Uma Linha de Produto de Software para Módulos de Aprendizagem Interativa

Danilo Leite Dalmon

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Mestre em Ciências

Programa: Mestrado em Ciência da Computação
Orientador: Prof. Dr. Leônidas de Oliveira Brandão

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro da FAPESP, processo 2010/06805-2

São Paulo, julho de 2012

Uma Linha de Produto de Software para Módulos de Aprendizagem Interativa

Esta versão da dissertação contém as correções e alterações sugeridas pela Comissão Julgadora durante a defesa da versão original do trabalho, realizada em 06/07/2012. Uma cópia da versão original está disponível no Instituto de Matemática e Estatística da Universidade de São Paulo.

Comissão Julgadora:

- Prof. Dr. Leônidas de Oliveira Brandão
- Prof. Dr. Marco Aurélio Gerosa
- Prof[a]. Dr[a]. Juliana Cristina Braga

Agradecimentos

Agradeço a todos os envolvidos direta e indiretamente na execução deste trabalho e na minha vida nesse período. Muito obrigado!

Ao meu orientador Prof. Leônidas, muito atencioso e sempre preocupado com as contribuições para os professores e alunos. À minha co-orientadora sorridente Profa. Anarosa e meu segundo co-orientador Prof. Seiji.

Aos que trabalharam diretamente comigo, no grupo de pesquisa do Laboratório de Informática na Educação (LInE), colegas de projeto Daniel, Eliane, Guilherme, Helena, Jonatas, Maria José, Márcia, Maurício, Patrícia, Reginaldo, Roberto e Romenig. Principalmente àqueles que revisaram meus textos, que programaram junto comigo, estudaram as mesmas disciplinas, conduziram os cursos de verão e a distância e também responderam às minhas entrevistas sempre com muita atenção e companheirismo.

A Cid, Cilene e Otávio por oferecerem o novo iMA. Ao Romenig, que foi sensacional por se sujeitar a ser o caso estudado com prontidão e competência.

Aos professores e amigos que me ajudaram muito com sugestões e conselhos extra-oficiais: Prof. Gerosa, Prof. Uirá, Cláudia, Flávio, Giuliano, Profa. Miriam, Prof. Nilson e outros que posso não lembrar mas não deixaram de contribuir.

À FAPESP pela bolsa de estudos. Ao CNPq que financiou as viagens aos eventos. Ao IME e à USP por toda a estrutura.

Aos colaboradores do LIQC, Prof. Suzuki, Cláudio, Edmilton, Eduardo, Eric, Juliana, Rodrigo, e outros que há muito tempo desenvolveram em mim o gosto pela ciência.

À família, meus pais Ângela e Luiz, e minha irmã Tábata pela paciência e suporte. À namorada Adriana pela presença e força incomensuráveis. Aos meus amigos de jogo (André, Alexandre, Bruno, Diego, Gustavo e Marcelo), do intercâmbio (Juliano, Pierre, Simone...), da mecatrônica (Alexandre, Gustavo, Nilson, Osmar...), da república (Bruno, Daniel, Ewerton, Hugo, Márcio...), do técnico (Rafael e Raphael), e indiretos pela namorada (Adriana, Dona Eliene, Jorge, Juliana, Rogério...) por entenderem quando dizia "não posso, tenho coisas do mestrado para fazer", inclusive nos fins de semana e feriados prolongados. Aos amigos do IME (André, Antônio, Eduardo...) por compartilharem da situação. Aos que eu não coloquei o nome, me desculpem, o espaço aqui é pequeno mas não o é em meu coração.

Ao universo, à natureza, a Deus, Tao, Alá, Bhraman, Samsara e os outros nomes do que é e mantém tudo.

Resumo

Dalmon, D. L. **Uma Linha de Produto de Software para Módulos de Aprendizagem Interativa**. 2012. 99 p. Dissertação (Mestrado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2012.

Aplicativos educacionais podem promover diversos benefícios a professores e alunos, desde a redução de tarefas repetitivas à realização de atividades impossíveis com o uso de apenas lousa e papel. Módulos de Aprendizagem Interativa (iMA) são uma família de aplicativos educacionais que fornecem atividades interativas integradas a Sistemas de Gerenciamento de Cursos. O desenvolvimento de iMA, similarmente ao de outros sistemas criados em contexto de projetos acadêmicos, enfrenta problemas relacionados a manutenção e evolução de *software*, que afetam suas contribuições à sociedade. Isso é provocado principalmente pela degradação do código com o tempo e dos métodos *ad-hoc* utilizados, sem sistematização do processo ou procedimentos explícitos para prevenção desses problemas. Com o objetivo de aprimorar esse processo, esta dissertação apresenta uma Linha de Produto de Software (LPS) criada para o desenvolver iMA. O método usado para criá-la envolveu análise do domínio, formado pelos iMA existentes, a elaboração de um modelo de sistema para definir as responsabilidades das características obrigatórias, variantes e opcionais aos aplicativos, e a implementação de um arcabouço de aplicação. Assim, essa LPS consiste nesse arcabouço e seus manuais de funcionamento interno, utilização e evolução, promovendo reúso de código, arquitetura e processo. A avaliação deste trabalho foi conduzida com uma prova de conceito e um estudo de caso. A prova de conceito descreve o desenvolvimento de um novo iMA, chamado *iTangran*, apresentando a factibilidade da utilização da LPS para essa tarefa. O estudo de caso investigou com maior profundidade o impacto da LPS sobre a criação da nova versão de um aplicativo existente, chamado *iVProg*. Os resultados obtidos mostram que o oferecimento de um processo e uma arquitetura que guiam as tarefas do programador de iMA teve grande influência na sua percepção de produtividade e satisfação, além de contribuírem para a qualidade do código criado e sua documentação, fatores essenciais para a prevenção dos problemas de desenvolvimento enfrentados atualmente.

Palavras-chave: Software Educacional, Linha de Produto de Software, Arcabouço de Aplicação, Módulo de Aprendizagem Interativa.

Abstract

Dalmon, D. L. **A Software Product Line for Interactive Learning Modules**. 2012. 99 p. Dissertação (Mestrado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2012.

Educational software can provide many benefits for teachers and students, from reducing repetitive tasks to offering assignments impossible with the use of only blackboard and paper. Interactive Learning Modules (iLM) are a family of educational systems which offer interactive assignments integrated with Learning Management Systems. The development of iLM, similarly to other software created in context of academic projects, faces issues related to software maintenance and evolution, which hinder their contributions to society. This is mainly caused by code degradation with time and to ad-hoc methods used, without a systematic process and explicit considerations to prevent these problems. With the goal of improving this process, this work presents a Software Product Line (SPL) for the development of iLM. The method used to create this SPL involved an analysis of the domain, which are the existing iLM, the creation of a system model in order to define the responsibilities of mandatory, variant and optional features among systems, and the implementation of an application framework. Therefore, the SPL consists of this framework and its manuals for internal operation, utilization and evolution, providing code, architecture and process reuse. Contributions of this work were evaluated by a proof of concept and a study case. The proof of concept describes the development of a new iLM called *iTangran*, presenting the possibility of using the SPL for this task. The study case investigated more deeply the SPL impact on the refactoring process of an existing iLM, *iVProg*. Results show that a guide and an architecture for the tasks undertaken by programmers have a significant influence on the perceived productivity and their satisfaction while working, also contributing to code and documentation quality, which are essential factors to prevent development problems such as those faced nowadays.

Keywords: Educational Software, Software Product Line, Application Framework, Interactive Learning Module.

Sumário

Lista de Abreviaturas **ix**

Lista de Figuras **xi**

Lista de Tabelas **xiii**

1 Introdução **1**
1.1 Problemática . . . 2
1.2 Justificativa . . . 3
1.3 Objetivos . . . 5
1.4 Abordagem . . . 6
1.5 Organização do Trabalho . . . 7

2 Fundamentação **9**
2.1 Módulos de Aprendizagem Interativa . . . 9
2.1.1 Contexto e Motivação . . . 9
2.1.2 Histórico de Desenvolvimento . . . 10
2.1.3 Funcionalidades Comuns . . . 13
2.2 Linha de Produto de Software . . . 15
2.2.1 Definição e Benefícios . . . 15
2.2.2 Estrutura . . . 16
2.2.3 Implementação . . . 17
2.2.4 Arcabouço de Aplicação . . . 18
2.3 Trabalhos Relacionados . . . 19
2.3.1 Desenvolvimento de Aplicativos Educacionais . . . 20
2.3.2 Engenharia de Software em Aplicativos Educacionais . . . 21

3 Métodos **25**
3.1 Análise de Domínio . . . 25
3.1.1 Análise de Requisitos . . . 25
3.1.2 Análise de Características . . . 26
3.2 Engenharia de Domínio . . . 27
3.2.1 Criação do Modelo de Sistema . . . 28
3.2.2 O Método PLUS . . . 29
3.2.3 Projeto e Implementação do Arcabouço . . . 32
3.2.4 Uso e Evolução do Arcabouço . . . 33

3.3 Avaliação . . . 34
3.3.1 Prova de Conceito - *iTangran* . . . 35
3.3.2 Estudos de Caso . . . 36
3.3.3 Casos de desenvolvimento de iMA - *iVProg* . . . 37

4 Uma LPS para iMA **41**
4.1 Análise de Domínio . . . 41
4.1.1 Requisitos dos iMA . . . 42
4.1.2 Funcionalidades dos iMA . . . 43
4.1.3 Modelagem das características . . . 45
4.2 Modelo de Sistema . . . 46
4.2.1 Componente Estrutural . . . 47
4.2.2 Componente de Domínio . . . 48
4.2.3 Componente de Atividades . . . 50
4.2.4 Componente de Extensões . . . 51
4.3 O Arcabouço de Aplicação . . . 52
4.3.1 Desenvolvimento . . . 52
4.3.2 Estrutura de Componentes . . . 55
4.4 Engenharia de Aplicação . . . 58
4.4.1 Estender o Modelo de Domínio . . . 59
4.4.2 Estender a Interface Gráfica de Domínio . . . 60
4.5 Evolução da LPS . . . 62
4.5.1 Módulos de Atividade . . . 62
4.5.2 Módulos de Operações . . . 63

5 Resultados e Discussões **65**
5.1 Prova de Conceito - *iTangran* . . . 65
5.1.1 Descrição do *iTangran* . . . 66
5.1.2 Discussão . . . 68
5.2 Estudo de Caso - *iVProg* . . . 69
5.2.1 Descrição do *iVProg* . . . 69
5.2.2 Desenvolvimento anterior à adoção da LPS . . . 71
5.2.3 Desenvolvimento após a adoção da LPS . . . 74
5.2.4 Análise dos Dados e Discussão . . . 78
5.2.5 Limitações e Ameaças à Validade . . . 82

6 Conclusões **85**
6.1 Considerações Finais . . . 86
6.2 Sugestões para Trabalhos Futuros . . . 88

Referências Bibliográficas **91**

Lista de Abreviaturas

CAI	Instrução Assistida por Computador (*Computer Assisted Instruction*)
DDM	Desenvolvimento Dirigido por Modelos
ETT	Tutores Rastreadores de Exemplos (*Example-Tracing Tutors*)
FODA	Análise de Domínio Orientada a Características (*Feature Oriented Domain Analysis*)
GI	Geometria Interativa
iMA	Módulo de Aprendizagem Interativa
IMS-LD	Especificação *IMS Learning Design*
LPS	Linha de Produto de Software
OA	Objeto de Aprendizagem
PLUS	Engenharia de Linha de Produto de Software baseada em UML (*Product Line UML-based Software Engineering*)
POA	Programação Orientada a Aspectos
SCORM	Especificação *Shareable Content Object Reference Model*
SGC	Sistema de Gerenciamento de Cursos

Lista de Figuras

1.1 Resultados do questionário aplicado sobre o contexto de desenvolvimento de aplicativos educacionais em outros grupos de pesquisa. 3

2.1 Interface gráfica dos iMA *iCG* e *iHanói*. 11
2.2 Um exemplo de atividade do *iGeom* integrada ao SAW. 12
2.3 Exemplos de telas dos iMA: (a) *iGeom*, (b) *iGraf*, (c) *iComb* e (d) *iVProg*. 13
2.4 Estrutura de uma LPS, com a engenharia de domínio, de aplicação e sua evolução. . 16

3.1 Exemplo de diagrama de características, baseado em (Kang *et al.*, 1990). 27
3.2 Exemplo de diagrama de casos de uso do método PLUS. 30
3.3 Exemplo de diagrama de classes do método PLUS. 31
3.4 Exemplo de diagrama de colaboração do método PLUS. 32

4.1 Diagrama de características da LPS para iMA. 45
4.2 Diagrama da estrutura do modelo de sistema para iMA. 47
4.3 Diagrama de classes simplificado para a modelagem das operações de domínio. 48
4.4 Diagrama de sequência simplificado para a execução de uma ação de domínio. 49
4.5 Diagrama de classes simplificado com os elementos de uma atividade. 50
4.6 Diagrama de casos de uso para a caracterítica de gerenciamento de atividades. 53
4.7 Diagrama de classes para a caracterítica de gerenciamento de atividades. 54
4.8 Diagrama de componentes do arcabouço para desenvolvimento de iMA. 55
4.9 Imagem da interface gráfica do arcabouço, formada pela base, com borda preta, e a específica de domínio, com borda tracejada, usando como exemplo o iMA *iVProg*. . . 56
4.10 Interface de atividade da instanciação teste do arcabouço. 61
4.11 Interface de autoria da instanciação teste do arcabouço. 61

5.1 Interface gráfica de atividade do *iTangran* com algumas peças movimentadas. 67
5.2 Diagrama de classes do *iTangran*. 67
5.3 Interface gráfica de atividade do *iVProg*. 70
5.4 Interface gráfica de atividade da nova versão do *iVProg* criada com o auxílio da LPS. 77

Lista de Tabelas

2.1 Classificação dos trabalhos relacionados quanto aos aplicativos desenvolvidos e a técnica de Engenharia de Software utilizada. 24

4.1 Funcionalidades independentes do domínio distribuídas nos iMA existentes 44
4.2 Resultado da análise do código do arcabouço segundo métricas relevantes para sua manutenção. 62

5.1 Resultado da análise do código da nova versão do *iVprog*. 79
5.2 Resultado da análise do código da versão anterior do *iVprog*. 80

Capítulo 1

Introdução

Aplicativos educacionais são um tipo específico de sistema computacional, cuja principal característica é o objetivo pelo qual é desenvolvido: servir de ferramenta para os processos de aprendizagem dos estudantes (Tchounikine, 2011). Assim, os sistemas educacionais contrastam com aqueles denominados *básicos*, cujos objetivos podem atender a quaisquer outras necessidades dos usuários (p.4). Nesta dissertação de mestrado usamos o termo em português *aplicativo* em vez do comumente usado com origem da língua inglesa *software*.

Diversos são os tipos de aplicativos educacionais, que receberam a atenção de pesquisadores e desenvolvedores em diferentes períodos, a seguir descrevemos alguns exemplos. Na década de 60, os chamados sistemas CAI (sigla em inglês de *Computer Assisted Instruction*, ou Instrução Assistida por Computador) eram considerados uma grande promessa para o aprimoramento da educação (Merril, 1984; Nicolson e Scott, 1986). Nos anos 80, aplicativos de tutoria inteligente estavam em destaque na área de Informática na Educação (Bloom, 1984). Com o passar do tempo, diversos outros tipos surgiram, como jogos educativos (Amory *et al.*, 1999), sistemas de simulação (van Dam *et al.*, 2005) e sistemas de gerenciamento de curso (Green, 2010). Mais recentemente, foi significativo o aumento no número de aplicativos educacionais relacionados a dispositivos móveis (Batista *et al.*, 2011) e redes sociais (Dotta, 2011).

De maneira similar, as contribuições para os processos de ensino e aprendizagem variam entre os sistemas. Há aplicativos que objetivam melhorar o desempenho dos alunos em exames de avaliação padronizados (Bloom, 1984). Outra possibilidade é influenciar, pelo uso da tecnologia, a motivação e o interesse (Raines e Clark, 2011). Um último exemplo são sistemas que promovem experiências de aprendizagem que são muito difíceis, ou impossíveis, de realizar sem a ajuda do computador (Tang, 2005). Alguns resultados de uso apresentados nesses trabalhos mostram sua efetividade em aprimorar o trabalho dos professores e as experiências dos alunos.

O processo de criar aplicativos educacionais é diferente daquele adotado para aplicativos *básicos* (Tchounikine, 2011). Dois exemplos de especificidades que devem ser consideradas são os aspectos de interfaces centradas no aprendiz (Quintana *et al.*, 2003) e de ferramentas profissionais para o professor (Hinostroza *et al.*, 2000). Interfaces centradas no aprendiz contrastam com as centradas no usuário de maneira que este último usa a ferramenta para executar uma tarefa enquanto que o primeiro a utiliza para aprender como se executa a tarefa. Mesmo que os aplicativos tenham intenção de melhorar o trabalho do professor, muitas vezes eles são criados sem considerar suas práticas cotidianas específicas, que também podem conter objetivos diferentes daqueles de um usuário comum.

O desenvolvimento de aplicativos educacionais é tema de diversas pesquisas acadêmicas. Nas últimas três edições do Simpósio Brasileiro de Informática na Educação (SBIE), cerca de metade dos artigos publicados possuíam uma contribuição na forma do desenvolvimento de um sistema (Dalmon *et al.*, 2012b). Apesar de haver artigos cuja contribuição se restringe ao aplicativo em si, alguns investigam as particularidades do processo de desenvolvimento. Artigos que relatam iniciativas nesse tema focam em organização da equipe (Winters e Mor, 2008), métodos de projeto instrucional (Meziane *et al.*, 1999), aspectos de programação (Spalter e van Dam, 2003) e o uso de técnicas de Engenharia de Software (Ahmed e Zualkernan, 2011; Conlan *et al.*, 2002; Reis, 2007).

Nesse contexto, este trabalho tem como principal tema o desenvolvimento de aplicativos educacionais. Mais especificamente, de uma família de sistemas similares, chamados Módulos de Aprendizagem Interativa (iMA) (Dalmon *et al.*, 2011c). O desenvolvimento de aplicativos, mesmo que eventualmente de forma implícita, remete necessariamente a aspectos provenientes da Engenharia de Software. Especificando novamente, este trabalho relata o uso da técnica Linha de Produto de Software (LPS) (Clements e Northrop, 2001) para o caso do desenvolvimento de iMA.

