ANÁLISE DE PONTOS DE FUNÇÃO EM SISTEMAS DESENVOLVIDOS USANDO MDA

Roque Elias Assumpção Pinel

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientadores: Geraldo Zimbrão da Silva
Rodrigo Salvador Monteiro

Rio de Janeiro
Junho de 2012

ANÁLISE DE PONTOS DE FUNÇÃO EM SISTEMAS DESENVOLVIDOS USANDO MDA

Roque Elias Assumpção Pinel

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

_____

Prof. Geraldo Zimbrão da Silva, D.Sc.

_____

Prof. Rodrigo Salvador Monteiro, D.Sc.

_____

Prof. Geraldo Bonorino Xexéo, D.Sc.

_____

Prof. Leonardo Gresta Paulino Murta D.Sc.

RIO DE JANEIRO, RJ - BRASIL

JUNHO DE 2012

Pinel, Roque Elias Assumpção

Análise de Pontos de Função em Sistemas Desenvolvidos Usando MDA / Roque Elias Assumpção Pinel. – Rio de Janeiro: UFRJ/COPPE, 2012.

XIII, 85 p.: il.; 29,7 cm.

Orientadores: Geraldo Zimbrão da Silva

Rodrigo Salvador Monteiro

Dissertação (mestrado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2012.

Referências Bibliográficas: p. 64-68.

1. Análise de Pontos de Função. 2. MDA. I. Zimbrão, Geraldo et al. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Agradecimentos

Aos meus pais, Roque Elias e Maritza, pelo dom da vida.

Aos meus avós, Roque e Irene pelo carinho, por me ensinarem os valores da vida e por me proporcionarem a oportunidade estudar.

Aos meus tios, Maria Izabel, Julia e Luiz Eduardo por todo o apoio ao longo desses anos, sem o qual eu não estaria aqui.

À Carol, minha noiva, por ser uma parte especial da minha vida, me compreendendo e principalmente me motivando nos momentos de desânimo.

Ao meu orientador Zimbrão, pelo conhecimento transmitido e conselhos que foram fundamentais para a elaboração deste trabalho.

Ao meu orientador Rodrigo, por sua presença em minha formação acadêmica e profissional. Pela paciência e apoio durante o desenvolvimento deste trabalho.

Aos professores Xexéo e Murta, por aceitarem o convite de participação da banca e contribuírem com este trabalho.

Ao CNPq e à FAPERJ, por apoiarem este trabalho.

Aos meus grandes amigos, que juntos caminharam comigo ao longo dessa jornada.

Aos amigos da Fundação COPPETEC, que apoiaram a realização deste trabalho.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

ANÁLISE DE PONTOS DE FUNÇÃO EM SISTEMAS DESENVOLVIDOS USANDO MDA

Roque Elias Assumpção Pinel

Junho/2012

Orientadores: Geraldo Zimbrão da Silva
Rodrigo Salvador Monteiro

Programa: Engenharia de Sistemas e Computação

Medição de software é tarefa crucial para o planejamento e desenvolvimento de sistemas de informação. A Análise de Pontos de Função (APF) foi desenvolvida para medir a complexidade das funcionalidades desses sistemas de informação. Seus métodos são independentes de tecnologia e podem ser aplicados diretamente na documentação de funcionalidades e de domínio. Contudo, a contagem deve ser realizada por um Analista de Métricas, sendo influenciada por conceitos subjetivos e representando grande consumidor de recursos. Este trabalho descreve a automatização da contagem de pontos de função utilizando modelos UML e a metodologia MDA. Nossa abordagem provê um método padrão de contagem com base no IFPUG, eliminando a subjetividade existente nos procedimentos tradicionais.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

FUNCTION POINT ANALYSIS ON SYSTEMS DEVELOPED USING MDA

Roque Elias Assumpção Pinel

June/2012

Advisors: Geraldo Zimbrão da Silva
Rodrigo Salvador Monteiro

Department: Systems and Computer Engineering

Software measurement is a crucial task for the planning and the developing of information systems. The Function Point Analysis (FPA) was developed to measure the complexity of the functionality of these systems. Its methods are independent of technology and can be applied directly to the specification of features and the domain. However, the counting should be performed by a Metrics Analyst, being under subjectivity and consuming time and large amount of resources. This work describes the proposal of automation of function point counting performed using UML models and the MDA methodology. Our approach provides a standard method of counting based on IFPUG, eliminating the subjectivity present in traditional procedures.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

AIE	Arquivo de *Interface* Externa
ALI	Arquivo Lógico Interno
APF	Análise de Pontos de Função
AR	Arquivo Referenciado
CE	Consulta Externa
CIM	*Computation Independent Model*
COSMIC	*Common Software Measurement International Consortium*
DET	*Data Element Type* (em português, TD)
DF	*Data Function* (em português, FD)
EE	Entrada Externa
EI	*External Input* (em português, EE)
EIF	*External Interface File* (em português, ALI)
EO	*External Output* (em português, SE)
EQ	*External Inquiry* (em português, CE)
FD	Função de Dados
FPA	*Function Point Analysis* (em português, APF)
FT	Função de Transação
FTR	*File Type Referenced* (em português, AR)
IEC	*International Engineering Consortium*
IFPUG	*International Function Point Users Group*
ILF	*Internal Logical File* (em português, ALI)
ISO	*International Organization for Standardization*
MDA	*Model Driven Architecture*
MOF	*Meta Object Facility*
OCL	*Object Constraint Language*
OMG	*Object Management Group*
PF	Ponto(s) de Função
PIM	*Platform Independent Model*
PSM	*Platform Specific Model*
RET	*Record Element Type* (em português, TR)
SE	Saída Externa

SLOC	*Source Lines of Code*
TD	Tipo de Dado
TF	*Transaction Function* (em português, FT)
TR	Tipo de Registro
UML	*Unified Modeling Language*
VAF	*Value Adjustment Factor*
XML	*Extensible Markup Language*

Capítulo 1

1 Introdução

A Análise de Pontos de Função (APF) (da sigla em inglês FPA, *Function Point Analysis*) foi definida em 1979 como um procedimento capaz de medir funcionalidades e complexidade de sistemas de informação (ALBRECHT, 1979). Sua aplicação faz uso de especificações de sistemas de informação, sendo dessa forma independente de tecnologia, em contraste a abordagem tradicional de linhas de código-fonte (da sigla em inglês SLOC, *Source Lines of Code*) (ABRAN, 2010). Desde seu surgimento, a APF tem sido mundialmente utilizada para medir o tamanho de aplicações e fornecer estimativas de custo de projetos (GARMUS, HERRON, 2000), com base em contagens já realizadas, possibilitando entregar aos clientes uma medida das funções lógicas do sistema de informação.

Ao ser proposta por ALBRECHT (1979), a APF não determinava uma metodologia de como o processo de contagem deveria ser realizado. Ao longo dos anos, diferentes grupos foram criados e diferentes padrões de contagem produzidos, *e.g.*, IFPUG (do inglês *International Function Point Users Group*) (IFPUG, 2010) e COSMIC (do inglês *Common Software Measurement International Consortium*) (COSMIC, 2009). Apesar do esforço no estabelecimento de métodos a serem seguidos em contagens, a análise continua a ser realizada pelo Analista de Métricas e dessa forma continua influenciada pela subjetividade do mesmo, visto que o Analista de Métricas é responsável por estudar os documentos do sistema de informação e realizar a contagem a partir dessas informações (ABRAN, 2010). Além disso, esse processo é conhecido por requerer horas de trabalho afinco e dedicação, sendo grande consumidor de recursos.

Com a formulação do padrão de modelagem UML (do inglês *Unified Modeling Language*) (BOOCH, RUMBAUGH, JACOBSON, 1998) grande parte dos sistemas de informação passou a utilizar a linguagem em suas especificações. Por ser uma representação gráfica padrão e independente de linguagem de programação, a UML melhora o canal de comunicação entre as partes envolvidas no desenvolvimento, inclusive com clientes. Hoje, documentos UML são fontes do Analista de Métricas durante a contagem de pontos de funções, já que APF não dependem da tecnologia utilizada no sistema de informação e pode ser realizada diretamente pelas especificações

do mesmo. Contudo, apesar de útil, a análise de documentos continua a ser um incomodo e grande consumidor de tempo.

Em 2001, a OMG (do inglês *Object Management Group*) (OMG, 2012) lançou um guia de definições sobre a geração de código com base em modelos, a Arquitetura Orientada à Modelos (da sigla em inglês MDA, *Model Driven Architecture*) (SIEGEL *et al.*, 2001). Essa metodologia utiliza, dentre outros padrões, a UML como linguagem de modelagem. Seus métodos permitem automatizar o ciclo de vida de projetos baseado em modelos UML, reduzindo o tempo de desenvolvimento e permitindo a padronização do código fonte. Nesse contexto, a abordagem MDA provê um ambiente adequado para a automatização da APF, possibilitando o estabelecimento de critérios objetivos para execução da contagem de pontos de função e reduzindo a subjetividade inerente ao processo manual tradicional.

Para tanto, este trabalho explora o uso do *framework* MDArte (MDARTE, 2012), que tem como base o *framework* AndroMDA (ANDROMDA, 2012). O MDArte é um *framework* de código aberto composto por cartuchos personalizáveis que especificam a geração de código para diferentes tecnologias, com base em modelos UML. Dessa forma, ele permite automatizar a APF, extraindo valores de modelos e gerando artefatos úteis na contagem. Artefatos como a classificação de elementos do sistema de informação gerado (e.g. Entidades, Serviços e Casos de Uso), dentre outras informações que são importantes para o Analista de Métricas. Nossa escolha está fundamentada no amadurecimento da ferramenta (PINEL *et al.*, 2011) e no número de sistemas de informações gerados pelo *framework* que estão em produção (MDARTE, 2011). Além disso, ele nos permite ter a acesso a um conjunto grande de sistemas que podem ser diretamente beneficiados pelo trabalho.

Este trabalho visa contar os pontos de função do que já foi realizado, *i.e.*, em sistemas de informação já desenvolvidos ou quase finalizados. Apesar da contagem ser aplicada a projetos já desenvolvidos, os resultados produzidos podem ser utilizados para ajustar e melhorar a precisão das regras da APF. Regras aplicadas para contar sistemas de informação ainda em desenvolvimento. A contagem dos pontos de função realizados é útil não só para comparações com as estimativas iniciais, mas principalmente para identificar distorções e auxiliar no ajuste dos critérios a serem aplicados em estimativas futuras.

Este trabalho encontra-se organizado em sete capítulos. O Capítulo 2 apresentará os conceitos relacionados à APF e à metodologia MDA. O Capítulo 3 debaterá os trabalhos relacionados. O Capítulo 4 discutirá a proposta do trabalho. O Capítulo 5 abordará a implementação da proposta do trabalho. O Capítulo 6 apresentará um estudo de caso e discutirá os resultados obtidos. Por fim, o Capítulo 7 tratará das conclusões e sugestões para trabalhos futuros.

Capítulo 2

2 Fundamentação Teórica

Neste capítulo serão apresentados conceitos essenciais para melhor compreensão do trabalho. Primeiramente, a APF será descrita, abordando como o processo é realizado seguindo o método IFPUG (2010). Em seguida, a metodologia MDA e suas características serão apresentadas.

2.1 Análise de Pontos de Função

GARMUS e HERRON (2000) e VAZQUEZ *et al.* (2011) definem a APF como uma técnica de medição das funcionalidades fornecidas por um sistema de informação do ponto de vista do usuário. Ela estabelece uma medida própria, o ponto de função, com o objetivo de tornar a medição independente da tecnologia utilizada na construção do sistema de informação. Conforme proposta por ALBRECHT (1979), a APF conta as seguintes características do sistema: arquivos utilizados, entradas, saídas e consultas, sendo cada característica considerada individualmente e contada conforme o peso atribuído. A Figura 2.1 apresenta os elementos envolvidos no processo de contagem de pontos de função.

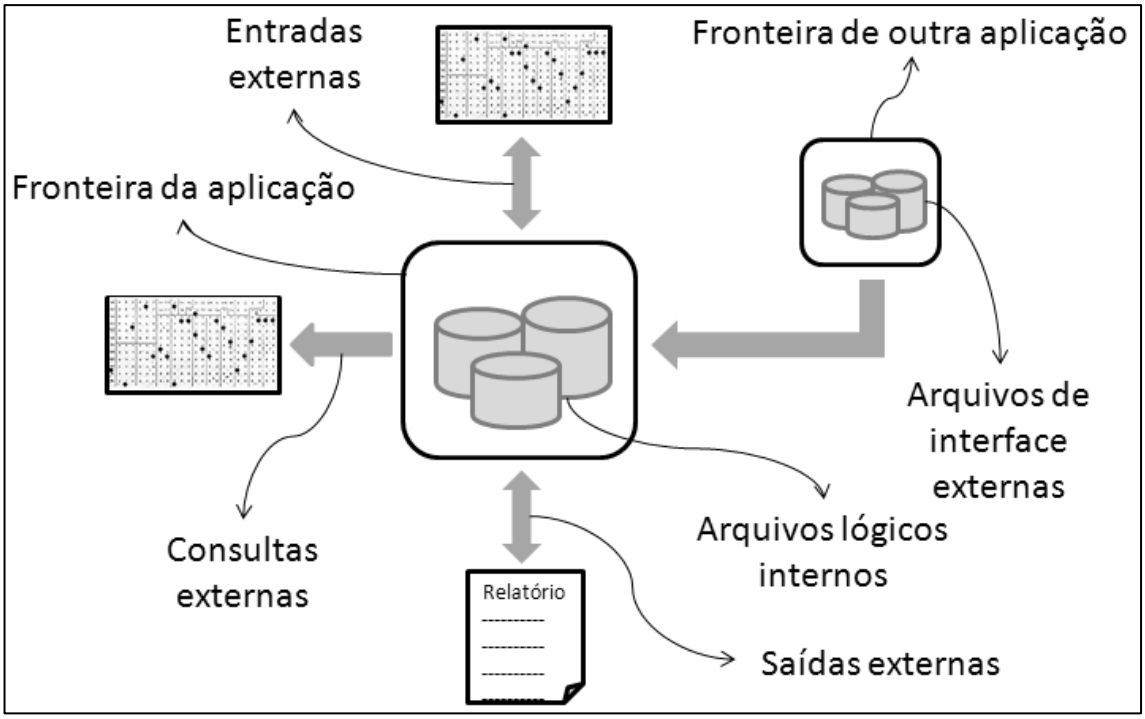

Figura 2.1 – Elementos da contagem de pontos de função (VAZQUEZ, SIMÕES, ALBERT, 2011).

Conforme descrito em VAZQUEZ *et al.* (2011), com o objetivo de resolver as inconsistências existentes entre os diferentes tipos de abordagens da metodologia proposta por ALBRECHT (1979) e estabelecer um método mais rigoroso de medição funcional, grupos de usuários de diversos países formaram um grupo de trabalho subordinado as diretrizes da ISO (do inglês *International Organization for Standardization*) (ISO, 2012) e do IEC (do inglês *International Engineering Consortium*) (IEC, 2012). Como resultado, foi estabelecido um conjunto de padrões internacionais chamado de norma ISO/IEC 14143 (ISO, 2012, IEC, 2012). Atualmente, cinco métodos de medição são reconhecidos pelo padrão, contudo este trabalho se restringe ao método adotado pelo IFPUG. A metodologia do IFPUG foi escolhida por ser largamente utilizada, tendo milhares de usuários (indivíduos, corporações, instituições de ensino), e por ser aplicada em mais de trinta países, incluindo o Brasil.

A versão proposta pelo IFPUG provê algumas modificações às regras originais. Ela é descrita em sete etapas, relacionadas a seguir, e pode ser realizada a partir de especificações do sistema de informação sendo contado (IFPUG, 2010).

1º) Determinar o tipo da contagem de pontos de função.

2º) Identificar a fronteira do sistema.

3º) Contar as Funções de Dados.

4º) Contar as Funções de Transações.

5º) Determinar o valor não ajustado de pontos de função.

6º) Determinar o fator de ajuste.

7º) Calcular o valor ajustado.

Neste trabalho, serão seguidas as regras do IFPUG da primeira a quinta etapa. As duas últimas etapas não fazem parte do escopo deste trabalho, visto que elas representam características especificas do sistema, que devem ser ajustadas manualmente. Além disso, apesar do padrão IFPUG ter sido aprovado pela ISO, as duas últimas etapas não aderem ao padrão de medição funcional, visto que contém requisitos tecnológicos e de qualidade (VAZQUEZ, SIMÕES, ALBERT, 2011).

Na primeira etapa, que consiste por determinar o tipo da contagem, o Analista de Métricas deve escolher um dentre os três tipos propostos, definidos na Tabela 2.1. Neste trabalho, serão consideradas todas as funções providas pelo sistema de informação em sua primeira instalação. Assim, será utilizado o tipo "Projeto em Desenvolvimento".

Tabela 2.1 – Tipos de contagem de pontos de função.

Nome	Definição
Projeto em Desenvolvimento	Mede as funções providas aos usuários na primeira instalação do *software*.
Projeto em Aprimoramento	Mede as modificações feitas em um projeto existente.
Aplicação	Mede as funcionalidades providas aos usuários em um projeto existente.

Além de estabelecer regras para a APF, o IFPUG também provê certificados para ferramentas que auxiliam o processo de contagem, reconhecendo três tipos: 1, 2 e 3 (IFPUG, 2010). No Tipo 1, a identificação das funções é feita pelo usuário e a ferramenta é utilizada para realizar os cálculos. No Tipo 2, a contagem é determinada de forma interativa, o usuário responde algumas perguntas e a ferramenta realiza a contagem com base nas respostas. Finalmente, no Tipo 3, a ferramenta realiza a contagem automaticamente com base em descrições do sistema de informação. Para

ferramentas envolvidas com a APF, esse certificado garante a eficiência do método adotado.

As próximas duas subseções descrevem os dois tipos de funções analisadas durante a contagem de pontos de função, relacionadas a terceira e a quarta etapa, respectivamente, Funções de Dados e Funções de Transações.

2.1.1 Funções de Dados

As Funções de Dados (FDs) podem ser definidas como funcionalidades fornecidas pela aplicação ao usuário, para atender às suas necessidades de armazenamento de dados. Elas são classificadas como Arquivos Lógicos Internos (ALIs) ou Arquivos de *Interface* Externa (AIEs). Um ALI está relacionado a dados que são criados ou mantidos pelo sistema sendo contado, enquanto um AIE trabalha com dados referenciados, externos a fronteira do sistema.

A seguir, serão apresentadas definições aprofundadas de seus dois tipos (ALI e AIE) e será descrito como a complexidade de uma FD é determinada.

2.1.1.1 Definição de Arquivo Lógico Interno (ALI)

De acordo com o IFPUG (2010), um ALI é:

- Um grupo de dados ou informações de controle;
- Identificável pelo usuário;
- Logicamente relacionado;
- Mantido dentro da fronteira do sistema sendo contado.

Por exemplo, em uma aplicação responsável por manter determinada entidade interna a fronteira do sistema, essa entidade seria classificada como um ALI.

2.1.1.2 Definição de Arquivo de Interface Externa (AIE)

De acordo com o IFPUG (2010), um AIE é:

- Um grupo de dados ou informações de controle;
- Identificável pelo usuário;
- Logicamente relacionado;

- Lido de fora da fronteira do sistema sendo contado.

Por exemplo, em uma aplicação que referenciasse determinada entidade externa a fronteira do sistema, essa entidade seria classificada como um AIE.

2.1.1.3 Determinação da Complexidade

Cada FD (ALI ou AIE) deve ser classificada com relação a sua complexidade funcional (baixa, média ou alta) com base em:

- Número de Tipos de Dados (TDs).
- Número de Tipos de Registros (TRs).

Assim, um TD é um campo único, reconhecido pelo usuário, não repetido (IFPUG, 2010). Dessa forma, o número de TDs representa a quantidade de campos, que se encaixam na definição anterior, e que podem ser contados pelo usuário. Vale lembrar que o campo reconhecível pelo usuário não necessariamente representa apenas um campo na base de dados, *i.e.*, um campo não precisa ser equivalente a um atributo de entidade ou tabela. Além disso, deve ser adicionado ao valor o número de associações identificadoras da FD.

Um TR é um subgrupo de dados, reconhecido pelo usuário, de um ALI ou um AIE (IFPUG, 2010). Cada FD tem pelo menos um TR, já que existe pelo menos um subgrupo de TDs, tendo valor adicional para os seguintes casos (VAZQUEZ, SIMÕES, ALBERT, 2011):

- Entidade associativa com atributos não chave;
- Subtipo (outro que não o primeiro subtipo);
- Entidade atributiva em um relacionamento que não seja mandatório.

Após ter classificado a FD e definido o número de TDs e de TRs, é possível determinar sua complexidade funcional utilizando a Tabela 2.2. Já o número de pontos de função atribuídos a FD deve ser obtido com base na Tabela 2.3, em que é utilizado o valor da complexidade funcional como entrada.

Tabela 2.2 – FD: complexidade funcional de ALI e AIE (IFPUG, 2010).

TR	TD		
	1 ~ 19	**20 ~ 50**	**> 50**
1	Baixa	Baixa	Média
2 ~ 5	Baixa	Média	Alta
> 5	Média	Alta	Alta

Tabela 2.3 – FD: valor não ajustado em unidades de ponto de função (IFPUG, 2010).

Complexidade	ALI	AIE
Baixa	7	5
Média	10	7
Alta	15	10

Para validação dos conceitos apresentados, considere o exemplo de uma aplicação que contém a tela presente na Figura 2.2. Considerando os conceitos apresentados e que a aplicação mantém "Pessoa", pode-se classificar "Pessoa" como uma FD do tipo ALI. Além disso, com base na figura, pode-se determinar que o número de TDs é igual a 3 e número de TRs é igual a 2, já que "CPF", "Nome" e "Telefone" são campos reconhecíveis pelo usuário e representam dois subgrupo de dados.

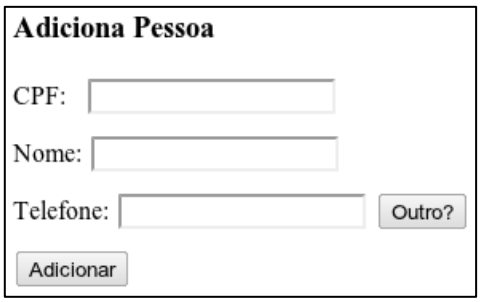

Figura 2.2 – Exemplo de tela que mantém uma FD do tipo ALI.

Por fim, com o número de TDs e de TRs, pode-se consultar a Tabela 2.2 e determinar que a complexidade funcional do ALI é "Baixa". A seguir, basta utilizar a Tabela 2.3 para obter o valor não ajustado de pontos de função igual a 7.

