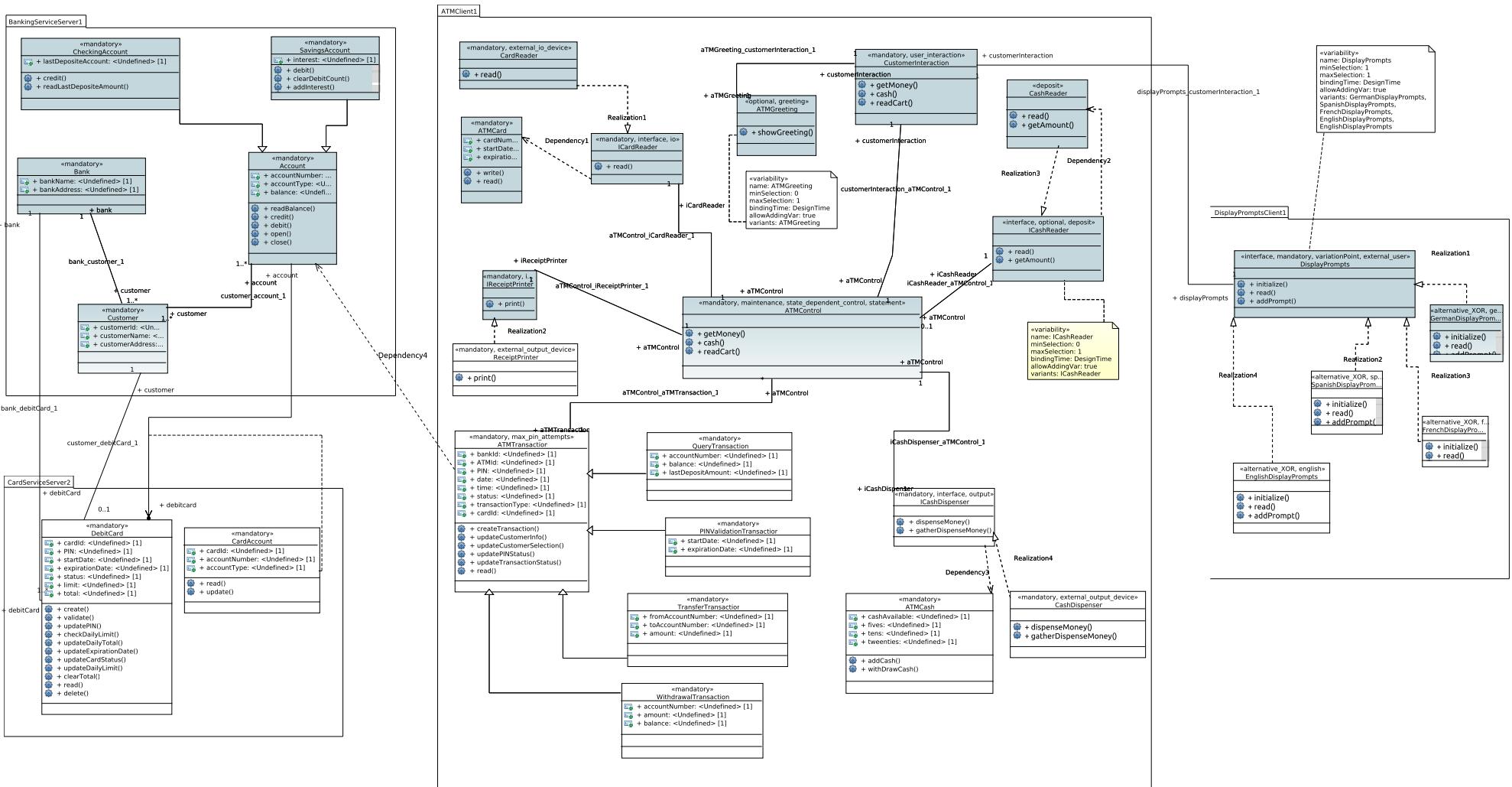


Figura 26: Banking com melhor ACLASS (CHOMA NETO, 2017)

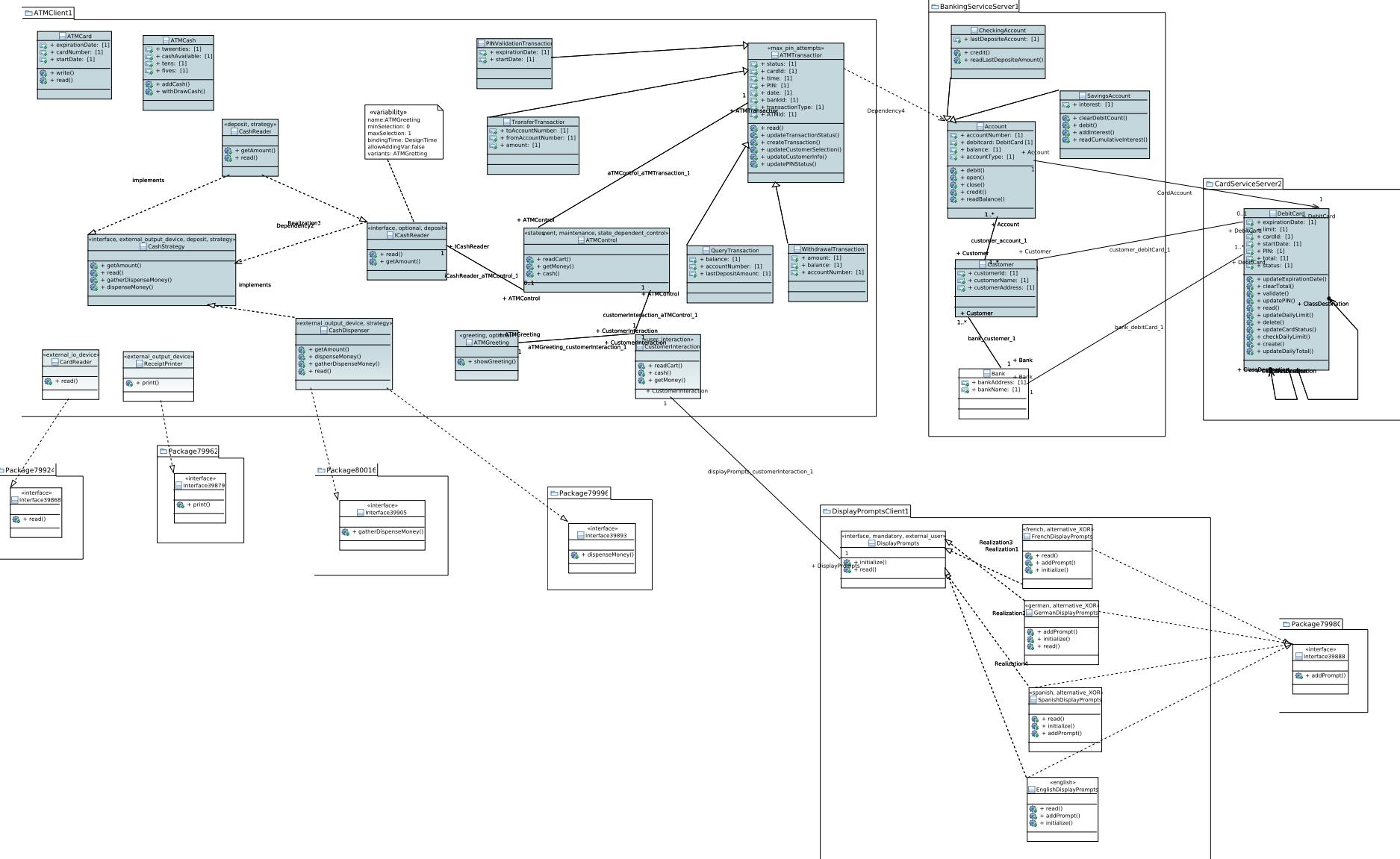


Figura 27: Banking com melhor COE e melhor ED (CHOMA NETO, 2017)