1.1 Problemática

Desenvolver aplicativos educacionais é uma tarefa sofisticada e complexa, além de envolver aspectos de computação e de educação, trata de outros específicos dessa interdisciplinaridade (Mor e Winters, 2007). Problemas como falta de organização e comunicação na equipe multidisciplinar (Spalter e van Dam, 2003), levantamento de requisitos educacionais mal formulados (McAndrew *et al.*, 2006) e associados a dificuldades de utilização por professores (Hinostroza *et al.* , 2000), são citados com frequência na literatura. Como é uma atividade de desenvolvimento de software[1], muitos dos problemas tratados pela Engenharia de Software são encontrados também na criação e evolução desses sistemas. Problemas sobre dificuldades de desenvolvimento específicos em aplicativos educacionais começaram a ser publicados por Nicolson e Scott (1986). Com o passar do tempo, artigos apresentam soluções de Engenharia de Software, como arquiteturas de componentes (Roschelle *et al.*, 1998), reengenharia (Choquet e Corbière, 2006), entre outras técnicas (Conlan *et al.*, 2002; Dodero *et al.*, 2007; Oberweis *et al.*, 2007). O problema tratado por esta pesquisa é próximo aos abordados por esses artigos.

No caso específico do grupo de pesquisa em que o trabalho foi realizado, ou seja, no desenvolvimento dos iMA, foram encontrados muitos dos problemas relatados nos artigos citados e em livros de Engenharia de Software, como por Sommerville (2007). Exemplos são: (i) dificuldade de manutenção em sistemas desenvolvidos há algum tempo ou por diferentes programadores; (ii) falta ou má qualidade de documentação; (iii) alta curva de aprendizagem do domínio ou das tecnologias envolvidas para novos programadores; (iv) falta de método sistemático de levantamento de requisitos e em outras etapas do desenvolvimento, entre outros. Dessa forma, os problemas aos quais este trabalho se remete são exatamente aqueles enfrentados pelos desenvolvedores de iMA.

Um artigo publicado por nós, no qual são apresentadas informações obtidas com questionários e entrevistas a programadores de outros grupos de pesquisa brasileiros, relata problemas enfrentados similares a esses (Dalmon *et al.*, 2012b). Essa situação é enfrentada por muitos desenvolvedores e

[1] Apesar de adotarmos o termo *aplicativo* em geral, decidimos manter a palavra em inglês quando usamos expressões compostas como "projeto de *software*" ou "desenvolvimento de *software*".

sua influência no andamento e nas contribuições dos projetos é grande. Relatos incluem grande dificuldade, insatisfação do programador durante a realização das tarefas e a frequente "perda de tempo" para tratar problemas provocados por má qualidade de código e falta de documentação dos aplicativos educacionais.

Mais detalhes sobre o contexto de desenvolvimento de iMA são descritos na seção 5.2, em que um estudo de caso é relatado com as experiências enfrentadas por um dos desenvolvedores do iMA chamado *iVProg* (Kamiya e Brandão, 2009). O problema escolhido para este trabalho pode parecer, num primeiro momento, porém, um estudo de levantamento de evidências foi realizado para justificar a sua relevância, apresentado a seguir.

1.2 Justificativa

A partir da leitura de artigos sobre o tema, da experiência durante o desenvolvimento deste trabalho e de conversas com outros pesquisadores e desenvolvedores, elaboramos a hipótese de que as situações enfrentadas pelo presente grupo de pesquisa são similares às de outros grupos do país. Para levantar indícios dessa hipótese, realizamos um estudo para o levantamento de evidências (Dalmon *et al.*, 2012b).

Para conhecer o contexto de desenvolvimento de outros grupos de pesquisa e compará-los com o nosso realizamos um questionário que, entre outras perguntas, pediu aos respondentes que indicassem o número de desenvolvedores envolvidos no projeto, a experiência em programação desses profissionais e o nível metodológico em que essa tarefa é realizada (sendo o nível 1 o desenvolvimento sem considerações metodológicas e o nível 4 a utilização sistemática de técnicas da Engenharia de Software). A figura 1.1 apresenta os resultados dessas questões respondidas para 22 projetos de pesquisa e desenvolvimento em universidades brasileiras.

Figura 1.1: *Resultados do questionário aplicado sobre o contexto de desenvolvimento de aplicativos educacionais em outros grupos de pesquisa.*

Esses resultados mostram que para a maior parte dos projetos o contexto de desenvolvimento em outros grupos de pesquisa é próximo ao nosso: poucos desenvolvedores (1 ou 2) com pouca experiência em programação (até 5 anos) e sem o uso de métodos sistemáticos (níveis 1 ou 2).

Para verificar se outros grupos de pesquisa enfrentam também problemas para realizar manutenção e evolução em seus sistemas realizamos entrevistas com seis programadores. As entrevistas

incluíram perguntas sobre o contexto de desenvolvimento e sobre problemas enfrentados pelos entrevistados durante as tarefas de programação. Em todos os casos foram relatadas dificuldades associadas à qualidade de software. Isso mostra que os problemas enfrentados por nosso grupo de pesquisa não são específicos e atingem muitos outros grupos de desenvolvedores.

Esse resultado pode ser considerado como uma evidência da abrangência da problemática descrita acima. Dessa forma, os problemas que este trabalho ataca não são exclusivos do nosso grupo ou ao desenvolvimento de iMA, e os resultados obtidos podem ser transpostos para situações de outros grupos de pesquisa em desenvolvimento de aplicativos educacionais, de acordo com o contexto. Dessa forma, esta pesquisa se justifica de maneira a tentar reduzir problemas enfrentados por muitas pessoas. A proporção de pesquisadores que desenvolvem aplicativos é importante, uma vez que cerca de metade dos artigos analisados no estudo possuem como contribuição a criação de um sistema.

Outras características do desenvolvimento de aplicativos educacionais em universidades fazem com que o uso de métodos sistemáticos seja relevante. Programadores são geralmente alunos de graduação, com pouca experiência profissional e curto tempo de participação no projeto, devido ao período de concessão da bolsa ou do trabalho de conclusão de curso. Se ao iniciar o projeto o programador possuir acesso a documentação de qualidade e processo bem definido, sua produtividade pode ser maior. Além disso, professores coordenadores de projetos de pesquisa são geralmente os "clientes", o que pode tornar as especificações e prazos menos definidos e mais flexíveis sem o uso de métodos sistemáticos, prejudicando o andamento do projeto.

Além da abrangência do problema, outras características podem ser destacadas sobre sua relevância. Primeiro foram considerados o efeito potencial da redução dos problemas analisados (Boehm *et al.*, 1976; Fayad *et al.*, 1999; Sommerville, 2007). Se um grupo de pesquisa que desenvolve aplicativos educacionais reduz os problemas de seu processo, ele passa a experimentar vários benefícios. Exemplos são: a redução do tempo de desenvolvimento, manutenção e evolução, a maior satisfação dos desenvolvedores e a maior qualidade do código dos sistemas. É possível, inclusive, conjecturar que há uma relação entre a qualidade dos sistemas e a efetividade dos aplicativos nos processos educacionais.

Esses benefícios podem ser ainda mais relevantes em grupos nos quais os desenvolvedores possuem pouca experiência profissional em programação ou têm formação em outras áreas do conhecimento. Nesses casos, a ocorrência de problemas no processo é maior (Spalter e van Dam, 2003), o que torna as estratégias de utilização de métodos sistemáticos mais importantes.

Por fim, o trabalho também se justifica pela influência dos aplicativos desenvolvidos nas atividades de professores e alunos. Se o tempo de manutenção é reduzido, um professor que encontra defeitos no aplicativo terá o defeito corrigido em menor tempo. O mesmo ocorre com a evolução, no caso do professor que usa o sistema levantar um novo requisito de funcionalidade. Uma outra possível influência é sobre a relação entre os programadores e outros desenvolvedores envolvidos na criação do aplicativo, se os programadores têm menos problemas em seu trabalho, essa comunicação pode ser facilitada.

1.3 Objetivos

De maneira coerente com a problemática e a justificativa apresentadas, o objetivo geral desta dissertação de mestrado é: *propor uma Linha de Produto de Software para o aprimoramento do desenvolvimento de iMA*. Em outras palavras, o objetivo deste trabalho é reduzir os problemas enfrentados no processo de desenvolvimento de aplicativos educacionais, como descritos anteriormente, utilizando a técnica LPS. A meta, de maneira mais geral que o objetivo, é contribuir para o desenvolvimento de aplicativos educacionais em contexto acadêmico como um todo, servindo de exemplo de iniciativa nessa direção.

Os critérios para analisar o cumprimento do objetivo descrito devem, portanto, avaliar o processo de criação de iMA, comparando sua execução anterior a este trabalho com a que utiliza seus resultados de desenvolvimento. São dois os métodos aplicados: (i) uma prova de conceito, com o objetivo de verificar a factibilidade da proposta apresentada pelo trabalho; e (ii) um estudo de caso de desenvolvimento de um iMA, que possui o objetivo de analisar com maior profundidade os efeitos e mecanismos causais dos problemas enfrentados no processo e a influência da LPS proposta.

Esses critérios foram escolhidos por seus aspectos qualitativos, o que é mais adaptado para os casos de desenvolvimento de iMA, que existem em pequena quantidade e estão disponíveis para uma análise mais próxima. Assim, os objetivos são avaliados com relação à percepção das pessoas envolvidas e pela análise sistemática do autor. Considerações adicionais sobre a os métodos de avaliação são feitas na seção 3.3.

Em complemento ao objetivo geral, é possível explicitar alguns objetivos mais específicos, subprodutos considerados ao longo da execução do trabalho. Abaixo são apresentados os principais deles:

- *Levantar e explicitar características e problemas do processo de desenvolvimento de aplicativos educacionais em contexto acadêmico*. Esse objetivo é uma consequência da análise do método de desenvolvimento de iMA anterior a este trabalho e também do estudo para o levantamento de evidências sobre que problemas ocorrem em outros grupos de pesquisa. Isso também é necessário para determinar os problemas a serem atacados, a abordagem deste trabalho e permitir a análise do seu efeito sobre os processos de desenvolvimento de iMA.

- *Adotar e divulgar o uso de métodos sistemáticos para o desenvolvimento de aplicativos educacionais*. Esse objetivo é uma consequência da abordagem escolhida, detalhada na próxima seção. Para fazer respeito à abrangência do problema escolhido, não basta aplicar no contexto específico, é necessário propagar a abordagem para outros projetos e grupos de pesquisa.

- *Reunir e documentar aspectos relacionados à família de sistemas iMA*. Durante o processo de determinação do modelo de domínio dos iMA, é preciso definir as características particulares dos aplicativos e da família como um todo. Esse objetivo, portanto, contribui para a explicitação e organização dos conhecimentos associados a cada sistema e ao conjunto deles.

- *Fornecer recursos concretos para serem reutilizados no desenvolvimento de novos iMA ou na refatoração dos existentes*. Como parte do objetivo geral, de aprimorar o desenvolvimento dos iMA, temos um objetivo específico de facilitar esse processo. Essa facilitação pode ser promovida pela utilização de recursos concretos (código-fonte e documentos) gerados por este trabalho.

- *Promover a reorganização, a refatoração e a adição de novas funcionalidades aos iMA existentes.* De maneira similar ao objetivo anterior, como parte do objetivo geral, este trabalho contribui fornecendo exemplos de novos aplicativos e funcionalidades e auxiliando projetos parceiros inseridos no grupo de pesquisa de desenvolvimento de iMA.

1.4 Abordagem

Para cumprir o objetivo descrito em direção a amenizar o problema selecionado, a abordagem deste trabalho foi a proposta de uma LPS para desenvolver iMA, e sua aplicação para criar sistemas novos ou refatorar existentes. Essa abordagem é uma forma de utilização de métodos sistemáticos, provenientes da Engenharia de Software, para o desenvolvimento de aplicativos educacionais visando o aprimoramento dos processos e produtos envolvidos.

A escolha dessa abordagem pode ser justificada da seguinte maneira: os problemas considerados por este trabalho são específicos da etapa de *desenvolvimento de software* na criação de aplicativos educacionais, o que torna natural a utilização da Engenharia de Software para embasar esse processo. Além disso, os aplicativos em questão formam uma família de sistemas similares, o que é exatamente a área de aplicação da LPS (Clements e Northrop, 2001).

De maneira sucinta, uma LPS organiza o desenvolvimento de um conjunto de aplicativos similares em duas partes: engenharia de domínio e engenharia de aplicação. A engenharia de domínio é responsável por todas as características comuns entre os aplicativos, enquanto que a engenharia de aplicação trata das partes específicas de cada um. Dessa forma, quanto mais compartilhamento de funcionalidades os aplicativos possuírem e quanto maior for o número deles, maiores são os benefícios da utilização dessa técnica (Pohl *et al.*, 2005).

Uma LPS pode ser implementada de diversas maneiras, dependendo do contexto em que ela é aplicada e de características dos sistemas a serem desenvolvidos (Clements e Northrop, 2001). No caso do presente trabalho, a LPS é entregue ao programador de iMA em duas partes: (i) um arcabouço de aplicação; e (ii) manuais de utilização para os desenvolvedores. Devido à falta de centralização de informações sobre a família como um todo, um modelo abstrato de sistema teve de ser criado para guiar a implementação das funcionalidades comuns. Usando esse modelo, um arcabouço de aplicação foi desenvolvido de maneira que possa ser utilizado com facilidade por programadores com pouca experiência, o que é comum nos contextos acadêmicos de pesquisa em aplicativos educacionais. Por sua vez, os manuais são necessários para guiar o uso da LPS, o reúso de processo promovido por eles é uma das principais contribuições deste trabalho. Outras considerações sobre essas escolhas são feitas na seção 2.2.

Para avaliar a LPS desenvolvida e a sua influência no processo de desenvolvimento de iMA foram escolhidos os métodos prova de conceito (Wazlawick, 2009) e estudo de caso (Gerring, 2006; Yin, 2008). Esses métodos foram escolhidos sobre métodos mais quantitativos pela impossibilidade de realizar experimentos que gerariam dados relevantes estatisticamente. A prova de conceito é uma forma de verificar a factibilidade de uma técnica ou processo, gerando um resultado simples para exemplificar a contribuição esperada, e foi aplicada sob a forma do desenvolvimento de um novo iMA, o *iTangran*, descrito na seção 5.1. Um estudo de caso já possui mais rigor e poder de análise, investigando detalhes dos mecanismos causais e das influências de uma intervenção, que neste trabalho é a adoção de uma LPS desenvolvida para a criação da nova versão de um iMA

existente, chamado *iVProg*.

As contribuições provocadas pela abordagem escolhida para este trabalho estão em várias áreas, incluindo o grupo de pesquisa no qual foi realizado, outros grupos com contexto de atuação similar e também pesquisadores em desenvolvimento de aplicativos de diferentes naturezas e em situações incomuns. O grupo de pesquisa que desenvolve iMA será o maior influenciado pelo trabalho. As contribuições vão da forma de programar os aplicativos até o planejamento a longo prazo dos projetos de pesquisa. A contribuição para outros grupos que também desenvolvem aplicativos educacionais pode ser na forma de um "caso exemplo" de aplicação da Engenharia de Software. Por fim, é possível contribuir com pesquisas e projeto de desenvolvimento em situações como os contextos acadêmicos de trabalhos em Informática na Educação.

1.5 Organização do Trabalho

O texto dos capítulos seguintes que descrevem o trabalho realizado está organizado da seguinte maneira:

- O capítulo 2, Fundamentação, apresenta os conceitos básicos para o entendimento do texto, descrevendo com mais detalhes os iMA e a técnica LPS. Esse capítulo também apresenta os trabalhos relacionados, abrangendo outras iniciativas de aplicação de técnicas de Engenharia de Software para o desenvolvimento de aplicativos educacionais.

- O capítulo 3, Métodos, descreve a forma com que o trabalho foi conduzido ao longo de sua execução. Assim, os métodos usados compreendem o de análise de domínio, o de desenvolvimento da linha de produto e aqueles usados para a avaliação do trabalho realizado, especificamente a prova de conceito e o estudo de caso.

- O capítulo 4, Uma LPS para iMA, apresenta o conteúdo do trabalho desenvolvido, que corresponde propriamente à Linha de Produto de Software para Módulos de Aprendizagem Interativa. Esse capítulo contém seções correspondentes a uma descrição geral, aos componentes da LPS: o modelo de sistema e o arcabouço de aplicação, e aos métodos criados para sua utilização e evolução.

- O capítulo 5, Resultados e Discussões, relata e discute as informações obtidas com as atividades de avaliação da LPS desenvolvida, incluindo os trabalhos de prova de conceito e estudo de caso.

- O capítulo 6, Conclusões, fecha o texto destacando as principais contribuições do trabalho, algumas considerações finais sobre os processos de desenvolvimento de aplicativos educacionais em contexto acadêmico e sugestões de trabalhos futuros.

Capítulo 2

Fundamentação

Este capítulo apresenta a fundamentação da dissertação, ou seja, os conceitos básicos envolvidos e a revisão da literatura abordando os trabalhos relacionados. Os conceitos básicos envolvidos estão presentes no título deste trabalho, na seção 2.1 apresentamos os *Módulos de Aprendizagem Interativa* e na seção 2.2 a *Linha de Produto de Software*. A seção 2.3 descreve os trabalhos relacionados que serviram como base, inspiração e fundamentação da pesquisa.

2.1 Módulos de Aprendizagem Interativa

Nosso objeto de estudo é o desenvolvimento de um grupo de aplicativos educacionais chamados Módulos de Aprendizagem Interativa, cuja sigla é iMA. Os iMA são aplicativos desenvolvidos pelo grupo coordenado pelo Prof. Dr. Leônidas de Oliveira Brandão, orientador desta pesquisa, desde 2000 (Brandão e Isotani, 2003). Apesar de até hoje apenas este grupo ter produzido esses sistemas, espera-se que uma das contribuições desta dissertação seja que outros desenvolvedores também passem a criar iMA.

Esta seção está dividida da seguinte maneira: primeiro são apresentados o conceito e a motivação dos iMA; em seguida é feito um relato do histórico de desenvolvimento dessa família de aplicativos; e por fim são apresentadas especificações de suas funcionalidades comuns. Considerações específicas sobre cada iMA utilizado diretamente neste trabalho são feitas ao longo do capítulo 5.