2.1.2 Funções de Transações

As Funções de Transações (FTs) podem ser definidas como requisitos de processamento fornecidos pelo sistema ao usuário. Elas são classificadas como Entrada Externa (EE), Saída Externa (SE) ou Consulta Externa (CE).

A seguir, serão apresentadas as definições aprofundadas de seus três tipos (EE, SE e CE) e será descrito como complexidade de uma FT é determinada.

2.1.2.1 Entrada Externa (EE)

De acordo com o IFPUG (2010), uma EE é uma transação que processa dados ou controla informações originados de fora da fronteira do sistema. Sua principal intenção é manter um ou mais ALIs ou mudar o comportamento do sistema.

Como exemplos de FTs do tipo EE, pode-se citar telas que permitem ao usuário adicionar, excluir ou alterar ALIs.

2.1.2.2 Saída Externa (SE)

De acordo com o IFPUG (2010), uma SE envia dados ou informações de controle para fora da fronteira do sistema. Sua principal intenção é prover informações ao usuário através de lógica de processamento que não seja apenas uma simples recuperação de dados ou informações do controle. Seu processamento deve conter pelo menos uma fórmula matemática ou cálculo, criar dados derivados que mantêm um ou mais ALIs, ou alterar o comportamento do sistema.

Como exemplos de FTs do tipo SE, pode-se citar relatórios que realizam algum cálculo para exibir um valor total.

2.1.2.3 Consulta Externa (CE)

De acordo com o IFPUG (2010), uma CE envia dados ou informações de controle para fora da fronteira do sistema. Sua principal intenção é apresentar informações ao usuário através da recuperação de dados ou informações de controle de ALIs e/ou AIEs. Diferentemente de uma SE, seu processamento não deve conter nenhuma fórmula matemática ou cálculo, e criar dados derivados. Nenhum ALI é mantido durante seu processamento e nem o comportamento do sistema é alterado. Além disso, uma CE sempre envolve duas telas do sistema, a que entra com as

informações de filtro da consulta e a que exibe os resultados. A APF sempre considerará esse par de telas quando identificado que uma FT do tipo CE.

Como exemplos de FTs do tipo CE, podemos citar consultas que retornam informações de dados, como ALIs.

2.1.2.4 Determinação da Complexidade

Cada FT (EE, SE ou CE) deve ser classificada com relação a sua complexidade funcional (baixa, média ou alta) com base em:

- Número de Arquivos Referenciados (ARs).
- Número de Tipos de Dados (TDs).

Um AR representa uma FD envolvida no processamento de uma FT (IFPUG, 2010). Ele pode ser um ALI lido ou mantido por uma FT, ou um AIE lido por uma FT. Dessa forma, o número de ARs representa a quantidade de FDs referenciadas pela FT em análise. No caso da FD ser um ALI, esse AR deve ser contado apenas uma vez, mesmo que esteja sendo lido e mantido pela mesma FT. Já a definição de um TD é exatamente a apresentada na seção sobre FDs. Ele é um campo único, reconhecido pelo usuário, não repetido (IFPUG, 2010), que também pode ser representado por mensagem exibida na aplicação e por um comando de execução de uma ação, como um botão.

Após ter classificado a FT e definido o número de TDs e ARs, as etapas seguintes são semelhantes as realizadas para FDs. Se a FT for uma EE, devemos utilizar a Tabela 2.4 para determinar sua complexidade funcional. Já se ela for uma SE ou uma CE, devemos utilizar a Tabela 2.5. O número de pontos de função atribuídos a FT deve ser obtido com base na Tabela 2.6, em que é utilizado o valor da complexidade funcional como entrada.

Tabela 2.4 – FT: complexidade funcional de EE (IFPUG, 2010).

AR	**TD**		
	< 5	**5 ~ 15**	**> 15**
< 2	Baixa	Baixa	Média
2 ~ 3	Baixa	Média	Alta
> 3	Média	Alta	Alta

Tabela 2.5 – FT: complexidade funcional de SE e CE (IFPUG, 2010).

AR	TD		
	< 6	**6 ~ 19**	**> 19**
< 2	Baixa	Baixa	Média
2 ~ 3	Baixa	Média	Alta
> 3	Média	Alta	Alta

Tabela 2.6 – FT: valor não ajustado em unidades de ponto de função (IFPUG, 2010).

Complexidade	EE	SE	CE
Baixa	3	4	3
Média	4	5	4
Alta	6	7	6

Como validação dos conceitos apresentados, considere a tela ilustrada pela Figura 2.2. Ela representa um exemplo de FT do tipo EE, em que um ALI "Pessoa" é mantido pela transação. Assim, pode-se calcular seu número de TDs da transação "Adicionar Pessoa" contando os campos únicos da tela (*i.e.*, "CPF", "Nome" e o comando representado pelo botão "Adicionar"), obtendo um valor igual a 3. Já seu número de ARs tem valor igual a 1, visto que apenas a FD "Pessoa" está envolvida no processamento da FT. A partir das informações presentes na Tabela 2.4 e na Tabela 2.6, pode-se verificar que complexidade funcional da FT é "Baixa" e seu número de pontos de função é igual a 3.

Agora, considere a tela apresentada pela Figura 2.3. Ela consiste de um relatório que envolve o cálculo de alguns campos, como o valor "Total". Assim, essa FT deve ser classificada como do tipo SE. O cálculo do número de TDs da transação deve ser feito contando os campos únicos da tela (*i.e.*, "Código", "Produto", "Valor Unit. (R$)", "Quantidade", "Total (R$)" e "Total"), obtendo um valor igual a 6, em que apenas as colunas da tabela são contabilizadas, não suas linhas. O número de ARs deve ser calculado analisando quantas FDs são lidas durante o processamento da FT, que em nosso exemplo é igual a 2, "Pedido" e "Item Pedido". A partir das informações

presentes na Tabela 2.5 e na Tabela 2.6, pode-se verificar que complexidade funcional da FT é "Média" e seu número de pontos de função é igual a 5.

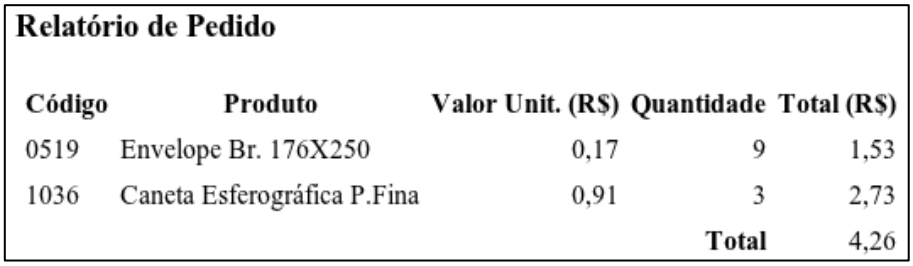

Figura 2.3 – Exemplo de tela de uma FT do tipo SE.

Conforme definido anteriormente, uma FT do tipo CE se difere de uma EE e uma SE, principalmente, por sua análise envolver duas telas da aplicação, sendo uma geralmente destinada a consulta e outra ao resultado. A Figura 2.4 ilustra duas telas de um sistema, Figura 2.4-a e Figura 2.4-b, que representam, respectivamente, a tela utilizada para fornecer os filtros da consulta e outra para exibir o resultado.

Figura 2.4 – Exemplo de tela de uma FT do tipo CE.

Primeiramente, verifica-se que a tela ilustrada pela Figura 2.4-a possui três campos únicos (*i.e.*, "CPF", "Nome" e o comando representado pelo botão "Consultar") e que a tela ilustrada pela Figura 2.4-b também possui três campos únicos (*i.e.*, "CPF", "Nome" e "E-mail"), resultando em um número de TDs igual a 6. Já o número de ARs tem valor igual a 1, visto que apenas a FD "Pessoa" é lida pela transação sendo contada. Dessa forma, a partir informações presentes na Tabela 2.5 e na Tabela 2.6, pode-se

verificar que complexidade funcional da FT é "Baixa" e seu número de pontos de função é igual a 3.

2.2 Model Driven Architecture

A MDA é uma iniciativa da OMG com foco na padronização do desenvolvimento orientado por modelos. Ela é composta por padrões e tecnologias que podem ser aplicados no processo de desenvolvimento de sistemas de informação. Conforme definida por FRANKEL (2003), o principal objetivo da MDA é promover a separação entre especificações de funcionalidades e suas implementações em nível específico de plataforma. Ela permite o desenvolvimento de sistemas através da criação de modelos, que são refinados a partir de sucessivas transformações realizadas durante o processo de desenvolvimento. Dessa forma, o processo de desenvolvimento de sistemas passa a ser direcionado pela atividade de modelagem e os modelos passam a ser o objetivo do processo (KLEPPE, BAST, WARMER, 2003).

CZARNECKI e HELSEN (2003) definem duas das principais categorias de transformação: a transformação de modelo para código e a transformação de modelo para modelo. A transformação de modelo para código consiste da geração de código-fonte, em linguagem de programação, a partir de um modelo UML. Já a transformação de modelo para modelo pode ser definida como a tradução de modelos, *i.e.*, como a transformação entre instâncias de um mesmo metamodelo ou de metamodelos diferentes. Na prática, a transformação entre modelos serve como base para a transformação de modelo em código, fornecendo camadas abstratas em nível de tecnologia.

Ao introduzir a MDA, um pouco antes da primeira versão do guia da OMG, SOLEY *et al.* (2000) apresentou alguns conceitos sobre a transformação de modelo em código, a classificando em dois tipos: total e parcial. A geração total de código estabelece que todo o código-fonte da aplicação é gerado a partir de modelos, não sendo necessário nenhuma ação adicional em nível de código. Já a geração parcial de código utiliza modelos para gerar apenas uma parcela do código-fonte, devendo a outra parte ser desenvolvida diretamente em linguagem de programação. Ambos os tipos oferecem vantagens em domínios distintos, em que o ponto positivo de um pode ser o ponto negativo de outro e vice-versa.

A geração total de código permite que os modelos utilizados no desenvolvimento estejam sempre em sincronismo com o código da aplicação, representando documentos importantes do sistema de informação. Contudo, devido ao limite de representação da linguagem UML, os casos em que essa abordagem pode ser aplicada são restritos. Além disso, esse tipo de geração está limitado pelo próprio meio de geração: tudo é gerado a partir de informações contidas em modelos, sem a possibilidade de realizar ajustes específicos em código, sendo necessário alterar o processo de geração de modelos em código. Na prática, o uso da geração total de código é improvável na maioria das aplicações reais.

Na geração parcial de código, as informações que não foram modeladas e são importantes para o funcionamento da aplicação são descritas em locais denominados pontos de implementação. Esses pontos são fornecidos pelo mecanismo de transformação para que os desenvolvedores possam expressar informações adicionais aos modelos. Dessa forma, características específicas da aplicação podem ser personalizadas em seus pontos de implementação, sem necessidade de propagar a todos as outras aplicações geradas pela mesma ferramenta de transformação. Devido a grande necessidade desse tipo de personalização (*e.g.*, ajustes de telas do sistema ou até mesmo de regras de negócio), esse tipo de abordagem de geração acaba atendendo melhor a maior parte dos projetos.

Apesar da geração parcial de código facilitar a personalização de aplicações, devido ao fácil acesso do desenvolvedor ao código-fonte, o uso de pontos de implementação também acarreta desvantagens. Como tanto os modelos quanto o código-fonte guardam informações do sistema em desenvolvimento, é preciso trabalho adicional na verificação do que realmente deve ser gerado a partir de modelos e o que deve ser escrito diretamente nos pontos de implementação. Além disso, a geração parcial também pode sofrer com a falta de detalhes da linguagem de modelagem utilizada, contudo em escala menor devido a menor quantidade de informações em modelos.

Embora o desenvolvimento com a geração parcial de código apresente trabalho adicional em pontos de implementação, grande parte do código fonte continua a ser gerado pela ferramenta de transformação. Com base em nossa experiência no desenvolvimento de sistemas de informação de grande porte (MDARTE, 2011) e

conforme apontado GUTTMAN e PARODI (2006), podemos aplicar o Princípio de Pareto e afirmar que em torno de 80% do código-fonte é gerado automaticamente. Ou seja, mesmo com o uso da abordagem de geração parcial, a ferramenta transformação continua a ser de grande ajuda no desenvolvimento de aplicações.

O Anexo 1 (MDARTE, 2012), disponível na comunidade do MDArte, também apresenta outras informações sobre a metodologia MDA e sobre o *framework* de geração parcial de código, o MDArte, que é utilizado na implementação deste trabalho e será descrito nas próximas seções. Além disso, os Anexos 2, 3 e 4 (MDARTE, 2012), também disponíveis na comunidade, apresentam, respectivamente, informações sobre a modelagem das camadas de domínio, serviços e apresentação, que serão úteis no decorrer dos capítulos seguintes.

Capítulo 3

3 Trabalhos Relacionados

Neste capítulo serão discutidos os trabalhos relacionados à contagem automática de pontos de função. Serão abordados trabalhos que realizam a análise através código-fonte e modelos.

A Seção 3.1 apresentará os trabalhos que usam o código-fonte diretamente na contagem de pontos de função. A Seção 3.2 apresentará os trabalhos que utilizam modelos, não necessariamente UML, como entrada do processo de contagem. A Seção 3.3 apresentará os trabalhos que se beneficiam da MDA para realizar a contagem.

3.1 Contagem a partir de Código-Fonte

Existem diferentes tipos de métricas para analisar e predizer o tamanho de um sistema de informação a ser desenvolvido. O termo "tamanho" pode ser definido como uma medida do volume do sistema, de comprimento e de funcionalidades que podem ser estimadas. Contudo, muitos especialistas concordam que comprimento (*i.e.* SLOC) não é a medida mais apropriada para um *software*. Apesar disso, grande parte dos trabalhos da década de 90 ainda considerava o código-fonte como uma boa fonte para a contagem automática de pontos de função, baseada na medida SLOC.

Os trabalhos de HO e ABRAN (1999) e PATON (1999) utilizam as linguagens de programação COBOL e C, respectivamente, para determinarem a fronteira do sistema de informação e realizarem a medição. Apesar de ser uma abordagem considerável, o uso direto do código-fonte desvaloriza a principal característica de pontos de função, sua independência de tecnologia.

3.2 Contagem a partir de Modelos

Devido aos problemas existentes na contagem baseada em código-fonte, muitos pesquisadores buscaram alternativas que garantissem maior independência das tecnologias aplicadas ao desenvolvimento de sistemas de informação. O trabalho de FETCKE *et al.* (1997) foi publicado próximo ao lançamento da linguagem de modelagem UML, e utiliza a abordagem de Orientação à Objetos de Jacobson (JACOBSON, 1992) para identificar os tipos de funções presentes em sistemas e

permitir a contagem. Dessa forma, foi proposto o uso de diagramas de classes na análise de funções de dados e casos de uso na análise de funções de transações. Contudo, o trabalho não abordou a automatização do processo, descrevendo apenas como os modelos deveriam ser utilizados.

O trabalho de UEMURA *et al.* (2001) também utiliza a abordagem de Jacobson, contudo ele descreve o uso de modelos UML na automatização da APF pelo método IFPUG. Primeiramente, foram definidas regras e heurísticas que apoiassem a contagem em modelos UML. O trabalho descreve a ferramenta desenvolvida para a automatização, que utiliza diagramas de classes para analisar as funções de dados e diagramas de sequência para as funções de transações. Os diagramas de classes possibilitam à ferramenta obter informações sobre os dados, seus atributos e operações, enquanto os diagramas de sequência fornecem informações sobre como os dados são utilizados pelo sistema de informação sendo contado. Além disso, o trabalho ressalta a possibilidade de se utilizar casos de uso quando diagramas de sequência não estiverem disponíveis, visto que os diagramas de sequência podem ser obtidos a partir das descrições de casos de uso.

Ambos os trabalhos (CANTONE, PACE, CALAVARO, 2004) e (HARPUT, KAINDL, KRAMER, 2005) apresentam ambientes semelhantes e realizam a contagem de forma semiautomática. Eles descrevem a transformação de modelos UML para modelos mais significativos à contagem, *i.e.*, que facilite o trabalho do Analista de Métricas. Eles utilizam diagramas UML (sequências, classes e casos de uso) e algumas regras de contagem para gerar modelos que contém informações pertinentes à APF.

LAVAZZA *et al.* (2008) também propõem a contagem a partir de modelos UML. Contudo, seus modelos devem ser construídos seguindo uma notação padrão significativa à contagem. Isso foi proposto para garantir que os modelos contenham informações necessárias para que a contagem seja realizada, diferentemente do método descrito por UEMURA *et al.* (2001). Nesse contexto, LEVESQUE *et al.* (2008) também descrevem a análise em modelos UML construídos conforme regras que visam melhorar o processo de contagem. Por descrever um processo de contagem automática, o trabalho tem por objetivo oferecer uma ferramenta de contagem para o padrão COSMIC, em nível industrial, e competir com outras que seguem o padrão IFPUG, como a descrita em (UEMURA, KUSUMOTO e INOUE, 2001).

Similar ao apresentado por FETCKE *et al.* (1997), BATISTA *et al.* (2011) não utilizam modelos para automatização do processo de contagem, *i.e.*, como entrada de uma ferramenta. O trabalho descreve como o Analista de Métricas deve utilizar modelos UML de requisitos e casos de uso para identificar as funções presentes no sistema de informação, seguindo o padrão IFPUG. Sua proposta principal reside na elaboração de uma ferramenta para auxiliar analistas nos cálculos envolvidos no processo de APF e no reconhecimento da ferramenta pelo IFPUG com certificação de Tipo 1.

O estudo apresentado por IORIO (2004) contém informações sobre pesquisas relevantes na área de medição em pontos de função baseada em modelos UML, incluindo o trabalho de UEMURA *et al.* (2001). Apesar do estudo não conter todos os trabalhos aqui relacionados, alguns pontos de sua pesquisa devem ser ressaltados:

- Durante a medição, modelos UML diferentes que representam a mesma regra de negócio podem resultar em valores diferentes de pontos de função.
- A conversão de modelos UML em modelos que facilitem a contagem pode causar distorções às informações originais e interferir na medição.
- O uso de padrões de modelagem pode trazer alguns benefícios aos próprios modelos, já que a margem de erro seria menor e sua identificação mais fácil.

3.3 Contagem a partir da MDA

A contagem de pontos de função realizada através da metodologia MDA pode ser compreendida como uma extensão ao uso de modelos, abordado na seção anterior. Ela aproveita os modelos construídos durante o desenvolvimento pela MDA e os utiliza no processo de análise sem exigir que sejam criados especificamente para esse propósito. Ademais, por serem utilizados no desenvolvimento, os modelos estão sempre em sincronia com o sistema de informação gerado, permitindo a medição do tamanho atual da aplicação.

ABRAHÃO *et al.* (2007) abordam a contagem de pontos de função pelo padrão IFPUG em modelos que utilizam o método Object-Oriented Hypermedia (OO-H) (GOMEZ, CACHERO, PASTOR, 2001). Além de estabelecerem regras para a análise de modelos, o trabalho realiza a automação do processo e a validação dos resultados a partir da base de dados Tukutuku (MENDES, MOSLEY, COUNSELL, 2005).

ABRAHAO e INSFRAN (2008) continua a abordar a contagem de pontos de função pelo padrão IFPUG, com foco no uso de modelos UML como fonte de informação para a contagem. O estudo utiliza diagramas de classes para identificar as FDs e casos de uso para identificar as FTs, enquanto os diagramas de sequências são utilizados para relacionar os dois tipos de funções. Além disso, o trabalho descreve a automatização do processo através da construção de uma ferramenta dedicada, que identifica as funções e calcula número de pontos de função de acordo com pesos configuráveis.

ABRAHÃO *et al.* (2010) retornam à análise de modelos OO-H tendo como objetivo a contagem pelo padrão COSMIC. Apesar de continuarem a utilizar diagramas de classes e diagramas de navegação como fonte de informações para contagem, o trabalho apenas estabelece regras de contagem, sem abordar a automatização do processo. Conforme descrito por (ABRAHÃO *et al.*, 2007), (ABRAHAO, INSFRAN, 2008) e (ABRAHÃO *et al.*, 2010), podemos inferir que os *frameworks* MDA utilizados no três trabalhos apresentam geração total de código, visto que apenas os modelos foram suficientes para identificar e contar os pontos de função, sem necessitar de informações adicionais sobre o sistema sendo contado.

Assim como em (ABRAHÃO *et al.*, 2010), temos em (MARÍN *et al.*, 2008a), (MARÍN, PASTOR, GIACHETTI, 2008b) e (MARÍN, PASTOR, ABRAN, 2010) estudos que também se apoiam em modelos OO-H para realizarem a contagem pelo padrão COSMIC. Os três trabalhos acompanham a evolução de uma pesquisa que envolve a definição de regras de contagem, a automatização do processo e a apresentação de um caso de estudo para verificação dos resultados. MARÍN *et al.* (2008b) também discutem, superficialmente, a eficiência do padrão COSMIC em sistemas desenvolvidos com MDA, apresentando apenas dados relativos a modelos OO-H.

O estudo apresentado por FRATERNALI *et al.* (2006) propõe a contagem de pontos de função pelo padrão IFPUG em modelos criados com WebML (CERI, FRATERNALI, BONGIO, 2000), uma linguagem conceitual desenvolvida pela Academia para modelar aplicações Web. O trabalho define algoritmos e heurísticas para coletar as informações a partir de modelos, realizando a APF em conjunto com o processo de geração de código através de um *plug-in* para a ferramenta CASE WebRatio (WEBRATIO, 2012), como ilustrado pela Figura 3.1.