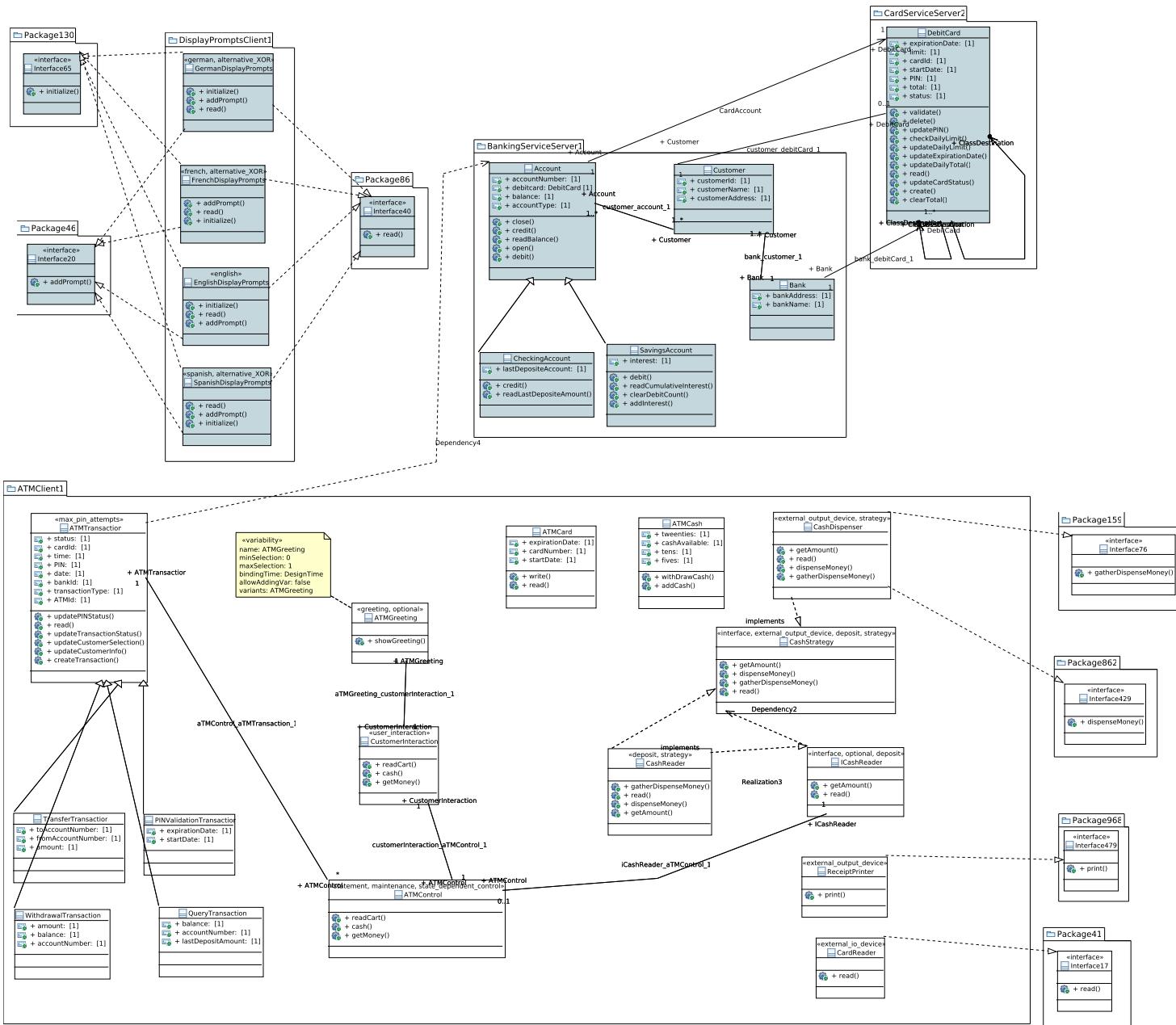


Figura 28: MM original (CHOMA NETO, 2017)

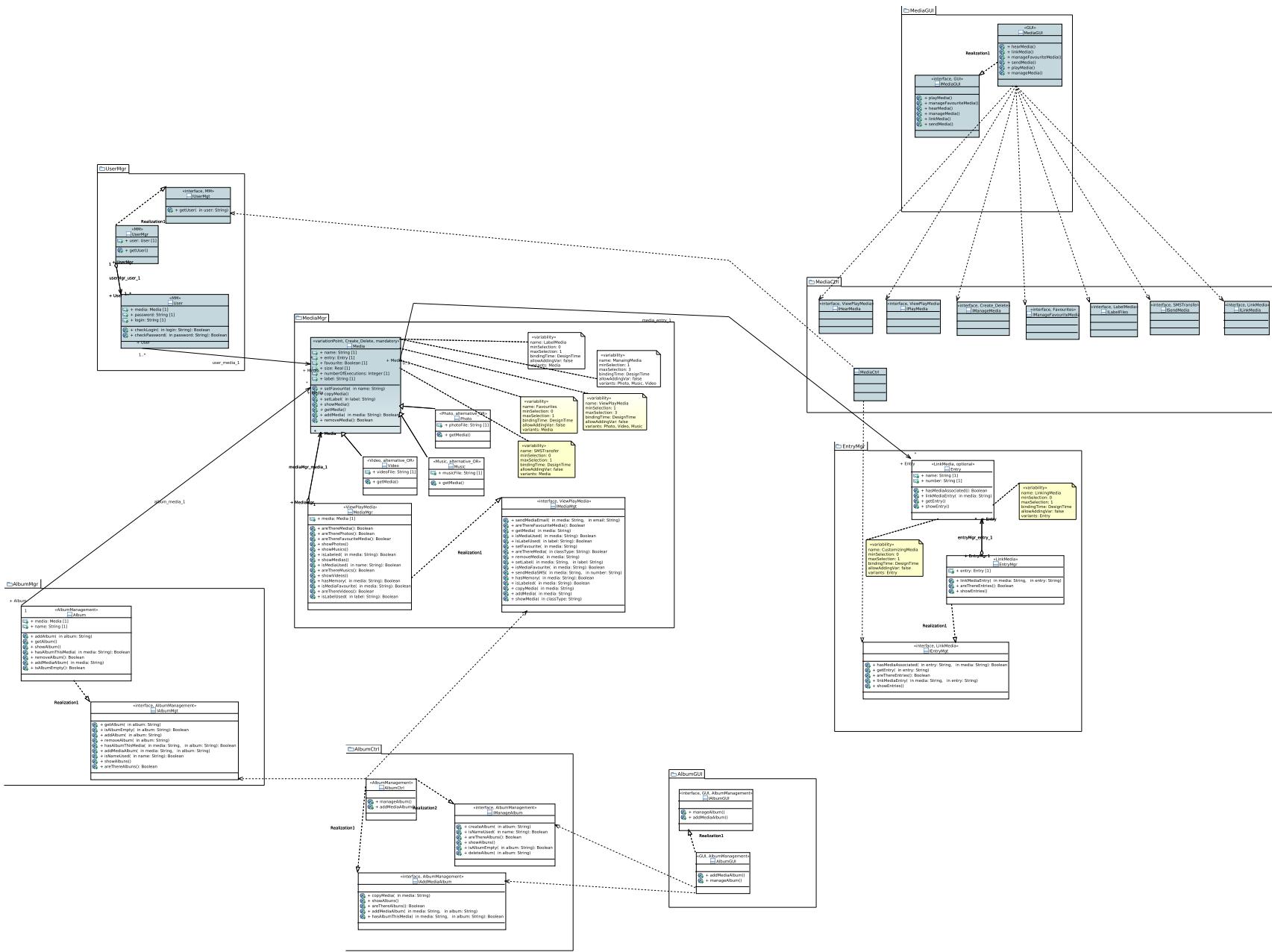


Figura 29: MM com Padrão de Projeto Strategy (CHOMA NETO, 2017)

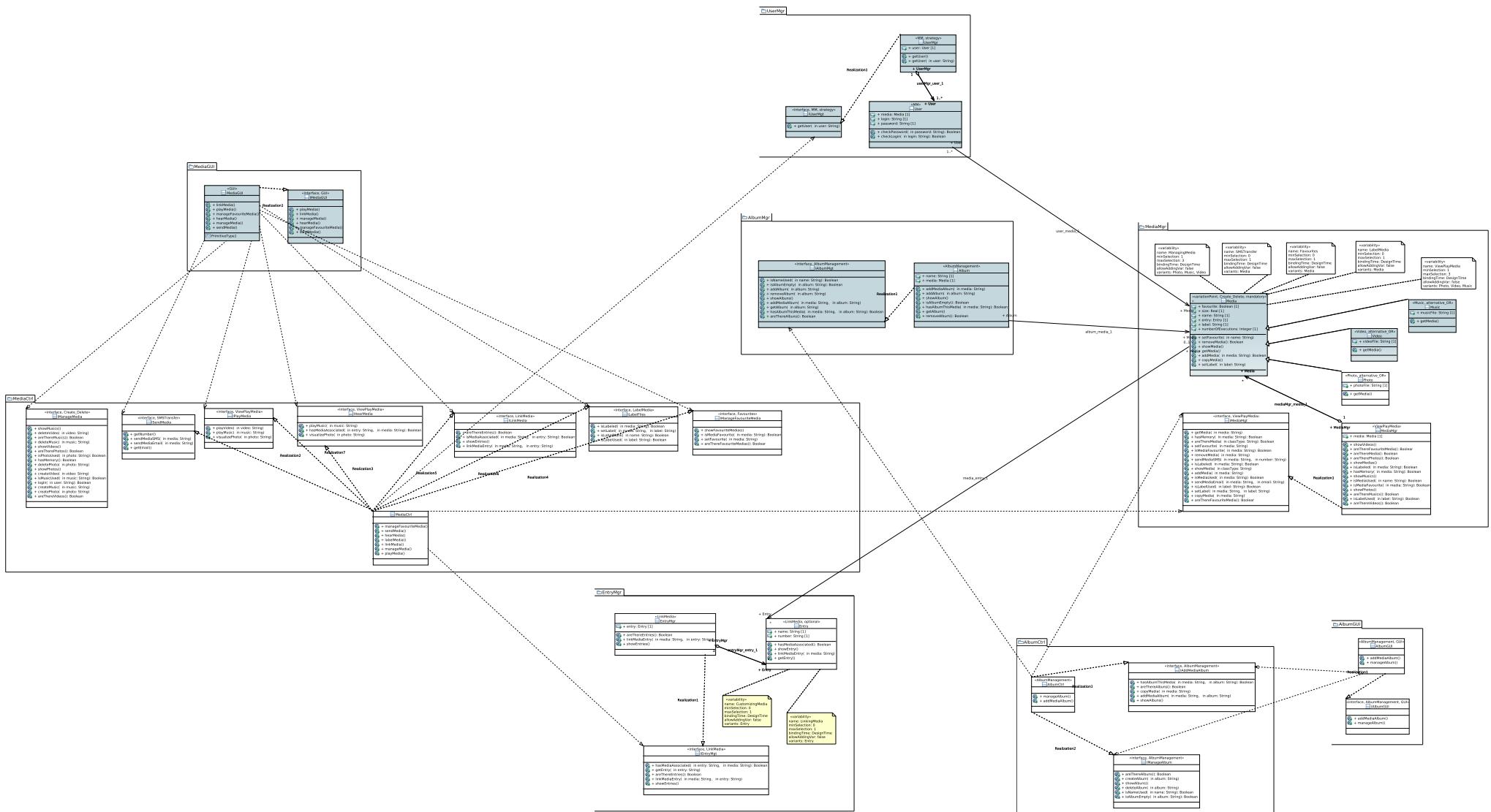


Figura 30: MM com melhor COE (CHOMA NETO, 2017)