2.1.1 Contexto e Motivação

Módulos de Aprendizagem Interativa (iMA) são aplicativos educacionais similares que formam uma família. O objetivo dessa família é facilitar e melhorar o trabalho do professor e simultaneamente fornecer ao aluno um ambiente motivador de atividades interativas (Dalmon *et al.*, 2011c). Os iMA foram concebidos para serem usados em conjunto com Sistemas Gerenciadores de Curso (SGC). A partir dessa afirmação, segue uma descrição do contexto de uso, da problemática e da motivação desses aplicativos.

Apesar de serem muito utilizados pelas instituições de ensino, os SGC não possuem como função primária o oferecimento de atividades, e apresentam limitações quando usados como aplicativos para atividades interativas (Rodrigues, 2011). A maior parte dos SGC oferece ao professor funcionalidades de autoria de atividades que promovem apenas interatividade simples, não permitindo a elaboração de ferramentas para experiências consideradas mais interessantes, por exemplo interati-

vidades do tipo *manipulação* e *construção* (Sims, 1997; Tang, 2005). Isso é particularmente grave no caso de disciplinas como a geometria e gráficos de funções matemáticas, o que pode torná-las desinteressantes e dificultar a aprendizagem quando são empregados apenas utilizando recursos tradicionais (Raines e Clark, 2011).

Para atender a essa demanda, instituições de ensino e professores utilizam aplicativos de atividade interativa específicas, que promovem tipos de interatividade que facilitam certas abordagens pedagógicas (Sampson, 2005). Um problema desse uso é que nem sempre esses aplicativos são integráveis aos SGC (Rodrigues *et al.*, 2010). Outra dificuldade importante encontrada pelos professores é a sua dificuldade de reutilização, ou seja, não podem ser usados em diferentes contextos, aulas, cursos e abordagens pedagógicas (Polsani, 2003).

Aplicativos de atividade integrados em SGC ou as próprias atividades fornecidas nos sistemas de gerenciamento têm vantagens para professores e alunos. Para o professor, a centralização das ferramentas economiza tempo e facilita o acesso a informações como respostas dos alunos, notas, etc. Para o aluno essa centralização reduz a curva de aprendizagem para uso e facilita o acesso às atividades. Reutilização diz respeito à possibilidade de utilizar um recurso em diversos contextos educacionais sem exigir esforço para adaptá-lo, sendo importante para professores que desejam autonomia para o planejamento e execução de suas aulas. Caso a instituição utilize ferramentas computacionais com baixo potencial de reúso, o professor deve modelar sua aula em função da ferramenta ou, se possível, escolher não utilizá-las.

Dentre as frentes de pesquisa que têm como objetivo vencer essas dificuldades, aumentando a possibilidade de reúso dos aplicativos, há repositórios (Richards *et al.*, 2002) e ferramentas de autoria (Moundridou e Virvou, 2003). Repositórios permitem aos professores acesso a um grande número de aplicativos e atividades, de maneira que possam reutilizar uma ferramenta que melhor se adapte à sua abordagem. Ferramentas de autoria permitem, por sua vez, criar atividades em um sistema de regras, com as quais o professor pode aplicar sua abordagem da maneira que preferir em diferentes situações.

Assim, uma das formas de instituições de ensino ou professores aproveitarem bem recursos computacionais para o cotidiano de um curso é a utilização de um SGC com aplicativos de atividades interativas reutilizáveis, seja com o uso de repositórios ou de ferramentas de autoria. Nessa situação entram os iMA, que foram concebidos para serem aplicativos educacionais de atividades interativas com ferramentas de autoria e integrados aos SGC. Para complementar as possibilidades de reúso desses aplicativos, está em andamento o desenvolvimento de um repositório específico para as atividades interativas criadas com esses sistemas (Rosa e Brandão, 2010).

Outra funcionalidade importante que os iMA podem oferecer é a avaliação automática de atividades, promovendo também o tipo de interatividade de retroação (*feedback*) (Brandão *et al.*, 2004; Isotani e Brandão, 2008). A partir da integração com SGC, é possível comunicar o resultado da avaliação automática das atividades dos alunos a um banco de dados do SGC para que o professor acompanhe de maneira simples os resultados de cada aluno utilizando as funcionalidades desse sistema (Rodrigues *et al.*, 2010).

2.1.2 Histórico de Desenvolvimento

A família de aplicativos educacionais iMA teve início com o *iGeom* (Brandão e Isotani, 2003), um sistema de Geometria Interativa (GI) (Kortenkamp, 1999). Sistemas de GI são aplicativos edu-

cacionais que, basicamente, oferecem possibilidades de criar e manipular objetos geométricos, atualizando as construções em tempo real. Assim, esses sistemas podem beneficiar consideravelmente professores e alunos de geometria e áreas relacionadas (Ruthven *et al.*, 2008).

No final da década de 1990, o Prof. Leônidas de Oliveira Brandão percebeu uma lacuna nas características dos sistemas de GI existentes: o fato de não serem aplicativos *Web* e serem sistemas proprietários. Dessa forma, em 2000, ele começou o desenvolvimento do *iGeom: Geometria Interativa na Internet*, um sistema de GI gratuito e que funciona também em navegadores *Web* (Brandão e Isotani, 2003). Da mesma forma que os outros sistemas de GI, o *iGeom* possui funcionalidades de ferramenta de autoria de atividades interativas para professores. Com o tempo e com o auxílio de alunos de iniciação científica e pós-graduação, muitas funcionalidades foram incluídas, inclusive algumas inéditas em sistemas de GI, como *scripts* e avaliação automática (Brandão *et al.*, 2004; Isotani, 2005).

Seguindo uma abordagem similar, ainda no início dos anos 2000, o grupo de pesquisa desenvolveu outros sistemas *Web* gratuitos, cujos equivalentes eram aplicativos locais (*desktop*) ou comerciais. Surgiram, então, aplicativos como o *iCG*, um sistema para introdução ao ensino de programação de computadores, e o *iHanói*, um aplicativo para o jogo "Torre de Hanói", muito usado em cursos de programação para o ensino de conceitos de recursão. Ambos estão disponíveis nas páginas *Web* (Brandão, 2012a) e (Brandão, 2012b). A figura 2.1 apresenta exemplos de tela dos iMA *iCG* e *iHanói*.

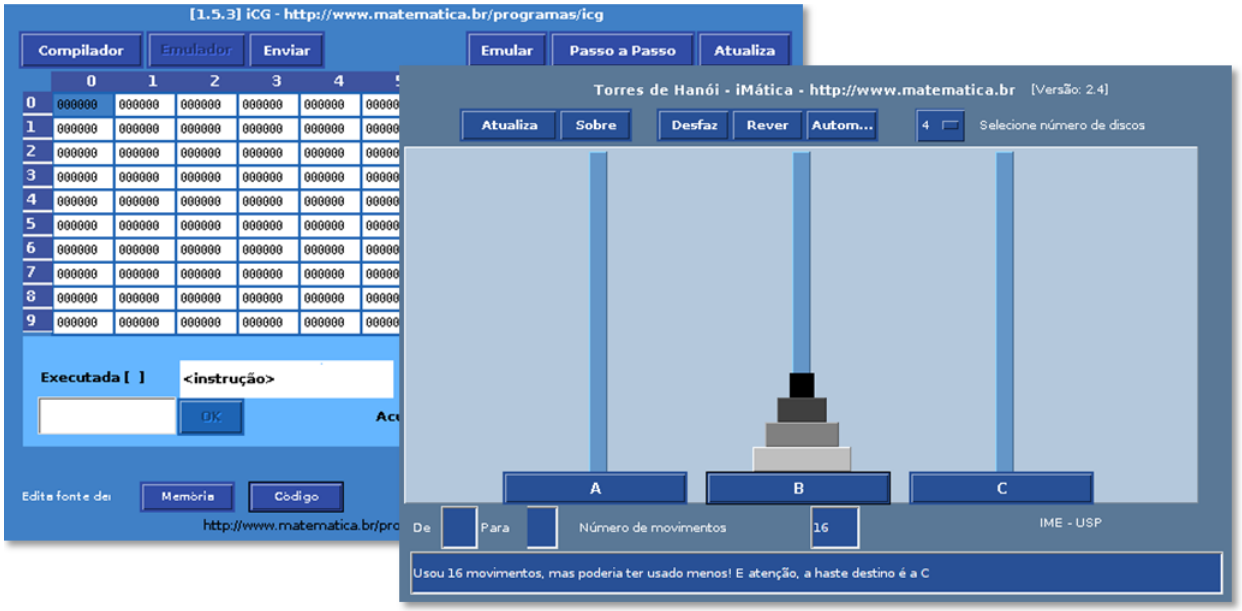

Figura 2.1: *Interface gráfica dos iMA iCG e iHanói.*

A demanda para o desenvolvimento de sistemas com mais funcionalidades seguindo o mesmo conceito era grande. Assim, com o trabalho de mestrado de Reginaldo do Prado, foi criado o *iGraf: Gráficos Interativos na Internet* (do Prado, 2008). Esse sistema é uma alternativa gratuita e que funciona plenamente em navegadores *Web* para sistemas de visualização de gráficos de funções matemáticas, como o WinPlot (Mpaka, 2010). Além disso, da mesma forma que o *iGeom*, oferece para os professores ferramentas de autoria e correção automática de atividades interativas e outras operações interativas com os gráficos de funções.

Com a popularização da Internet e da *Web*, por volta de 2004 os SGC também se tornaram muito populares (Green, 2010), o que fez surgir uma demanda de utilização dos sistemas desenvolvidos pelo grupo de pesquisa de maneira integrada em SGC. Para essa integração, deveria haver compatibilidade da parte dos aplicativos e da parte do SGC. Isso resultou no desenvolvimento de um novo sistema, o *SAW: Sistema de Aprendizagem pela Web*, princial contribuição do projeto foi do mestrado de Janine Moura (Moura, 2007). O SAW é um SGC que possui o primeiro protocolo implementado para esse conjunto de aplicativos, tendo sido testado com o *iGeom*, *iGraf* e *iCG*. A figura 2.2 mostra um exemplo de atividade do *iGeom* integrada ao SAW.

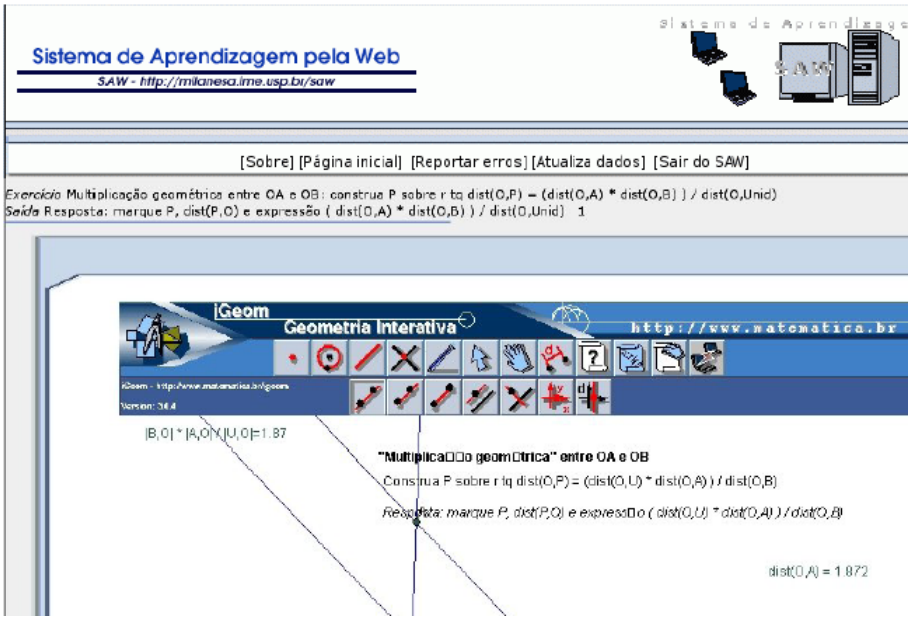

Figura 2.2: *Um exemplo de atividade do iGeom integrada ao SAW.*

Com o trabalho do SAW, sistemas compatíveis com esse SGC, como o *iGeom* e o *iGraf*, foram denominados Módulos de Aprendizagem (MA), termo que hoje se tornou Módulos de Aprendizagem Interativa (iMA), para dar mais destaque à interatividade. Assim, o SAW foi uma prova de conceito para a família iMA, definindo uma especificação para suas funcionalidades: compatibilidade com o SGC e ferramentas de autoria de atividades interativas.

Com o aprimoramento do conceito de iMA, outros dois projetos de mestrado culminaram no oferecimento de aplicativos pertencentes à família. O trabalho de Alexandre Eisenmann ofereceu o sistema *iComb: Combinatória Interativa na Internet*, um iMA para o ensino de Combinatória (Eisenmann, 2009) e o trabalho de Reginaldo Kamiya resultou no *iVProg: Programação Visual na Internet*, o iMA para auxiliar o ensino de introdução à Programação de Computadores na forma de um sistema de programação visual (Kamiya, 2010). Atualmente os principais integrantes da família iMA são: *iGeom*, *iGraf*, *iComb* e *iVProg*. A figura 2.3 apresenta exemplos representativos de suas telas principais.

Nos últimos anos, o SGC de código livre *Moodle* conquistou o título de SGC com maior número de usuários registrados (Moodle.org, 2012). Dessa forma, a integração de iMA no *Moodle* poderia contribuir significativamente com o aumento de usuários dos sistemas do grupo de pesquisa. Isso foi realizado com o trabalho de mestrado de Patrícia Rodrigues, que criou um módulo de funcionalidades para o SGC *Moodle* que implementa o protocolo de comunicação com os iMA, chamado

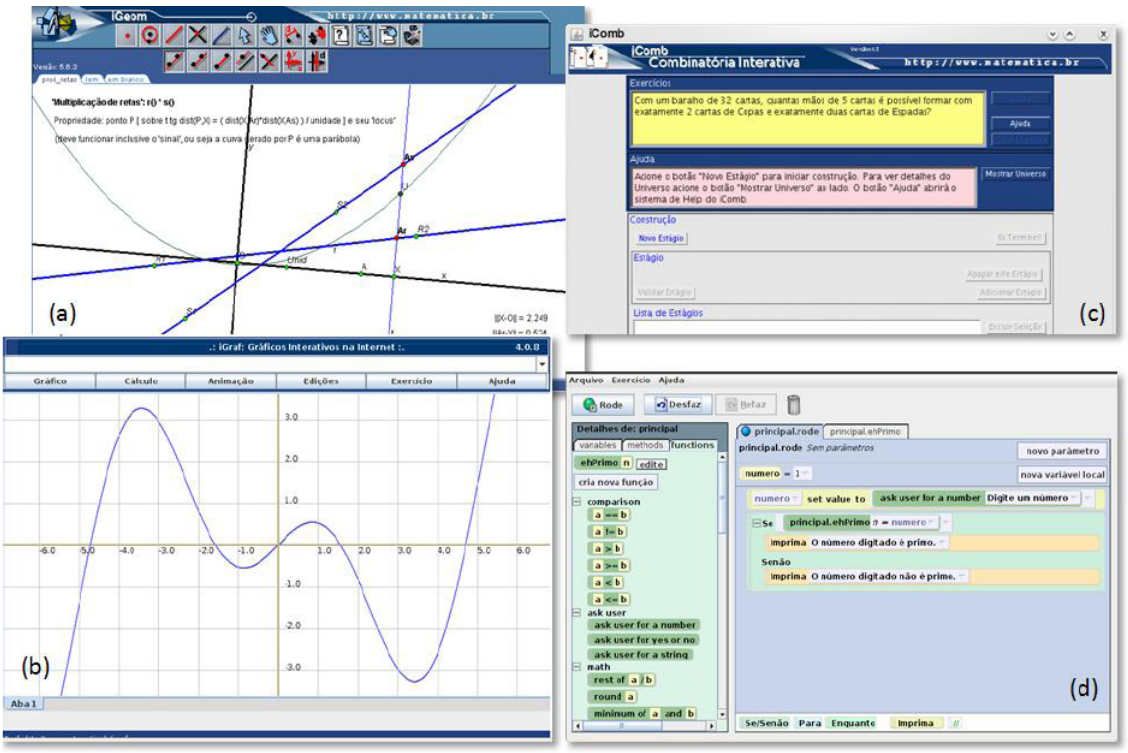

Figura 2.3: *Exemplos de telas dos iMA: (a) iGeom, (b) iGraf, (c) iComb e (d) iVProg.*

iTarefa (Rodrigues, 2011). Recentemente vários cursos foram ministrados com a utilização dos iMA no *Moodle* pelo módulo *iTarefa*, como o relatado por Rodrigues *et al.* (2011).

As atividades realizadas sobre os iMA em outros projetos ao longo da execução deste trabalho incluem adição de funcionalidades e refatoração do código. Os responsáveis pela execução dessas atividades são em geral alunos de pós-graduação sob orientação do coordenador do projeto. Atualmente, com a evolução da linha de produto e motivada pelas dificuldades de desenvolvimento, as tarefas de refatoração e documentação passaram a receber mais atenção. Uma vez descrita a história dessa família de aplicativos, a próxima seção apresenta as funcionalidades comuns compartilhadas por eles.

2.1.3 Funcionalidades Comuns

As definições das funcionalidades obrigatórias comuns aos iMA são provenientes dos trabalhos relacionados aos SGC SAW e ao *iTarefa*. Neles são definidas duas características: (i) ser *Applet* (Oracle, 2012a); e (ii) possuir um protocolo de comunicação específico. Ser *Applet* significa ser um aplicativo que funciona em páginas *Web* especificamente para a linguagem *Java*. Esse requisito visa facilitar o uso e o acesso, já que não há instalação ou dependência de sistema operacional e qualquer um com o endereço da página consegue usá-lo, uma vez que os navegadores atuais trazem consigo uma máquina virtual *Java* compatível.

O protocolo de comunicação visa integrar o uso do iMA como aplicativo de atividade com o SGC no qual a página *Web* está. Essa integração possui três vias de troca de informação: (i) o envio pelo SGC ao iMA da atividade a ser realizada, em conjunto com dados de configuração; (ii) a disponibilização pelo iMA ao SGC da resposta feita pelo aluno e (iii) do resultado da correção

automática dessa resposta, calculada pelo iMA. Esses requisitos são necessários para promover ao professor e ao aluno um controle das atividades realizadas e das respostas, podendo organizá-las em relatórios e gerar automaticamente as notas pelo sistema de notas do SGC (Rodrigues *et al.*, 2010). Uma nova versão do protocolo iMA é descrita em um relatório na página *Web* deste projeto (Dalmon e Brandão, 2012).