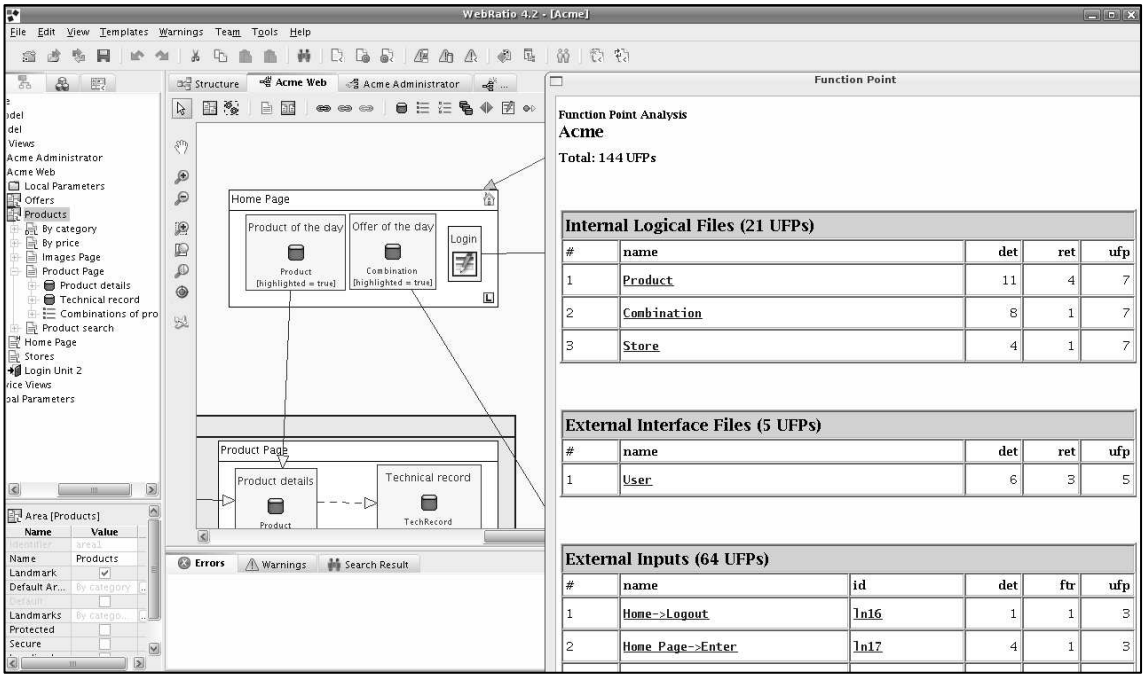

Figura 3.1 – *Plug-in* de APF para WebRatio (FRATERNALI, TISI, BONGIO, 2006).

Devido a fatores que limitariam a automatização da contagem, o trabalho calcula apenas o valor não ajustado de pontos de função. Além disso, conforme definido pelo IFPUG, a ferramenta construída por FRATERNALI *et al.* (2006) pode ser considerada de Tipo 3, por depender apenas de modelos para realizar a APF. Assim como em (ABRAHÃO *et al.*, 2010), a partir da abordagem de análise dos modelos, podemos verificar que o trabalho descreve a APF em um ambiente MDA de geração total de código, em que todas as informações necessárias para a contagem estão modeladas e podem obtidas no momento da geração.

Apesar dos trabalhos aqui relacionados apresentarem resultados e validações de seus estudos, nenhum deles teve foco em *frameworks* MDA de geração parcial de código, que trabalham com informações adicionais às modeladas, muitas vezes descritas diretamente em linguagem de programação. Contudo, este trabalho não tem por objetivo o uso de código-fonte diretamente na APF, visto que isso acarretaria em limitações tecnológicas, além de explorar pouco as vantagens oferecidas pela MDA. Assim, será apresentada uma abordagem híbrida do problema, utilizando modelos UML e código-fonte em conjunto, com o apoio do *framework* MDArte. Para criar uma camada flexível entre código-fonte de sistemas a serem contados e a ferramenta APF desenvolvida, o código-fonte será utilizado a partir de valores extraídos, e não diretamente. Isso é

possível com o uso de analisadores de dependências, *i.e.*, com o uso de ferramentas externas que auxiliam essa etapa, conforme será explorado no próximo capítulo.

Capítulo 4

4 Proposta

Medir o tamanho é uma tarefa crucial em projetos de software, tanto na fase inicial contratual quanto durante sua manutenção. Ser capaz de medir a "quantidade de funcionalidades" a ser entregue aos clientes é essencial para o planejamento de desenvolvimento, preservando a rentabilidade dos projetos e ações de cronograma (EBERT *et al.*, 2004). Nesse contexto, o estabelecimento da automatização da APF apresenta várias vantagens. Um procedimento considerado demorado e custoso passa a ser realizado em tempo muito inferior e automaticamente (FRATERNALI, TISI, BONGIO, 2006). Além disso, como a ferramenta de automatização contém regras e heurísticas a serem aplicadas na contagem, torna-se menos necessário a alocação de especialista para o procedimento (LEVESQUE, BEVO, CAO, 2008), evitando também uma interpretação subjetiva por parte do próprio especialista (LAVAZZA, DEL BIANCO, GARAVAGLIA, 2008), que seria o responsável pela análise tradicional (*i.e.*, manual).

Em ambientes de desenvolvimento com *frameworks* MDA de geração parcial de código, as equipes de desenvolvimento adotam modelos para representar o comportamento da aplicação. Quando os modelos falham em representar tal comportamento, essas informações são descritas em código-fonte, nos conhecidos pontos de implementação. Assim, tanto os modelos quanto o código-fonte são fontes importantes para extração de informação, conforme ilustrado pelo diagrama da Figura 4.1.

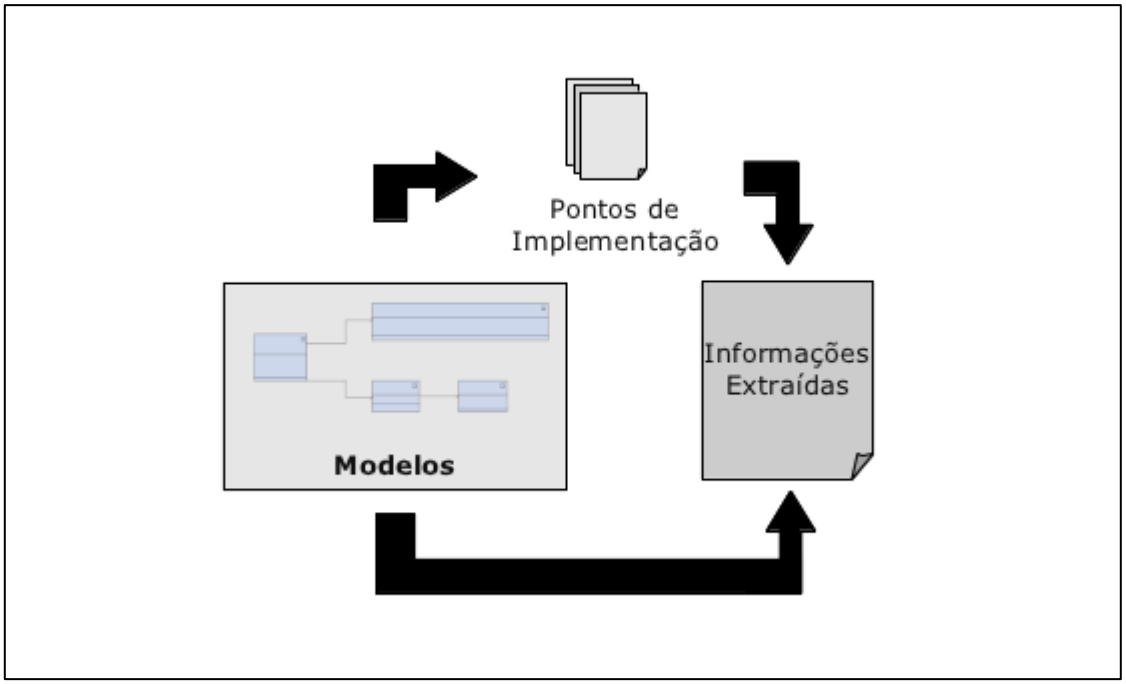

Figura 4.1 – Diagrama de extração de informações.

As informações extraídas de modelos e de pontos de implementação permitem a geração de artefatos úteis não apenas à análise de métricas de *software*, mas também para procedimentos que apoiam o ciclo de desenvolvimento, *e.g.*, grafos para análise de impacto e/ou violações de arquitetura, como descrito no trabalho de SILVA *et al. (* 2010). Esses artefatos podem representar diferentes informações, conforme o tipo de análise a ser empregada.

Este trabalho tem como foco a automação da APF, pelo padrão IFPUG, em sistemas de informação, que foram desenvolvidos pela geração parcial de código. Dessa forma, por realizarmos a contagem de pontos de função no que já foi desenvolvido, estaremos medindo as funções já realizadas, que estão modeladas e codificadas.

A extração de informações do código-fonte da aplicação, representado pelos pontos de implementação, garante a proposta certa independência da linguagem de programação utilizada no desenvolvimento. Essas informações são compostas de relações de dependência entre os elementos que compõem a aplicação, em nível genérico, servindo para complementar as informações obtidas com base em modelos.

A partir da análise em conjunto das relações de dependências e das informações obtidas a partir dos modelos, é possível estabelecer um cruzamento dos dados fornecidos e identificar os dois tipos de funções propostas pela APF: dados e transações.

Após a identificação, o núcleo de contagem baseia-se nessas informações para obter o valor em pontos de função, como indicado pelo diagrama da Figura 4.2.

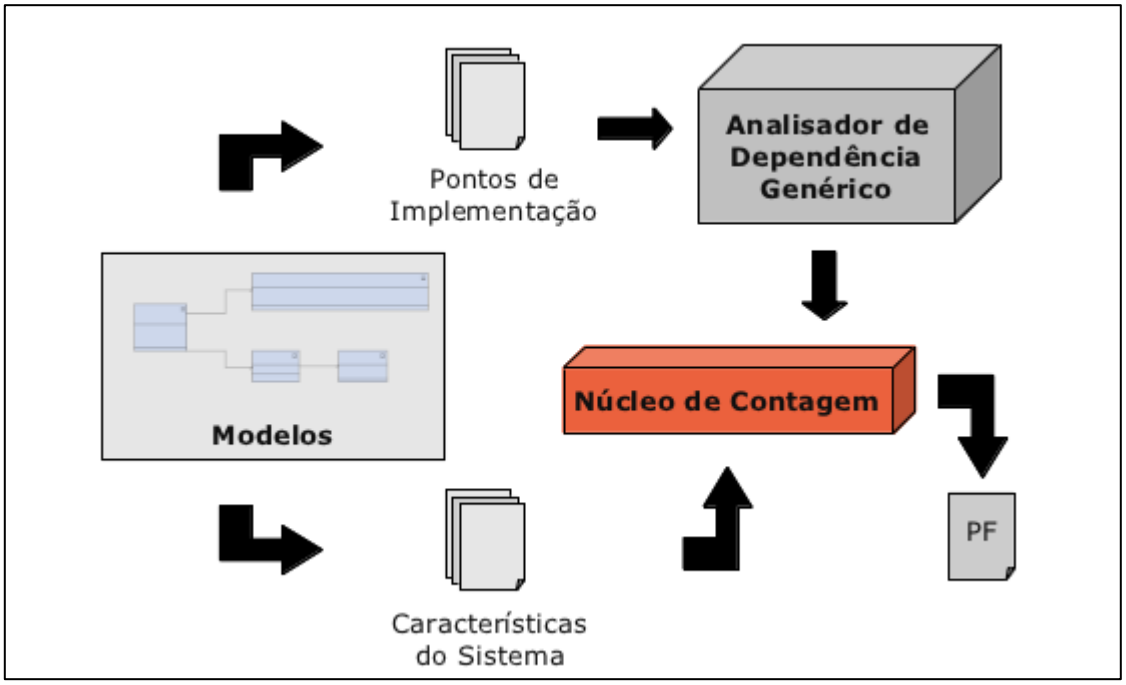

Figura 4.2 – Diagrama da proposta de automação da APF.

A relação de dependência obtida do analisador de dependência contém informações que não foram necessariamente modeladas, *e.g.*, a relação de casos de uso que acessam determinada entidade do sistema de informação.

Já as características do sistema extraídas dos modelos descrevem elementos de modelo e suas peculiaridades, como exemplificado a seguir:

- Entidades: são obtidas a partir de modelos, representando os elementos do tipo "classe" com o estereótipo *«Entity»*.
- Serviços: são obtidos a partir de modelos, representando os elementos do tipo "classe" com o estereótipo *«Service»*.
- Telas: são obtidos a partir de modelos, representando os elementos do tipo "atividade" com o estereótipo *«FrontEndView»* de diagramas de atividades relacionados a elementos do tipo "caso de uso" com o estereótipo *«FrontEndUseCase»*.

A interpolação desses dados permite obter o cruzamento de informações necessárias para a concretização da APF.

De acordo com a proposta descrita anteriormente, a Seção 4.1 e a Seção 4.2 abordarão a identificação e contagem de funções de dados (FDs) e de funções de transações (FTs), respectivamente. Por fim, a Seção 4.3 apontará as diferenças entre a proposta de contagem e o método IFPUG.

4.1 A Contagem de Funções de Dados

As FDs são relacionadas a elementos da camada de domínio de sistemas de informação. Elas são classificadas conforme a fronteira do sistema, como internas (ALI) e externas (AIE).

A Figura 4.3 apresenta o fluxograma de identificação de FDs. O processo é realizado, primeiramente, com base nos artefatos gerados com as características do sistema de informação. Nessa etapa, o núcleo de contagem verifica todas as entidades pertencentes à camada de domínio, além obter informações sobre seus atributos e métodos. Em seguida, o núcleo utiliza as características, de casos de uso e serviços modelados, para consultar as relações de dependências e verificar, para cada entidade, se ela é mantida pelo sistema, *i.e.*, se a FD é uma ALI. Em caso negativo, a FD representada pela entidade é classificada como AIE.

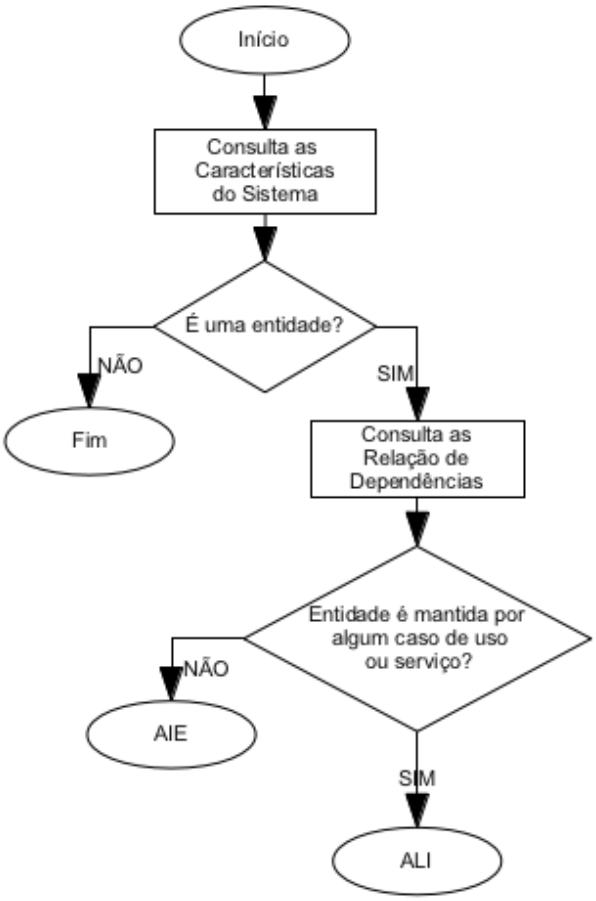

Figura 4.3 – Fluxograma de identificação de uma FD.

Cada FD possui um número de TDs e um número de TRs, contados como descrito a seguir:

a) TD: contado pelo somatório do número de atributos da entidade e o número de atributos de cada entidade herdada, recursivamente, se existir. Durante o somatório, são desconsiderados todos os atributos identificadores. A esse valor é adicionado o número associações que identificação a FD, caso exista alguma.

b) TR: foi aproximado para o valor 1 (um), em virtude de ser o valor mais comum na maioria dos casos e ter fornecido bons resultados em (UEMURA, KUSUMOTO e INOUE, 2001). Além disso, pela definição do IFPUG, o número de TRs representa o subgrupo de dados reconhecidos pelo usuário, o que não é possível de ser contado com base nas características do sistema e nas relações de dependências.

Conforme descrito na Seção 2.1.1.3 do Capítulo 2, por meio da combinação do número de TDs e de TRs, associado a uso de tabelas do IFPUG, é possível atribuir o valor de complexidade à FD e posteriormente determinar seu valor não ajustado de pontos de função.

4.2 A Contagem de Funções de Transações

As FTs são compostas, majoritariamente, por componentes da camada de apresentação de sistemas de informações. Assim como as FDs, elas são classificadas conforme a fronteira do sistema alvo da análise, como entradas externas (EE), saídas externas (SE) e consultas externas (CE).

O processo de identificação de FTs pode ser separado em duas etapas, como ilustrado pela Figura 4.4. O fluxograma da Figura 4.4-a representa a primeira etapa do processo, descrevendo como as FTs do tipo EE e CE são identificadas. Nessa etapa, apenas uma parte de uma FT do tipo CE é identificada, visto que uma CE é composta por uma tela de consulta e uma de resultados. Assim, apenas as FTs do tipo CE, provenientes de telas de consultas, serão identificadas nessa primeira etapa. Na segunda etapa do processo, conforme descrito pelo fluxograma da Figura 4.4-b, são identificadas as FTs do tipo SE e os pares de CEs já identificadas na primeira etapa.

Na primeira etapa, com base nos artefatos gerados com as características do sistema de informação, o núcleo de contagem verifica se cada tela do caso de uso possui ou não campos de entrada de informações, *i.e.*, se a FT possui parâmetros. Dessa forma é possível fazer um levantamento de quais telas são candidatas a serem identificadas como EE ou CE. Em seguida, o núcleo utiliza as relações de dependências para analisar se as FTs provenientes das telas mantêm entidades do sistema. Em caso afirmativo, a FT é classificada como EE, do contrário, será classificada como CE, visto que representaria a tela de consulta do par descrito anteriormente.

Na segunda etapa, o núcleo volta a utilizar os artefatos gerados com as características do sistema de informação e verifica novamente se a tela possui campos de entrada. Ao contrário da primeira etapa, em caso negativo, o processo continua e verifica se a tela é resultado de alguma tela de consulta, *i.e.*, se representaria o par de uma CE já identificada na primeira etapa. Se sim, a FT é classificada como um par CE, senão será classificada como um SE.

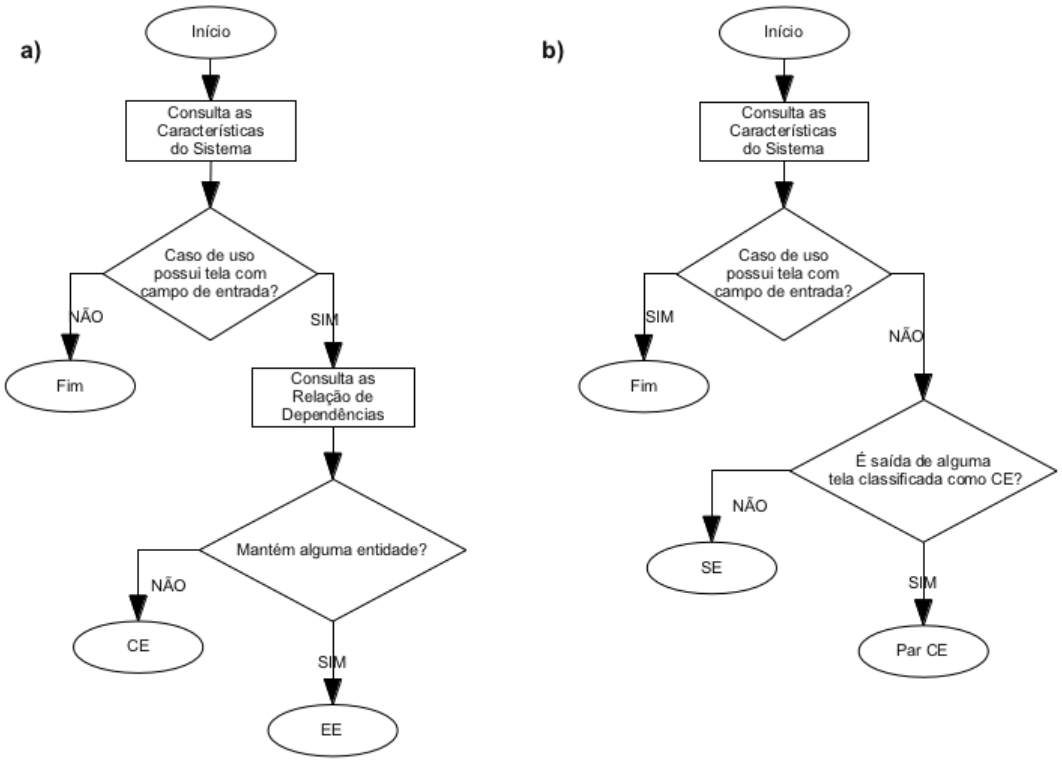

Figura 4.4 – Fluxograma de identificação de uma FT.

Uma vez tendo identificado todas as FTs, a contagem é iniciada. Cada FT possui um número de TDs e um número de ARs, contados conforme:

a) TD: para uma EE, representa o número de parâmetros da transação de entrada. Para uma SE, representa o número de parâmetros de saída. Por fim, para uma CE, representa a soma entre o número de parâmetros de entrada e o número de parâmetros de saída.

b) AR: representa o número de ALI e AIE referenciados pela FT. Cada FD é contada apenas uma vez, mesmo que muitos de seus atributos sejam utilizados pela FT.

Similar à atribuição de complexidade de FDs e conforme descrito pela Seção 2.1.2.4 do Capítulo 2, a atribuição é realizada com base nos valores de TD e AR. Mediante a combinação desses valores é possível obter o valor de complexidade da FT e determinar seu valor não ajustado de pontos de função.

4.3 Diferenças entre a Proposta e o Método IFPUG

Apesar de este trabalho utilizar o método IFPUG, alguns aspectos da contagem se tornaram particulares a proposta. Essas diferenças não impedem que a APF seja realizada, apenas altera um pouco sua quantificação total, mantendo sua grandeza. Elas foram necessárias para que a contagem pudesse ser realizada automaticamente no ambiente proposto.

A seguir, essas diferenças serão abordadas em duas subseções, uma para FDs e outra para FTs.

4.3.1 Diferenças na Análise de Funções de Dados

Poucas são as diferenças envolvendo a análise de FDs da proposta e do método IFPUG. Provido de informações sobre os modelos, através dos artefatos com as características do sistema, e sobre as dependências, o núcleo de contagem consegue identificar exatamente o tipo de cada FD: ALI ou AIE. Contudo, a etapa de contagem possui algumas peculiaridades.

Durante a contagem, o número de TDs é determinado estritamente a partir dos artefatos com as características do sistema, que permite ao núcleo ter acesso ao número de atributos de cada entidade do sistema e seus relacionamentos. O número de TDs é então contado seguindo as práticas propostas pelo IFPUG. Já a contagem do número TRs é realizada pela aproximação do valor 1 (um), por razões já descritas, que desacordam com o método IFPUG.

Por aproximar o valor do número de TRs, pela Tabela 2.2 é possível verificar que o núcleo de contagem estará restringindo a complexidade da FD entre baixa ou média, visto que a apenas o número de TDs estará variando. Essa diferença com o IFPUG ressalta a maior limitação da contagem de FDs pela proposta.

4.3.2 Diferenças na Análise de Funções de Transações

Assim como para a análise de FDs, também é possível levantar diferenças entre a análise de FTs e o método IFPUG. Da mesma forma, essas diferenças não constituem irregularidades da proposta, e sim a abordagem escolhida para a automatização da APF.