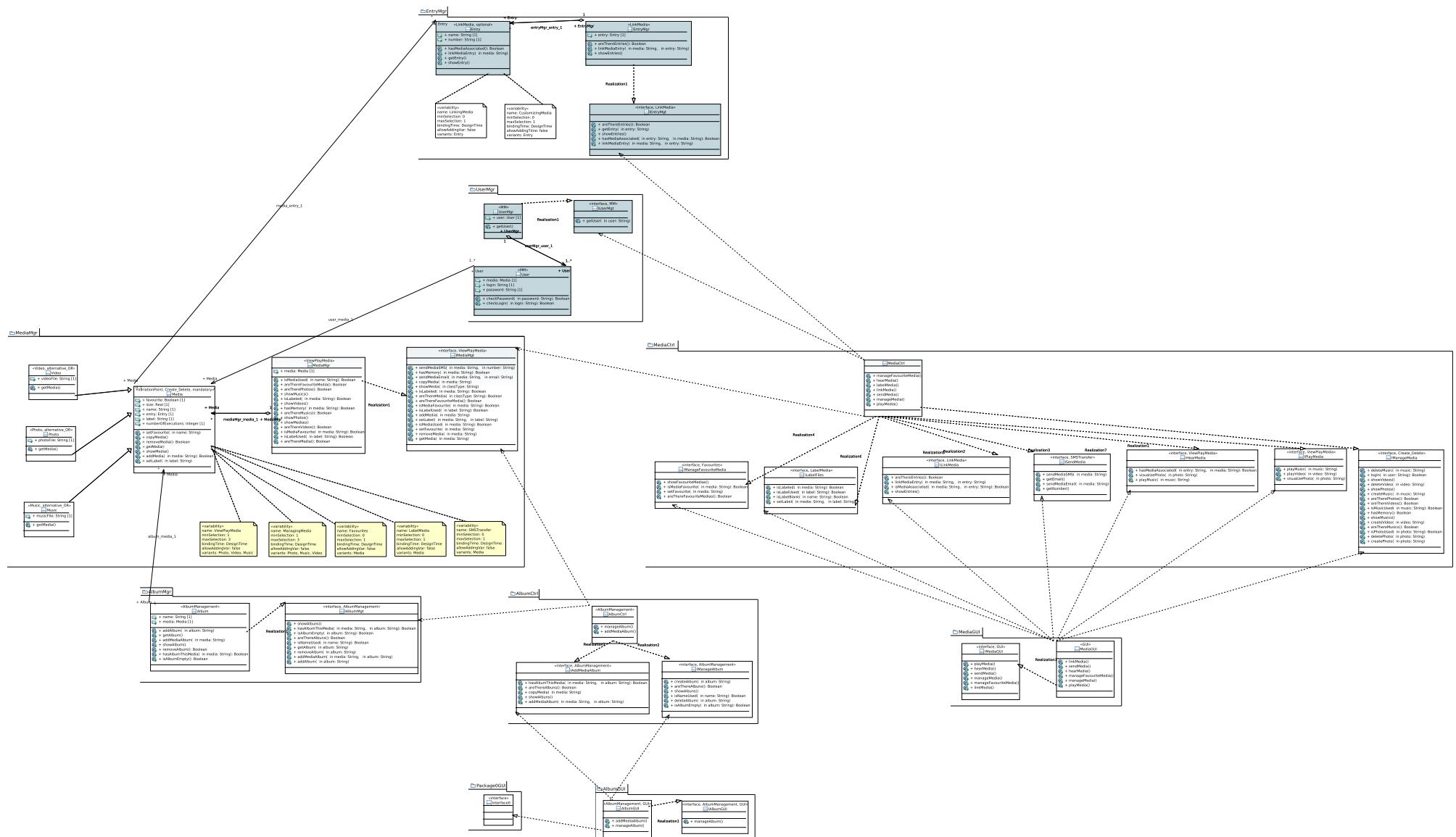


Figura 31: MM com melhor ACLASS e melhor ED (CHOMA NETO, 2017)

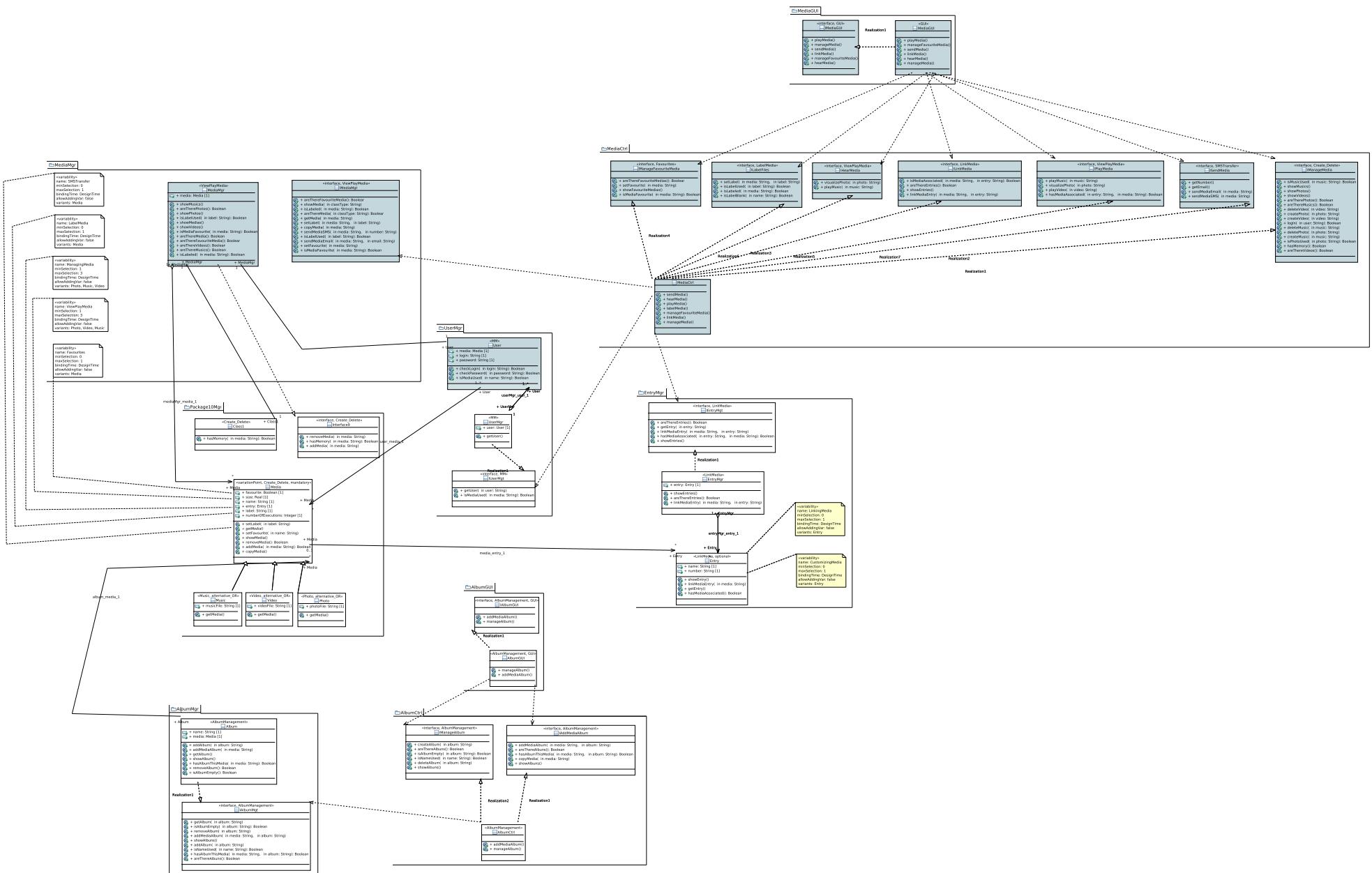


Figura 32: Banking original (LAVEZZO, 2017)