Além desses requisitos computacionais com motivação educacional, os iMA também têm outras funcionalidades comuns, que por não serem necessárias para a compatibilidade com os outros sistemas, geralmente não são descritas de forma explícita. De qualquer forma, os iMA existentes promovem ou planejam promover todas elas, que são: (i) ferramentas de autoria de atividades pelo professor; (ii) interatividade nas atividades; e (iii) correção automática da solução do aluno.

Ferramentas de autoria para professores são funcionalidades que permitem ao usuário (normalmente professor) criar uma atividade para ser realizada por outro usuário, normalmente aluno. Essa atividade é guardada em um arquivo e contém informações sobre o enunciado, as ações que o aluno pode realizar e o gabarito. Para que professores possam realizar a tarefa de autoria, a interface não pode exigir conhecimentos de computação. Isso contrasta com sistemas que permitem maior liberdade para os comportamentos do aplicativo durante a atividade mas exigem que o autor use *scripts* ou linguagens de programação (Moundridou e Virvou, 2003). Essa funcionalidade faz parte da abordagem dos iMA para aumentar a possibilidade de reúso descrita na seção anterior.

Um requisito para os iMA é que suas atividades devem ser *interativas*. A definição de atividade interativa ainda não é totalmente precisa, diversos autores definem o conceito de interatividade de diversas maneiras, dependendo do contexto e do foco de cada um (Chou *et al.*, 2010; Downes e McMillan, 2000; Sims, 1997). No caso dos iMA, a interatividade interessante para o aluno é aquela em que ele não use o computador apenas para resolver um exercício, mas que o faça para testar, simular, criar, conjecturar e outras ações consideradas relevantes para a aprendizagem (Tang, 2005). Existem classificações de tipos de interatividade, e os tipos que promovem essas ações do aluno são descritos como interatividade de simulação e de construção (Ohl, 2001; Sims, 1997).

A última funcionalidade comum aos iMA é a correção automática das soluções dos alunos. Apesar disso, o iMA *iVProg* ainda não a possui, por ser o mais novo membro da família. Com grande motivação educacional, a retroação (ou *feedback*) imediata, promovida por essa funcionalidade, possui efeitos muito positivos a alunos e professores (Isotani e Brandão, 2008; Koedinger e Aleven , 2007). Alunos têm a informação sobre o que ainda precisam estudar ou um conceito que não conhecem bem no momento apropriado. E professores podem se dedicar a tarefas mais sofisticadas, como resolver as dúvidas e ajudar alunos com dificuldades, enquanto o aplicativo realiza a operação repetitiva de corrigir a solução de todos.

A partir das informações sobre esses três aspectos dos iMA: contexto, histórico e funcionalidades comuns, é possível criar uma ideia geral sobre a família de aplicativos educacionais necessária para compreender o desenvolvimento da LPS feito neste trabalho. Informações mais detalhadas sobre as funcionalidades dos sistemas são apresentadas nas seções 4.1 e 4.2, e também no caso específico do iMA *iVProg*, na seção 5.2.

2.2 Linha de Produto de Software

Os iMA formam uma família de aplicativos similares, o que motivou a escolha da LPS como técnica para melhorar seus processos de desenvolvimento (Clements e Northrop, 2001). Nesta seção descrevemos com mais detalhes essa técnica e como foi definida sua abordagem nesta pesquisa de mestrado.

2.2.1 Definição e Benefícios

Linha de Produto de Software (LPS) pode ser definida como um conjunto de técnicas e métodos com o objetivo de facilitar e aprimorar o processo de desenvolvimento de famílias de aplicativos similares (Clements e Northrop, 2001). Essa técnica utiliza resultados obtidos na Engenharia de Software no desenvolvimento de aplicativos individuais em conjunto com boas práticas empíricas de desenvolvimento de famílias de aplicativos (Pohl *et al.*, 2005).

As contribuições da LPS podem ocorrer em quatro áreas: (i) negócios; (ii) arquitetura; (iii) processo; e (iv) organização (van der Linden *et al.*, 2007). A área de negócios diz respeito ao planejamento estratégico da instituição, que decide utilizar a LPS para desenvolver aplicativos em um domínio específico. Assim, a instituição deve tomar decisões em muitos aspectos em função da linha de produto, como escolha de mercado e dos produtos. No caso dos iMA, a definição da família de aplicativos similares evoluiu com o tempo internamente ao grupo de pesquisa, o que culminou neste projeto de desenvolvimento da LPS.

A arquitetura é relativa à parte interna dos aplicativos que serão desenvolvidos. Com o uso da LPS, os aplicativos possuem uma arquitetura comum e podem se beneficiar também de reúso de código. Outra influência da LPS é no processo de desenvolvimento, definindo como serão executadas as etapas de análise de requisitos, implementação e outras. Os reúsos de arquitetura e processo são mais indiretos para o programador quando trabalha em uma LPS, como é relatado no capítulo 5, mas seus benefícios de economia de tempo e esforço são consideráveis (Pohl *et al.*, 2005). Essas duas formas de reúso são importantes para o desenvolvimento em ambientes acadêmicos por facilitar o trabalho de novos programadores e reduzir as perdas com a saída de programadores experientes.

Por fim, a LPS também afeta a organização das pessoas que trabalham no projeto, definindo as tarefas pelas quais cada pessoa é responsável. Para este trabalho, por exemplo, em vez de cada programador trabalhar com um iMA, como era antes, o autor deste texto foi responsável somente pelos aspectos comuns, enquanto que os outros continuaram a trabalhar em seus iMA.

Todas essas mudanças requerem um custo muito grande por parte da instituição que adota a LPS (van der Linden *et al.*, 2007). Dessa forma, para que as vantagens da adoção da LPS sejam maiores do que esse custo, vários produtos devem ser previstos para produção. Estimativas dizem que entre três e quatro aplicativos desenvolvidos em uma LPS são suficientes para cobrir os custos iniciais (Clements e Northrop, 2001). As vantagens da adoção de uma LPS para produzir famílias de aplicativos são muitas. Em um nível mais alto, a instituição se torna mais unida por executar um projeto envolvendo mais pessoas, o que facilita a troca de informações, a organização dos profissionais e até as estratégias de vendas. Em um nível mais detalhado, o aplicativo recebe benefícios de maior reúso de código, reúso de arquitetura, e reúso de um processo sistemático de desenvolvimento, além disso, o trabalho dos profissionais é facilitado em várias etapas do desenvolvimento dos diferentes produtos.

2.2.2 Estrutura

Uma LPS é formada por três partes: (i) a engenharia de domínio; (ii) a engenharia de aplicação; e (iii) a evolução da própria LPS (Pohl *et al.*, 2005), como mostra a figura 2.4, adaptada de (Oberweis *et al.*, 2007). A **engenharia de domínio** é responsável pelo domínio de funcionalidades, ou seja, a área de atuação dos integrantes da LPS, definindo quais os possíveis aplicativos podem pertencer à família. É também responsável pela criação e manutenção do núcleo da LPS, que consiste no processo, arquitetura e código a serem reaproveitados no desenvolvimento dos aplicativos. A **engenharia de aplicação** é responsável pela utilização desse núcleo para a criação dos aplicativos. Dessa forma, a engenharia de domínio produz recursos para reúso, enquanto a engenharia de aplicação produz sistemas pelo reúso desses recursos. **A evolução da LPS** é responsável pelo ciclo de vida da LPS como um todo, da criação, crescimento e amadurecimento e ao fim da sua vida útil.

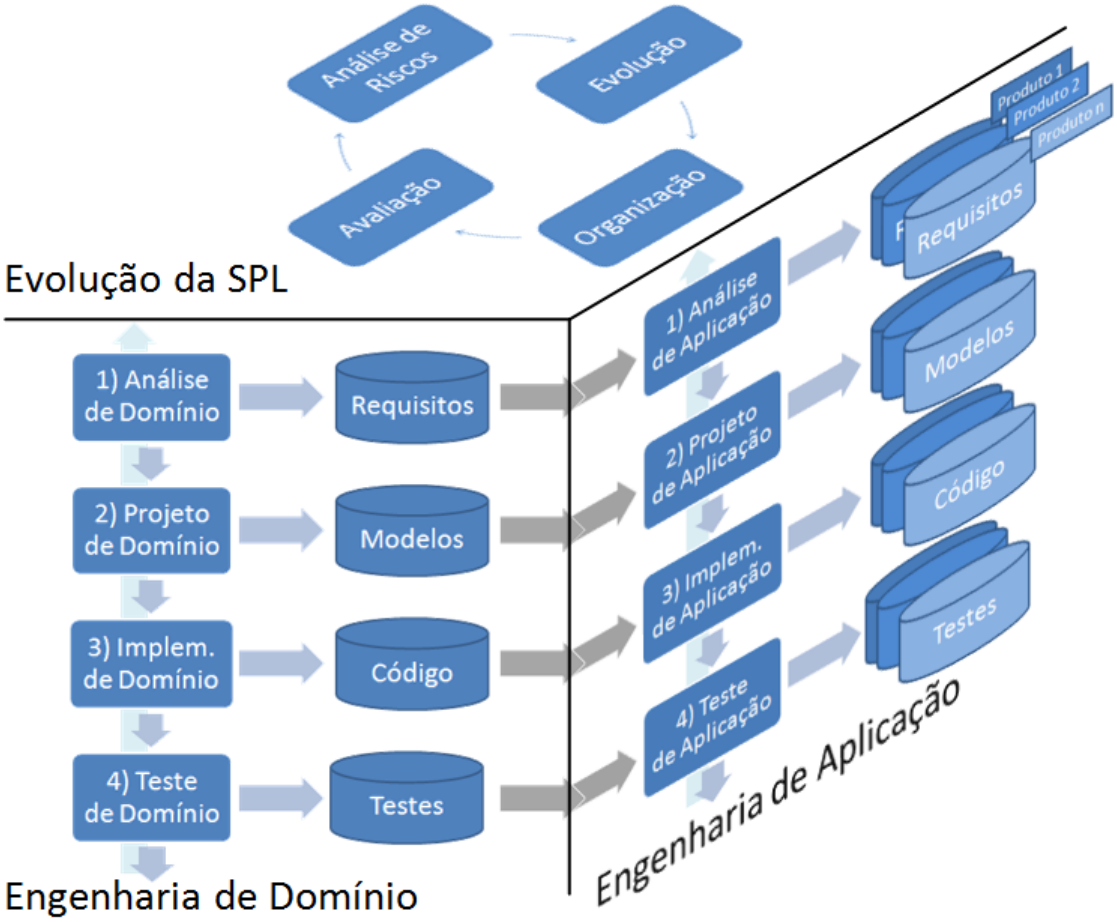

Figura 2.4: *Estrutura de uma LPS, com a engenharia de domínio, de aplicação e sua evolução.*

A engenharia de domínio realiza a análise de domínio, na qual são definidas as funcionalidades que farão parte do núcleo da linha e as que serão responsabilidade de cada aplicação. A partir desse resultado, o núcleo pode ser projetado e implementado, seguindo as orientações definidas pelo processo e pela organização da LPS. Os capítulos 3 e 4 descrevem respectivamente os métodos usados e os resultados do desenvolvimento da engenharia de domínio da LPS para iMA.

Após a primeira entrega do núcleo, resultado da engenharia de domínio, a engenharia de aplicação pode ser realizada. Essa etapa segue seu próprio processo de desenvolvimento para cada

produto sendo criado, assim é possível que programadores trabalhem paralelamente na engenharia de domínio e em várias instâncias de engenharia de aplicação. A engenharia de aplicação utiliza os recursos criados pela engenharia de domínio para produzir seu aplicativo. A seção 4.4 descreve o procedimento definido para a engenharia de aplicação da LPS para iMA. Muitas vezes, funcionalidades criadas na engenharia de aplicação podem ser incluídas no núcleo da linha para fazer parte de outros produtos.

Ao longo do tempo a LPS evolui, o que a permite mudar gradualmente o domínio em que atua, a definição das funcionalidades comuns nos aplicativos, o foco dos produtos sendo desenvolvidos ou mesmo deixar de criar novos aplicativos e se concentrar apenas em dar manutenção aos existentes. Essa parte da LPS recebe mais influência com relação ao aspecto dos negócios envolvidos.

2.2.3 Implementação

Como foi descrito, a LPS é uma técnica de desenvolvimento que atua em diversos aspectos da criação de aplicativos, inclusive na criação do código-fonte. Porém, ao adotar a LPS, a instituição não define as técnicas que serão utilizadas durante a implementação. É possível existir linhas de produto que atuam no mesmo domínio mas são implementadas de maneiras diferentes por causa das escolhas de técnicas envolvidas. Dessa forma, é necessário definir quais serão utilizadas na implementação da LPS para iMA.

Os principais critérios para escolher a forma de implementação são facilitar e aprimorar o projeto e a implementação de uma família de aplicativos similares, e estar de acordo com o contexto dos iMA (Clements e Northrop, 2001). Existem várias possibilidades de técnicas, aplicadas inclusive em aplicativos educacionais, como os exemplos: (i) arquitetura de componentes (Roschelle *et al.*, 2000; van Dam *et al.*, 2005); (ii) Programação Orientada a Aspectos (POA) (Ateyeh e Lockemann, 2006; Pankratius, 2005); e (iii) Desenvolvimento Dirigido por Modelos (DDM) (Choquet e Corbière, 2006; Dodero *et al.*, 2007). Arquitetura de componentes atende aos critérios, mas existem diversas maneiras de utilizá-la na prática, o que pode tornar esse uso complicado demais, inclusive para programadores sem experiência em orientação a objetos, como é frequente no desenvolvimento de iMA. POA e DDM oferecem muitas vantagens para os usuários, mas exigem uma grande especialização dos desenvolvedores da LPS. Isso impossibilita a aplicação dessas técnicas em nosso contexto, pois o grupo de pesquisa não possui experiência ou tempo hábil para se especializar.

Outras duas técnicas que possuem objetivo de promover reúso de código e facilitar a criação de aplicativos similares são as bibliotecas de classes e os arcabouços de aplicação (em inglês *Application Frameworks*). Bibliotecas de classes ou API (sigla para *Application Programming Interface* ou Interface de Programação de Aplicativos) são conjuntos de funções prontas disponíveis para o programador e geralmente são relacionadas a um domínio (Chan e Lee, 1996). Ambas as técnicas estão de acordo com os critérios citados acima, pois não requerem nenhum conhecimento adicional além de programação orientada a objetos para utilizá-las. A principal diferença entre biblioteca de classes e arcabouços é que arcabouços fornecem, além de uma interface de programação, uma arquitetura reutilizável e a *inversão de controle*. A inversão de controle faz com que o arcabouço controle o fluxo de dados, o que, no caso das bibliotecas, é feito apenas pelo aplicativo sendo desenvolvido e não pela própria API. A próxima seção apresenta uma descrição mais detalhada sobre os arcabouços de aplicação.

2.2.4 Arcabouço de Aplicação

A técnica de arcabouço de aplicação é adaptada para a implementação da LPS para iMA, pois é uma forma de organizar o uso de uma arquitetura de componentes e possui um maior potencial de reúso de código quando comparada a bibliotecas de classes, provocado pelo reúso de arquitetura e de processo. O reúso de processo promovido pela LPS é especificamente o aproveitamento do método de utilização do arcabouço desenvolvido.

Arcabouços de aplicação, constituídos de código-fonte e documentação, fornecem um conjunto de funções prontas, um conjunto de funções pré-definidas a serem implementadas e uma arquitetura padrão com o objetivo de facilitar a criação de aplicativos em um dado domínio (Fayad e Johnson , 2000; Guerra, 2010). As principais características de um arcabouço de aplicação são: (i) o fornecimento de funcionalidades prontas na forma de uma API, o que promove reúso de código; (ii) o fornecimento de uma arquitetura padrão para os aplicativos com a inversão do controle do fluxo de informações, que permite limitar o trabalho do programador a configurar pontos específicos da aplicação ao invés de criar o seu funcionamento interno; e (iii) possuem modularidade e a possibilidade de extensão do número de funcionalidades ou da arquitetura, possivelmente pela inclusão de código criado para os aplicativos da família.

Existem outros tipos de arcabouço (Fayad e Schmidt, 1997), definidos de acordo com a sua função, como arcabouços de infraestrutura e comunicação e de integração de *middleware*. Arcabouços de aplicação têm a particularidade de serem voltados a um domínio específico para o qual serão criados aplicativos para o usuário final.

De maneira similar à LPS, a criação de arcabouços de aplicação exige um esforço inicial por parte da instituição que, em geral, é compensado com a criação de três aplicativos que o utilizam (Fayad e Johnson, 2000). No contexto dos iMA, os quatro aplicativos existentes podem ser refatorados com o arcabouço proposto, o que pode justificar esse esforço inicial.

Com relação ao uso por desenvolvedores, um arcabouço de aplicação pode ser dividido em duas partes: a que o programador utiliza sem modificar, chamada de *frozen spots* e a que ele utiliza pela extensão de uma funcionalidade, chamada de *hot spots* (Fayad *et al.*, 1999). Considerando arcabouços criados com conceitos de orientação a objeto (Fayad e Schmidt, 1997), a utilização de *frozen spots* é feita pela chamada de operações de interfaces ou classes públicas, enquanto que a utilização de *hot spots* pode ser feita pela herança de classes abstratas, a sobrescrita de métodos ou outras técnicas de flexibilização baseadas em conceitos de orientação a objetos. Atualmente técnicas mais sofisticadas são usadas, como a injeção de dependências, reflexão, metadados, configurações e anotações (Guerra, 2010).

O processo de implementação de arcabouços de aplicação pode variar substancialmente de acordo com as tecnologias utilizadas, como é mostrado nos exemplos de (Fayad e Johnson, 2000; Fayad *et al.*, 1999; Guerra, 2010). Cada arcabouço é desenvolvido para um contexto e os principais fatores durante esse processo são dependentes das tecnologias envolvidas.

No caso dos iMA, é necessário que a criação do arcabouço utilize o menor número de tecnologias possível, com o objetivo de facilitar a curva de aprendizado. Assim, para estar de acordo com os trabalhos do grupo de pesquisa, a lingugagem *Java* é usada e a implementação de componentes é feita com conceitos básicos de orientação a objeto (Fayad e Schmidt, 1997), sem a utilização de outras tecnologias ou outros arcabouços além do que é oferecido pela distribuição padrão da linguagem.