Durante a identificação do tipo de cada FT, a proposta se restringe a um perfil de sistemas de informação, com telas com características bem nítidas. Dessa forma, uma

EE consiste de uma tela de entrada, uma SE de uma tela de saída, e uma CE de uma tela de entrada e outra de saída. Nessas condições, a proposta não leva em consideração conceitos mais modernos ou até mesmo telas que carregam informações em seu próprio corpo, *e.g.*, uma tela que além de listar os resultados, permite ao usuário editar os valores diretamente na listagem.

Além disso, outra diferença ao IFPUG reside na falta de informações para núcleo de contagem verificar se uma FT altera o comportamento do sistema ou realiza alguma operação matemática. Tanto os artefatos com as características do sistema quanto as relações de dependências pecam em fornecer tais informações.

Por conseguinte, ao influenciar a classificação do tipo de cada FT, essas características da análise acabam por deixar de contar valores relevantes à contagem, como número de TDs e número de ARs. Ainda sim, para as FTs identificadas, o núcleo realiza a contagem apropriada, quase em total conformidade com as práticas do IFPUG.

O número de TDs é contado conforme o IFPUG, com exceção da contagem de mensagens (*e.g.*, sucesso, aviso ou erro), que não é possível de ter sua existência verificada com base nos artefatos com características do sistema e nas relações de dependências, visto que representa a forma como o sistema de informação funciona. Já o número de ARs é contado exatamente com no IFPUG, analisando cada FT e verificando o número de FD únicas acessadas.

Capítulo 5

5 Implementação

Muitos são os sistemas de informação que foram e estão sendo desenvolvidos com o *framework* MDArte (MDARTE, 2011). Apesar de facilitar o trabalho do desenvolvedor e do analista, o MDArte não oferece nenhum suporte a medição do tamanho dos sistemas gerados por ele. Em muitos momentos do processo de desenvolvimento, não apenas no início de sua elaboração, medidas como pontos de função são importantes para a equipe responsável pelo projeto e também para o cliente. Poder contar automaticamente o número de pontos de função de um sistema de informação fornece grande redução de custo e tempo, além de padronizar os resultados obtidos.

Como o MDArte é um *framework* de geração parcial de código, os modelos utilizados por ele não são suficientes para que a APF seja realizada, tornando o uso informações de presentes em código-fonte uma etapa mandatória da contagem. Ainda assim, para que o código-fonte não seja diretamente utilizado pela ferramenta responsável pela contagem (*i.e.*, para que a ferramenta possa ser independente da linguagem de programação utilizada pelo sistema a ser analisado), são apenas utilizadas relações de dependências de elementos do código-fonte. Neste trabalho, o extrator de dependências Dependency Finder (DEPENDENCY FINDER, 2012) foi escolhido para geração das relações de dependências com base no código-fonte em Java, linguagem que compõe os sistemas de informações gerados pelo MDArte.

Neste capítulo, será abordada a implementação da ferramenta Ligeiro, desenvolvida por este trabalho para a automatização da APF. A ferramenta realiza a contagem automática de pontos de função em aplicações geradas pelo *framework* MDArte. Assim, no decorrer das próximas seções, a proposta apresentada no Capítulo 4 será detalhada e seus conceitos concretizados com a confecção da ferramenta de contagem.

A Seção 5.1 descreverá a implementação da proposta, apresentando a ferramenta Ligeiro, sua estrutura e funcionamento. Além disso, também será apresentado o

processo de construção dos arquivos com as características dos sistemas e das relações de dependências.

Por fim, analogamente ao capítulo anterior, as duas últimas seções, Seção 5.2 e Seção 5.3, descreverão a contagem de FDs e FTs, respectivamente, no contexto da implementação. Ou seja, o processo de contagem de ambos os tipos de funções será apresentado tendo em vista a ferramenta elaborada e as características dos sistemas de informação alvos.

5.1 Descrição da Implementação

Sistemas de informação desenvolvidos com o MDArte são gerados com base na estrutura de camadas exibida pela Figura 5.1. Esses sistemas são compostos por três camadas de componentes: apresentação, negócio e persistência. Todas as camadas e seus componentes são modelados em UML, e esses modelos são utilizados como entrada do processo de geração.

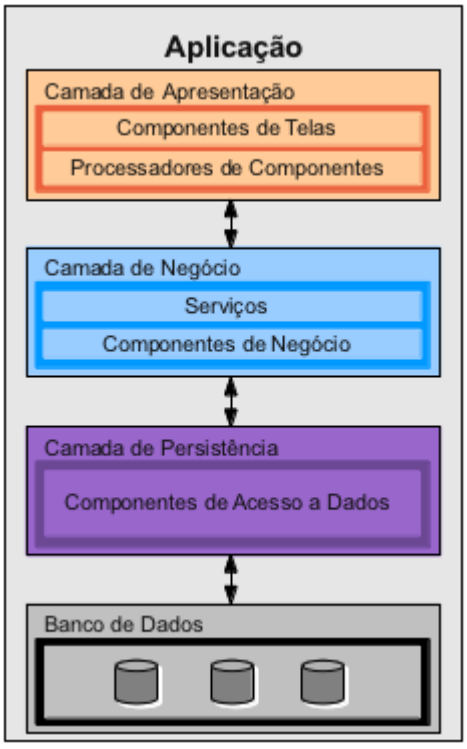

Figura 5.1 – Estrutura de camadas de aplicações geradas pelo MDArte (ANDROMDA, 2012).

A modelagem de cada componente depende majoritariamente da camada a qual ele pertence. Na camada de persistência, entidades são modeladas com elementos UML do tipo classe e devem conter o estereótipo *«Entity»*, como ilustrado pela Figura 5.2-a.

Na camada de negócio, serviços também são modelados com elementos UML do tipo classe, que devem conter o estereótipo *«Service»*, como exibido pela Figura 5.2-b. Por fim, na camada de apresentação, casos de uso têm seus fluxos descritos por diagramas atividades, como mostrado pela Figura 5.2-c. Além disso, a cada caso de uso é atribuído um elemento UML do tipo classe, conhecido como "controle", que é responsável por lidar com as informações fornecidas ao sistema, *e.g.*, a entrada de dados por usuários. Os métodos de classe de controle guardam o código que é executado quando uma ação é realizada por um usuário, *e.g.*, quando um usuário clica em um botão da tela do sistema.

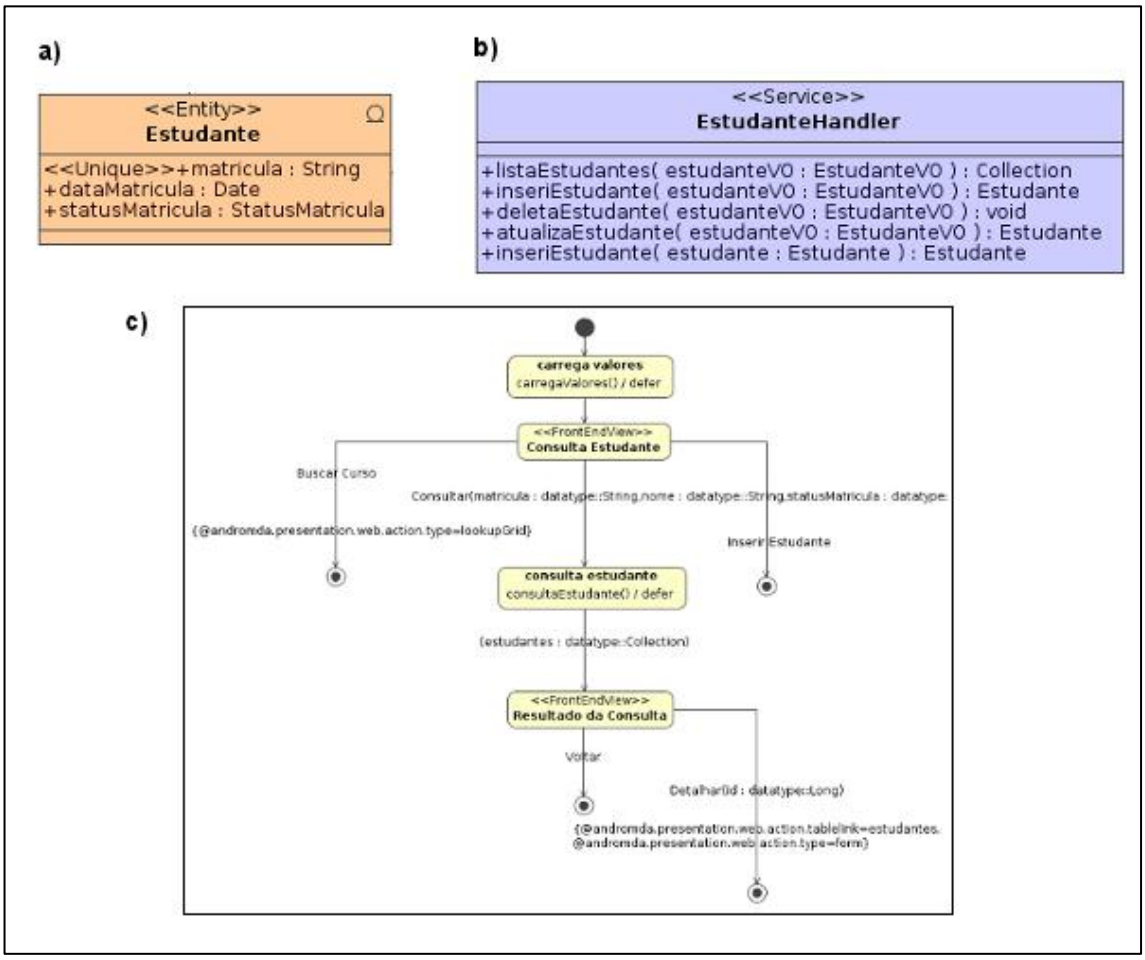

Figura 5.2 – Elementos UML comuns ao MDArte.

A implementação da proposta de automatização da APF em *frameworks* MDA de geração parcial de código foi possível com o uso do *framework* MDArte e com a criação da ferramenta Ligeiro, conforme ilustrado pela Figura 5.3. Nesse contexto, o MDArte tem o papel de gerar artefatos, com base em modelos UML, que descrevem componentes de sistemas de informação, sendo utilizados diretamente pelo Ligeiro e também através do extrator de dependências Dependency Finder. Como resultado, o

Ligeiro realiza a APF e calcula automaticamente o valor não ajustado de pontos de função.

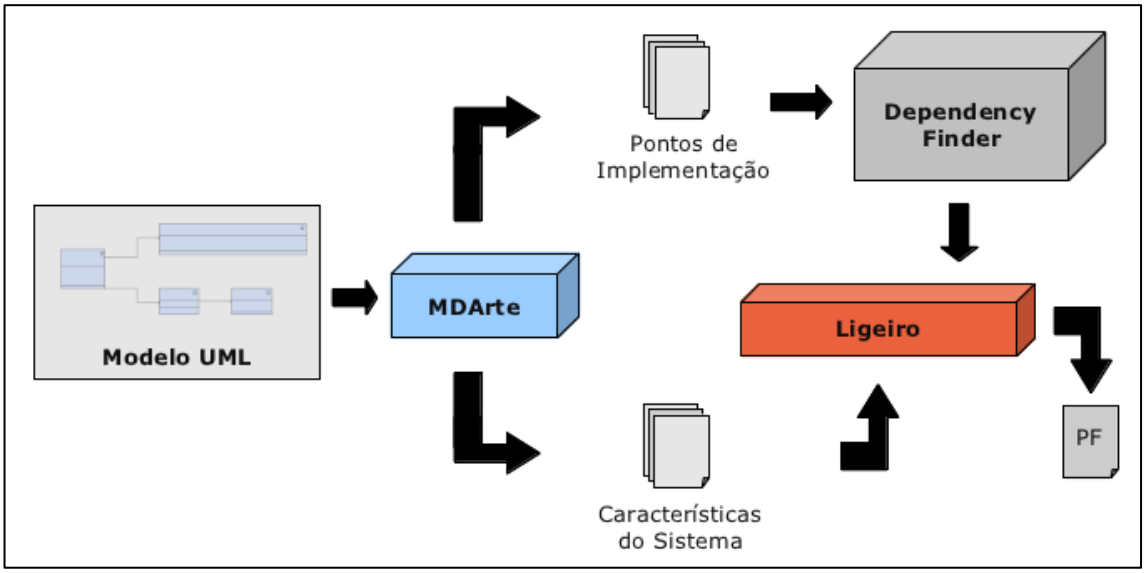

Figura 5.3 – Diagrama da implementação da proposta.

A ferramenta foi desenvolvida em duas partes, como biblioteca que pode ser utilizada por outras aplicações e como um *plug-in* da ferramenta de desenvolvimento Eclipse (ECLIPSE, 2012). A ferramenta Eclipse foi escolhida para acolher o Ligeiro por ser largamente utilizada por projetos que desenvolvem com o MDArte, representando uma vantagem em seu uso. A Figura 5.4 exibe a tela do Ligeiro para Eclipse. Através dos botões "mais" e "menos", das áreas de estatísticas e dependências, é possível adicionar os arquivos com as características do sistema e com as relações de dependência, respectivamente. Além disso, o *plug-in* permite ao usuário carregar um arquivo de configuração, em XML (do inglês, *Extensible Markup Language*), que contém os valores de complexidade da IFPUG, representado pelas tabelas do Capítulo 2, que serão efetivamente utilizados nos cálculos dos pontos de função.

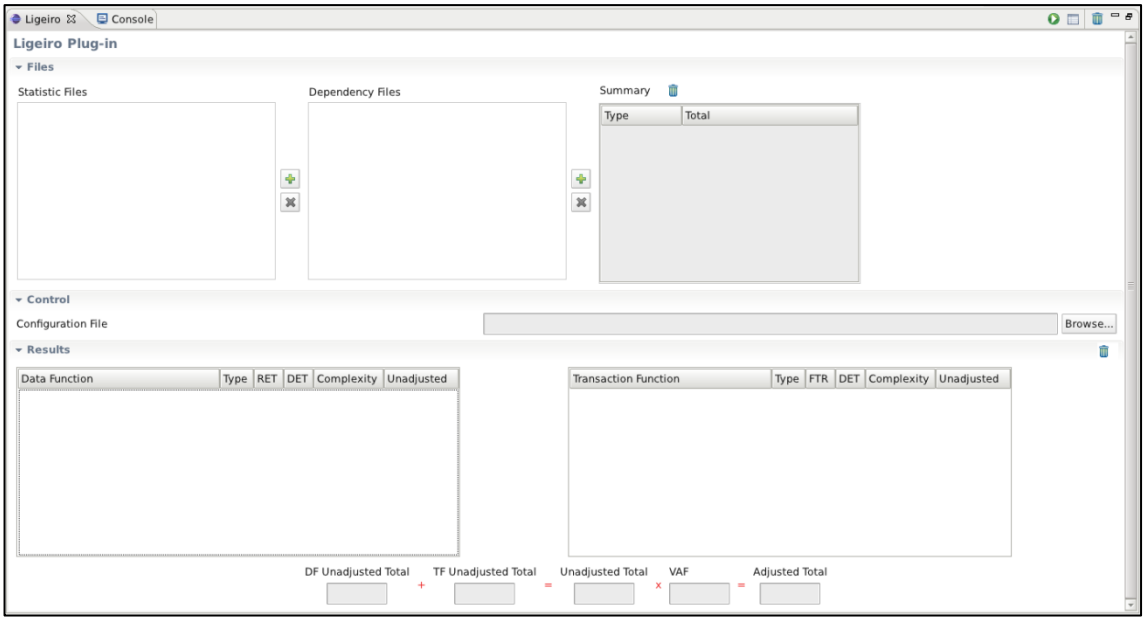

Figura 5.4 – *Plug-in* do Ligeiro para Eclipse.

Ambos os tipos de artefatos, os arquivos com as características do sistema e os pontos de implementação (fonte para construção das relações de dependências), são necessários para que a APF seja realizada. Dessa forma, as próximas duas subseções descreverão as características do sistema e as relações de dependências.

5.1.1 Características do Sistema

Durante o desenvolvimento de aplicações com uso do MDArte, em geral, apenas artefatos do código-fonte e pontos de implementação são gerados pelo *framework*. A geração dos artefatos com as características do sistema é opcional e foi criada para obtenção de estatísticas e realização da APF. Esses artefatos permitem que o Ligeiro não dependa diretamente de modelos no formato UML e possa receber como entrada um sumário dos componentes modelados, permitindo que a ferramenta futuramente possa ser utilizada em outros ambientes. Esses artefatos são gerados no formato XML e cada arquivo descreve apenas uma camada da aplicação. Assim, para cada modelo e cada camada da aplicação presente no modelo, será gerado pelo menos um arquivo que descreva as características dos componentes modelados. A Figura 5.5 ilustra um arquivo, gerado pelo MDArte, com as características de entidades da camada de persistência.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<statistics>
 <application>SistemaAcademico</application>
 <type>entities</type>
 <entities>
  <entity>
   <name>br.mdarte.exemplo.academico.cd.Pessoa</name>
   <attributes>
    <size>1</size>
    <attribute identifier="false">
     <name>nome</name>
     <type>java.lang.String</type>
    </attribute>
   </attributes>
   <methods>
    <size>2</size>
    <method modifier="true">
     <name>setNome</name>
     <return>
      <type>void</type>
     </return>
     <parameters>
      <size>1</size>
      <parameter>
       <name>nome</name>
       <type>java.lang.String</type>
      </parameter>
     </parameters>
    </method>
    <method modifier="false">
     <name>getNome</name>
     <return>
      <type>java.lang.String</type>
     </return>
     <parameters>
      <size>0</size>
     </parameters>
    </method>
   </methods>
  </entity>
  <entity>
   <name>br.mdarte.exemplo.academico.cd.Estudante</name>
   <extends>br.mdarte.exemplo.academico.cd.Pessoa</extends>
   <attributes>
    <size>0</size>
   </attributes>
   <methods>
    <size>0</size>
   </methods>
  </entity>
 </entities>
 <entitiesTotal>2</entitiesTotal>
</statistics>
```

Figura 5.5 – Exemplo de XML com as características de entidades da camada de persistência.

Com base no artefato contendo as características de uma entidade, é possível verificar quais são seus atributos e métodos. Além disso, o artefato também informa se um método é responsável ou não por modificar a entidade. No exemplo da Figura 5.5, a entidade "br.mdarte.exemplo.academico.cd.Pessoa" possui um atributo "nome" e dois métodos, "setNome" e "getNome", sendo que apenas o método "setNome" é capaz de modificar a entidade, quando executado.

Análogo ao arquivo gerado para a camada de persistência, o artefato com as características de serviços da camada de negócio é gerado como ilustrado pela Figura 5.6. Ele contém informações sobre cada serviço, listando as características de seus métodos e algumas estatísticas. Além disso, para cada serviço, é informado seu ponto de implementação relacionado e alguns outros aspectos da classe de serviço.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<statistics>
 <application>SistemaAcademico</application>
 <type>services</type>
 <modules>
  <module>
   <name>estudante</name>
   <services>
    <size>1</size>
    <service>
     <name>br.mdarte.exemplo.academico.cs.estudante.EstudanteHandler</name>
     <implementationName>br.mdarte.exemplo.academico.cs.estudante.EstudanteHand
lerBeanImpl</implementationName>
     <otherNames>
      <size>2</size>
      <value>br.mdarte.exemplo.academico.cs.estudante.EstudanteHandlerBI</value
>
      <value>br.mdarte.exemplo.academico.cs.estudante.pbi.EstudanteHandlerPBI</
value>
     </otherNames>
     <methods>
      <size>5</size>
      <method>
       <name>listaEstudantes</name>
       <implementationName>handleListaEstudantes</implementationName>
       <return>
        <type>java.util.Collection</type>
       </return>
       <parameters>
        <size>1</size>
        <parameter>
         <name>estudanteVO</name>
         <type>br.mdarte.exemplo.academico.vo.EstudanteVO</type>
        </parameter>
        <parameter>
         <name>paginacao</name>
         <type>java.lang.Integer</type>
        </parameter>
       </parameters>
      </method>
      <method>
```

Figura 5.6 – Exemplo de XML com as características de serviços da camada de negócio.

Com base no exemplo anterior, tem-se que o serviço de nome "br.mdarte.exemplo.academico.cs.estudante.EstudanteHandler" possui ponto de implementação de nome "br.mdarte.exemplo.academico.cs.estudante.EstudanteHandlerBeanImpl" e cinco métodos, dentre eles o "listaEstudantes". Apesar de serem meramente descritivas, essas informações possibilitam utilizar as relações de dependências e localizar dependências

relacionadas ao serviço em questão, conforme abordaremos nas seções seguintes sobre o processo de contagem.

O MDArte também gera artefatos com as características de casos de uso da camada de apresentação. Esses arquivos seguem a estrutura exibida pela Figura 5.7 e contém informações sobre diagramas de atividade e classes de controle associados a casos de uso.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<statistics>
 <application>SistemaAcademico</application>
 <type>usecases</type>
 <modules>
  <module>
   <name>sistema</name>
   <useCases>
    <size>4</size>
    <useCase>
     <name>br.mdarte.exemplo.academico.uc.sistema.estudante.atualiza.AtualizaEst
udante</name>
     <controller>
      <name>br.mdarte.exemplo.academico.uc.sistema.estudante.atualiza.AtualizaEs
tudanteControle</name>
      <implementationName>br.mdarte.exemplo.academico.uc.sistema.estudante.atual
iza.AtualizaEstudanteControleImpl</implementationName>
      <methods>
       <size>2</size>
       <method>
        <name>carregaValores</name>
        <return>
         <type>void</type>
        </return>
        <parameters>
         <size>4</size>
         <parameter>
          <name>mapping</name>
          <type>org.apache.struts.action.ActionMapping</type>
         </parameter>
         <parameter>
```

Figura 5.7 – Exemplo de XML com as características de casos de uso da camada de apresentação.

A figura anterior exibe o trecho de um artefato com algumas características do caso de uso "br.mdarte.exemplo.academico.uc.sistema.estudante.atualiza.AtualizaEstudante". Analisando o artefato de características, é possível identificar que esse caso de uso está associado a classe de controle "br.mdarte.exemplo.academico.uc.sistema.estudante.atualiza.AtualizaEstudanteControle ", e que ela possui dois métodos, sendo um deles o "carregaValores". Assim como para serviços, essas informações são úteis para associar elementos de modelos com a relação de dependências extraídas dos pontos de implementações.

5.1.2 Relações de Dependências

A geração de relações de dependências permite que informações relevantes à APF sejam extraídas do código-fonte e ao mesmo tempo cria uma camada de abstração entre linguagem de programação do sistema de informação sendo contado e a ferramenta de contagem. O extrator Dependency Finder foi escolhido devido a sua boa capacidade de lidar com a linguagem Java, dentre outros fatores já citados. Ao receber um pacote JAR como entrada, o Dependency Finder gera um arquivo XML com as relações de dependências daquele pacote.

A Figura 5.8 ilustra um arquivo XML gerado pelo Dependency Finder. Ele contém informações sobre a classe de controle do caso de uso "br.mdarte.exemplo.academico.uc.sistema.estudante.atualiza.AtualizaEstudante", que foi exemplificada pela Figura 5.7. Para cada classe presente no pacote JAR, o extrator lista as dependências de seus elementos, construtores (ou construtor, caso exista apenas um) e métodos. As dependências listadas consistem de classes, construtores e métodos que esses elementos acessam diretamente durante a execução. Por exemplo, com base na figura, pode-se verificar que o método "carregaValores" da classe "br.mdarte.exemplo.academico.uc.sistema.estudante.atualiza.AtualizaEstudanteControleImpl" depende da entidade "br.mdarte.exemplo.academico.cd.Estudante", contudo ele acessa apenas métodos do tipo "get", que não modificam suas informações.

Esse tipo de análise é semelhante às realizadas pela ferramenta Ligeiro e permitem, por exemplo, identificar FTs que mantém pelo menos uma FD do sistema de informação. Nas próximas duas seções, o uso das relações de dependências e os outros artefatos serão explicados detalhadamente, assim como regras de contagem de pontos de função.