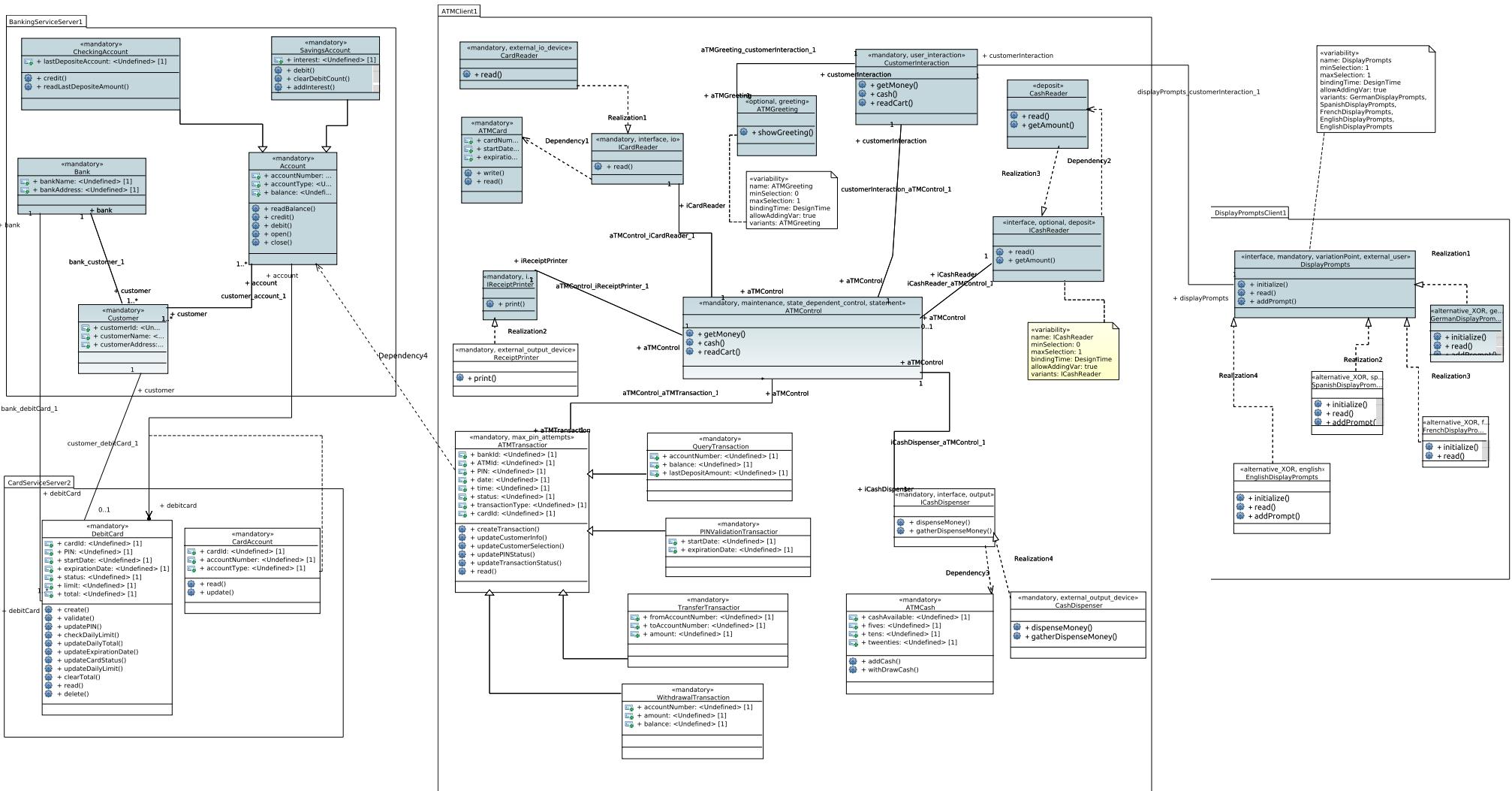


Figura 33: Banking gerada pela OPLA-Tool (LAVEZZO, 2017)

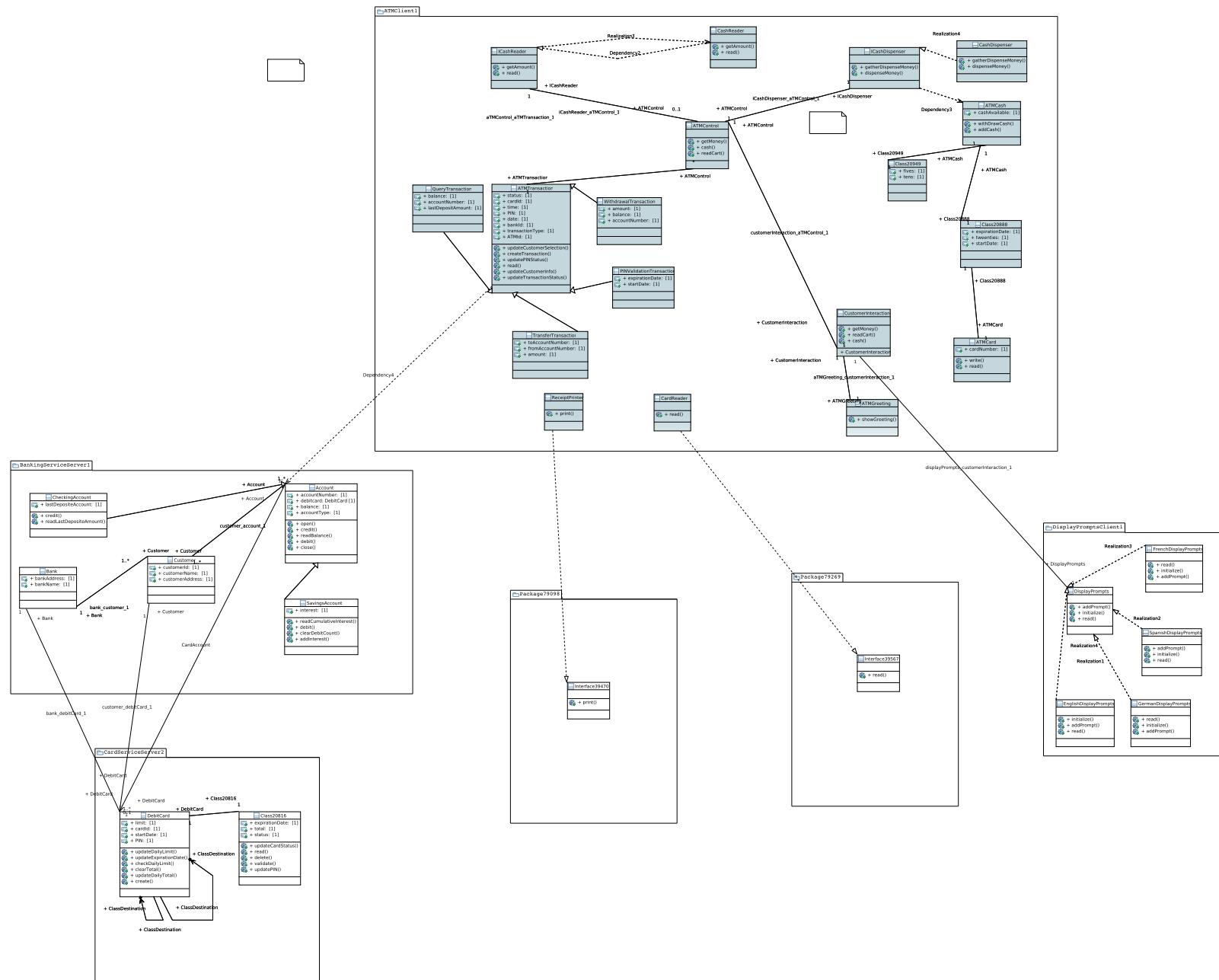


Figura 34: MM original (LAVEZZO, 2017)