As informações sobre LPS fundamentais para este trabalho foram apresentadas abordando descrição geral, estrutura e a forma de específica de implementação escolhida, o arcabouço de aplicação. O método conduzido para o desenvolvimento da LPS para iMA é apresentado no capítulo 3, enquanto que as características finais da linha estão no capítulo 4. Antes disso, a próxima seção é uma revisão bibliográfica, descrevendo o contexto acadêmico em que esta pesquisa se insere e seus trabalhos relacionados.

2.3 Trabalhos Relacionados

Esta seção apresenta os trabalhos relacionados encontrados na literatura. Esses foram divididos em duas categorias, uma relacionada aos aspectos gerais de desenvolvimento de aplicativos educacionais e outra que trata especificamente da etapa de programação ou da utilização de técnicas de Engenharia de Software nesse contexto. Antes são feitas considerações sobre o método empregado e o escopo.

Método

O método para realizar a revisão da literatura foi o seguinte. Selecionado um tópico de interesse para o trabalho, por exemplo, desenvolvimento de aplicativos educacionais, uma pesquisa na *Web* com os termos em português e em inglês foi feita nos sítios de busca *Google Scholar* (Google, 2012) e *Citeseer* (The Pennsylvania State University, 2012). Dessa pesquisa, os trabalhos com mais citações e com títulos aderentes aos termos foram selecionados e lidos. Além disso, durante a leitura, artigos citados por eles relevantes a esta pesquisa também foram selecionados. Por fim, quando o artigo era importante, procuramos nos mesmos sítios os artigos que o citam.

Apesar de não haver um método rígido para a realização dessa revisão, houve várias iterações nas buscas dos termos mais relevantes. Em cada busca foram considerados resultados menos relevantes na classificação usadas pelos sítios, até a quinta página em geral. Uma revisão sistemática da bibliografia não foi feita principalmente pela definição tardia do tema de pesquisa, que evoluiu durante mais de um ano no período do programa de mestrado do autor.

Para indicar a falta de artigos sobre o tema de estudo do processo de desenvolvimento de aplicativos educacionais, buscas feitas na ferramenta *Google Scholar* com os termos em inglês (*"product line"* ou *framework* ou *component* ou *development*) e (*educational software* ou *elearning*) no título retorna menos de 400 resultados no total. Para esse motor de busca especificamente, esse é um resultado bem pequeno, sendo que uma busca equivalente para o termo *elearning* retorna mais de 4000 artigos. Outro fator que reduz o número de artigos no tema é que mesmo com os termos procurados no título ou resumo, grande parte dos artigos apresenta exemplos de aplicativos educacionais ou de utilização dessas ferramentas em salas de aula, considerando pouco os aspectos do processo de desenvolvimento.

Escopo

Muitas vezes há uma confusão entre os termos *aplicativo educacional* e *objeto de aprendizagem*. Na área de Informática na Educação, "Objeto de Aprendizagem" é qualquer recurso digital reutilizável (Wiley, 2000). Assim, tanto arquivos de texto ou apresentações de slides quanto aplicativos

educacionais podem ser objetos de aprendizagem. Nos trabalhos relacionados, há artigos de desenvolvimento de objetos de aprendizagem como aplicativos educacionais, mas também há alguns que apresentam técnicas para reúso de recursos digitais como documentos.

O domínio dos trabalhos relacionados apresentados é o de desenvolvimento de aplicativos educacionais, seja em termos gerais (seção 2.3.1), ou especificamente na aplicação de Engenharia de Software em seu desenvolvimento (seção 2.3.2).

2.3.1 Desenvolvimento de Aplicativos Educacionais

O desenvolvimento de aplicativos educacionais pode ser dividido em duas grandes etapas: projeto instrucional e projeto e implementação de software. O projeto instrucional exige uma equipe multidisciplinar que envolve, em geral, professores, projetistas instrucionais, programadores, administradores de ambientes educacionais, responsáveis pelo currículo, pedagogos, coordenadores, entre outros (Winters e Mor, 2008). Essa variedade de profissionais pode dificultar a especificação de requisitos para a etapa de projeto de software (Douglas, 2001). Além disso, esses profissionais podem não possuir experiência em computação, o que pode também prejudicar essa especificação e provocar mudanças frequentes nos requisitos (Spalter e van Dam, 2003).

Quando o aplicativo é desenvolvido em ambiente acadêmico, pode haver fatores adicionais influentes no projeto. Um deles é a relação do aplicativo com projetos de pesquisa, o que provoca a descontinuidade do desenvolvimento ou do suporte uma vez que o projeto de pesquisa se encerra. Além disso, no caso em que os programadores são alunos, sua experiência em programação é pequena e caso alcancem certa maturidade durante o andamento do projeto, podem se desligar quando terminarem seus estudos, prejudicando consideravelmente o futuro do aplicativo (Spalter e van Dam, 2003).

Projeto Instrucional, também chamado de projeto de sistemas instrucionais, é a etapa de desenvolvimento de aplicações educacionais na qual são definidos a abordagem pedagógica do aplicativo com relação ao aluno e o conjunto de atividades proporcionadas (Squires e Preece, 1999a). Essa é uma etapa essencial no processo de desenvolvimento de aplicativos educacionais, cuja literatura apresenta diversas contribuições, como métodos para o projeto instrucional (Boyle, 2003; Nash, 2005; Squires e Preece, 1999a) e padrões de projeto educacionais (McAndrew *et al.*, 2006; Sampson, 2005). Um artigo especificamente adapta conceitos de métodos de desenvolvimento ágil da Engenharia de Software para todo o processo de criação de recursos digitais educacionais (Geller *et al.*, 2009).

Por exemplo, Quintana *et al.* (2003) apresenta distinções entre o desenvolvimento de aplicativos profissionais e educacionais. Aplicativos profissionais devem ser projetados considerando o usuário, que o utiliza para executar uma tarefa. Aplicativos educacionais, por outro lado, devem ser projetados considerando o usuário como aprendiz, que o utiliza para aprender a executar uma tarefa, o que faz com que sua utilização seja diferente e consequentemente seu desenvolvimento.

O livro escrito por Tchounikine (2011) apresenta muitas considerações importantes no desenvolvimento de aplicativos educacionais. Ele define fatores importantes no processo, computacionais, pedagógicos e transdisciplinares, que muitas vezes são desconsiderados inclusive em projetos de pesquisa. Outras contribuições são a descrição das possibilidades de papéis exercidos pela computação em cada projeto, e a importância da "capitalização", ou explicitação, do conhecimento para a área. Essa capitalização consiste em destacar as lições aprendidas dos projetos explicitamente, para que

outros projetos as aproveitem. Hoje em dia ela é pouco realizada, uma vez que desenvolvedores apresentam ou relatam erros similares em muitos artigos.

Apesar dessa área de pesquisa ter alcançado muitos avanços, de acordo com a pesquisa bibliográfica realizada, existe uma falta de distinção entre os termos "projeto de aplicativos educacionais" e "desenvolvimento de aplicativos educacionais" (em inglês, respectivamente, *Design of Educational Software* e *Development of Educational Software*). Há trabalhos que consideram "desenvolvimento de aplicativos educacionais" como apenas a etapa de projeto instrucional (Boyle, 2003; Nash, 2005; Squires e Preece, 1999a) e há trabalhos que consideram como todo o processo, desde o projeto instrucional até o projeto de software e implementação (Hinostroza *et al.*, 2000; Merril, 1984; Meziane *et al.*, 1999). Esta última é utilizada nesta dissertação pois o foco está especificamente na etapa de projeto de sofware e implementação, apresentada a seguir.

2.3.2 Engenharia de Software em Aplicativos Educacionais

A etapa de projeto e implementação de software é a que a computação sofre menos influência das outras áreas e os métodos utilizados são providos pela Engenharia de Software, apesar de existirem métodos de projeto instrucional que envolvem etapas iterativas entre projeto e implementação (Boyle, 2003). Ao analisar um aplicativo educacional, é possível considerar dois tipos de qualidade: pedagógica ou instrucional, e computacional ou de software. A qualidade pedagógica influencia diretamente as experiências de uso por alunos ao estudarem (Matos *et al.*, 2010; Squires e Preece, 1999b). Os métodos de projeto instrucional visam assegurar ou aumentar essa qualidade (Mor e Winters, 2007).

A qualidade computacional também possui uma influência sobre as experiências de aprendizagem (Schleyer e Johnson, 2003). Por exemplo, um aplicativo com defeitos pode desmotivar alunos e influenciar negativamente a aprendizagem. No caso, baixa qualidade computacional pode provocar dificuldades durante a manutenção, como a lentidão na atualização e correção de defeitos pelos desenvolvedores (Ben-Menachem e Marliss, 1997). Esses e outros problemas de qualidade computacional podem reduzir o número de usuários do aplicativo e consequentemente sua contribuição à comunidade (Mor e Winters, 2007).

A seguir, é apresentada uma descrição cronológica de trabalhos que relatam o uso de técnicas da Engenharia de Software nos processos de desenvolvimento de aplicativos educacionais. Comentários sobre os trabalhos são feitos em seguida.

Em 1986, Nicolson e Scott (1986) relataram dificuldades durante o projeto de software e implementação de aplicativos educacionais. Para resolver esses problemas, propuseram três iniciativas: (i) o uso de *software toolboxes*; (ii) a criação de arcabouços com funcionalidades de CAI (*Computer Aided Instruction*); e (iii) o emprego de consultores em CAI durante o desenvolvimento. As *software toolboxes* propostas se assemelham às modernas bibliotecas de classes, uma técnica de reúso sistemático de código que pode aprimorar consideravelmente a implementação de aplicativos educacionais, de maneira similar a arcabouços com as funcionalidades do domínio (no caso, CAI). A inclusão de especialistas em CAI no desenvolvimento entra na equipe multidisciplinar, o que cumpriria um papel de intermediário entre os grupos de profissionais da área pedagógica e da área da computação. Essas sugestões estão de acordo com a proposta deste trabalho, sendo que as duas primeiras diferem basicamente com relação ao domínio aplicado e às técnicas utilizadas, que nos últimos 25 anos obtiveram muitos avanços.

Posteriormente, Roschelle *et al.* (1998) explicam que a produção da comunidade de Informática na Educação continha, na sua maioria, aplicativos isolados e incompatíveis, o que torna o processo de desenvolvimento mais lento e caro. Assim, resultados de um projeto de criação de uma família de aplicativos usando arquitetura baseada em componentes para algumas áreas do ensino fundamental são apresentados. Esse tipo de arquitetura apresenta vantagens como reúso de código e modularidade, e desvantagens como problemas de comunicação entre os componentes e aumento da complexidade do gerenciamento das equipes de implementação. O uso de tecnologias emergentes da época, como *JavaBeans* (Oracle, 2012c) e repositórios poderia facilitar ainda mais o uso de componentes.

Do mesmo grupo de pesquisa, um outro projeto com objetivos similares, chamado ESCOT, foi realizado em seguida (Repenning *et al.*, 2001; Roschelle *et al.*, 1999, 2000). Com o uso dessas "novas" tecnologias, os artigos apresentam três contribuições principais: (i) a importância da definição dos componentes; (ii) a importância da utilização de convenções simples de comunicação entre eles; e (iii) a necessidade de estender essas técnicas para comportamentos dinâmicos, novos tipos de dados e configurações. Essas são contribuições bem específicas para o uso de componentes *JavaBeans*, que estão incluídas nas técnicas de Engenharia de Software atuais de maneira mais genérica (Schmidt *et al.*, 2000).

Conlan *et al.* (2002) realizaram um trabalho com relação à comunicação entre componentes baseada em metadados para o domínio de sistemas de aprendizagem adaptativos, cuja adaptação é feita com a escolha de componentes usando os metadados. Uma contribuição para a implementação dos componentes é a utilização dos metadados também para a definição das abstrações a serem implementadas. De maneira similar, Bote-Lorenzo *et al.* (2004) relatam o uso da especificação IMS-LD (Berggren *et al.*, 2005) para especificar fluxos de atividades colaborativas. Essas especificações são usadas para criar aplicativos a partir de componentes de *software* providos por instituições externas. Outros trabalhos que se dedicaram à aquitetura de componentes para desenvolver aplicativos educacionais são (Boyle, 2003; Douglas, 2001; Polsani, 2003).

Um projeto similar ao ESCOT, chamado *Exploratories*, também usou as contribuições da arquitetura em componentes para facilitar o processo de projeto de software e implementação de aplicativos educacionais (van Dam *et al.*, 2005). Além das vantagens já apresentadas, esse mesmo projeto permitiu a descoberta de várias particularidades do processo de desenvolvimento de aplicativos educacionais que não necessariamente estão ligadas a problemas de Engenharia de Software (Spalter e van Dam, 2003).

Pankratius (2005) fez um trabalho de reengenharia com aplicativos educacionais existentes utilizando Programação Orientada a Aspectos (POA) (Kiczales e Hilsdale, 2001). Com esse paradigma de programação ele apresenta resultados de organização, redução da redundância do conjunto de aplicativos da sua instituição e aprimoramento considerável do trabalho de manutenção desses aplicativos. O trabalho de Ateyeh e Lockemann (2006) também utiliza POA para a organização e reúso de aplicativos educacionais.

Também com o intuito de realizar a reengenharia de aplicativos educacionais, Choquet e Corbière (2006) usaram um arcabouço baseado na técnica de Desenvolvimento Dirigido por Modelos (DDM) (Hailpern e Tarr, 2006). Os autores conseguiram unificar o projeto instrucional e o projeto de software de uma forma que facilitou o processo inteiro de desenvolvimento. Em uma outra utilização da técnica DDM (Dodero *et al.*, 2007), os autores foram capazes de superar as dificuldades de definição

de modelos pedagógicos ou níveis de abstração durante o projeto instrucional e a especificação dos requisitos para a implementação, que pode ser automatizada a partir dos modelos. Isso mostra que o uso de técnicas de Engenharia de Software pode trazer muitos benefícios ao desenvolvimento de aplicativos educacionais.

No trabalho apresentado por Oberweis *et al.* (2007), foi criada uma LPS para Produtos de Informação Digital, que compreende principalmente documentos e apresentações de slides educacionais. Além da LPS, os autores apresentam com detalhes definições e adaptações para LPS de recursos digitais educacionais, o que não pode ser usado diretamente por este trabalho, cujo objeto de pesquisa são aplicativos de atividade. Há outros artigos citados aqui em que não é descrito explicitamente o recurso criado com as técnicas de Engenharia de Software aplicadas. Nesses casos, suas contribuições diretas para esta pesquisa são as técnicas e o contexto, mas não o produto final e seus resultados.

Trabalhos brasileiros recentes incluem o desenvolvimento de uma LPS de aplicativos educacionais de colaboração baseados no modelo 3C (comunicação, coordenação e cooperação) e no *Groupware Workbench* (Gadelha *et al.*, 2010), e uma LPS para o desenvolvimento redes sociais (Oliveira e Gerosa, 2011). O interessante deste último trabalho é a avaliação utilizada para verificar a modelagem de domínio e os produtos criados com a LPS. Esse método de avaliação inspirou o utilizado nesta pesquisa, com a diferença de que nosso objetivo é avaliar o processo de desenvolvimento.

Três últimos artigos apresentam estratégias para o desenvolvimento de famílias de aplicativos educacionais. Em (Reis, 2007), é apresentado um processo e arquitetura em camadas similar a uma LPS que usam repositórios de componentes. Há o relato do uso de análise de características para guiar o processo de modelagem, produção e implementação de aplicativos em repositórios de componentes, mostrando uma prova de conceito no domínio de circuitos lógicos (Stuikys e Damasevicius, 2008). Por fim, Ahmed e Zualkernan (2011) relatam o desenvolvimento de uma LPS para aplicativos educacionais baseado no modelo ED^2, e também um exemplo de uso em um curso para ensino de engenharia de qualidade.

Os trabalhos apresentados aqui podem ser classificados quanto ao tipo de aplicativo educacional, se esses promovem recursos ou funcionalidades genéricas (como documentos, formulários e fóruns) ou atividades interativas (como simulações, animações e interatividades de manipulação). Além disso, podem também ser divididos com relação à técnica de Engenharia de Software usada, se prevalece o reúso de código (como bibliotecas, especificações e componentes) ou se também há reúso de arquitetura e processo (como arcabouços e LPS). A tabela 2.1 apresenta os trabalhos citados classificados quanto a esses dois aspectos.

Essa tabela mostra sete projetos de desenvolvimento na categoria de aplicativos com funcionalidades genéricas usando técnicas de apenas reúso de código, cinco projetos na categoria de genéricas-processo e arquitetura, três na atividade-código, e dois na de atividade-processo e arquitetura. Não foram incluídos no quadro mais de um artigo do mesmo projeto, apesar de serem citados no texto. Há menos trabalhos relacionados a aplicativos de atividade e reúso de arquitetura e processo. Porém, a data de publicação desses artigos mostra que a tendência mais recente é a de desenvolver aplicativos de atividade com técnicas que promovem os três tipos de reúso, o que pode revelar uma tendência à qual este trabalho está incluído.

	Aplicativos com funcionalidades genéricas	Aplicativos de atividades interativas
Técnica apenas com reúso de código	(Douglas, 2001), (Boyle, 2003), (Polsani, 2003), (Pankratius , 2005), (Ateyeh e Lockemann , 2006), (Dodero *et al.*, 2007) e (Choquet e Corbière, 2006).	(Nicolson e Scott, 1986), (Roschelle *et al.*, 1998), (Conlan *et al.*, 2002), (van Dam *et al.*, 2005) e (Bote-Lorenzo *et al.*, 2004).
Técnica também com reúso de processo e arquitetura	(Oberweis *et al.*, 2007), (Reis , 2007), (Gadelha *et al.*, 2010) e (Oliveira e Gerosa, 2011),	(Stuikys e Damasevicius, 2008) e (Ahmed e Zualkernan, 2011).

Tabela 2.1: *Classificação dos trabalhos relacionados quanto aos aplicativos desenvolvidos e a técnica de Engenharia de Software utilizada.*

Em resumo, a literatura apresenta diversos esforços para aprimorar o desenvolvimento de aplicativos educacionais. Os trabalhos relacionados descritos aqui contribuíram para esta pesquisa com as diferentes técnicas utilizadas, os vários tipos de sistemas desenvolvidos e principalmente com as lições aprendidas específicas do contexto de cada artigo. Apesar da maior parte deles não relatar o desenvolvimento de aplicativos de atividades interativas, percebemos que não apenas o reúso de código é importante, mas o de processo e arquitetura também são muito influentes no desenvolvimento de aplicativos educacionais.