```
<?xml version="1.0" encoding="utf-8" ?>
<dependencies>
 <package confirmed="yes">
 <name>br.mdarte.exemplo.academico.uc.sistema.estudante.atualiza</name>
  <class confirmed="yes">
   <name>br.mdarte.exemplo.academico.uc.sistema.estudante.atualiza.AtualizaEstudant
eControleImpl</name>
   <outbound type="class"
confirmed="yes">br.mdarte.exemplo.academico.uc.sistema.estudante.atualiza.AtualizaE
studanteControle</outbound>
   <feature confirmed="yes">
    <name>br.mdarte.exemplo.academico.uc.sistema.estudante.atualiza.AtualizaEstudan
teControleImpl.AtualizaEstudanteControleImpl()</name>
    <outbound type="feature"
confirmed="yes">br.mdarte.exemplo.academico.uc.sistema.estudante.atualiza.AtualizaE
studanteControle.AtualizaEstudanteControle()</outbound>
    <inbound type="feature"
confirmed="yes">br.mdarte.exemplo.academico.uc.sistema.estudante.atualiza.AtualizaE
studanteControleFactory.static {}</inbound>
   </feature>
   <feature confirmed="yes">
    <name>br.mdarte.exemplo.academico.uc.sistema.estudante.atualiza.AtualizaEstudan
teControleImpl.carregaValores(org.apache.struts.action.ActionMapping,
br.mdarte.exemplo.academico.uc.sistema.estudante.atualiza.CarregaValoresForm,
javax.servlet.http.HttpServletRequest,
javax.servlet.http.HttpServletResponse)</name>
    <outbound type="class"
confirmed="no">br.mdarte.exemplo.academico.cd.Estudante</outbound>
    <outbound type="feature"
confirmed="no">br.mdarte.exemplo.academico.cd.Estudante.getId()</outbound>
    <outbound type="feature"
confirmed="no">br.mdarte.exemplo.academico.cd.Estudante.getMatricula()</outbound>
    <outbound type="feature"
confirmed="no">br.mdarte.exemplo.academico.cd.Estudante.getNome()</outbound>
    <outbound type="feature"
confirmed="no">br.mdarte.exemplo.academico.cd.Estudante.getStatusMatricula()</outbo
und>
    <outbound type="class"
confirmed="no">br.mdarte.exemplo.academico.cs.estudante.EstudanteHandlerBI</outboun
d>
    <outbound type="feature"
confirmed="no">br.mdarte.exemplo.academico.cs.estudante.EstudanteHandlerBI.listaEst
udantes(br.mdarte.exemplo.academico.vo.EstudanteVO, java.lang.Integer)</outbound>
    <outbound type="class"
```

Figura 5.8 – Exemplo de XML com relação de dependência de um controle de caso de uso.

5.2 A Contagem de Funções de Dados

A proposta definida pelo trabalho determina que as FDs sejam contadas com base nas características do sistema de informação e nas relações de dependência, em conjunto. Na prática, o uso desses arquivos permite que o Ligeiro consiga analisar e identificar os tipos de funções para então determinar sua complexidade e chegar ao valor não ajustado de pontos de função.

Em sua implementação, o Ligeiro utiliza a lista de todas as relações de dependências para obter as relações que realmente são relevantes a APF, *i.e.*, para obter as relações de dependências de componentes do sistema que possuam pontos de

implementação. Para isso, o Ligeiro utiliza as características do sistema para permitir a identificação desses componentes e poder reduzir a lista de relações a ser trabalhada.

Para cada componente que possua ponto de implementação, serviços e controles de casos de uso, o Ligeiro verifica seus métodos e obtém suas dependências. Nesta etapa, as dependências dos métodos são analisadas em busca daqueles que acessam entidades do sistema. Assim, o Ligeiro utiliza novamente as características do sistema para descobrir as entidades que foram modeladas e poder identificar esses métodos.

Uma vez que os métodos que utilizam entidades são identificados, o Ligeiro verifica se dentre as dependências de cada método desses componentes estão métodos modificadores de entidades. Ou seja, a ferramenta analisa as dependências em busca de métodos de entidades que permitam mantê-las. Essa informação sobre todos que permitem manter entidade é obtida com nos arquivos com as características do sistema, utilizando o atributo "modifier" da *tag* XML "method", conforme pode ser verificado no arquivo ilustrado pela Figura 5.5.

As entidades, que possuem seus métodos modificadores como dependências desses componentes, são classificadas como FDs do tipo ALI. Todas as outras entidades modeladas são classificadas como FDs do tipo AIE, representando entidades que provavelmente são mantidas por outro sistema de informação.

Após a identificação das FDs, as características do sistema são novamente utilizadas para que seja realizada a contagem do número de TDs. Assim, o Ligeiro realiza o somatório do número de atributos da entidade e do número de atributos herdados de outras entidades, desconsiderando todos os atributos identificadores, e adiciona o número de associações que identificam a entidade, obtendo o número de TDs.

Aproximando o valor do número de TRs como 1 (um), conforme descrito pela proposta, e utilizando o arquivo de configurações com os valores de complexidade, a ferramenta determina a complexidade de cada FD e calcula o valor não ajustado de pontos de função.

5.3 A Contagem de Funções de Transações

A contagem de FTs envolve, além da análise de FDs já identificadas, o uso de informações da camada de apresentação para que seja possível classificar os três diferentes tipos e calcular o valor em pontos de função. Na prática, o Ligeiro utiliza o arquivo com as características do sistema para identificar os componentes da camada de apresentação e as relações de dependência para descobrir se uma FT mantém alguma FD e assim poder classifica-la. Assim como na proposta, a implementação está dividida em duas etapas, consistindo a primeira da classificação de FTs do tipo EE e parte CE, e a segunda etapa da classificação de FTs do tipo SE e a outra parte do par CE.

Na primeira etapa, com base nas características da camada de apresentação, exemplificadas pela Figura 5.7, o Ligeiro obtém a lista de telas de cada caso de uso do sistema de informação alvo da APF. Para cada tela presente na lista, a ferramenta verifica se ela contém pelo menos um parâmetro de entrada (*e.g.*, campo de texto que permitam a entrada de informações), e em caso positivo, o Ligeiro recupera a relação de dependências da classe de controle do caso de uso. A partir dessa relação, é verificado se os métodos da classe de controle, que são executados por algum botão da tela, dependem de alguma FD do tipo ALI. Se o método depender de um ALI, a FT é então classificada como EE, pois ela mantém uma entidade do sistema de informação. Senão, ela é classificada como parte do par de CE, representando a primeira tela da dupla.

Na segunda e última etapa, a lista de telas de cada caso de uso é novamente consultada em busca daquelas que ainda não foram classificadas, *i.e.*, que não possuem nenhum parâmetro de entrada. Essas telas podem representar relatórios, resultados de consultas, *etc*. A seguir, para cada tela ainda não classificada, o Ligeiro consulta as características do sistema e verifica se ela é saída de alguma tela já classificada como um par de CE. Em caso positivo, a FT é classificada como a segunda parte do par CE, sendo associada a outra tela. Por fim, as telas que ainda não foram classificadas são definidas como FTs do tipo SE, por representarem telas que apenas exibem valores.

Tendo classificado todas as FTs, a ferramenta inicia a contagem do número de TDs. Para as EEs, o número de TDs é igual ao número de campos da tela (parâmetros) mais o comando, representado pelo o botão da ação. Para SEs, o número de TDs é igual ao número campos da tela (*e.g.*, informações em texto) mais o número de colunas de cada tabela presente na tela, caso exista alguma. Para CEs, o número de TDs é igual ao

somatório do número de parâmetros e comando da tela de consulta mais o número de campos da tela de resultados, também considerando as tabelas, como feito para SEs.

O Ligeiro também realiza a contagem do número de ARs. Ele é calculado através da busca de entidades, que foram classificadas como ALI ou AIE, fornecidas pelo arquivo de características do sistema, dentre os elementos contidos na relação de dependências da FT. Como descrito na proposta, cada FD é contada apenas uma vez, mesmo que mais de um atributo seja acessado. Além disso, para as FTs do tipo CE, o número de ARs representa o total de FDs distintas acessadas pelo par.

Utilizando o arquivo de configurações com os valores de complexidade, a ferramenta determina a complexidade de cada FT e calcula o valor não ajustado de pontos de função.

5.4 Executando o Ligeiro em Sistemas de Informação Reais

A ferramenta desenvolvida por este trabalho, o Ligeiro, foi executada em alguns sistemas de informação reais desenvolvidos com o MDArte, como pode ser verificado pela amostra da Tabela 5.1. Ela contém os valores obtidos pelo Ligeiro ao realizar a APF automática em dois sistemas, nomeados Sistema A e Sistema B.

Tabela 5.1 – Resultados da APF automática em sistema de informação reais.

	Valor Não Ajustado (PF)	
	Sistema A	**Sistema B**
Funções de Dados	338	33
Funções de Transações	95	35
Total	433	68

A Tabela 5.2 apresenta um sumário com as características contadas dos dois sistemas de informação. Apesar da grande diferença de tamanho dos sistemas amostrados, pode-se verificar que a contagem de pontos de função possui valor proporcional.

Tabela 5.2 – Características dos sistema de informação reais submetidos ao APF automática.

	Valor Contado	
	Sistema A	**Sistema B**
Dependências	43	16
Entidades	52	5
Serviços	6	2
Casos de Usos	25	10
Linhas de Código (geradas e implementadas)	363722	53437

Capítulo 6

6 Estudo de Caso

Neste capítulo será abordado um estudo de caso utilizado para experimentar e avaliar a ferramenta Ligeiro, desenvolvida para realizar a contagem automática de pontos de função com base em modelos e informações extraídas de pontos de implementação. Este estudo de caso foi introduzido por VAZQUEZ *et al.* (2011) como um exemplo prático da aplicação do processo de contagem de pontos de função. Como proposto, o exemplo consiste de um sistema de informação responsável por registrar a entrada e saída de funcionários de uma organização. Deve ser observado que o exemplo foi seguido conforme sua descrição original, contudo adaptações foram necessárias para se adequar a tecnologia utilizada no desenvolvimento, o *framework* MDArte.

Os valores aqui apresentados como resultados da contagem automática foram obtidos após inúmeras iterações e aperfeiçoamentos da ferramenta Ligeiro.

A Seção 6.1 abordará a contagem pontos de função de FDs do estudo de caso proposto. Dessa forma, o domínio do sistema será explorado para que seja mais bem compreendido o papel de cada entidade. No processo de contagem, serão utilizados valores propostos pelo IFPUG, que foram anteriormente introduzidos no Capítulo 2.

A Seção 6.2 apresentará a contagem de pontos de função de FTs do mesmo estudo. Primeiramente, os casos de usos que compõem o sistema de informação serão brevemente descritos e exemplificados por modelos UML, utilizados pelo framework MDArte. Para cada caso de uso, serão identificadas e classificadas as FTs que o representam. Tendo classificado as FTs, elas também serão contadas com base nos valores do IFPUG.

A Seção 6.3 descreverá e comparará os resultados obtidos pela APF através da ferramenta Ligeiro e pelos valores obtidos por VAZQUEZ *et al.* (2011). A seção também analisará as divergências encontradas e proverá informações sobre os fatores que resultaram nessa diferença.

6.1 A Contagem de Funções de Dados

A Figura 6.1 ilustra o diagrama de classes do estudo de caso. O diagrama contém três entidades:

- "Person": pessoa do sistema.
- "Reason": justificativa, de alteração de ponto ou inclusão em data passada, inserida por uma pessoa.
- "Track": apontamento de uma pessoa.

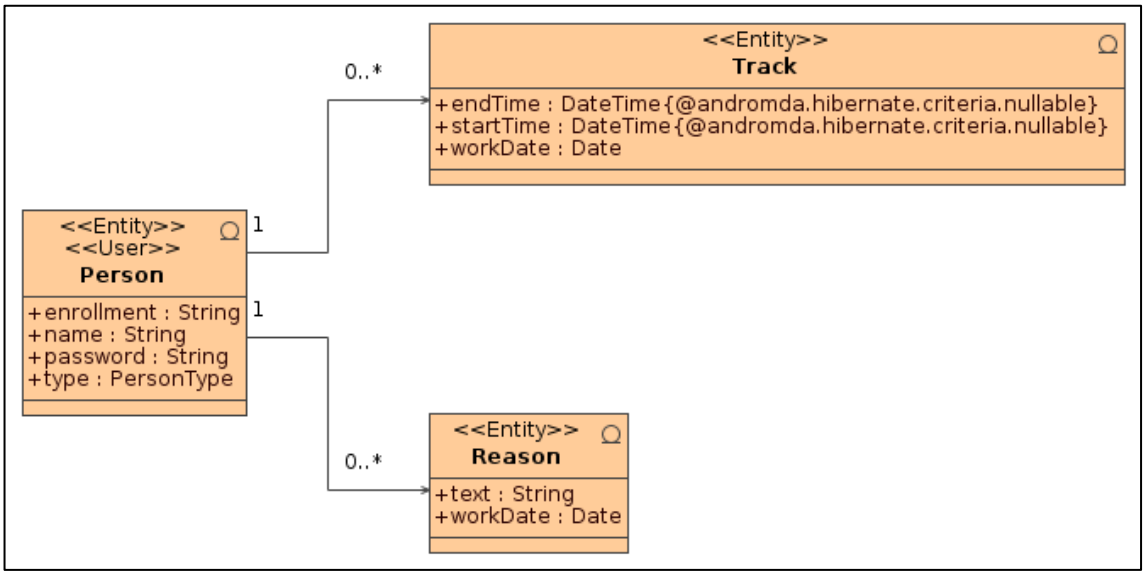

Figura 6.1 – Diagrama de classes do estudo de caso.

A entidade "Person" é mantida fora da fronteira do sistema, visto que ela é parte do controle de acesso, como pode ser inferido pelo estereótipo *«User»*. Assim, como descrito no capítulo anterior, a identificação das FDs é realizada procurando-se pelos métodos modificadores de entidade dentre as dependências do sistema de informação. Como resultado tem-se:

- ALI: "Reason" e "Track".
- AIE: "Person".

Após identificar cada FD, o processo de contagem de TDs e de TRs é iniciado. O número de TDs é definido pelo somatório do número de atributos com o número de atributos herdados, excluindo-se os atributos identificadores, mais o número de

associações identificadoras (*e.g.*, as entidades "Reason" e "Track" são identificadas pela associação com "Person"). Já o número de TRs tem seu valor aproximado de 1 (um), por se tratar de como o usuário identifica o sistema, conforme explicado no capítulo sobre a proposta. Aplicando os valores de Tabela 2.2 e Tabela 2.3, o total não ajustado de pontos de função para as FDs está definido em Tabela 6.1.

Tabela 6.1 – Resultado da contagem de FDs.

	Tipo	TR	TD	Complexidade	Valor
Person	AIE	1	4	Baixa	5
Reason	ALI	1	3	Baixa	7
Track	ALI	1	4	Baixa	7

6.2 A Contagem de Funções de Transações

O estudo de caso proposto por VAZQUEZ *et al.* (2011) contém cinco casos de usos, conforme apresentado pelo diagrama da Figura 6.2. Ele descreve um sistema em que o trabalhador registra suas entradas e saídas da organização, sendo necessário justifica-las quando se esquecer de registrar um apontamento ou for alterar um já registrado. Cada trabalhador tem acesso apenas às suas informações, e também pode consultar o total de horas trabalhadas em determinado período. O gerente pode emitir relatórios de presença de todos os funcionários.

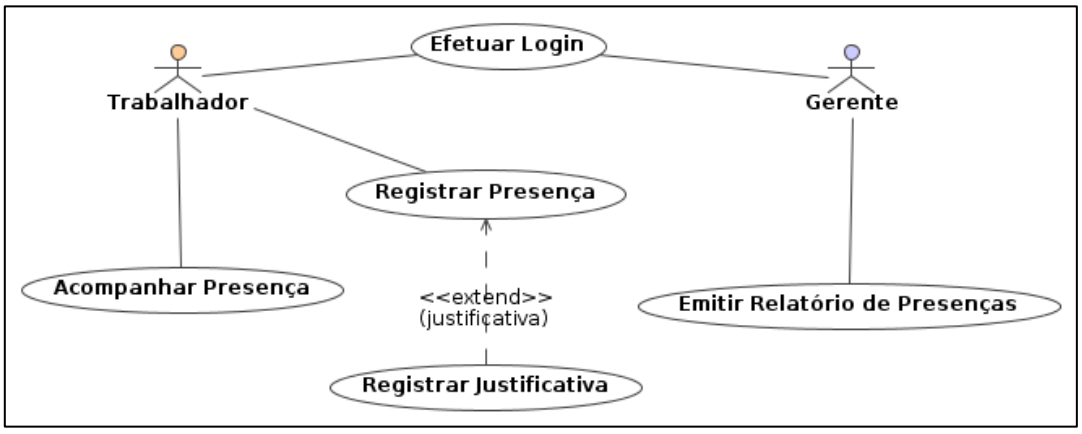

Figura 6.2 – Diagrama de casos de uso original do estudo de caso (VAZQUEZ, SIMÕES, ALBERT, 2011).

Nesse contexto, o caso de uso "Efetuar Login" será desconsiderado das análises seguintes, visto que o MDArte é responsável por gerar o mecanismo de *login*, o que não

possibilitaria sua comparação com o caso descrito no estudo de caso. Além disso, devido a limitações de modelagem e geração, o caso de uso "Registrar Justificativa" foi agregado ao "Registrar Presença". A Figura 6.3 exibe o diagrama de casos de uso que serão contados pela ferramenta Ligeiro. Os casos de uso "RegisterTime", "SearchTime" e "ReadReport" representam, respectivamente, os casos de uso "Registrar Presença", "Acompanhar Presença" e "Emitir Relatório de Presenças".

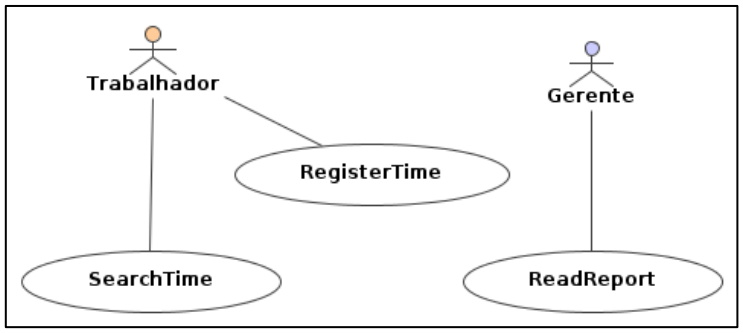

Figura 6.3 – Diagrama de casos de uso do estudo de caso.

O diagrama de atividades da Figura 6.4 ilustra o fluxo do caso de uso "RegisterTime", em que todas as atividades com estereótipo *«FrontEndView»* representam telas do sistema de informação, que devem ser geradas pelo MDArte. Essas telas simbolizam as FTs e são processadas pelo Ligeiro durante a APF.

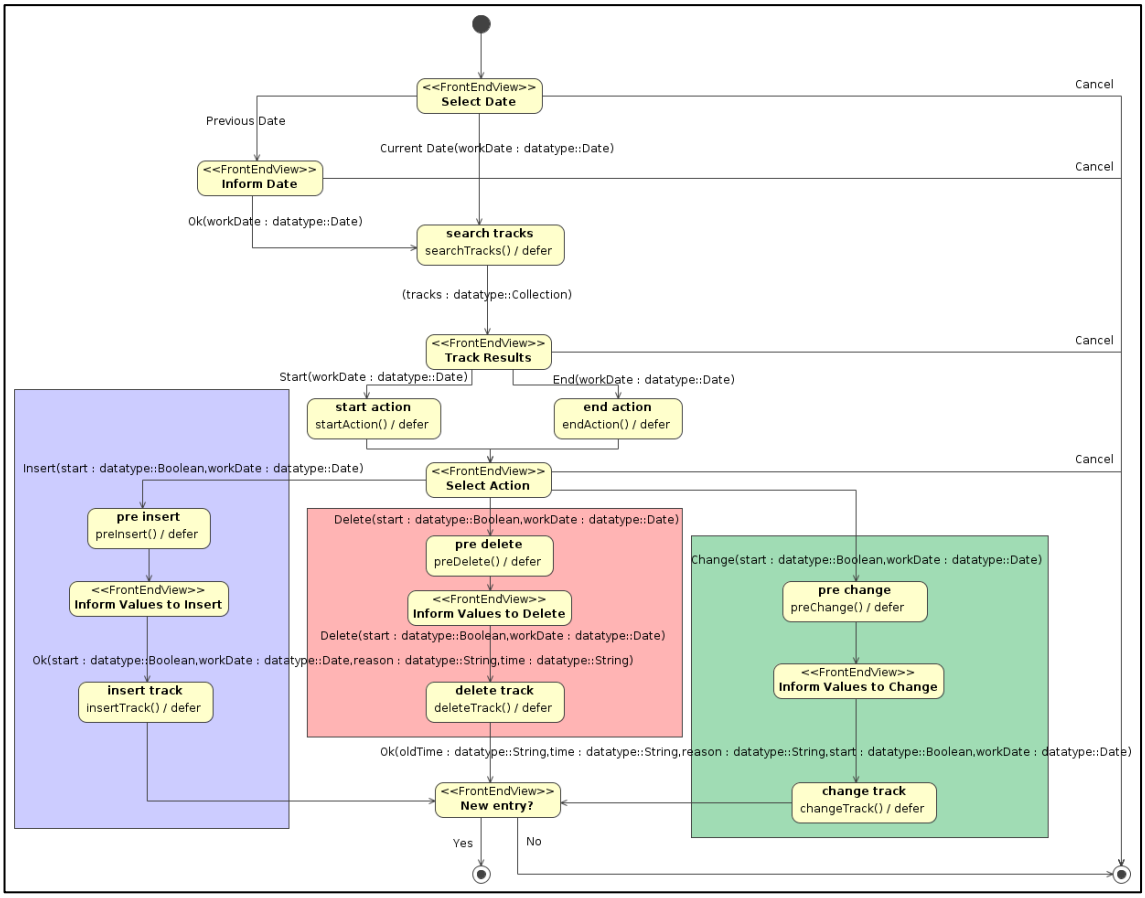

Figura 6.4 – Diagrama de atividades do caso de uso "RegisterTime".

A análise de FTs é realizada conforme descrito no capítulo sobre a implementação da ferramenta de APF, começando pela listagem das telas da aplicação e verificando as dependências de seus pontos de implementação em busca de métodos modificadores de entidades. Em seguida, cada FT é classificada e números de TDs e de ARs são calculados.

Com base nas relações de dependências, tem-se para o caso de uso "RegisterTime", que as FTs "Inform Values to Change", "Inform Values to Delete" e "Inform Values to Insert" são classificadas como EE, por conterem campos de entrada nas telas da aplicação e acessarem métodos modificadores de entidades. O número de TDs dessas FTs é contado conforme:

- "Inform Values to Change": possui valor 4, por conter um comando ("Ok") e três parâmetros ("old", "reason" e "time"). Vale ressaltar que os outros dois campos presentes no modelo são ocultos ao usuário. Campos ocultos representam valores de entrada, que, em geral, guardam informações

relevantes ao caso de uso (*e.g.*, o identificador de entidade que é mantida pelo caso de uso), não sendo relevante ao usuário.

- "Inform Values to Delete": possui valor 1, por conter um comando ("Ok"). Os dois parâmetros presentes no modelo são ocultos ao usuário, sendo utilizados internamente pelo sistema.

- "Inform Values to Insert": possui valor 3, por conter um comando ("Ok") e dois parâmetros ("reason" e "time"). Os outros dois parâmetros presentes no modelo são ocultos ao usuário.