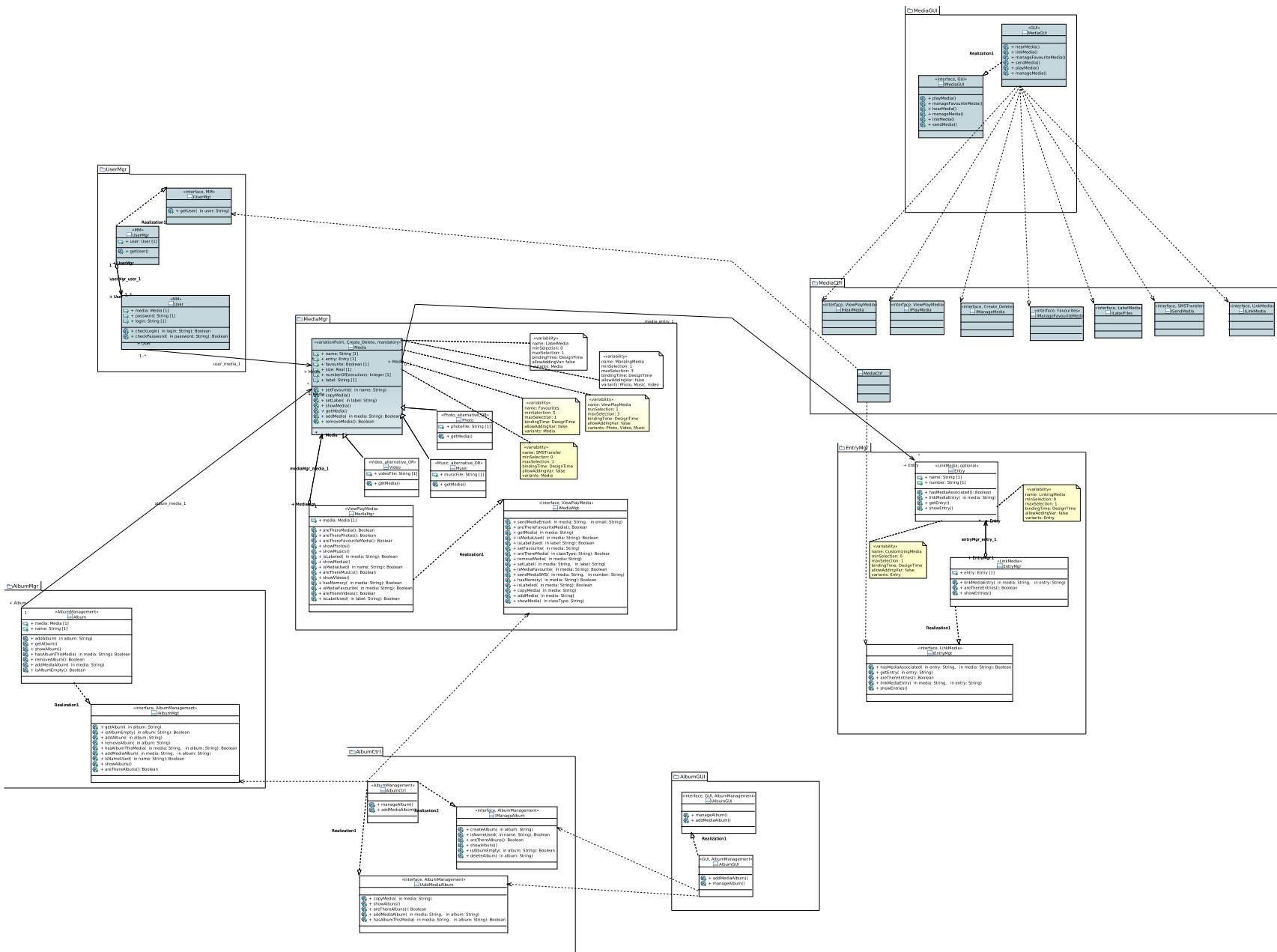


Figura 35: MM com melhor DC (LAVEZZO, 2017)

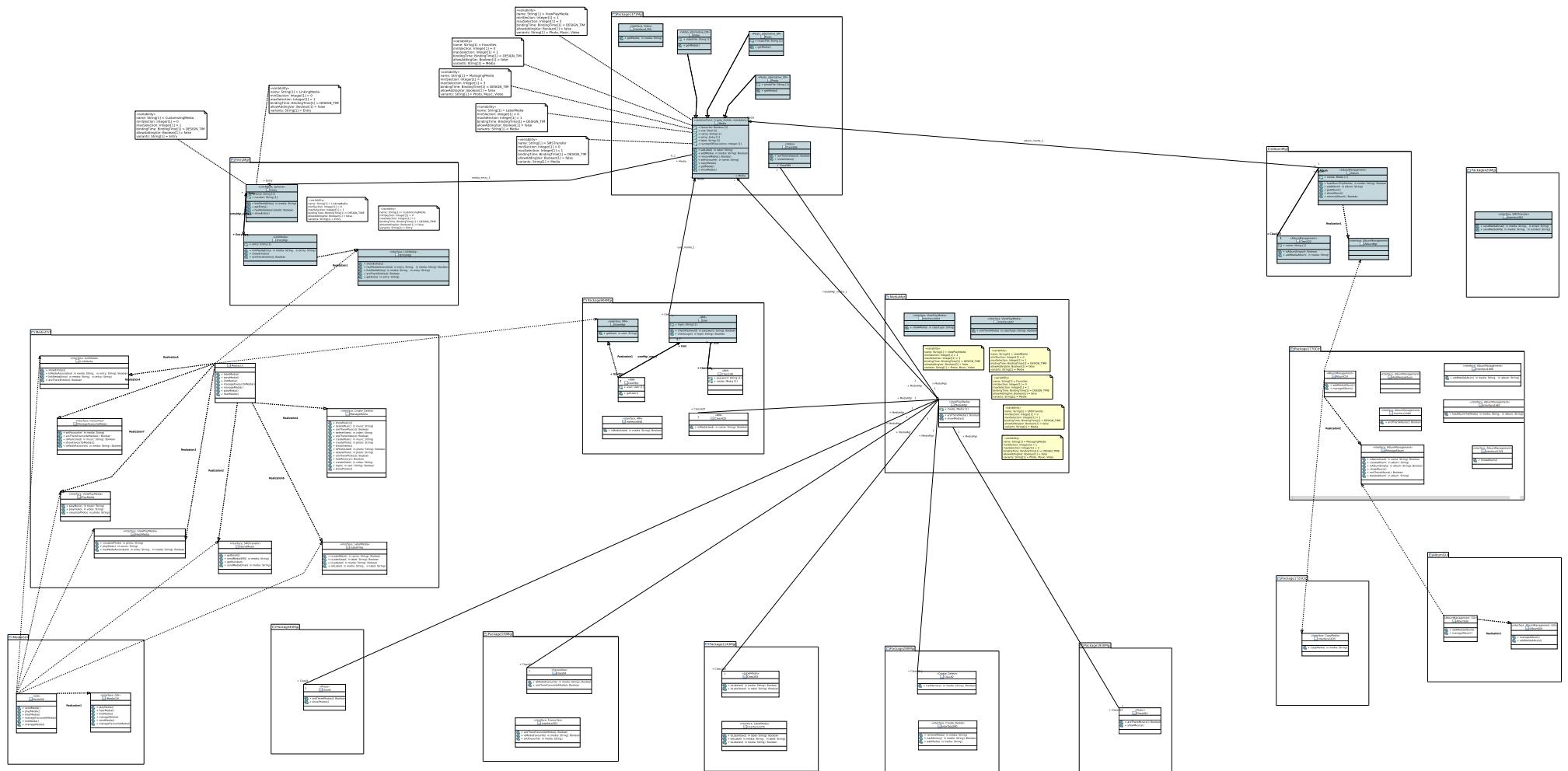


Figura 36: MM com melhor ED (LAVEZZO, 2017)

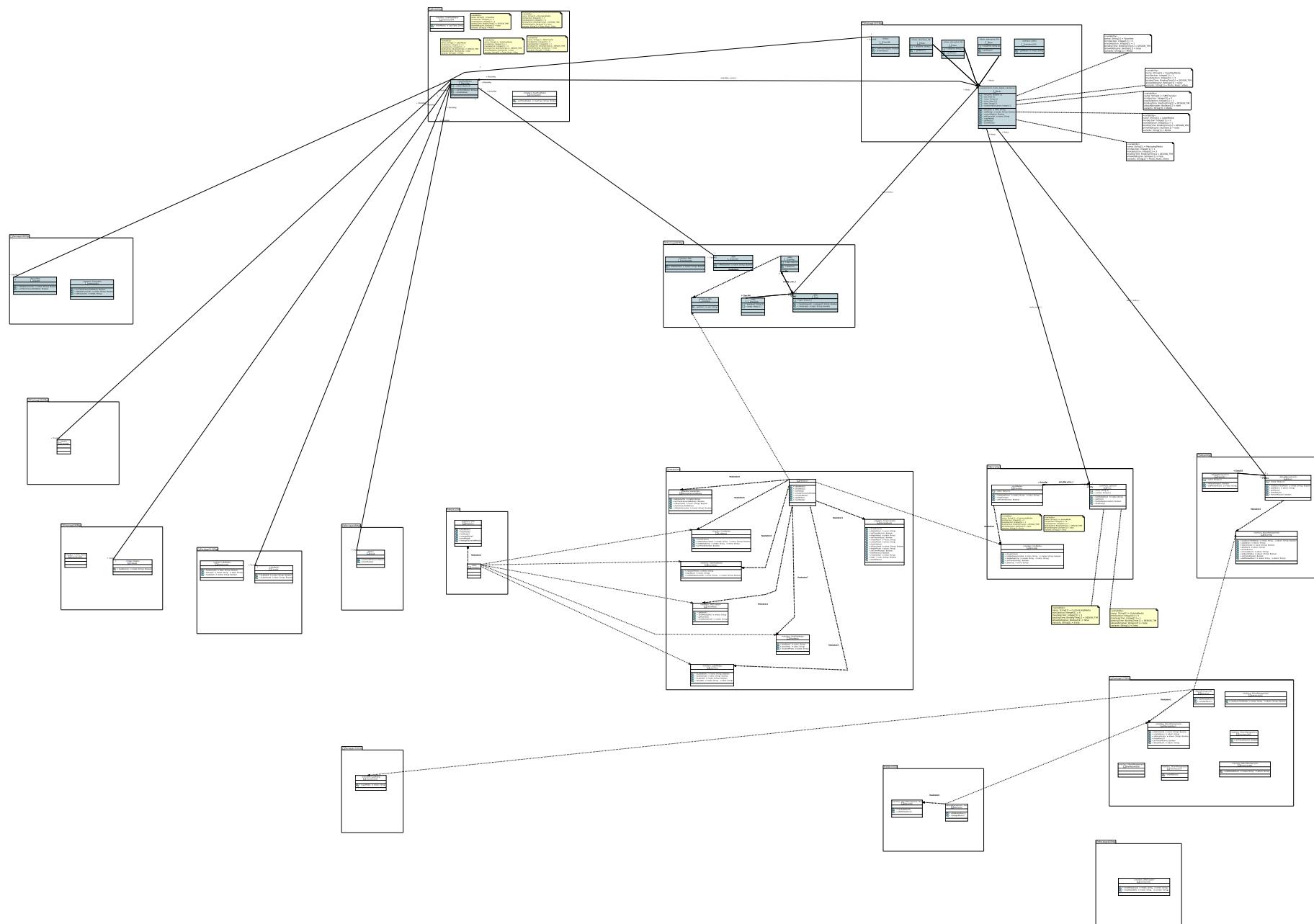


Figura 37: MM com melhor Eleg (LAVEZZO, 2017)