Infelizmente, poucos artigos relatam consistentemente uma análise mais detalhada do trabalho realizado, com vantagens e desvantagens, métodos de avaliação e descrição dos contextos (sistema, tamanho da equipe, experiência dos programadores), o que limita a explicitação do conhecimento, de acordo com Tchounikine (2011). Isso ainda pode ser considerada uma característica da área de Informática na Educação (Dalmon *et al.*, 2012b), que deve evoluir nos próximos anos, com a influência de trabalhos que explicitam suas lições aprendidas.

Este trabalho pretende também contribuir dessa maneira. O próximo capítulo apresenta os métodos utilizados para desenvolver a LPS para iMA e para avaliá-la, de maneira a permitir a explicitação do conhecimento sobre como o trabalho foi executado.

Capítulo 3

Métodos

Este capítulo apresenta os métodos utilizados na execução deste trabalho. Para o desenvolvimento da LPS proposta, os métodos foram divididos em dois grupos: de análise de domínio e de engenharia de domínio. Há uma seção específica para cada grupo. Para a avaliação da LPS criada outros métodos foram utilizados, descritos na última seção do capítulo.

Uma característica importante do método utilizado, que permeia todas as etapas é o desenvolvimento iterativo. Dessa forma, sempre que necessário, uma etapa já realizada era revista de acordo com informações obtidas durante etapas posteriores. Isso foi necessário pelo andamento do projeto, que amadureceu aos poucos e exigiu a consideração de fatores que foram percebidos em períodos posteriores aos que são usualmente, ou idealmente, analisados.

3.1 Análise de Domínio

A análise de domínio, na implantação de uma LPS, consiste em determinar os requisitos dos aplicativos que farão parte da família de produtos a partir de um estudo detalhado do domínio ao qual a LPS pertence (Clements e Northrop, 2001). No caso da LPS para iMA, ela é realizada a partir de aplicativos existentes. Assim, a análise de domínio deve estudar primeiramente os requisitos utilizados no desenvolvimento dos aplicativos existentes e em seguida os requisitos novos, ou desejados, para esses sistemas. A etapa seguinte da análise de domínio foi a aplicação do método FODA (*Feature Oriented Domain Analysis*) (Kang *et al.*, 1990) a partir dos requisitos mapeados.

3.1.1 Análise de Requisitos

Os iMA desenvolvidos pelo grupo de pesquisa formam uma família de aplicativos similares. A LPS para iMA deve ser usada para criar produtos que atendam aos mesmos requisitos utilizados no desenvolvimento dos iMA existentes, que definem os requisitos da própria LPS. Além disso, ela pode incluir requisitos adicionais, representando uma evolução e refatoração desses sistemas.

Essa análise se deu em duas partes: análise das características gerais dos iMA, e análise das suas funcionalidades. A análise das características gerais foi realizada pela leitura dos trabalhos publicados sobre os aplicativos, comparação com aplicativos apresentados em outros trabalhos, e em conversas e reuniões do grupo de pesquisa. Assim foi possível analisar os principais aspectos dos iMA como uma família de aplicativos. Os resultados dessa etapa são apresentados na descrição desses sistemas, na seção 2.1.

A análise das funcionalidades dos iMA existentes foi realizada da seguinte maneira. Como o *iGeom* foi o primeiro iMA criado e o que possui o maior número de funcionalidades, foi iniciado por ele o levantamanto de funcionalidades para servirem de base à família. Em seguida, as funcionalidades dos outros iMA foram listadas e comparadas com as do *iGeom*. O último passo foi classificar cada funcionalidade de acordo com seu uso na família de aplicativos, nesse caso elas poderiam ser **comuns**, caso pertencessem ou pudessem ser criadas em mais de um iMA ou **específicas**, caso contrário.

Em conjunto com os requisitos determinados a partir das funcionalidades dos sistemas existentes, a LPS para iMA pode incluir também requisitos desejados ou planejados para esses aplicativos individualmente. Esses requisitos foram determinados a partir de conversas e reuniões com os desenvolvedores dos aplicativos da família, uma vez que não há documentação dos projetos com relação a esse aspecto.

Os requisitos levantados nessa etapa de análise foram formulados considerando principalmente seu caráter funcional de forma a serem suficientes para a definição das características nas etapas posteriores. Uma definição mais precisa e uma análise quantitativa dos requisitos deve ser feita como trabalho futuro. A classificação dos requisitos e das funcionalidades quanto à ocorrência nos iMA é importante para a aplicação de métodos sistemáticos para a análise de domínio. Na subseção seguinte é apresentado o método utilizado para essa tarefa.

3.1.2 Análise de Características

Um método sistemático de análise de domínio para uma LPS é o **FODA** (***Feature Oriented Domain Analysis***) (Kang *et al.*, 1990). Esse método descreve como realizar a análise de domínio, o que resultará em um diagrama de características (termo traduzido do inglês *features*). Após a definição dos requisitos nas etapas descritas acima, o método FODA foi aplicado com os objetivos princiais de sistematizar e resumir o conjunto de requisitos da LPS para iMA, e de documentar o processo da análise de domínio. Esse método foi escolhido por ser o mais utilizado em projetos de LPS para a modelagem de características e por ser compatível com o método usado na definição da arquitetura, apresentado na seção 3.2.2.

A análise de domínio pelo método FODA é dividida em três partes: (i) análise do contexto; (ii) modelagem do domínio; e (iii) modelagem da arquitetura. A análise do contexto trata vários aspectos, como a definição do escopo da LPS, os limites do domínio no qual atuará, os possíveis produtos (aplicativos) que poderão ser criados usando a LPS e a interação com outros domínios. Essa tarefa pode ser feita em texto apenas ou usando diagramas de estrutura e de contexto.

A modelagem do domínio pode ser dividida ainda em três partes: (i) análise das características, (ii) modelagem de relacionamentos e entidades e (iii) análise funcional. A análise das características é a principal atividade e a mais conhecida do método FODA. Uma característica é um aspecto, qualidade ou propriedade de um aplicativo que seja visível e distinto ao usuário final (Kang *et al.*, 1990). Essa é uma forma de abstrair conceitos de uma maneira que usuários e desenvolvedores consegue se entender bem, além de permitir de maneira simples e precisa a especificação de aplicativos dentro de uma família. Um aplicativo pode ser representado por um conjunto de características escolhidas dentre as disponíveis para a família à qual pertence.

Assim, uma família de aplicativos pode ser definida por um conjunto de características comuns a todos os seus integrantes e algumas específicas a algum deles, e representada por um diagrama de

características, como exemplificado na figura 3.1, adaptada de (Kang *et al.*, 1990). O diagrama de características apresenta de forma gráfica as características presentes em uma família de produtos e as classifica em: (i) obrigatórias, que todos os aplicativos possuem, como a campainha e o relógio no caso do microondas; (ii) opcionais, que os aplicativos podem ter ou não, indicadas pela terminação da linha em um círculo, como o comando "descongelar comida"; e (iii) variantes, que os aplicativos devem escolher uma variante da característica (como o tipo de campainha) ou podem escolher (como o tipo do descongelamento, não mostrado na figura), e são indicadas pelo arco entre as linhas de cada uma. As regras de composição apresentam relações entre características e atributos dos produtos. O exemplo do microondas também será usado na próxima seção.

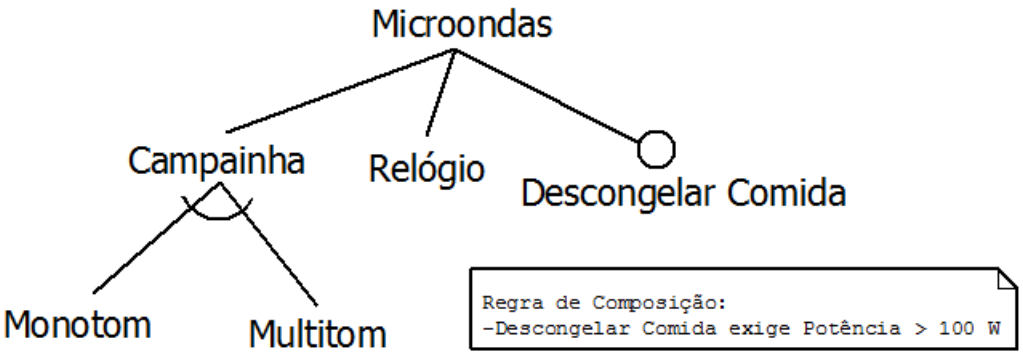

Figura 3.1: *Exemplo de diagrama de características, baseado em (Kang et al., 1990).*

As outras etapas, modelagem de relacionamentos e entidades e análise funcional, também têm resultados em forma de diagramas. Esses diagramas podem ser considerados precursores da linguagem UML atual (Miles e Hamilton, 2006), que se consolidou no final dos anos 90, quase dez anos depois da proposta do método FODA. A modelagem de relacionamentos e entidades gera um diagrama similar ao diagrama de classes, enquanto que a análise funcional assemelha-se à análise de casos de uso. Utilizamos a linguagem UML para essas etapas, em vez do método FODA, como é explicado na seção 3.2.2.

A modelagem da arquitetura representa componentes, camadas e outros padrões arquiteturais em diagramas após uma análise sistemática dos elementos constituintes dos aplicativos da família. Por usarmos outros métodos para a definição da arquitetura, o foco do método FODA nesta dissertação está na análise do contexto e na modelagem do domínio.

3.2 Engenharia de Domínio

Engenharia de Domínio é o processo de utilização de conhecimento específico de um domínio para a produção de recursos genéricos dentro desse domínio (Clements e Northrop, 2001). Assim, é a etapa da implantação de uma LPS em que o domínio é transformado em recursos para serem aproveitados na engenharia de aplicação, ou seja, o processo que desenvolve o núcleo e dá corpo à LPS. Esses recursos podem ser de diversos tipos, promovendo reúso de processo, arquitetura e código-fonte (Pohl *et al.*, 2005).

A implantação de uma LPS envolve aspectos de negócios, organização, processo e arquitetura (van der Linden *et al.*, 2007). Pode-se afirmar que os dois últimos são mais próximos ao projeto de software propriamente dito e à implementação envolvida. Assim, antes de descrever em detalhe os

métodos usados no desenvolvimento da engenharia de domínio, são descritos em linhas gerais os métodos para os aspectos de negócios e organização.

Como este trabalho está inserido em um grupo de pesquisa, os aspectos ligados aos negócios foram decididos durante a evolução da sua pesquisa. Atualmente o foco dos produtos e da contribuição almejada pelo grupo contém: facilitar o trabalho dos professores quando utilizam computadores em seu trabalho, aumentar o número de profissionais que o fazem, facilitar abordagens pedagógicas que promovam o aprendizado significativo dos alunos, aumentar a motivação dos alunos durante os estudos, entre outros objetivos (Dalmon *et al.*, 2011c). Esse é o caminho trilhado pelos atuais iMA, o que permite à LPS simplesmente manter os aspectos de negócios existentes no grupo de pesquisa e nos iMA, mas organizando-os de maneira sistemática.

Com relação à organização das pessoas que trabalham no grupo de pesquisa, os iMA foram desenvolvidos, em geral, individualmente pelos autores dos trabalhos citados (do Prado e Brandão, 2006; Eisenmann e Brandão, 2009; Isotani e Brandão, 2008; Kamiya e Brandão, 2009), com eventuais contribuições do coordenador da pesquisa, com exceção do *iGeom*, que foi principalmente desenvolvido pelo próprio coordenador. Essa organização é influenciada pela dinâmica do programa de pós-graduação do instituto, assim, cada aluno de mestrado contribuiu para um dos iMA e concluiu seus estudos fornecendo uma versão do aplicativo. Com o presente trabalho e a LPS proposta, será necessário ao menos um desenvolvedor para engenharia de domínio, sendo responsável por todas as etapas: projeto, implementação, teste e documentação. Assim, a presente organização de trabalho será pouco alterada, preservando o esquema em que cada desenvolvedor poderá concentrar seus esforços na engenharia de aplicação de um aplicativo específico, além do responsável pelo núcleo da LPS, que durante a execução deste trabalho é o próprio autor. Para a avaliação do uso da LPS, a organização é abordada na seção 3.3.

Os aspectos de arquitetura e processo são apresentados a seguir. Para a implementação do núcleo, foi criado um modelo de sistema e a partir dele um arcabouço de aplicação. Os processos de definição da utilização do núcleo (engenharia de aplicação) e da evolução da LPS são apresentados em seguida na última subseção.

3.2.1 Criação do Modelo de Sistema

Ao longo do desenvolvimento da engenharia de domínio da LPS para iMA, um modelo de sistema foi criado com o objetivo de especificar as funcionalidades definidas na análise de domínio como genéricas e compatíveis entre si. Por genérica dizemos que as funcionalidades da engenharia de domínio devem ser independentes do domínio específico de cada aplicação. Ou seja, devem ser desenvolvidas de forma a funcionar da mesma maneira nos domínios da geometria e da programação visual, por exemplo. Além disso, elas devem ser compatíveis entre si, cada uma usando os resultados das outras e todas em uma mesma arquitetura coerente.

Esse esforço foi necessário porque os iMA existentes foram desenvolvidos individualmente. A união das funcionalidades comuns aos iMA em um núcleo exigiu a adaptação nesses dois quesitos. O modelo de sistema criado serviu de base para a implementação do arcabouço de aplicação.

Para descrever o método usado para a criação do modelo de sistema é necessário relatar a evolução e definição do tema deste trabalho. O projeto de mestrado do autor, em seu início, tinha como tema a criação de funcionalidades de tutoria inteligente no *iGeom* (Dalmon *et al.*, 2010a,b, 2011b). O paradigma de tutoria inteligente escolhido é chamado de *Example-Tracing Tutors* (ETT

ou Tutores Rastreadores de Exemplos) (Aleven *et al.*, 2009). Com o desenvolvimento do projeto, percebemos que o paradigma escolhido poderia ser desenvolvido de maneira independente de domínio sem muito esforço adicional, se comparado ao da tarefa de adicioná-lo especificamente ao *iGeom*.

A partir desse momento, o tema deste trabalho evoluiu até se separar do desenvolvimento das funcionalidades de tutoria e se focou principalmente nos aspectos comuns, o desenvolvimento da LPS. Porém, apesar da separação, o paradigma ETT serviu de base para a definição do modelo de sistema, ou seja, a modelagem usada para garantir a generalidade e a compatibilidade das funcionalidades no núcleo da LPS foi derivada desse paradigma.

A adaptação do modelo do paradigma ETT para o contexto dos iMA foi feita usando modelagem orientada a objetos, principalmente com o emprego de padrões de projeto (Gamma *et al.*, 1994). Após uma primeira versão do modelo o arcabouço começou a ser criado, e com o andamento do desenvolvimento tanto o modelo quanto o arcabouço evoluíram, gerando no final a versão atual. O resultado desse processo é apresentado na seção 4.2.

Por fim, as funcionalidades de tutoria inteligente são consideradas na versão final da LPS como características opcionais, como mostrado nas seções 4.1.3. O paradigma ETT seria adicionado aos iMA por dois componentes, o de autoria e o de atuação. O componente de autoria deu origem à característica de *funcionalidades sobre atividades* e o componente de atuação às *funcionalidades sobre operações de domínio*. Os módulos que as implementam são descritos na seção 4.5.

O modelo de sistema teve grande influência nas tarefas realizadas pelos métodos utilizados para o desenvolvimento da LPS: FODA e PLUS, descrito a seguir. A partir dessa modelagem, o domínio dos iMA pode ser descrito e projetado por esses métodos sem grandes modificações. Assim, os métodos escolhidos foram suficientes para a modelagem e projeto do domínio principalmente pelas escolhas realizadas no modelo de sistema.

3.2.2 O Método PLUS

O projeto de software do arcabouço que forma o núcleo da LPS para iMA foi executado seguindo o método **PLUS** (***Product Line UML-based Software Engineering Method***, que pode ser traduzido como Método de Engenharia de Software para Linha de Produto baseado em UML) (Gomaa, 2004). Esse método foi escolhido principalmente pelo fato de ser baseado em UML (Miles e Hamilton, 2006), o que facilita o processo, e por ser utilizado em um livro dedicado ao método e que apresenta exemplos de projetos de linhas de produto. Outras referências com métodos de projeto de linha de produto também foram consultados, mas o nível de detalhe de cada etapa do processo ou o número de exemplos não eram suficientes para serem usados neste trabalho (Clements *et al.*, 2006; Pohl *et al.*, 2005).

O método PLUS consiste, basicamente, em aplicar a criação de diagramas UML, adaptados ao contexto de uma família de aplicativos, para projetar o núcleo de uma LPS. Ele é dividido em cinco etapas: (i) modelagem de características, (ii) análise de casos de uso, (iii) análise estática, (iv) análise dinâmica e (v) projeto arquitetural. A etapa de modelagem de características é similar à análise de domínio executada pelo método FODA, apresentado na seção anterior. Nela as características de todos os aplicativos da família são listadas e posteriormente classificadas em obrigatórias, opcionais ou variantes. As outras etapas do método PLUS são apresentadas a seguir.

Análise de Casos de Uso

Uma vez listadas e classificadas as características dos aplicativos da LPS, a etapa de análise de casos de uso é executada para cada uma. Assim, os casos de uso são relacionados às características, o que permite também separar as etapas seguintes dessa maneira. Uma influência dessa separação é a possibilidade de considerar as características opcionais e variantes em diferentes períodos do desenvolvimento da LPS, quando as obrigatórias já tiverem sido projetadas.

O método PLUS adapta a análise de casos de uso da linguagem UML para o contexto de LPS, incluindo a variabilidade nessa análise. Os casos de uso no método PLUS são classificados em obrigatório, opcional ou variante. Essa variabilidade é uma especialização em maior nível de detalhe daquela realizada na modelagem das características. Casos de uso **obrigatórios** pertencem a todos os aplicativos inseridos na LPS, enquanto que os **opcionais** são escolhidos para pertencer ao aplicativo durante a engenharia de aplicação. Casos de uso **variantes** ou **alternativos** podem ser escolhidos entre os elementos de uma lista. A Figura 3.2 mostra um exemplo de diagrama de casos de uso do método PLUS.