Já o número de ARs é contado com base relações de dependências e analisando quantas FDs são referenciadas por cada FT. Tem-se que as FTs "Inform Values to Change" e "Inform Values to Delete" acessam as FDs "Reason" e "Track", enquanto a FT "Inform Values to Insert" acessa as FDs "Person", "Reason" e "Track". Logo, utilizando a Tabela 2.4 e a Tabela 2.6, é possível obter os valores presentes na Tabela 6.2.

Tabela 6.2 – Resultado da contagem de FTs do tipo EE.

	Tipo	**AR**	**TD**	**Complexidade**	**Valor**
Inform Values to Change	EE	2	4	Baixa	3
Inform Values to Delete	EE	2	1	Baixa	3
Inform Values to Insert	EE	3	3	Média	4

Com base nas características do sistema, tem-se que as FTs "New Entry?", "Select Action" e "Select Date" são classificadas como SE, por não possuírem nenhum campo de entrada em suas telas. Seus números de TDs são calculados conforme:

- "New Entry": possui valor 1, por conter um comando e nenhum parâmetro.

- "Select Action": possui valor 1, por conter um comando e nenhum parâmetro reconhecível pelo usuário. Os dois parâmetros indicados no modelo ("start" e "workDate") guardam informações relevantes apenas para o funcionamento do caso de uso, desconhecidos pelo usuário.

- "Select Date": possui valor 1, por conter um comando e nenhum parâmetro reconhecível pelo usuário. O parâmetro oculto "workDate" é utilizado internamente pela aplicação.

Já o número de ARs de cada SE identificada é calculado, como anteriormente, com base nas relações de dependências. As FTs "New Entry" e "Select Action" não acessam nenhuma FD e possuem valor 0 (zero). Já a FT "Select Date" acessa apenas a FD "Track" e possui valor 1.

A Tabela 6.3 contém o resultado da contagem das FTs do caso de uso "RegisterTime" classificadas como SE, com base na Tabela 2.5 e na Tabela 2.6.

Tabela 6.3 – Resultado da contagem de FTs do tipo SE.

	Tipo	AR	TD	Complexidade	Valor
New Entry	SE	0	1	Baixa	4
Select Action	SE	0	1	Baixa	4
Select Date	SE	1	1	Baixa	4

Da mesma forma, tem-se que a FD "Inform Date / Track Results" é classificada como CE. Ela é composta por duas telas, uma de busca ("Inform Date") e outra de resultado ("Track Results"). Seu número de TDs é igual a 4, sendo o resultado obtido pela contagem de ambas as telas. A tela "Inform Date" contém um comando ("Ok") e um parâmetro ("workDate"). Já a tela "Track Results" contém uma tabela com duas colunas, representado no modelo por "tracks". Apesar das colunas da tabela "tracks" não estarem exibidas no diagrama da Figura 6.4, elas estão presentes no modelo através do uso de valor etiquetado específico, que permite gerar esse tipo de informação nos artefatos com as características do sistema. Seu número de ARs é obtido pela a análise das dependências do método "searchTracks" do controle associado ao caso de uso e possui valor igual a 1, por referenciar a FD de nome "Track".

O diagrama de atividades da Figura 6.5 descreve o fluxo do caso de uso "SearchTime". Esse caso de uso representa claramente um mecanismo de consulta, em que uma tela recebe os filtros da pesquisa e a outra exibe os resultados. Dentro do contexto da APF, o Ligeiro classifica a FT "Search Time / Search Result" como CE devido a presença de campos de entrada na tela "Search Time" e o fato de nenhuma FD

ser mantida pela FT. Além disso, todos os parâmetros da tela “Search Result” são apenas de apresentação (*i.e.*, *plaintext*), o “name” e o “hourTotal”.

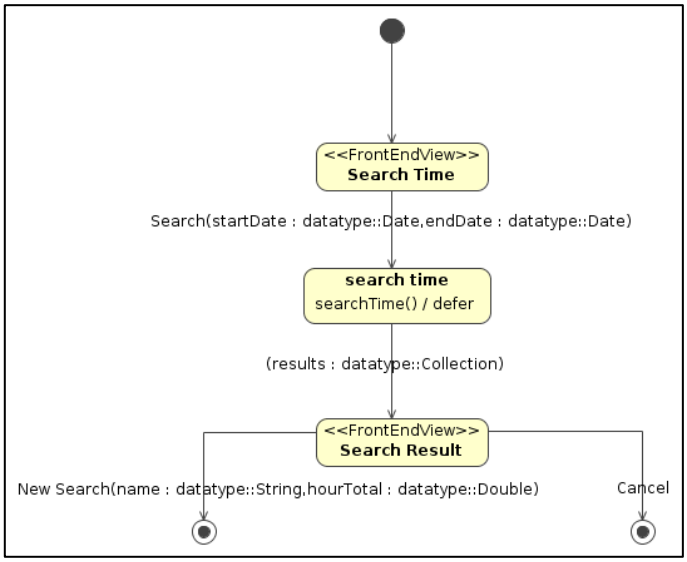

Figura 6.5 – Diagrama de atividades do caso de uso “SearchTime”.

Assim, a contagem dessa FT define que o número de TDs é igual a 9. A tela “Search Time” contém um comando (“Search”) e dois parâmetros (“startDate” e “endDate”). Já a tela “Search Result” dois parâmetros (“hourTotal” e “name”). Assim, como na contagem da outra CE, deve-se ser levado em consideração o número de colunas da tabela de resultados, que contém quatro colunas (“dayStr”, “hourStr”, “reason” e “type”). Seu número de ARs possui valor igual a 3, por referenciar as FDs “Person”, “Reason” e “Track”.

Por fim, o diagrama de atividades da Figura 6.6 apresenta o fluxo do caso de uso “ReadReport”. Assim como o caso de uso “SearchTime”, ele também representa um sistema de busca, tendo como diferença básica o botão de imprimir o resultado, ou seja, o relatório. A FT denominada “Search Report / Search Result” é classificada como CE, exatamente por se tratar de uma tela de pesquisa e outra de resultados, conforme descrito para a FT “Search Time / Search Result”.

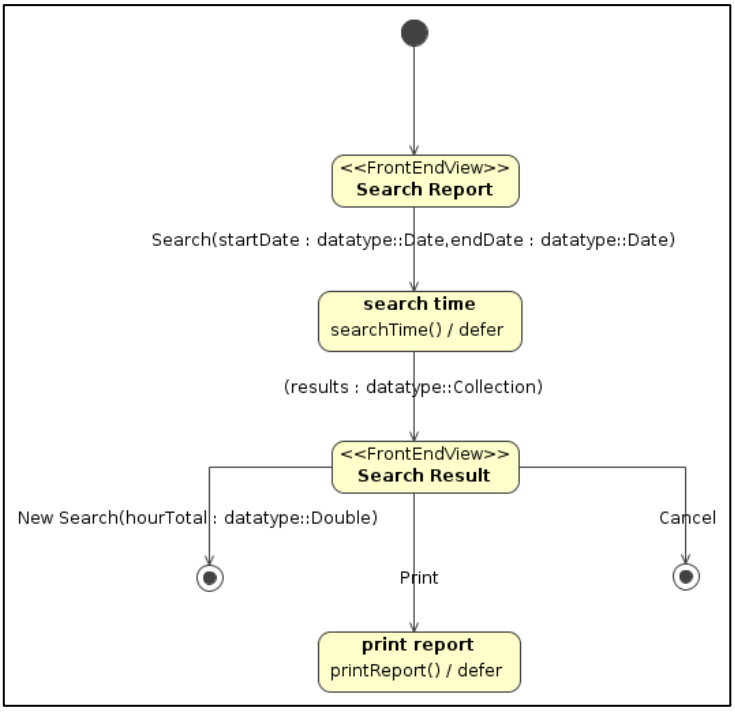

Figura 6.6 – Diagrama de atividades do caso de uso "ReadReport".

Assim como a outra CE, seu número de TDs também é igual a 8. A tela "Search Report" contém um comando ("Search") e dois parâmetros ("startDate" e "endDate"). Já a tela "Search Result" contém um parâmetro ("hourTotal"). Além disso, a tabela de resultados ("results") apresenta quatro colunas ("hourTotal", "personEnrollment", "personName" e "reasonTotal"). Seu número de ARs possui valor igual a 3, por referenciar as FDs "Person", "Reason" e "Track".

A Tabela 6.4 contém os valores obtidos para todas as FTs classificadas como CE, com base na Tabela 2.5 e na Tabela 2.6.

Tabela 6.4 – Resultado da contagem de FTs do tipo CE.

	Tipo	AR	TD	Complexidade	Valor
Inform Date / Track Results	CE	1	4	Baixa	3
Search Report / Search Result	CE	3	8	Média	4
Search Time / Search Result	CE	3	9	Média	4

6.3 Analisando os Resultados

Os resultados apresentados nas duas últimas seções podem ser sumarizados com o uso da ferramenta Ligeiro, como exibido pela tela da Figura 6.7. Além de contar os valores já apresentados, o Ligeiro também utiliza informações do arquivo de

configurações para calcular a complexidade de cada função e o total ajustado em pontos de função, de acordo com o valor de ajuste representado na figura por VAF.

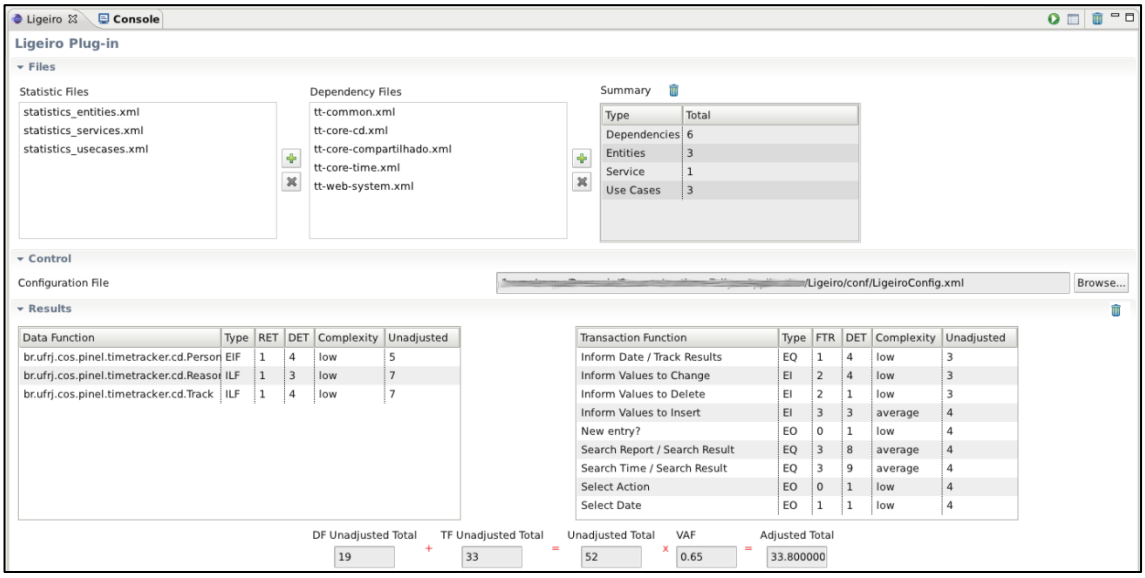

Figura 6.7 – Resultado da APF do estudo de caso utilizando o Ligeiro.

A Tabela 6.5 contém a comparação entre os casos de uso modelados e os descritos por VAZQUEZ *et al.* (2011). Além de listar os resultados obtidos, com o auxílio das tabelas do IFPUG apresentadas no Capítulo 2, ela também relaciona, por linha, funções semelhantes entre ambas às abordagens (*e.g.*, "Search Report / Search Result" e "Emitir Relatório de Presença"). Em complementar, a Tabela 6.6 representa a legenda aos campos destacados na Tabela 6.5. Ainda nesse contexto, o gráfico de colunas da Figura 6.8 exibe o valor não ajustado de pontos de função para cada par semelhante, permitindo uma fácil comparação dos resultados das duas abordagens (automática e manual).

Tabela 6.5 – Comparando os resultados obtidos pelo Ligeiro e por VAZQUEZ *et al.* (2011).

Ligeiro	Tipo	TR / AR	TD	Complexi-dade	PF	VAZQUEZ *et al.* (2011)	Tipo	TR / AR	TD	Complexi-dade	PF
Funções de Dados											
Person	AIE	1	4	Baixa	5	Pessoa	AIE	1	4	Baixa	5
Reason	ALI	1	3	Baixa	7	Justificativa	ALI	1	3	Baixa	7
Track	ALI	1	4	Baixa	7	Apontamento	ALI	1	4	Baixa	7
Funções de Transações											
Inform Date / Track Results	CE	1	**4**	Baixa	3	Consulta Apontamento Diário	CE	1	**5**	Baixa	3
Inform Values to Change	EE	2	**4**	Baixa	3	Alteração de Apontamento	EE	2	**5**	Média	4
Inform Values to Delete	EE	2	**1**	Baixa	3	Exclusão de Apontamento	EE	2	**2**	Baixa	3
Inform Values to Insert	EE	**3**	3	Média	4	Registro de Ponto	EE	**1**	3	Baixa	3
New entry?	SE	0	1	Baixa	**4**						
Search Report / Search Result	**CE**	3	**8**	Média	4	Emitir Relatório de Presença	**SE**	3	**9**	Média	5
Search Time / Search Result	**CE**	3	**9**	Média	4	Acompanhar Presença	**SE**	3	**10**	Média	5
Select Action	SE	0	1	Baixa	**4**						
Select Date	SE	1	1	Baixa	**4**						
						Apontamento c/ Justificativa	EE	2	5	Média	**4**
Total					**52**	**Total**					**46**

Tabela 6.6 – Legenda da comparação de resultados da Tabela 6.5.

Exibição de Mensagens	Padrão de Modelagem
Limite de Tecnologia e Representação	Limite de Representação
Operações de Cálculo	Outros (em desacordo)

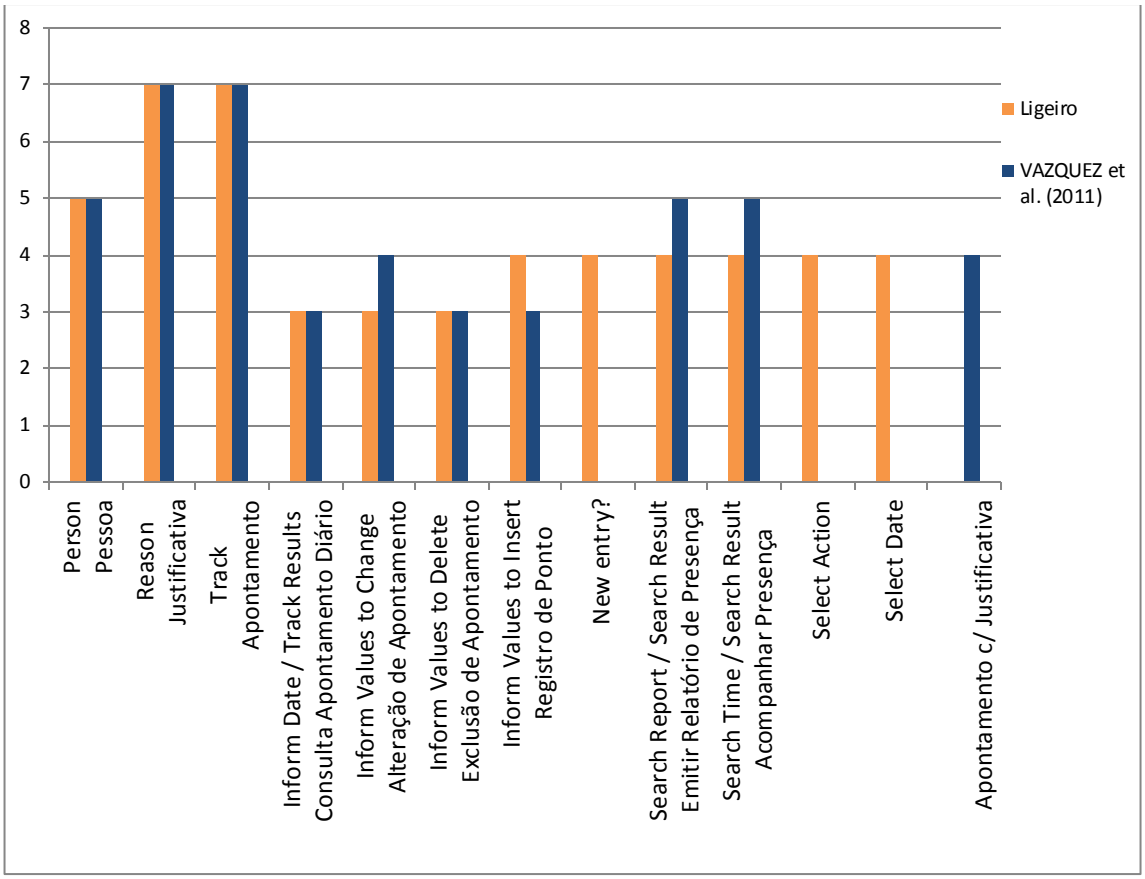

Figura 6.8 – Gráfico com os resultados obtidos pelo Ligeiro e por VAZQUEZ *et al.* (2011).

Ao observar os resultados presentes na tabela, verifica-se que os valores totais calculados pelas abordagens são diferentes, tendo o processo automático obtido o maior valor. Essa disparidade resulta, em parte, do número de FTs identificadas em cada contagem, que advém da diferença de modelagem de casos de uso iguais. Apesar desses casos de uso serem os mesmos em especificação, o padrão de modelagem requerido pelo MDArte pode ser apontado como fonte da divergência.

Por ser um *framework* gerador de código com base em modelos, o MDArte espera que os modelos de entrada possuam certas características necessárias para a ativação da geração e também para a geração correta do código-fonte para a tecnologia da aplicação. Dessa maneira, modelos criados para a geração, e que por fim serão utilizados na APF, passam a conter valores que não estavam descritos na especificação do estudo de caso.

Analisando o par semelhante de "Inform Date / Track Results" e "Consulta Apontamento Diário", pode-se verificar que ambas as funções foram classificadas como CE e atribuídas a complexidade "Baixa". Contudo, existe uma pequena diferença no número de TDs contados em cada FT, que poderia influenciar na atribuição da complexidade caso as tabelas de complexidade (como as fornecidas no Capítulo 2) tivessem outros intervalos. Essa diferença existe, pois a análise manual conta também as mensagens de avisos da transação (que independe do número de mensagens e deve ser contada uma vez), não sendo possível de ser contado automaticamente com as informações fornecidas, *i.e.*, apenas com os artefatos com as características do sistema e as relações de dependências.

Assim como no caso anterior, a desconsideração das mensagens de avisos também pode ser observada nos pares semelhantes "Inform Values to Change" e "Alteração de Apontamento", e "Inform Values to Delete" e "Exclusão de Apontamento". Especialmente no par "Inform Values to Change" e "Alteração de Apontamento", a mudança no valor do número de TDs em uma unidade foi suficiente para que a complexidade atribuída a função fosse diferente da contagem manual. Como resultado, o valor não ajustado de pontos de função foi calculado como 3, em vez de 4.

Já o par semelhante "Inform Values to Insert" e "Registro de Ponto" também obteve a mesma classificação em ambas as abordagens. Contudo, devido a diferença no número de ARs contados, a complexidade atribuída durante contagem automática foi superior à manual, resultando em um valor não ajustado de pontos de função também diferente. Como na análise automática o número de ARs é contado com base nas relações de dependências, e não no que o usuário identifica diretamente como FD, esse valor fica sujeito a pequenos aspectos da tecnologia utilizada pela aplicação. Em nível de código, essa FT realmente utiliza três FDs ("Person", "Reason" e "Track"), e não apenas a FD "Track". Ademais, conforme introduzido na descrição deste estudo de

caso, o caso de uso "Registrar Presença" precisou agregar as funcionalidades do caso de uso "Registrar Justificativa", que justifica o fato da FD "Reason" ser referenciada pela FT "Inform Values to Insert".

Além do diferente número de funções identificadas pela a análise automática, que resultou em funções sem par semelhante, duas FTs (e suas semelhantes) chamam a atenção pela diferença entre suas classificações (em negrito):

- "Search Report / Search Result" e "Emitir Relatório de Presença";
- "Search Time / Search Result" e "Acompanhar Presença".

Ambos os casos representam áreas do sistema de informação que retornam alguma informação ao usuário, com base em uma consulta realizada. Sendo, dessa forma, classificado pelo Ligeiro como FTs do tipo CE. Contudo, conforme descrito por VAZQUEZ *et al.* (2011), e implementado na aplicação gerada pelo MDArte, essas duas funções são responsáveis por calcular e exibir o total de horas trabalhadas, que, conforme a definição do IFPUG, é uma característica de FTs do tipo SE.

Essa diferença na classificação de FTs é explicada pela falta de informação existente nos artefatos com as características do sistema e nas relações de dependências. Primeiramente, os modelos não sabem se determinado método de controle de caso de uso é responsável ou não por realizar alguma operação matemática, significando que os artefatos com as características do sistema também não conterão essa informação. Já as relações de dependências apenas contêm referências a elementos que são necessários para a execução dos diversos componentes da linguagem, sem ser capaz de fornecer qualquer dado relacionado a fórmulas matemáticas ou qualquer tipo de cálculo.

Assim como nos pares semelhantes já analisados, verifica-se que essas duas FTs também possuem números de TDs inferiores, em uma unidade, aos valores contados em suas FTs semelhantes identificadas na abordagem manual. Isso também é explicado pela não contagem das mensagens de aviso, que não é possível apenas com as informações disponíveis a ferramenta.

Para a análise de FDs, vale ressaltar que apesar do resultado encontrado na contagem ter sido o mesmo para as duas abordagens, as funções poderiam divergir quanto ao número de TRs, visto que o Ligeiro aproxima o número de TRs como 1 (um),

conforme descrito no Capítulo 5. Essa aproximação é justificada pelo fato desse número representar a quantidade de subgrupos lógicos de dados reconhecidos pelo usuário, o que não é possível de ser contando automaticamente, com base nas características do sistema e nas relações de dependências.

De forma complementar, a Tabela 6.7 apresenta a comparação entre os resultados obtidos pela contagem manual, seguindo as regras do Ligeiro, e os resultados obtidos por VAZQUEZ *et al.* (2011). Os valores também podem ser interpretados com o auxílio da legenda definida pela Tabela 6.6. A contagem manual foi realizada diretamente nas especificações dos casos de uso, levando em consideração a descrição de cada tela e forma como o autor teria originalmente contado as funções do estudo de caso.

É possível verificar três grupos de divergência:

- O composto por FTs que se diferenciam em uma unidade na contagem do número de TDs, representado a não contagem de mensagens exibidas ao usuário;
- O composto por FTs que se diferenciam quanto ao número de ARs, representado por "Registro de Ponto", que apresenta esse valor pela interpretação do contador (Analista de Métricas) quanto ao número de FD efetivamente utilizadas pela FT;
- O composto por FTs que apresentam diferença de classificação, por não ser considerado na contagem se a FT realiza ou não alguma operação matemática.

Apesar dos resultados aqui apresentados, tanto na contagem automática quanto na manual, esses valores seriam mais bem avaliados se uma contagem manual fosse realizada diretamente, por um especialista, na aplicação gerada pelo MDArte.

Tabela 6.7 – Comparando os resultados obtidos pela contagem manual seguindo as regras do Ligeiro e por VAZQUEZ *et al.* (2011).

	Contagem Manual (Regras do Ligeiro)					VAZQUEZ *et al.* (2011)				
	Tipo	**TR / AR**	**TD**	**Complexidade**	**PF**	**Tipo**	**TR / AR**	**TD**	**Complexidade**	**PF**
Funções de Dados										
Pessoa	AIE	1	4	Baixa	5	AIE	1	4	Baixa	5
Justificativa	ALI	1	3	Baixa	7	ALI	1	3	Baixa	7
Apontamento	ALI	1	4	Baixa	7	ALI	1	4	Baixa	7
Funções de Transações										
Consulta Apontamento Diário	CE	1	**4**	Baixa	3	CE	1	**5**	Baixa	3
Alteração de Apontamento	EE	2	**4**	Baixa	3	EE	2	**5**	Média	4
Exclusão de Apontamento	EE	2	**1**	Baixa	3	EE	2	**2**	Baixa	3
Registro de Ponto	EE	**2**	**2**	Média	4	EE	**1**	**3**	Baixa	3
Emitir Relatório de Presença	**CE**	3	**8**	Média	4	**SE**	3	**9**	Média	5
Acompanhar Presença	**CE**	3	**9**	Média	4	**SE**	3	**10**	Média	5
Apontamento c/ Justificativa	EE	2	**4**	Baixa	3	EE	2	**5**	Média	4
	Total				**43**	**Total**				**46**

Capítulo 7

7 Conclusão e Trabalhos Futuros

Este trabalho descreveu um novo método de análise automática de pontos de funções em aplicações geradas com base em modelos, através de *frameworks* MDA de geração parcial de código. Foi apresentado como as características de sistemas de informação, presentes em modelos, são obtidas e como as relações de dependências são extraídas de parte de seus códigos-fonte (*i.e.*, dos pontos de implementação). A partir dessas informações é descrito como a análise automática de pontos de função é realizada, demonstrando o processo de identificação dos dois tipos de funções (dados e transações) e como é estabelecida a contagem de suas características. Como resultado da automatização, tem-se o valor não ajustado de pontos de função, de acordo com o método IFPUG.