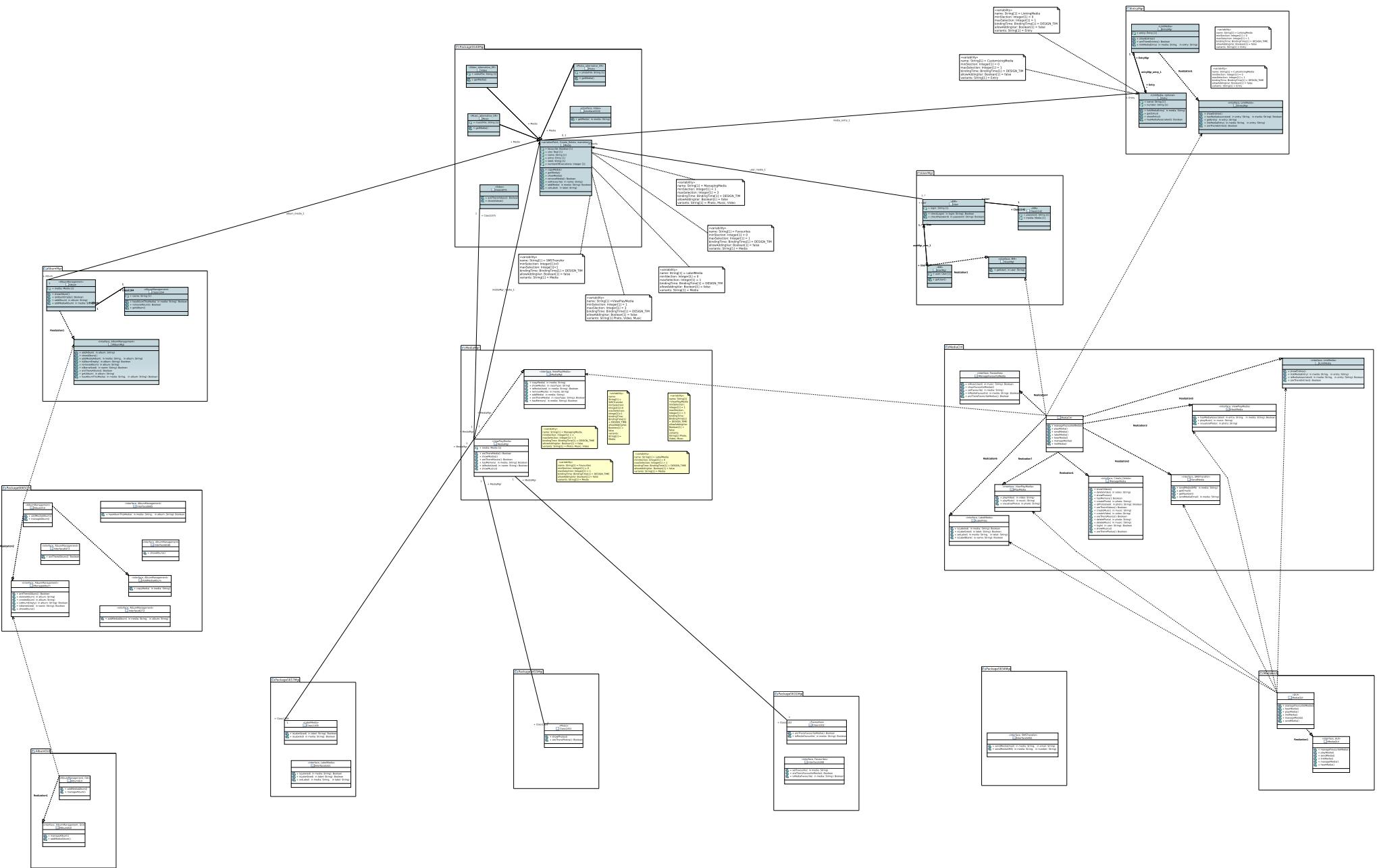


Figura 38: AGM original (DIAS, 2016)

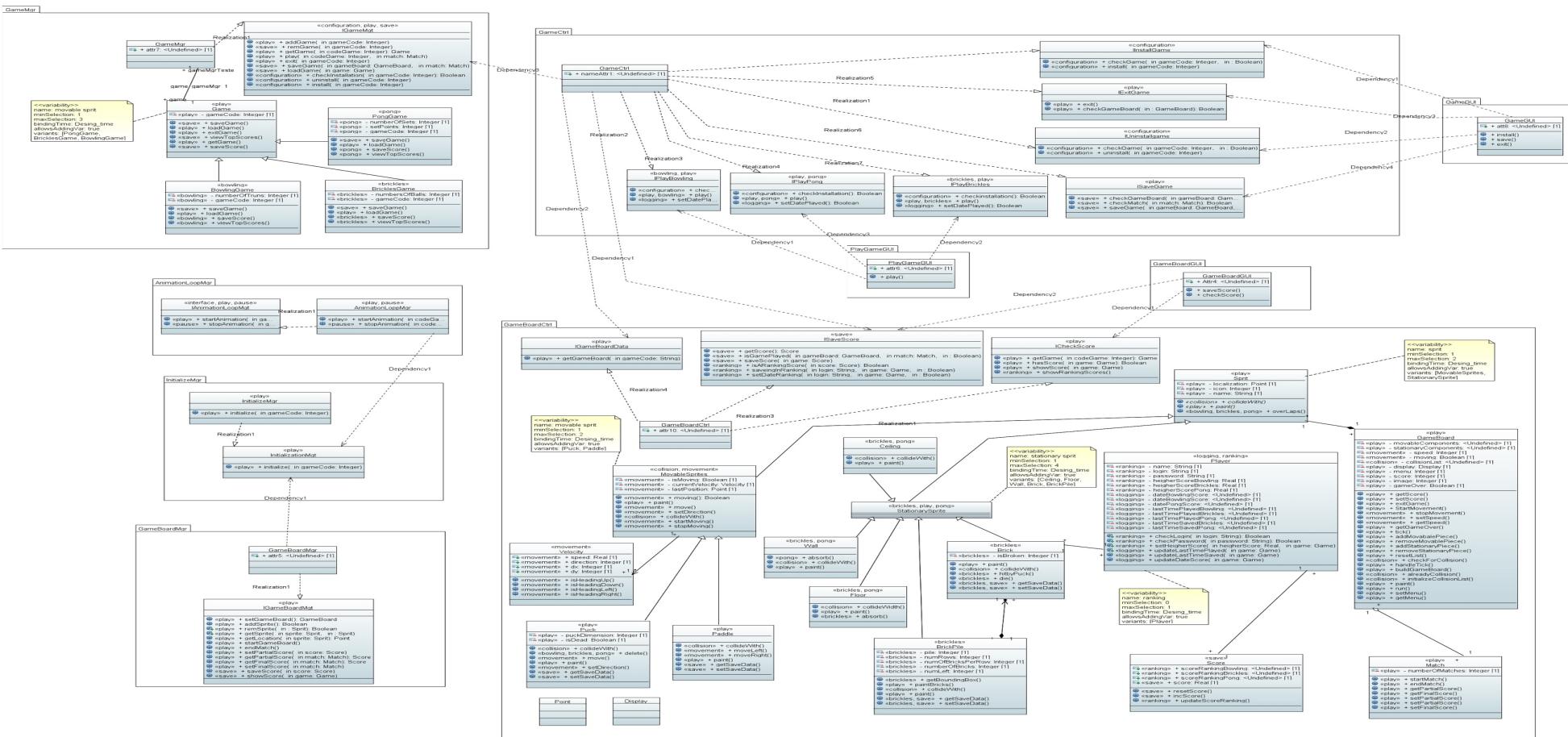


Figura 39: AGM com melhor ED e melhor FM (DIAS, 2016)