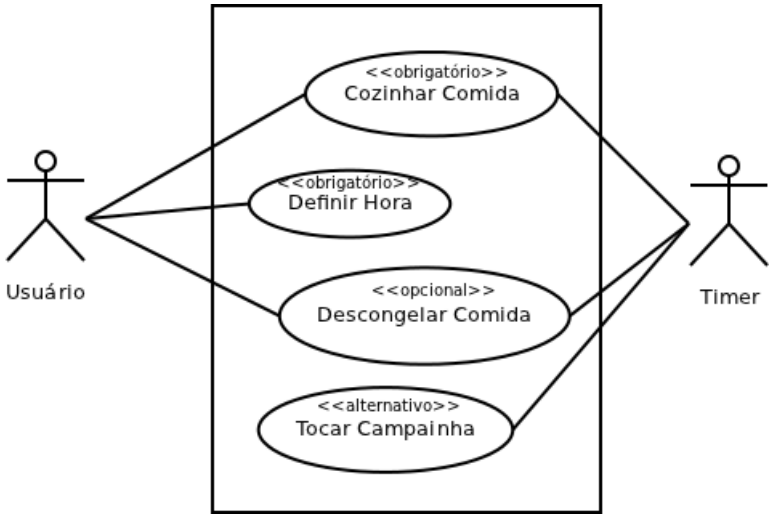

Figura 3.2: *Exemplo de diagrama de casos de uso do método PLUS.*

Nesse exemplo para o caso do projeto de um sistema de controle para uma linha de produto de microondas (Gomaa, 2004), o diagrama de casos de uso mostra casos de uso obrigatórios, opcionais e alternativos. Os casos de uso *Cozinhar Comida* e *Definir Hora* são obrigatórios, pois devem estar presentes em todos os microondas da linha. O caso de uso *Descongelar Comida* é opcional, apenas alguns microondas, mais completos, têm essa funcionalidade. Por fim, o caso de uso *Campainha* é alternativo, pois em cada microondas é possível ter uma campainha diferente, ou seja, uma campainha deve ser escolhida entre a lista de disponíveis.

Os casos de uso de uma característica permitem relacioná-la às classes, ou componentes arquiteturais. Dessa forma é possível determinar as classes responsáveis pelas funcionalidades de cada característica. As classe e a arquitetura são definidas nas etapas posteriores de análise estática e projeto arquitetural.

Análise Estática

A etapa de análise estática trata da modelagem do domínio e das funcionalidades selecionadas para a LPS em forma de classes, da mesma forma que o projeto de software com UML para aplicativos individuais. A adaptação proposta pelo método PLUS para o contexto de LPS é similar àquela para os casos de uso. Assim, as classes são classificadas como obrigatórias, opcionais e variantes. A figura 3.3 mostra um exemplo de diagrama de classes em um projeto de LPS.

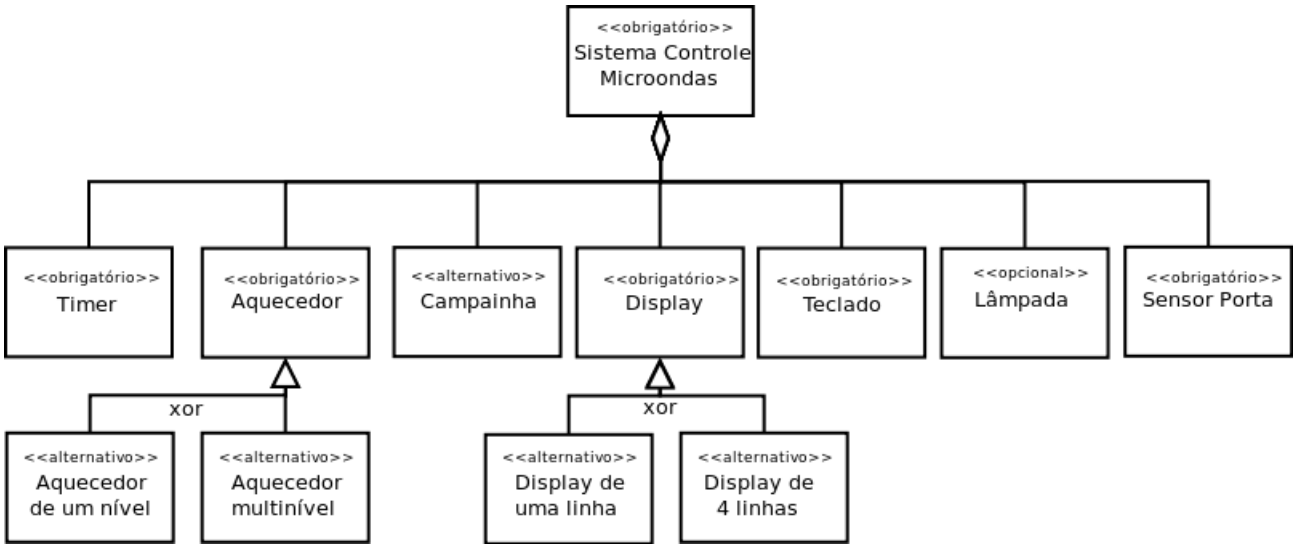

Figura 3.3: *Exemplo de diagrama de classes do método PLUS.*

Usando o mesmo exemplo da seção anterior, o da linha de produtos de sistemas de controle de microondas, o diagrama de classes mostra elementos obrigatórios, opcionais e alternativos. A classe central é a de controle, uma agregação de classes que representam interfaces dos vários dispositivos sensores e atuadores dos microondas. As classes *Timer*, *Teclado* e *Sensor de Porta* são obrigatórias, e todos os microondas da linha têm iguais. Por outro lado, as classes do *Aquecedor* e do *Display* são obrigatórias, mas podem variar entre dois tipos mutualmente exclusivos (representado pela expressão *xor*). A classe *Lâmpada* é opcional, o que indica que alguns microondas têm, outros não.

Análise Dinâmica

Uma vez definidas as classes que modelam as funcionalidades dos aplicativos e classificadas de acordo com a variabilidade dos casos de uso que fazem parte, a análise dinâmica pode ser executada. Similarmente, são usados os diagramas UML de sequência ou de colaboração. Incluindo a variabilidade definida na análise estática, é necessário criar diagramas para cada uma das variações, como mostra a figura 3.4.

Ainda usando o exemplo da linha de produto de controladores de microondas, esse diagrama de colaboração mostra apenas a parte variante do passo “4” de um caso de uso com relação a uma das classes alternativas. Nesse caso a classe alternativa é a *Display*. Assim, para o exemplo da figura, a alternativa de *display* de uma linha foi escolhida, e o diagrama criado. Essa análise é necessária para cada alternativa pelo fato das diferentes alternativas poderem se comunicar de diferentes maneiras com os outros elementos do sistema.

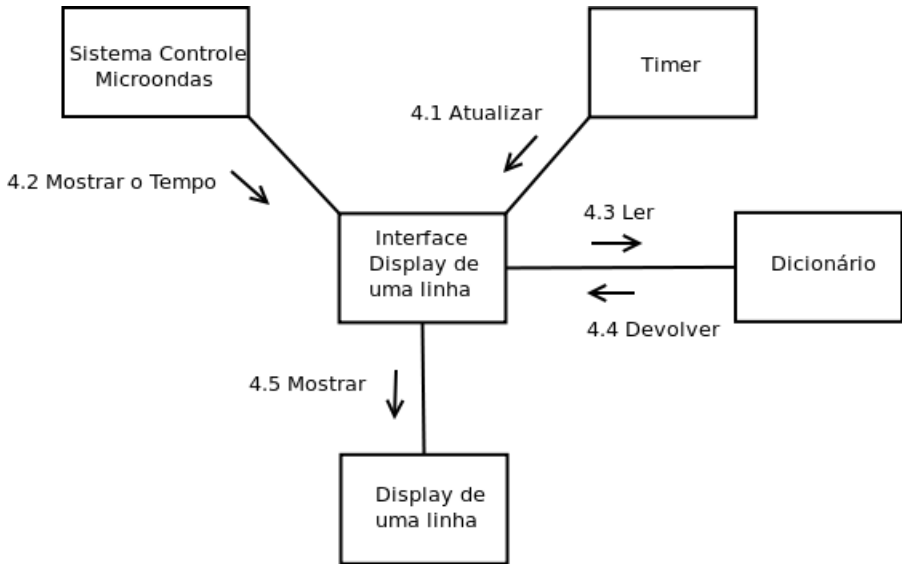

Figura 3.4: *Exemplo de diagrama de colaboração do método PLUS.*

Projeto Arquitetural

O projeto arquitetural refina a análise estática e a análise dinâmica com informações mais precisas sobre a comunicação dos objetos e encapsula classes em componentes de acordo com as características da LPS. Essa etapa também inclui a variabilidade como classificação dos componentes e outros elementos envolvidos.

Ao fim dessa etapa, todas as especificações necessárias para a implementação do núcleo da LPS estão definidas. Isso envolve um processo iterativo de revisão das outras etapas, uma vez que novas informações são coletadas ao longo do projeto. Na seção seguinte é relatado o método utilizado para o a criação do modelo de sistema, incorporado pelo projeto e implementação do arcabouço de aplicação da LPS para iMA, utilizando o método PLUS.

3.2.3 Projeto e Implementação do Arcabouço

O projeto e implementação do arcabouço de aplicação que forma o núcleo da LPS para iMA seguiu basicamente três conceitos: (i) o método PLUS, descrito na seção 3.2.2; (ii) o modelo de sistema apresentado na seção anterior; e (iii) o método de desenvolvimento em espiral (Boehm, 1986). A metodologia em espiral definiu as várias iterações de projeto e implementação do arcabouço, sendo que uma iteração posterior era mais detalhada e avançava para a conclusão da tarefa. O modelo de sistema forneceu uma arquitetura prévia e o projeto orientado a objetos das funcionalidades principais. Por sua vez, o método PLUS foi usado para o projeto de software, em suas várias etapas e nas primeiras iterações de implementação.

No início da execução dessa tarefa, não havia uma distinção nítida entre a aplicação do método PLUS para o projeto do arcabouço e o modelo de sistema. Com o tempo, percebemos que o modelo de sistema atuava em pontos específicos do arcabouço que mudaram pouco com o passar do tempo, já que os requisitos de generalidade e compatibilidade entre as funcionalidades eram atendidos desde a primeira versão. Enquanto isso, o método PLUS atuava em todo o arcabouço, em partes do projeto e do código que mudaram muito ao longo da implementação. Ao final, a distinção entre os

dois era clara, de forma que o arcabouço era uma forma de implementar o modelo, o que é melhor discutido na seção 4.2.

Também ao longo do desenvolvimento do arcabouço, o método PLUS que aplicamos foi ligeiramente modificado. De forma que as características do arcabouço eram classificadas apenas em obrigatória ou opcional, e as classes que fazem parte do código em obrigatórias ou variantes. Isso foi possível para as características porque o arcabouço foi concebido de forma que sua inteface usuário pode ser alterada completamente durante a instanciação, fazendo com que todas as características sejam variantes. No caso das classes, elas nunca são opcionais individualmente porque decidimos que apenas as características como um todo podem ser opcionais, com o objetivo de simplificar o projeto de software.

Essas considerações foram importantes nas primeiras iterações de desenvolvimento, sendo que nas últimas as tarefas eram essencialmente relacionadas a detalhes técnicos de implementação e teste, enquanto que as primeiras incluíam grandes mudanças no projeto. Ao total foram seis iterações, cada uma incluindo um elemento ao arcabouço, passando de um pequeno protótipo da interface do usuário na primeira, para um projeto de software detalhado demais na terceira iteração e simplificações da arquitetura em sequência nas iterações seguintes.

As ferramentas utilizadas foram o ambiente Eclipse (The Eclipse Foundation, 2012), os sistemas de gerenciamento de versões de código Git (Software Freedom Conservancy, 2012) e o servidor Redmine (Centro de Competência em Software Livre, 2012). Essas ferramentas devem passar a ser usadas pelos outros programadores de iMA, de maneira a uniformizar os métodos de implementação do grupo de pesquisa. A linguagem de programação utilizada é *Java*, a mesma em que os iMA foram desenvolvidos.

Para testar o arcabouço duas técnicas foram usadas, testes de unidade para as operações de classes chave, e uma instanciação de teste para o método de utilização do arcabouço como um todo. Os testes de unidade formam classes de código associadas a algumas classes do arcabouço. A instanciação de teste é um exemplo de como ele pode ser instanciado, contendo um conjunto de código mínimo para testar todas as suas funcionalidades. Isso foi necessário pelo fato do arcabouço ser um aplicativo semi-acabado, então para testá-lo como usuário final era usada essa instanciação. Um fato interessante é que o código dessa instanciação de teste foi pouco modificado ao longo de todas as iterações de desenvolvimento, provavelmente pela sua relação maior com o modelo de sistema do que com o restante do código do arcabouço.

Por fim, um fator importante para o andamento do projeto e implementação do arcabouço foi a comunicação com outros desenvolvedores. Devido à pouca experiência em desenvolvimento de arcabouço do autor, e pelo fato de arcabouços serem em geral mais complicados de implementar do que aplicativos individuais, a ajuda de programadores mais experientes foi crucial em momentos de crise. Essas contribuições foram especialmente importantes para o projeto de como o núcleo da LPS deve ser utilizado e evoluído, cujo método é apresentado a seguir. Os resultados da execução do método descrito aqui são apresentados na seção 4.3.

3.2.4 Uso e Evolução do Arcabouço

O objetivo considerado para a definição da forma de utilização do arcabouço foi reduzir ao máximo a curva de aprendizado, para facilitar o trabalho de um programador experiente e permitir que um inexperiente o utilize após algum tempo de estudo. As considerações feitas para atingir esse

objetivo, durante o projeto de software, incluíram a concentração dos pontos flexíveis do arcabouço (*hotspots*) em apenas dois componentes, um para as funcionalidades de domínio e outro para a interface com o usuário específica de domínio.

Outra característica do método utilizado foi a aplicação de apenas conceitos simples de orientação a objeto, principalmente herança e associação, para que programadores inexperientes não precisem estudar conceitos mais sofisticados para utilizar o arcabouço. Além disso, as funcionalidades providas pelo arcabouço foram as mínimas possíveis, para que o programador não precise também aprender a usar muitas funcionalidades.

Por fim, uma documentação abrangente foi criada. Essa documentação compreende comentários *Javadoc* (Oracle, 2012d), um manual contendo a descrição do arcabouço e um manual com o passo a passo de como utilizá-lo para criar um iMA. O passo a passo contém também uma descrição completa da instanciação de teste usada durante o desenvolvimento do arcabouço que serve de exemplo para o programador.

O método usado para a definição de como o arcabouço deve ser extendido ou evoluído de maneira a aprimorar seu funcionamento ou aumentar o número de funcionalidades providas foi composto de duas iniciativas. Uma delas é manter a qualidade do código alta (Ben-Menachem e Marliss, 1997) e documentar o melhor possível o código do arcabouço de maneira que futuros desenvolvedores consigam realizar a evolução facilmente. A outra iniciativa é a utilização de uma arquitetura de *plug-ins*, na qual a adição de funcionalidades não se relaciona com o funcionamento do arcabouço.

Com relação à qualidade de código, a implementação teve como objetivo seguir as recomendações de (Ben-Menachem e Marliss, 1997; Fowler *et al.*, 1999; Gamma *et al.*, 1994). Para a documentação, uma estratégia similar à descrita acima foi usada. Seguindo os manuais para a utilização do arcabouço, foi elaborado um terceiro manual, dessa vez com o objetivo de apresentar o passo a passo para a criação de uma nova funcionalidade em forma de *plug-in*. Caso o programador queira adicionar uma funcionalidade no arcabouço sem usar *plug-ins*, ele deve usar o manual que descreve seu funcionamento interno e modificar seu código. Todos os recursos desenvolvidos estão disponíveis em (Dalmon e Brandão, 2012).

Plug-ins são componentes que funcionam como sub-aplicativos de maneira que adiciona alguma funcionalidade ou comportamento ao aplicativo principal (Clayberg e Rubel, 2008). O aplicativo principal deve fornecer uma interface para o *plug-in* acessar seus dados e oferecer esses serviços. A principal vantagem desse método é que o aplicativo principal não é alterado no processo de adição de uma nova funcionalidade, apesar de restringir, pela interface fornecida, as funcionalidades que podem ser adicionadas dessa maneira. No caso do arcabouço, dois tipos de *plug-in* são especificados, e descritos com mais detalhes na seção 4.5.

3.3 Avaliação

Esta seção se destaca das outras deste capítulo por tratar dos métodos usados para avaliar as contribuições do trabalho realizado, em vez de desenvolver a pesquisa. As contribuições almejadas, conforme descritas na introdução, são sobre aspectos do desenvolvimento de iMA. Dessa forma as avaliações devem ser feitas durante o desenvolvimento realizado após a entrega do núcleo da LPS aos programadores de iMA.

Os métodos de avaliação apresentados aqui possuem mais características qualitativas do que quantitativas. Apesar de menos usados nas pesquisas em áreas de ciências exatas e tecnologia, métodos qualitativos são tão válidos para avaliar contribuições de trabalhos quanto métodos quantitativos, mas em situações diferentes (Gerring, 2006). Métodos quantitativos são mais adaptados a casos com muitas amostras, enquanto que os qualitativos são para poucas. Analogamente, os quantitativos analisam comportamentos gerais em populações, enquanto que os qualitativos analisam mais profundamente alguns casos, representativos ou particulares.

No caso do presente trabalho, uma possível avaliação usando um método quantitativo seria o experimento (Basili *et al.*, 1986). Nele, seriam convidados programadores, cujas experiências deveriam ser similares entre si, para a realização de uma atividade de desenvolvimento usando a LPS desenvolvida. As vantagens desse método seriam a quantidade de dados a partir do número de participantes e do número de sessões. Porém, acreditamos que as desvantagens se destacam. O processo de desenvolvimento de um iMA não poderia ser bem simulado em um número pequeno de sessões do experimento (um número grande de sessões não é factível), uma vez que é longo e envolve muitos aspectos, inclusive educacionais. Outra dificuldade seria encontrar programadores dispostos a participarem dos experimentos com uma experiência similar entre si, e similar com a usualmente encontrada nos desenvolvedores de iMA.

O objetivo geral apresentado na introdução foi *aprimorar o desenvolvimento de aplicativos educacionais em contexto acadêmico, no caso dos iMA pela utilização de uma LPS*. Assim, a avaliação deve analisar o efeito da utilização da LPS criada no desenvolvimento de iMA. A análise deve gerar evidências positivas da influência do uso da LPS nesse processo de forma a considerar o trabalho bem sucedido. Para realizar essa análise foram escolhidos a prova de conceito e o estudo de caso, apresentados nas próximas seções.