Não obstante, o trabalho também apresenta o desenvolvimento da ferramenta Ligeiro, que segue a proposta elaborada e facilita a contagem automática de pontos de função. Apesar de não fazer parte da proposta, como explicado na Seção 2.1 do Capítulo 2, o Ligeiro também pode se configurado para que o valor ajustado de pontos de função seja calculado, sendo flexível a necessidade do projeto alvo da contagem.

Com o objetivo de validar a proposta e a ferramenta desenvolvida, foi utilizado o estudo de caso proposto por VAZQUEZ *et al.* (2011). Os valores obtidos na abordagem automática da APF foram comparados com os da abordagem manual e suas diferenças analisadas. Foi verificado que apesar dos limites existentes na automatização, ela apresentou bom desempenho, considerando o que realmente foi desenvolvido em modelos e pontos de implementação. A contagem automática de Funções de Dados (FDs) apresentou bom desempenho e a de Funções de Transações (FTs), apesar de também ter sido próxima a manual, sugeriu algumas melhorias, como a contagem de mensagens exibidas ao usuário e a verificação se algum cálculo matemático é realizado ou não pela FT.

Esses valores seriam mais bem avaliados se uma contagem manual fosse realizada diretamente, por um especialista, na aplicação gerada pelo MDArte. Assim, pode-se considerar como trabalho futuro, a realização da contagem manual, por um

especialista IFPUG, de uma aplicação gerada pelo MDArte, que seria também submetida ao processo automático. Dessa forma, a comparação realmente seria feita com relação a mesma aplicação, e não a mesma especificação, podendo levar ao refinamento das regras de contagem para que o valor calculado se aproxime ainda mais com o proposto pelo IFPUG.

Também como trabalho futuro, pode-se considerar o refinamento da contagem do número de TRs de FDs, visto que este trabalho usa um valor aproximado para o mesmo. Isso poderia ser feito, por exemplo, explorando as relações entre entidades modeladas, como associações e composições. Outro ponto significativo é poder futuramente adicionar ou utilizar mais informações dos modelos, para diminuir ao máximo a o uso das relações de dependências.

Além disso, este trabalho tem como contribuição, o ambiente necessário para a criação de uma base de dados contendo as características dos modelos submetidos à APF e os valores obtidos na contagem. Essa base possibilitaria estimar a complexidade, em pontos de função, de sistemas de informação que ainda não foram implementados, *i.e.*, que possuem apenas as especificações em modelos. Esse procedimento seria extremamente útil na fase inicial de projetos que dependem da contagem de pontos função para terem seus escopos aprovados ou contratos assinados.

Referências Bibliográficas

ABRAHÃO, S., DE MARCO, L., FERRUCCI, F., GRAVINO, C., SARRO, F., 2010. "A COSMIC measurement procedure for sizing web applications developed using the OO-H method". *In Proceedings of the Workshop on Advances in Functional Size Measurement and Effort Estimation - FSM '10*, ACM Press , pp. 1-8, New York, New York, USA.

ABRAHAO, S., INSFRAN, E., 2008. "A Metamodeling Approach to Estimate Software Size from Requirements Specifications". *In 2008 34th Euromicro Conference Software Engineering and Advanced Applications*, IEEE, pp. 465-475.

ABRAHÃO, S., MENDES, E., GOMEZ, J., INSFRAN, E., 2007. "A Model-Driven Measurement Procedure for Sizing Web Applications: Design, Automation and Validation", In: *Model Driven Engineering Languages and Systems*, Springer Berlin / Heidelberg, pp. 467-481.

ABRAN, A., 2010, *Software Metrics and Software Metrology*, 1 ed., IEEE Computer Society, New Jersey, New Jersey, USA.

ALBRECHT, A. J., 1979, "Measuring Application Development Productivity". *Proceedings of the Joint SHARE/GUIDE/IBM Application Development Symposium*, New York, New York, USA.

ANDROMDA, 2012, "AndroMDA". Disponível em http://www.andromda.org. Acesso em: 22 jan. 2012, 20:56:07.

BATISTA, V. A., PEIXOTO, D. C. C., BORGES, E. P., PÁDUA, W., RESENDE, R. F., PÁDUA, C. I. P. S., 2011. "ReMoFP: A Tool for Counting Function Points from UML Requirement Models". *Advances in Software Engineering*, Hindawi Publishing Corp, pp.1-7.

BOOCH, G., RUMBAUGH, J., JACOBSON, I, 1998, *The Unified Modeling Language User Guide*, 2 ed., Addison-Wesley.

CANTONE, G., PACE, D., CALAVARO, G., 2004. "Applying function point to unified modeling language: conversion model and pilot study". *METRICS '04*

Proceedings of the Software Metrics, 10th International Symposium, IEEE, pp. 280-291, Washington, DC, USA.

CERI, S., FRATERNALI, P., BONGIO, A., 2000. "Web Modeling Language (WebML): a modeling language for designing Web sites", In *Computer Networks*, v. 33, n. 1-6, pp. 137-157.

COSMIC, 2009, "COSMIC Measurement Manual: Release 3.0.1". *Common Software Measurement International Consortium*.

CZARNECKI, K., HELSEN, S., 2003. "Classification of Model Transformation Approaches". *OOPSLA2003 Workshop on Generative Techniques in the Context of MDA*, Anaheim, CA, USA.

DEPENDENCY FINDER, 2012, "Dependency Finder". Disponível em http://depfind.sourceforge.net. Acesso em: 09 abr. 2012, 19:31:03.

EBERT, C., DUMKE, R., BUNDSCHUH, M., SCHMIETENDORF, A., DUMKE, R., 2004, *Best Practices in Software Measurement*, 1 ed., Springer Berlin.

ECLIPSE, 2012, "Eclipse". Disponível em http://www.eclipse.org. Acesso em: 20 abr. 2012, 08:57:35.

FETCKE, T., ABRAN, A., NGUYEN, T., 1997. "Mapping the OO-Jacobson approach into function point analysis". *In Proceedings of TOOLS USA 97: International Conference on Technology of Object Oriented Systems and Languages*, pp. 192-202, Santa Barbara, CA, USA.

FRANKEL, D. S., 2003, *Model Driven Architecture: Applying MDA to Enterprise Computing*, 1 ed., Wiley.

FRATERNALI, P., TISI, M., BONGIO, A., 2006. "Automating Function Point Analysis with Model Driven Development". *In Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research*, Proceedings of CASCON 2006, ACM, New York, New York, USA.

GARMUS, D., HERRON, D., 2000, *Function Point Analysis: Measurement Practices for Successful Software Projects*, 1 ed., Addison-Wesley.

GOMEZ, J., CACHERO, C., PASTOR, O., 2001. "Conceptual Modeling of Device-Independent Web Applications", *IEEE Multimedia*, v. 8, n. 2, pp. 26-39.

GUTTMAN, M., PARODI, J., 2006, *Real-Life MDA: Solving Business Problems with Model Driven Architecture*, 1 ed., Morgan Kaufmann.

HARPUT, V., KAINDL, H., KRAMER, S., 2005. "Extending Function Point Analysis of Object-Oriented Requirements Specifications". *In 11th IEEE International Software Metrics Symposium (METRICS'05)*, IEEE, pp. 39-39, Washington, DC, USA.

HO, V. T., ABRAN, A., 1999. "A Framework for Automatic Function Point Counting From Source Code". *International Workshop on Software Measurement (IWSM'99)*, Lac Supérieur, Québec, Canada.

IFPUG, 2010, "Function Point Counting Practices Manual: Release 4.3". *International Function Point Users Group*.

IEC, 2012, "International Engineering Consortium". Disponível em http://www.iec.org. Acesso em: 23 jan. 2012, 15:14:38.

IORIO, T., 2004. "IFPUG Function Point analysis in a UML framework". *Proceedings of Software Measurement European Forum*, Roma, Italia.

ISO, 2012, "International Organization for Standardization". Disponível em http://www.iso.org. Acesso em: 23 jan. 2012, 15:13:02.

JACOBSON, I., 1992, *Object Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley Professional.

KLEPPE, A., WARMER, J., BAST, W., 2003, *MDA Explained: The Model Driven ArchitectureTM: Practice and Promise*, 1 ed., Addison-Wesley.

LAVAZZA, L. A., DEL BIANCO, V., GARAVAGLIA, C., 2008. "Model-based functional size measurement". *In Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement - ESEM '08*, ACM Press, p. 100, New York, New York, USA.

LEVESQUE, G., BEVO V., CAO, D. T., 2008. "Estimating software size with UML models". *In Proceedings of the 2008 C3S2E conference on - C3S2E '08*, ACM Press, p. 81, New York, New York, USA.

MARÍN, B., CONDORI-FERNANDEZ, N., PASTOR, O., ABRAN, A., 2008a. "Measuring the Functional Size of Conceptual Models in an MDA Environment", *In 20th International Conference on Advanced Information Systems Engineering*. pp. 33-36.

MARÍN, B., PASTOR, O., ABRAN, A., 2010. "Towards an Accurate Functional Size Measurement Procedure for Conceptual Models in an MDA Environment", *Data & Knowledge Engineering*, v. 69, n. 5, pp. 472-490.

MARÍN, B., PASTOR, O., GIACHETTI, G., 2008b. "Automating the Measurement of Functional Size of Conceptual Models in an MDA Environment", In: *Software Process and Product Measurement*, Springer Berlin / Heidelberg, pp. 170-183.

MDARTE, 2011, "MDArte: Framework de Desenvolvimento". *Linux Magazine Especial*, v. 6, pp. 40-41, São Paulo, SP, Brasil, Junho 2011.

MDARTE, 2012, "MDArte". Disponível em http://www.softwarepublico.gov.br/ver-comunidade?community_id=9022831. Acesso em: 22 jan. 2012, 20:31:35.

MENDES, E., MOSLEY, N., COUNSELL, S., 2005. "Investigating Web size Metrics for Early Web Cost Estimation", In *Journal of Systems and Software*, v. 77, n. 2, pp. 157-172.

OMG, 2012, "Object Management Group". Disponível em http://www.omg.org. Acesso em: 22 jan. 2012, 20:25:12.

PATON, K., 1999. "Automatic Function Point Counting Using Static and Dynamic Code Analysis". In *International Workshop on Software Measurement (IWSM'99)*, Lac Supérieur, Québec, Canada.

PINEL, R. E. A., CARMO, F. B. do, MONTEIRO, R. S, ZIMBRÃO, G., 2011, "Improving Tests Infrastructure through a Model-Based Approach". *ACM SIGSOFT Software Engineering Notes*, v. 36, n. 1 (Jan), pp. 1-5.

SIEGEL, J., the OMG Staff Strategy Group, 2001, "Developing in OMG's Model Driven Architecture". *OMG white paper*.

SILVA, M. A. N., FREITAS, K., MONTEIRO, R. S., SOUZA, J. M., 2010, "Automatically generating Component Dependency Diagrams in a Model Driven Development Scenario". In *I Brazilian Workshop on Model - Driven Development - CBSOFT*, Salvador, BA, Brazil.

SOLEY, R., the OMG Staff Strategy Group, 2000. "Model Driven Architecture". *OMG white paper*.

UEMURA, T., KUSUMOTO, S., INOUE, K., 2001. "Function-point analysis using design specifications based on the unified modelling language". *Journal of Software Maintenance: Research and Practice*, v. 13, n. 4, pp.223-243.

VAZQUEZ, C. E., SIMÕES, G. S., ALBERT, R. M., 2011, *Análise de Pontos de Função: Medição, Estimativas e Gerenciamento de Projetos de Software*, 11 ed. São Paulo, SP, Brasil, Editora Érica.

WEBRATIO, 2012, "Site Development Studio". Disponível em http://www.webratio.com. Acesso em: 18 mar. 2012, 17:56:12.

Anexos

Anexo 1

1 Introdução ao MDArte

1.1 Introdução

Em 2001, a OMG[1] publicou a MDA, uma metodologia para determinar diretrizes a serem seguidas na geração de código a partir de modelos. Ela propõe o uso de padrões que facilitam o trabalho com modelos e códigos gerados, como o emprego da UML na representação de modelos.

A MDA é uma abordagem que permite reduzir tempo e custo no desenvolvimento de sistemas de informação, visto que em torno de 80% do código da aplicação é gerado com base nas informações modeladas. Não obstante, isso permite que modelos, parte importante da documentação, estejam sempre representando o estado atual do sistema em desenvolvimento.

Devido a essas características, muitos grupos iniciaram a produção de ferramentas que possibilitassem a geração de código a partir de modelos ou que apoiassem essa atividade. No Brasil, somado a necessidade do Governo Brasileiro de ter uma ferramenta padrão de desenvolvimento, tivemos a criação do projeto MDArte[2], uma extensão do conhecido framework AndroMDA[3], que realiza a geração de código a partir da abordagem MDA.

O MDArte é um framework composto por um *kernel* e cartuchos. O *kernel* representa o núcleo de geração, responsável pela leitura de modelos e por transmitir suas informações aos cartuchos, que determinam o código a ser gerado. Cada cartucho é organizado de forma a agregar características de mesma tecnologia, sendo que diferentes tipos de cartuchos estão disponíveis: EJB, Hibernate, Java, JUnit e Struts.

---

[1] http://www.omg.org/
[2] http://www.softwarepublico.gov.br/dotlrn/clubs/mdarte
[3] http://www.andromda.org

1.2 Introdução à MDA

A MDA corresponde a um novo paradigma que envolve novas metodologias no desenvolvimento de software. Ela se baseia no antigo princípio de separação da especificação das funcionalidades do sistema dos detalhes de implementação.

Para tanto, a MDA divide o desenvolvimento em três níveis de modelagem: o Modelo Independente de Computação (*Computation Independent Model* – CIM), o Modelo Independente de Plataforma (*Platform Independent Model* – PIM) e o Modelo Específico de Plataforma (*Platform Specific Model* – PSM). A Figura 1 representa o esquema de transformação envolvendo os três níveis de modelos propostos.

O CIM é considerado o modelo de nível mais alto. Ele descreve o sistema dentro de seu ambiente (domínio de negócio), apresentando o que é esperado que o sistema fizesse, sem detalhes de como isto será feito. Apesar de ser uma representação importante, ela é pouco utilizado por ferramentas MDA, visto as informações fornecidas não são suficientes para representar etapas internas de sistemas de informações. Dessa forma, frameworks como MDArte utilizam um representação próxima ao PIM, que posteriormente é transformado em PSM e, finalmente, esse é transformado em código-fonte.

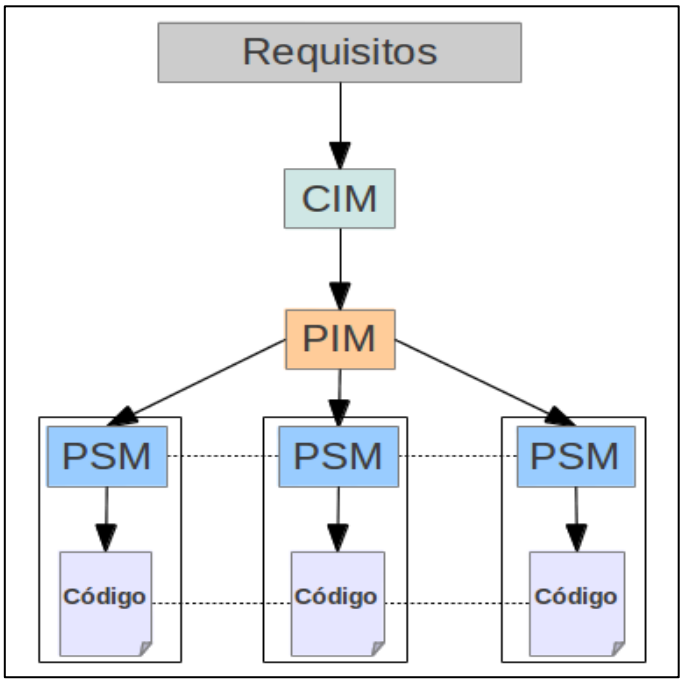

Figura 1 – Esquema de transformação entre CIM, PIM, PSM e código.

O PIM representa uma visão computacional independente de plataforma de implementação de negócio, ou seja, o PIM mostra as funções e a estrutura do sistema mas esconde informações específicas de tecnologia. Já o PSM, representa o sistema em nível mais baixo, especificando elementos particulares de plataforma ou tecnologia, sendo uma representação mais próxima do código.

A MDA encoraja a especificação de sistemas de informação em PIM, com o apoio do padrão UML. A UML é a linguagem utilizada para especificar modelos de aplicações de maneira que esses sejam independentes de plataforma. Dessa forma, a MDA utiliza o padrão MOF (*Meta Object Facility*) para mapear os elementos presentes no PIM e obter o PSM.

O MOF é padrão advindo da própria UML. Ele propõe a arquitetura de meta modelagem para definir a UML. Essa arquitetura permite às ferramentas MDA terem modelos UML, conhecidos por *metamodelos*, que internamente representam como o modelo UML de um sistema de informação deve ser trabalhado e seu código gerado. Os *metamodelos* são compostos por elementos *metafacades*, que efetivamente indicam essa representação e fornecem informações aos módulos (cartuchos) responsáveis pela geração de código.

Em linhas gerais, a MDA toma como entrada um modelo PIM e o transforma em um modelo PSM. O modelo PSM é então utilizado para gerar o código-fonte do sistema de informação. As etapas de modelagem, transformação e geração são apoiadas por padrões como UML e MOF, dentre outros.

1.3 A Estrutura do MDArte

O MDArte é um framework de geração de código por modelos UML. Ele possibilita a geração de sistemas de informação a partir de PIMs que os especifiquem. Isso é possível através de uma estrutura bem definida baseada nos conceitos da metodologia MDA.

Sua estrutura pode ser entendida por um *kernel* e seus cartuchos. O *kernel* é o núcleo responsável por carregar as informações dos modelos de entrada e apoiar todos os mecanismos de transformação. Ele utiliza diretamente o padrão MOF, possuindo uma camada conhecida como *metafacades*, que mapeia os elementos de entrada e os

disponibiliza aos cartuchos. A Figura 2 apresenta o modelo de relacionamento entre *metafacades* e cartuchos.

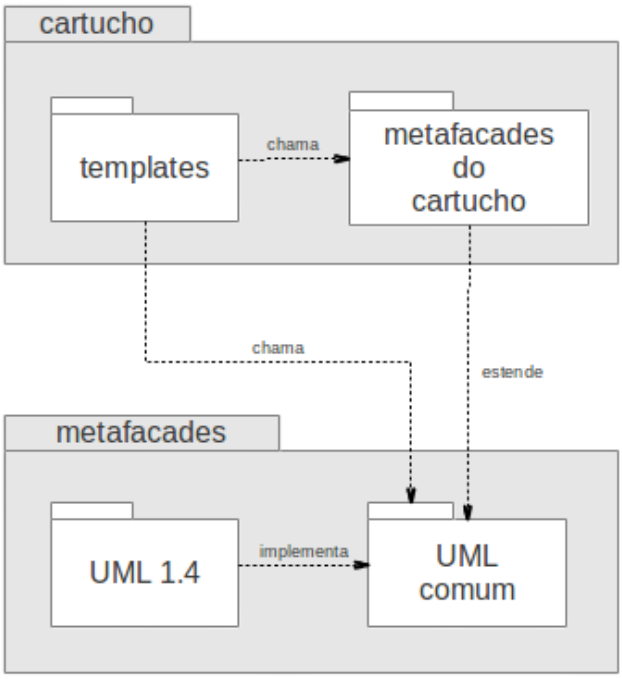

Figura 2 – Relacionamento entre *metafacades* e cartuchos.