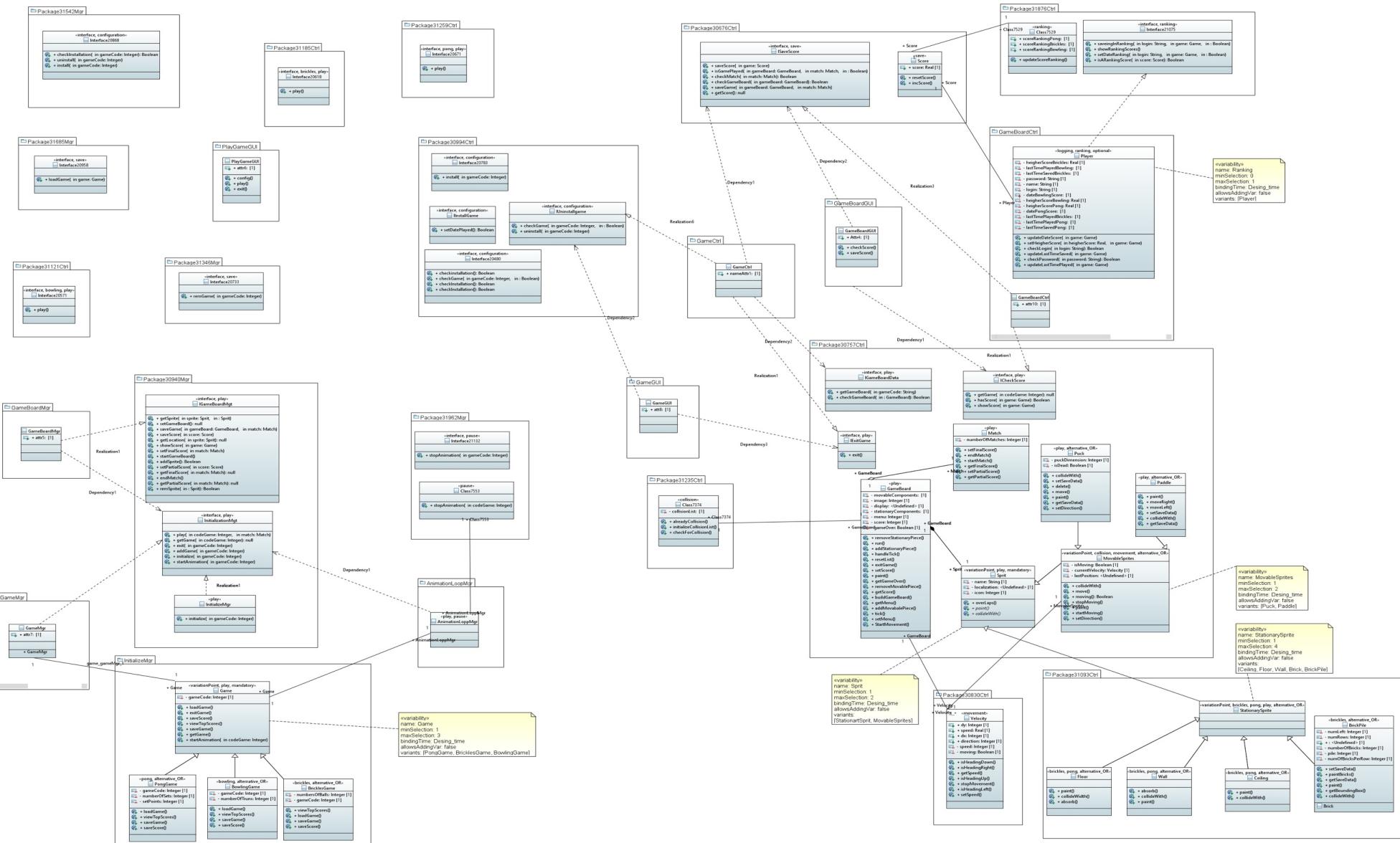


Figura 40: AGM com melhor CM (DIAS, 2016)

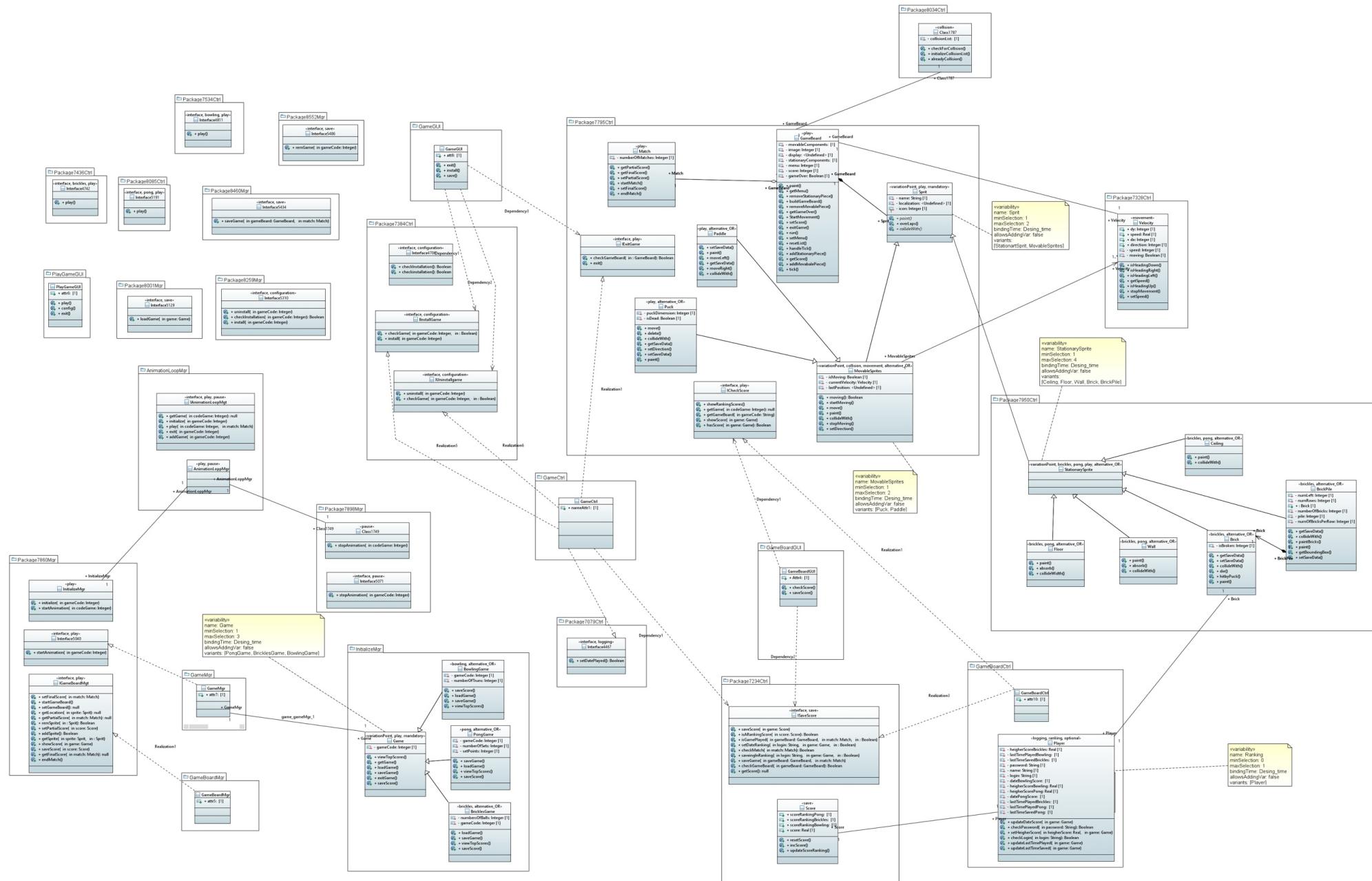


Figura 41: MM original (DIAS, 2016)

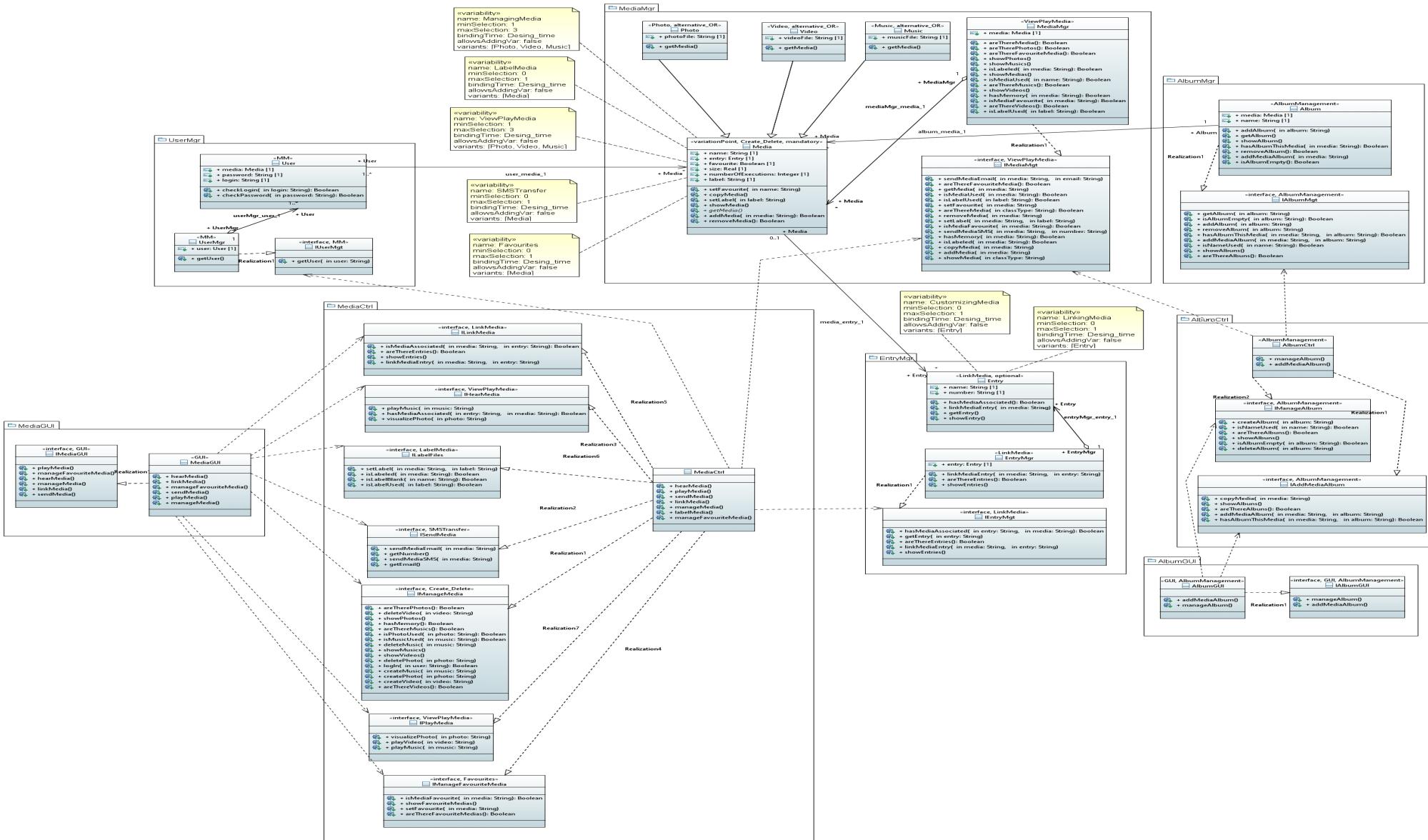


Figura 42: MM com melhor ED (DIAS, 2016)

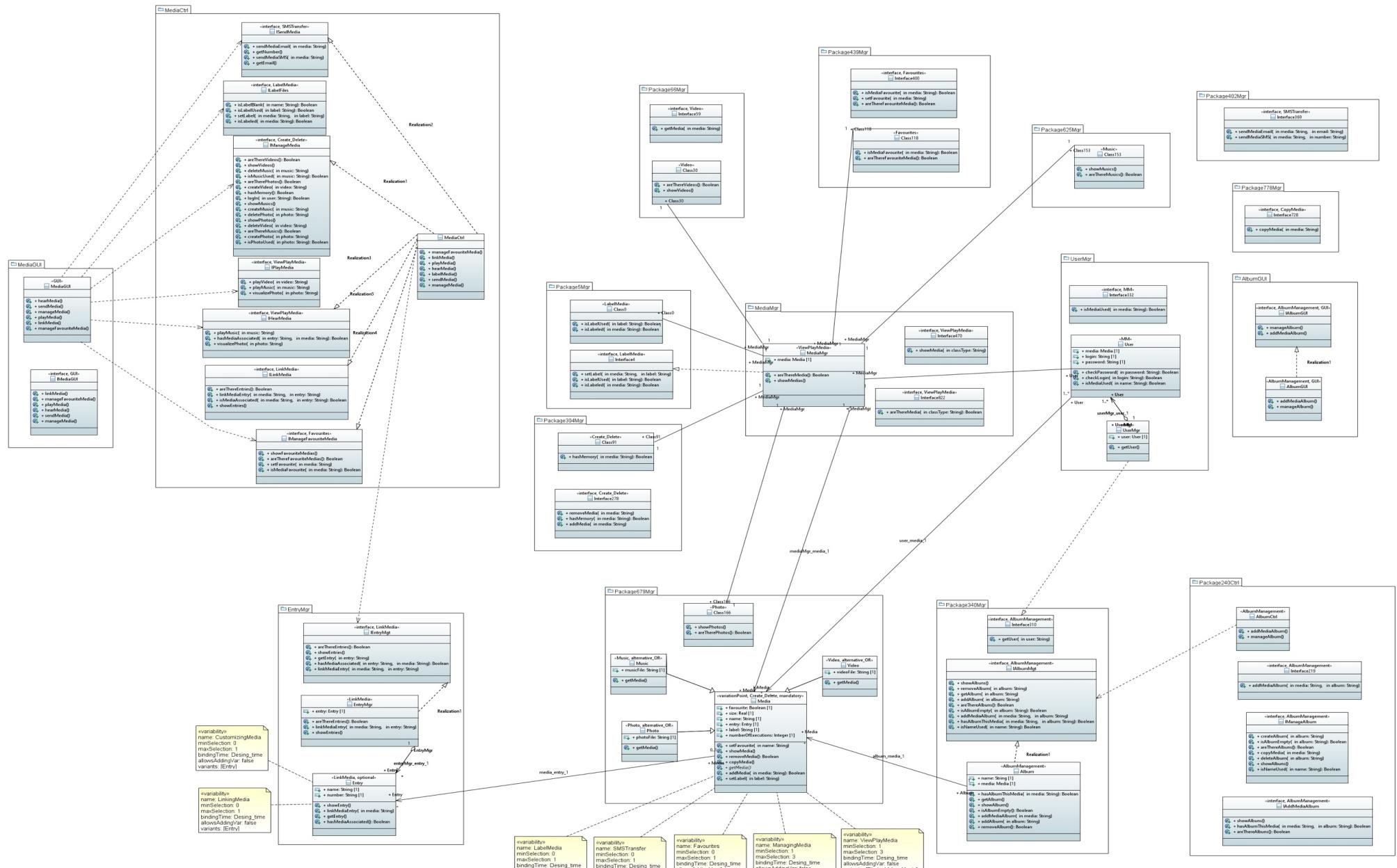


Figura 43: MM com melhor CM (DIAS, 2016)

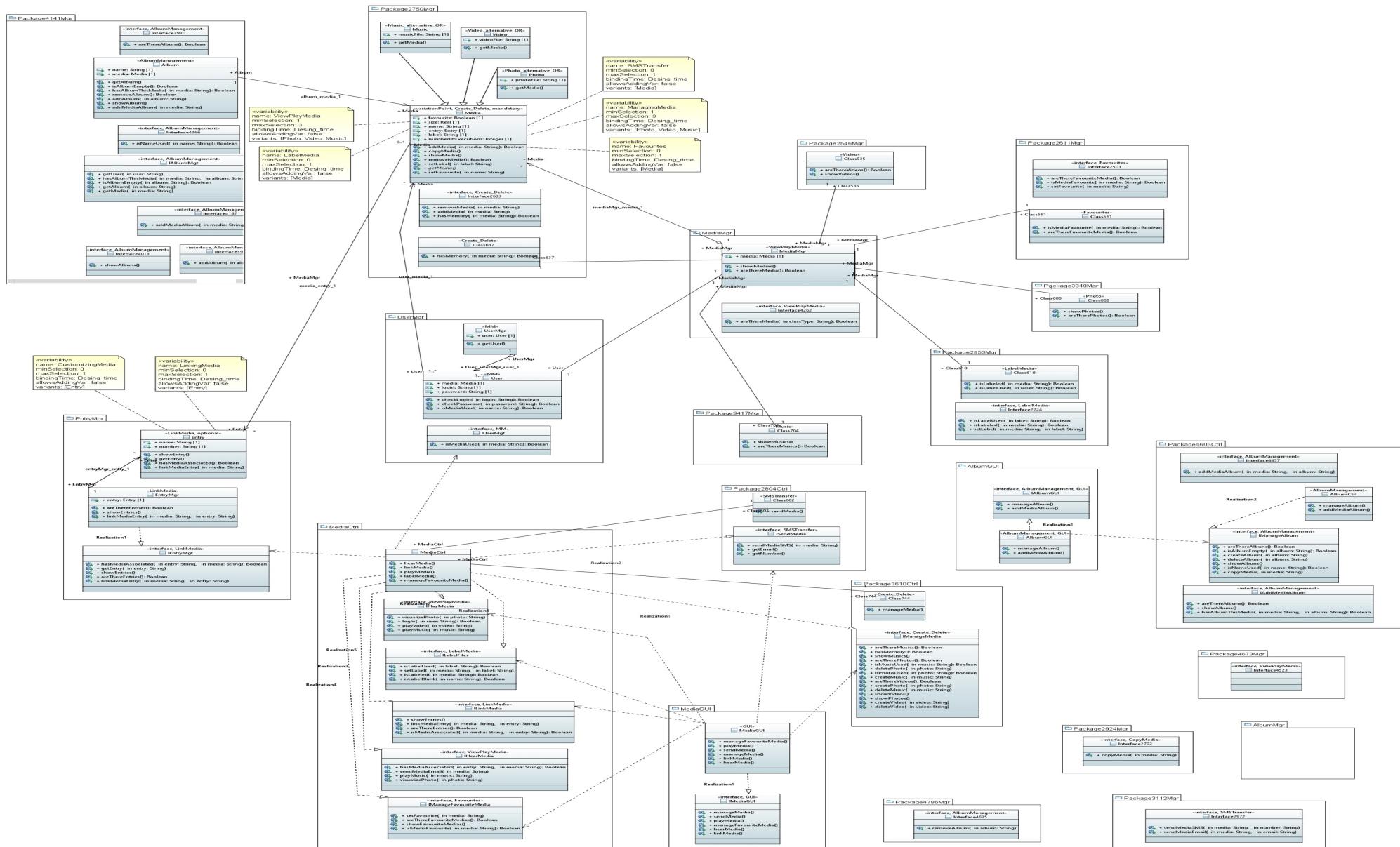


Figura 44: MM com melhor FM (DIAS, 2016)