A partir do modelo apresentado é possível verificar que o conjunto de *metafacades* é subdividido em grupos menores, que representam a hierarquia de *metafacades*. O subgrupo UML comum consiste da visão geral os elementos UML esperados como entrada do MDArte, sem considerar a versão da UML. Dessa forma, se faz necessário um conjunto de *metafacades* específicos, por exemplo, da versão 1.4 da UML, permitindo representar situações especificas da versão. Isso possibilita a evolução do framework no sentido de suportar diferentes tipos de modelos com entrada sem que a parte comum às versões precise ser rescrita.

Analogamente, com o objetivo de manter a independência entre cartuchos e versões UML, os *metafacades* comuns a UML são estendidos pelos cartuchos e outra camada de abstração é criada. Os cartuchos passam a representar suas próprias características dentro de seu escopo, e.g., como determinado elemento UML deve ser tratado e interpretado para uma devida geração. Esses elementos passam a estar disponíveis ao subgrupo de *templates*, que representam os gabaritos de códigos. Os *templates* consistem da parte que realmente decide o que será gerado, através das informações obtidas dos *metafacades* por chamadas diretas.

Contudo, como os *templates* são realmente acionados para geração? Cada cartucho possui também quatro arquivos responsáveis por guardar as configurações de *templates* e *metafacades*, dentre outras informações.

- **cartridge.xml**: Guarda a lista de todos os *templates* ativos presentes no cartucho. Para cada *template* é definido um conjunto de características que se verificadas no modelo de entrada, acionam o *template* para geração. Essas características consistem, na maior parte dos casos, de elementos *metafacades* e suas propriedades.

- **metafacades.xml**: Guarda a lista de todos os *metafacades* do cartucho, que estão disponíveis para a geração. Para cada *metafacade* é definido o elemento UML ao qual ele está relacionado e que características devem ser atendidas para que o mapeamento ocorra.

- **namespace.xml**: Define todas as propriedades que podem ser configuradas para o *namespace* do cartucho, i.e., define que parâmetros podem ser passados ao cartucho. Dependendo de como a propriedade é trabalhada no processo de geração, qualquer mudança nos valores dessas propriedades pode ter efeito direto no resultado final da geração, a aplicação. Assim, cada projeto que utiliza o MDArte para geração de sistemas de informação pode configurar os cartuchos para que a geração tenha o comportamento desejado.

- **profile.xml**: Defini que informações dos profiles UML podem ser utilizadas pelo cartucho. Essas informações consistem de extensões UML (estereótipos e valores etiquetados) que podem ser atribuídos ao elementos UML do modelo de entrada.

A definição do padrão UML prevê o uso de estereótipos e valores etiquetados para que os modelos descritos por UML possam agregar mais valor e serem mais precisos em sua representação. Na próxima seção, serão abordados os aspectos importantes da geração e descrito como estereótipos e valores etiquetados são utilizados.

1.4 Aspectos da Geração

O MDArte realiza a geração automática de código a partir de modelos UML. Em sua versão estável, o MDArte trabalha apenas com a versão 1.4 do padrão UML. Contudo, a versão experimental suporta não só as versões 1.4 e 2.0 da UML, como também o padrão de modelagem Eclipse UML 2.x. Isso é possível através de sua estrutura flexível e, principalmente, pelo uso do padrão MOF, que permite aos cartuchos lidarem com *metafacades* dos elementos UML lidos sem manter uma relação sólida com o padrão de modelagem utilizado.

Com o objetivo de aumentar a adaptabilidade de modelos UML e favorecer a criação de modelos refinados, a UML propõe três tipos de extensibilidade, dos quais todos são utilizados pelo MDArte para aumentar o poder de geração. Os modelos produzidos para geração com o framework utilizam estereótipos e valores etiquetados para descrever seus elementos, e framework utiliza restrições UML para realizar as validações desses modelos.

1.4.1 Estereótipos

O uso de estereótipos permite que o vocabulário UML seja estendido para se cria novos elementos, derivados de outros que já existam, com o objetivo de representar melhor problemas particulares do domínio a ser especificado.

Os estereótipos são apresentados entre aspas angulares (*« »* ou, se não disponível, << >>) e adicionada acima dos nomes dos elementos UML. A Figura 3 apresenta elemento UML "Pessoa" do tipo classe com o estereótipo *«Entity»*, que o torna uma entidade do sistema especificado pelo modelo.

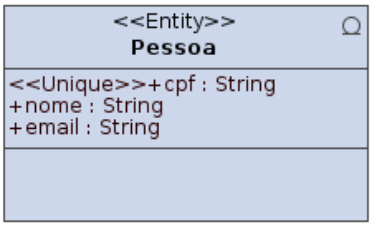

Figura 3 – Exemplo de elemento UML com estereótipo

Pela figura, também é possível verificar o uso do estereótipo *«Unique»* no elemento UML "cpf" do tipo atributo, que nesse contexto define que o elemento "cpf"

deve ser único para cada elemento “Pessoa”. Portanto, os estereótipos tiveram participação importante na especificação de características do domínio em questão, em que cada “Pessoa” deve ter um “cpf” único.

A partir desse pequeno exemplo, o cartucho Hibernate poderia ser ativado para gerar todas a estrutura relacionada a manipulação da entidade “Pessoa”. Desde o script SQL a componentes Java que permitem lidar facilmente com a entidade.

1.4.2 Valores Etiquetados

Assim como os estereótipos, os valores etiquetados sevem para melhorar a representação de domínios em modelos UML. Apesar de apresentarem pequenas diferenças nas versões UML 1.4 e 2.0, eles podem ser compreendidos como propriedade adicional aos estereótipos.

Diferentemente de estereótipos, os valores etiquetados permitem a adição de valores livres (ou não) aos elementos UML. Eles são representados por *strings* separadas pelo caractere ponto '.' e podem ter (ou não) o caractere arroba '@' como prefixo, aparecendo abaixo do nome do elemento. A Figura 4 ilustra o mesmo elemento presente na Figura 3, contudo ele agora possui um valor etiquetado associado.

Utilizando o nome do valor etiquetado associado ao elemento UML “Pessoa”, é possível inferir que ele está relacionado a camada de persistência e define um comentário livre que pode ser associado a entidade. Contudo, não são todos os valores etiquetados que permitem a entrada de texto livre como parâmetro. A UML também suporta a criação de valores etiquetados do tipo enumerado, em que é possível selecionar apenas um valor dentre os apresentados, como uma combo-box. Isso permite limitar entradas que seriam inválidas para o objetivo do valor etiquetado.

Em adição ao exemplo descrito na seção anterior, o cartucho Hibernate receberia o valor associado ao valor etiquetado ilustrado e geraria o comentário no script SQL de criação do banco de dados, assim como no arquivo *hbm.xml* da entidade.

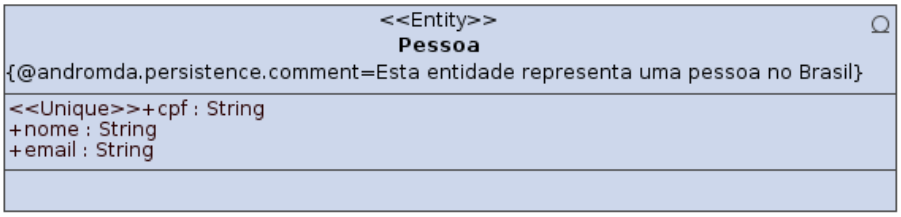

Figura 4 – Exemplo de elemento UML com valor etiquetado.

1.4.3 Restrições

As restrições (do inglês *constraints*) representam o suporte do padrão UML para lidar com situações que, por exemplo, precisem de alguma validação. Para tanto, o MDArte utiliza a linguagem declarativa OCL para descrever suas restrições UML, que procuram garantir que os modelos utilizados estão no formato esperado como entrada e são válidos para geração de código.

Elas são apresentadas entre chaves e exibidas abaixo do nome do elemento. A Figura 5 apresenta parte do modelo UML de *metafacades* do cartucho JUnit. Ele contém o elemento UML "JUnitActivityGraph" do tipo classe com uma restrição OCL. Pelo texto contido na restrição, é possível inferir que estando no contexto da *metafacade* "JunitActivityGraph", o atributo "firstAction" não pode ser vazio, ou seja, deve estar definido.

Esse é um exemplo prático de como as restrições são importantes para garantir o bom funcionamento do framework. Nesse mesmo exemplo, ao utilizar um diagrama de atividade em que não existe estado inicial, uma exceção é lançada e o processo de geração de código é interrompido, exibindo uma mensagem relacionada ao problema encontrado.

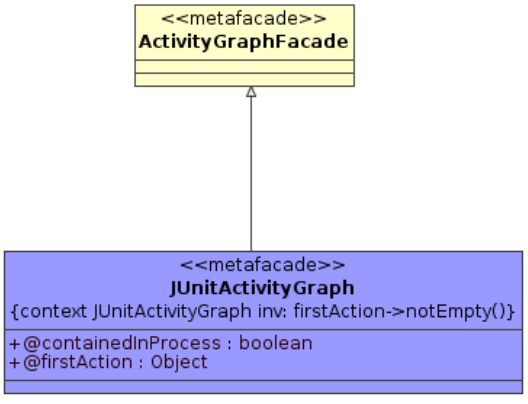

Figura 5 – Exemplo de elemento UML com restrição OCL.

1.5 Projetos Gerados com o MDArte

O primeiro passo da utilização do MDA no desenvolvimento de sistemas de informação consiste na geração da estrutura do projeto, que pode variar de acordo com o conjunto de tecnologias que serão utilizadas pela aplicação. Nessa etapa, são definidas as configurações iniciais e os cartuchos que estarão disponíveis durante a geração do código-fonte a partir dos modelos. Contudo, essas informações podem ser alteradas a qualquer momento, garantindo total configuração do ambiente.

O MDArte e seus projetos utilizam a ferramenta Maven[4] para gerenciar a construção de projetos e geração de código. Assim, as dependências de projetos são armazenadas conforme o padrão definido pelo Maven, em um repositório local *.maven* que pode ser encontrado na raiz do diretório do usuário. Esse diretório varia conforme o sistema operacional utilizado e guarda as ultimas versões dos componentes do MDArte e dos projetos gerados por ele.

[4] http://maven.apache.org/

Anexo 2

1 Modelagem da Camada de Domínio

A camada de domínio tem como finalidade definir as entidades e seus relacionamentos representando todo o conjunto de dados contidos no domínio de negócio apoiado pela aplicação. Todos os dados representados nessa camada poderão ser persistidos (salvos) em uma base de dados. A modelagem da camada de dados deve ser feita através de diagramas de classes da UML. Todas as classes que representam entidades deverão possuir o estereótipo *«Entity»*. Resumidamente, a camada de dados deve conter:

- Classes representando entidades.
- Atributos de classes e seus tipos de dados.
- Relacionamentos entre classes, dos tipos associação, generalização ou dependência.
- Cardinalidades mínimas e máximas dos relacionamentos do tipo associação.

1.1 O Que São Entidades e Relacionamentos?

O modelo ER (*Entity-Relationship model*), proposto por Peter Chen, corresponde a uma abstração de elementos do mundo real pertencentes ao domínio da aplicação procurando reter as suas principais características necessárias ao processamento de dados em sistemas de informação. As entidades do modelo ER correspondem a objetos reais do domínio da aplicação. Entidades contêm atributos que detalham informações específicas do objeto. Por exemplo, em um domínio de aplicação voltado para a gestão de um ambiente acadêmico podemos identificar as entidades Aluno, Professor, Curso, Disciplina, Inscrição, etc. Para a entidade Aluno podemos identificar atributos como Nome, Matrícula, Data de Ingresso, etc.

Os objetos reais de um domínio de aplicação não são estáticos ou mesmo isolados uns aos outros. Ao invés disso, podemos identificar durante a análise do domínio da aplicação interações entre os mesmos. Essas interações são representadas através de relacionamentos entre entidades no modelo ER.

1.2 Representação do Modelo de Entidade-Relacionamento Através de Diagramas de Classe da UML

Dentre os diagramas disponíveis na UML o diagrama de classes é o mais adequado para modelagem de dados, principalmente por possuir um mapeamento direto com o modelo ER, onde as classes correspondem a entidades e relacionamentos a interações entre entidades.

Ao usar o diagrama de classes para modelar a camada de dados de uma aplicação gerada pelo MDArte devem ser observados os seguintes padrões:

- Diagramas de classes devem estar contidas no pacote *<PACOTE_PROJETO>.cd*, onde *<PACOTE_PROJETO>* é o pacote definido para o projeto.
- Entidades devem receber o estereótipo *«Entity»*.
- Atributos de entidades devem ter visibilidade pública.

Pela Figura 1 é possível verificar como uma entidade é modelada.

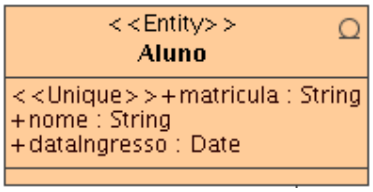

Figura 1 – Exemplo de modelagem da camada de dados.

Anexo 3

1 Modelagem da Camada de Serviços

A camada de serviço tem como finalidade abrigar as regras de negócio da aplicação. Ela é responsável por realizar ações sob as informações da camada de dados. A modelagem da camada de serviço deve ser feita através de diagramas de classe UML. Todos os serviços deverão possuir o estereótipo *«Service»*. Resumidamente, a camada de serviços deve conter:

- Classes representando serviços.
- Métodos de classes representando as operações daquele serviço.
- Relacionamento entre classes, dos tipos generalização e dependência.

1.1 O Que São Serviços?

Os serviços provêm um padrão para implementar código de regras de negócio, tipicamente encontradas em aplicações empresariais. Esses códigos eram comumente utilizados para o mesmo tipo de problema, e a solução era sua reimplementarão pelos desenvolvedores. O uso de serviços possibilita manipular esses problemas comuns oferecendo acesso ao domínio de persistência, integridade transacional e segurança.

No modelo de classes, os serviços são representados por classes que recebem o estereótipo *«Service»*. As classes especificadas se tornarão os serviços (API) da aplicação. Os serviços definidos no modelo se tornarão disponíveis através de Session Beans.

Os Session Beans são componentes de negócio. A lógica de negócio dos componentes EJB (site oficial) se encontra nesses componentes. Existem dois tipos de Componentes Session Bean, o Stateless Session Bean e o Stateful Session Bean. O *Stateless* é um componente de negócio que não mantém conversação com o usuário, não há garantia que chamadas sucessivas de métodos remotos vão ser feitas no mesmo objeto. O *Stateful* é um componente que mantém estado, com ele temos a garantia que chamadas sucessivas de métodos remotas, feitas por um mesmo cliente, serão processadas por um mesmo objeto.

Os *beans* EJB precisam ser modelados em um diagrama de classes. As classes destes *beans* precisam ter o estereótipo *«Service»*. Todas as classes de serviço devem estar no pacote <*PACOTE_PROJETO*>.*cs*. Além disso, o pacote *cs*, ou um pacote interno, deve ter o estereótipo *«ModuloServico»*. A Figura 1 ilustra a modelagem de um serviço de nome "EstudanteHandler".

Figura 1 – Exemplo de modelagem da camada de serviços.

Anexo 4

1 Modelagem da Camada de Apresentação

A camada de apresentação consiste na interface de comunicação com usuário. Seu objetivo é expor a lógica de negócios ao usuário e possibilitar sua interação com a aplicação. A modelagem da camada de apresentação deve ser feita através de diagramas de atividades, que determinam o fluxo de casos de usos.

1.1 O Que é o Diagrama de Atividades?

O Diagrama de atividade é um diagrama definido pela UML, e representa os fluxos conduzidos por processamentos. É essencialmente um gráfico de fluxo, mostrando o fluxo de controle de uma atividade para outra. Comumente isso envolve a modelagem das etapas sequenciais em um processo computacional.

No MDArte, um caso de uso deverá possuir o estereótipo *«FrontEndUseCase»*. Cada caso de uso deverá ter um diagrama de atividades associado. As atividades com estereótipo *«FrontEndView»* serão transformadas em telas da aplicação, gerando a camada de apresentação para o usuário. As demais atividades do diagrama indicarão interações com o servidor, chamadas na camada de controle.

A Figura 1 exemplifica um diagrama de atividade para o caso de uso "Consultar Aluno". Em que os estados "Consulta Estudante" e "Resultado da Consulta" possuem o estereótipo *«FrontEndView»*, e o estado "consulta estudante" representa a ligação com o método do controle, que efetivamente realizará a consulta.

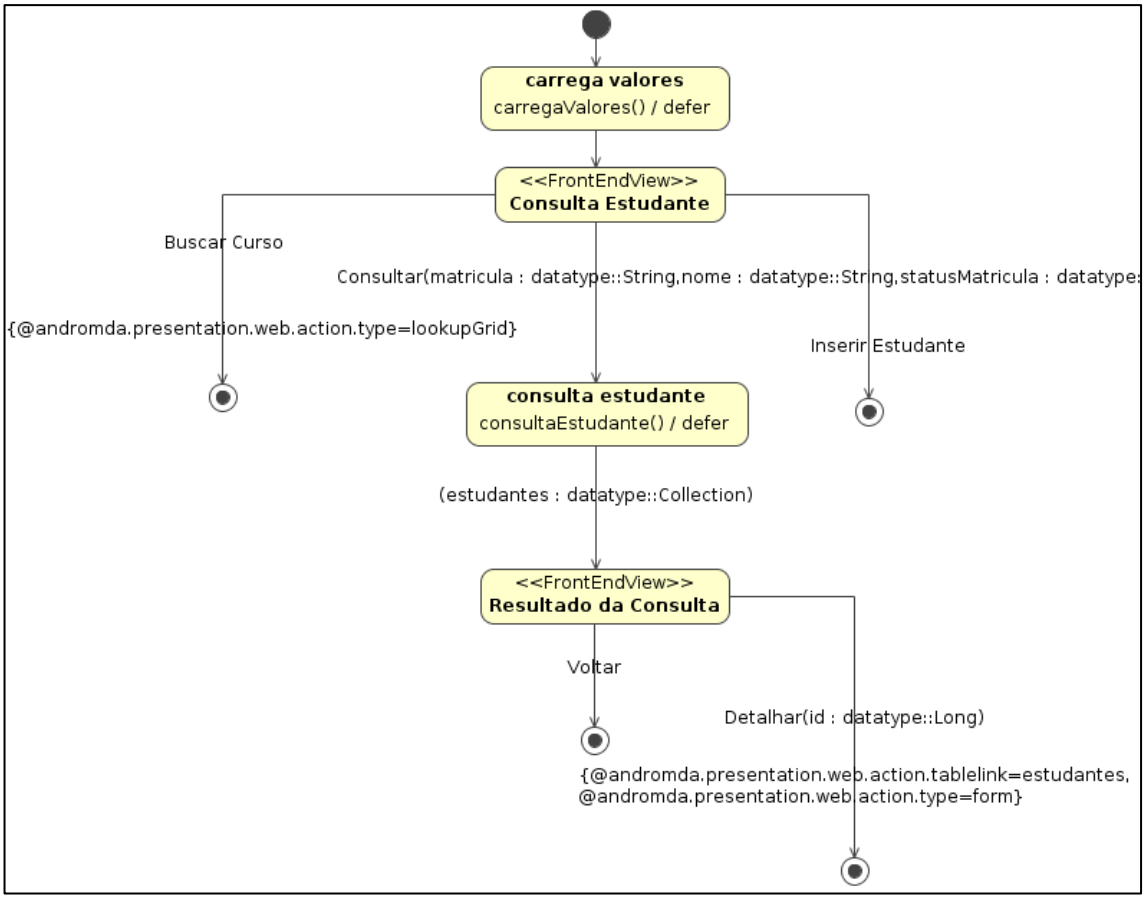

Figura 1 – Exemplo de modelagem da camada de apresentação.

O diagrama de atividades deverá ser criado no domínio do caso de uso correspondente, para que o MDArte possa identificá-lo e gerar o fluxo corretamente. Além disso, o diagrama de atividades deve ter uma classe associada, representando a classe de controle daquele caso de uso. Essa classe conterá todos os métodos que representem requisições ao servidor, como ações de botões.

Conforme indicado, na Figura 1, o estado "consulta estudante" representa uma chamada do sistema, executada após a ação sobre o botão "Consultar", representada pela transição de entrada do elemento. A partir da Figura 2 é possível verificar a relação existente entre o caso de uso "ConsultaEstudante" e o diagrama de atividades "ConsultaEstudanteAD", além da relação existente entre o diagrama de atividades e a classe de controle "ConsultaEstudanteControle". Também é possível verificar que a classe de controle possui os dois métodos referenciados na Figura 1.

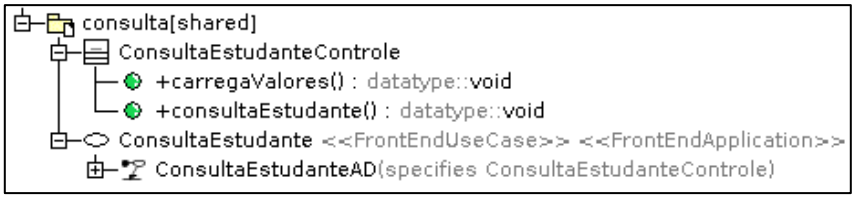

Figura 2 – Exemplo de relação entre caso de uso, diagrama de atividades e classe de controle.