3.3.1 Prova de Conceito - *iTangran*

A prova de conceito é, antes de ser um método de avaliação das contribuições de uma pesquisa, uma prática de mercado. Com ela, é possível provar que um conceito é possível. Por exemplo, no caso deste trabalho, pode haver duas provas de conceito, uma é o núcleo da LPS desenvolvido, e outra é um iMA criado com esse núcleo. Elas provam, respectivamente, que foi possível criar na prática a engenharia de domínio da LPS proposta e que esses recursos permitem a criação de um iMA. Nas categorias de pesquisa descritas por Wazlawick (2009), a prova de conceito seria o resultado de uma pesquisa de *apresentação de produto*.

Assim, a avaliação promovida pela prova de conceito é uma resposta "sim ou não" para as perguntas: "O seu trabalho deu certo? Ele funciona e gerou algo concreto para provar?". Normalmente, em trabalhos acadêmicos em que a prova de conceito é um dos principais resultados, propriedades do produto final, como suas funcionalidades, são apresentadas e discutidas.

A prova de conceito que este trabalho apresenta é o produto do desenvolvimento de um iMA usando a LPS desenvolvida, chamado *iTangram*. Esse aplicativo é uma prova de que a pesquisa gerou recursos concretos que permitem, de alguma forma, desenvolver iMA. Detalhes do sistema e algumas considerações sobre o seu processo de desenvolvimento são apresentados na seção 5.1.

A justificativa para o uso desse método como avaliação do trabalho desenvolvido é o tempo necessário para a aplicação do método *estudo de caso* em vários casos. Em um programa de mestrado, em geral, não há tempo suficiente para desenvolver uma ferramenta e em seguida planejar e

executar estudos profundos. Mesmo assim, algumas considerações sobre o processo de uso da LPS criada serão feitas de maneira a se aproximar de um estudo de caso, mas sem a base teórica ou um procedimento mais sistemático. Na próxima seção uma breve descrição desse método é feita.

3.3.2 Estudos de Caso

Um estudo de caso é um método de pesquisa de intensa análise de uma unidade individual, o caso. Apesar de poder envolver métodos quantitativos de coleta e análise de dados, em geral é associado a pesquisa qualitativa. Para o estudo desse assunto, dois livros foram consultados (Gerring, 2006; Yin, 2008), além de artigos que aplicam esse método na área da Engenharia de Software (Cavaye e Cragg, 1995; Shanks, 1997). A seguir descrevemos brevemente o método, para apresentar como foi usado na seção seguinte.

Esse método pode ser visto como um conjunto de diretivas sobre como aplicar diferentes formas de análise em um caso, com o objetivo de descobrir informações, propriedades, mecanismos causais e levantar outras questões de pesquisa. Assim, as etapas de execução e análise dos resultados devem considerar vários métodos, sejam quantitativos ou qualitativos, para garantir a qualidade do estudo. É possível dividir a realização de um estudo de caso em quatro etapas principais: (i) planejamento, (ii) execução, (iii) análise dos resultados e (iv) elaboração do relatório.

A etapa de planejamento considera, basicamente, a definição das questões de pesquisa, a seleção dos casos, a escolha dos métodos de coleta de dados e de análise dos resultados e a forma de relatar o estudo. As questões de pesquisa definem o escopo do trabalho e os possíveis casos a serem estudados, além de ajudar a definir os métodos de coleta e análise.

Para selecionar um caso, é necessário considerar diversos fatores, como disponibilidade, facilidade de acesso e coleta de dados e seu tipo. O tipo de um caso define a sua relação com a população de casos existentes. Por exemplo, um caso típico é aquele que pode ser facilmente encontrado em uma população, com um número grande de ocorrências, enquanto que um caso extremo não necessariamente é típico ou raro, mas possui uma característica cuja gradação é a maior ou menor dentro da população considerada. Para as alturas de pessoas brasileiras, um caso típico é o homem com 1,75 m de altura, enquanto que um caso extremo é um homem com mais de 2,10 m.

Um estudo de caso geralmente é aplicado quando não é possível analisar um número grande de casos ou quando o interesse da pesquisa é exatamente a investigação profunda de um caso, seja para responder ou levantar questões de pesquisa. Por isso, a escolha dos tipos de caso é muito relevante para definir a maneira com que o estudo é conduzido. Se o estudo considera apenas casos típicos, a forma de generalização dos resultados é significativamente diferente se forem usados apenas casos extremos. No último caso, porém, as contribuições serão em torno das causas ou efeitos da característica extrema do que foi analisado.

Além disso, os casos devem ser selecionados com relação à intervenção realizada na pesquisa. É possível escolher casos com ou sem intervenção (variação temporal) e ter casos de controle ou não (variação espacial). A situação ideal é aquela que se aproxima de um experimento, na qual há variação temporal e espacial, ou seja, há pelo menos um caso que recebe a intervenção e outro caso de controle que permanece sem estímulos.

A execução do estudo deve seguir vários métodos, coletando dados de diferentes naturezas. Isso é necessário para que a análise possa fazer a triangulação dos dados. A triangulação é uma análise que compara os efeitos e as causas de pontos de vista diferentes, cada um proveniente de um tipo de

informação coletada. Isso permite que as conclusões tomadas pelo estudo não deixem de considerar dados que poderiam ser relevantes para os fenômenos estudados.

Por exemplo, para os estudos de caso realizados na Engenharia de Software, os tipos de dados considerados podem ser: (i) contexto de desenvolvimento, incluindo características do projeto, da instituição, do código, do cliente (Cavaye e Cragg, 1995); (ii) as pessoas envolvidas, suas experiências, motivações, relações e influências; (iii) os acontecimentos ao longo do tempo, imprevistos, entre outros (Shanks, 1997). Quanto maior a variedade de dados considerada, maior a confiabilidade nas conclusões feitas durante a análise dos resultados.

A utilização do estudo de caso nesta pesquisa é justificada pelo fato de estarmos interessados em verificar a influência da LPS desenvolvida no desenvolvimento de iMA, de maneira a cumprir o objetivo geral da pesquisa. Outros fatores são: (i) o número disponível de casos não é grande, apenas aqueles envolvidos com iMA; (ii) a facilidade de acesso a dados é alta, pois o desenvolvimento é realizado no mesmo grupo; e (iii) a possibilidade de investigar detalhes dos processos em busca de melhorias provocadas pela LPS.

Apesar disso, algumas desvantagens para o método existem, como a falta de tempo para realizar iterações de coleta e análise de dados, o que também influencia a maturidade do estudo e das conclusões feitas. Outra consideração pode ser feita com relação à quantidade de dados coletada, durante o tempo analisado. Porém, o estudo de caso realizado não tem como objetivo esgotar o tema, mas iniciar uma cultura de avaliação de processos de desenvolvimento no grupo de pesquisa e na área de informática na educação. A próxima seção trata com mais detalhes a forma com que o estudo de caso foi aplicado.

3.3.3 Casos de desenvolvimento de iMA - *iVProg*

Durante a execução deste trabalho, outros quatro projetos de desenvolvimento de iMA eram realizados, para *iGeom*, *iGraf*, *iComb* e *iVProg*. Assim, todos foram considerados como casos para o estudo. Apesar do planejamento ter considerado os quatro casos, durante a coleta de dados apenas o projeto de desenvolvimento de uma nova versão do *iVProg* atingiu um estágio de forma a gerar dados sobre a utilização da LPS criada. A seguir apresentamos o planejamento para os estudos de caso e por fim a adaptação para o caso único do *iVProg*.

O tipo de estudo seria o de variação temporal apenas. Assim, o processo de desenvolvimento dos iMA seria analisado antes de passar a utilizar a LPS e depois, com a utilização do arcabouço e dos manuais preparados. Todos os projetos sofreriam essa intervenção. Ter variação espacial com um caso *de controle* (que em nenhum momento sofreria a intervenção de passar a utilizar a LPS) não conduziria ao objetivo do grupo de pesquisa, pois o projeto procura contribuir ao máximo para o desenvolvimento dos iMA, e escolher um para não receber a contribuição seria deixá-lo de lado, apesar de ser relevante pelo aspecto metodológico.

Além da forma construtiva dos estudos de caso, podemos definí-los quanto a sete aspectos (Gerring, 2006): (i) hipótese; (ii) validade da generalização; (iii) efeito causal; (iv) escopo da proposição; (v) população de casos; (vi) força causal; e (vii) variação útil. Essas considerações servem para justificar a adoção do método e definir sua validade interna, de forma a aprimorar a análise e a generalização dos resultados

- A *hipótese* dos estudos de caso é: o uso da LPS para iMA afeta o desenvolvimento desses sistemas positivamente. A LPS, sendo uma técnica de Engenharia de Software, tem o objetivo

de aprimorar o desenvolvimento e os aplicativos produzidos em um processo. Dessa forma, a hipótese é de que isso ocorre também no caso específico estudado. Essa hipótese se justifica pelo fato de que o contexto em que o desenvolvimento dos iMA ocorre é atípico com relação aos contextos em que as técnicas de Engenharia de Software são geralmente aplicadas, conforme será verificado durante o estudo. Outra justificativa é que a LPS pode ter sido desenvolvida de maneira não ótima, de forma que é necessário verificar quais seus pontos fortes e quais seus pontos fracos com relação às outras aplicações de LPS para aumentar sua contribuição positiva ao processo de desenvolvimento.

- A *validade de generalização* diz respeito à possibilidade de generalizar as conclusões dos estudos de caso para outras situações. Para um estudo de caso com boa validade de generalização podemos dizer que as conclusões obtidas poderão ser generalizáveis para contextos de desenvolvimento similares ao encontrado aqui, possivelmente para aplicativos de tamanho similares, não necessariamente educacionais. O contexto de desenvolvimento do caso é bem particular, o que influencia fortemente o processo de desenvolvimento e o uso da LPS e deve limitar a generalização das conclusões do estudo de caso apenas para contextos de desenvolvimento muito similares.

- O *efeito causal* trata do efeito esperado pela intervenção do estudo de caso, que é a adoção da LPS como método de desenvolvimento. O efeito da aplicação de uma LPS ao processo de desenvolvimento de uma família de aplicativos é bem conhecido (van der Linden *et al.*, 2007). Porém, para o contexto particular do desenvolvimento dos iMA é possível que seja um pouco diferente. Além disso, os mecanismos causais dos efeitos da aplicação de uma LPS podem também variar por causa desse contexto.

- O *escopo da proposta* é específico ao desenvolvimento dos iMA, pois muitas decisões de projeto são específicas para as características desses aplicativos e para o próprio contexto de desenvolvimento. Dessa forma, não é possível aumentar o escopo para aplicativos educacionais em geral ou outros grupos. Esse aspecto do estudo de caso é bastante ligado à validade de generalização. Por ser um escopo bem restrito, é possível aproveitar esse fato para aprofundar as investigações.

- A *população de casos disponíveis* para estudo é composta de quatro projetos de desenvolvimento de iMA: (i) *iGeom*; (ii) *iGraf*; (iii) *iComb*; e (iv) *iVProg*. Cada projeto possui aspectos particulares e comuns com os demais. Por exemplo, o principal programador do *iGeom* é o único com experiência profissional em desenvolvimento e o *iVProg* é o único aplicativo que está sendo refatorado completamente, os outros são apenas adaptados.

- A *força causal* descreve a intensidade da influência da intervenção do estudo de caso na variável de análise, aqui seria a intensidade da influência da adoção da LPS como método de desenvolvimento no andamento do processo de desenvolvimento. Como vemos na literatura, a adoção de métodos sistemáticos de desenvolvimento e técnicas de Engenharia de Software tem uma influência grande no desenvolvimento. Dessa forma podemos dizer que a força causal da intervenção planejada é grande, o que facilitará a coleta de dados no modelo de estudo de caso.

- A *variação útil* fala da variação entre os casos, mas como os casos possíveis aqui são muito poucos, ela é rara. Essa variação tem maior influência no estudo quando o número de casos é maior. A disponibilidade de dados é restrita aos casos que serão analisados, uma vez que não é possível coletar dados de outras fontes como artigos ou relatórios, já que esses materiais não focaram nas informações relevantes sobre a variável de análise: o processo de desenvolvimento.

O planejamento do estudo de caso compreende, além da sua definição, as etapas de coleta e análise de dados. Os dados relevantes sobre o processo de desenvolvimento dos iMA são compostos de: (i) o código e sua evolução ao longo do tempo; (ii) os programadores, sua forma de trabalho e sua relação com a tarefa de desenvolvimento; e (iii) o uso dos aplicativos. Esses tipos de dados foram inspirados dos usdos em Shanks (1997), porém, os processos considerados aqui são mais simples, envolvendo menos pessoas e sistemas menores, do que os do trabalho citado.

A *coleta de dados* foi feita de duas maneiras: (i) o código fonte do aplicativo; e (ii) entrevistas com pessoas responsáveis pelo desenvolvimento. O código do aplicativo tem informações sobre o produto final, e reflete a forma de realizar o processo. Seria preferível analisar informações armazenadas em uma ferramenta de controle de versões de código, mas isso não era usado anteriormente à intervenção. As entrevistas têm o objetivo de fornecer informações sobre o programador e o andamento do desenvolvimento em geral. Dessa forma, seriam realizadas uma entrevista antes da intervenção e entrevistas mensais com o programador utilizando a LPS, e também ao final uma entrevista com o coordenador do projeto. Os roteiros das entrevistas são disponibilizados junto com as suas transcrições na página do projeto (Dalmon e Brandão, 2012).

A *análise de dados* é feita pela análise do código e das transcrições das entrevistas. A análise de código é feita com relação a métricas de código (Li e Henry, 1993) e com uma leitura superficial e subjetiva do código. Essas métricas podem ser relacionadas a vários fatores objetivos, como desacoplamento e critérios de manutenção de software, enquanto que a leitura gera informações sobre o aspecto do código para um novo programador. As métricas usadas durante a análise são as propostas por Chidamber e Kemerer (1994), pelos motivos apresentados pelos autores. O aplicativo usado para calcular as métricas dos códigos foi o *plugin* para Eclipse chamado *Metrics* (Sourceforge, 2012).

As informações contidas nas entrevistas transcritas são resumidas em quadros, de maneira a facilitar sua análise, e em seguida apresentadas ao próprio entrevistado para que ele confirme as análises realizadas. São três tipos de informações buscadas nas entrevistas: (i) o método usado para o desenvolvimento; (ii) a percepção de desempenho de programação; e (iii) a opinião do programador com relação ao processo. Essas informações foram escolhidas por abrangerem uma descrição do processo realizado, resultados obtidos com esse novo processo, todos sob o olhar do programador e sua opinião explicitamente, de maneira um pouco similar a (Yin, 2008).

Durante a análise, o método usado no desenvolvimento, descrito pelo programador e confirmado pelo coordenador, serve de base para todas as análises. Os resultados obtidos, em termos de desempenho de programação e da qualidade do código obtido devem ser avaliados de forma a relacionar todos as informações coletadas. O desempenho do desenvolvimento é muito difícil de avaliar (Sentas *et al.*, 2005), de forma que neste trabalho só será considerada a percepção desse aspecto pelas pessoas envolvidas. Por fim, a opinião do programador tem o objetivo de levantar indícios da satisfação e facilidade de uso, qualidade da organização e apresentação dos recursos, entre outros aspectos.

Apesar desse planejamento ter sido realizado para os quatro casos considerados, com o passar do tempo, apenas o processo do *iVProg* gerou resultados significativos de uso da LPS (informações pós-intervenção). Dessa forma, a avaliação da LPS foi realizada com apenas um estudo de caso completo. Informações superficiais dos outros casos também foram consideradas, mas com menos ênfase. O fato de haver apenas um programador e de não existirem outros documentos além do código do aplicativo, impedem que sejam consideradas outras informações a serem coletadas, como pontos de vista de várias pessoas e dados sobre versões do aplicativo, o que apresenta dados sobre o andamento real do desenvolvimento.

A maior mudança do estudo de múltiplos casos para um caso único é a abrangência dos resultados, uma vez que não é possível fazer comparações e deduções das informações obtidas de diferentes processos de desenvolvimento. Porém ainda é analisada a influência da LPS suficientemente para avaliar o desenvolvimento apresentado nesta dissertação.

O estudo de caso planejado está limitado ao período de coleta de dados e ao caso de desenvolvimento escolhido. Dessa forma, os indícios do aprimoramento do processo de desenvolvimento provocado pela LPS devem considerar essas limitações. Apesar de ser possível analisar os resultados de forma a transferí-los para outras situações, para que conclusões mais abrangentes e definitivas possam ser feitas sobre um fenômeno, vários estudos de caso (avaliando diferentes amostras da população) durante toda sua duração devem ser feitos.

Os resultados obtidos com o estudo de caso do *iVprog* são apresentados na seção 5.2. As informações dos outros casos são apresentadas nas transcrições das entrevistas disponibilizadas em (Dalmon e Brandão, 2012), algumas considerações são feitas na seção 6.1. O próximo capítulo apresenta o resultado do desenvolvimento da LPS, em sua versão final, sucedido do capítulo das avaliações.

Capítulo 4

Uma LPS para iMA

Este capítulo apresenta a Linha de Produto de Software para Módulos de Aprendizagem Interativa desenvolvida. Após algumas considerações iniciais, apresentamos os resultados da análise de domínio (os iMA existentes), o modelo de sistema derivado dessa análise e em seguida o arcabouço desenvolvido que implementa esse modelo. Por fim, são descritos os métodos definidos para a utilização do arcabouço para a criação de novos iMA (engenharia de aplicação) e para a evolução do arcabouço, incluindo nele novas funcionalidades.

Analisando sua construção, a LPS foi criada a partir dos iMA existentes, dos quais foram separadas as funcionalidades particulares de cada aplicativo daquelas compartilhadas entre eles (seção 4.1). Em seguida, um modelo foi criado de maneira que essas funcionalidades compartilhadas fossem compatíveis com qualquer domínio (seção 4.2). Esse modelo serviu de base para a implementação do arcabouço (seção 4.3), que pode ser usado com o auxílio de processos pré-definidos para a criação de novos iMA (seção 4.4). O objetivo desse desenvolvimento é facilitar o trabalho de desenvolvedores de iMA e aprimorá-lo em comparação aos processos realizados no desenvolvimento dos iMA existentes.

Assim, a LPS se constitui nos produtos da análise e modelagem de domínio, como o diagrama de características e o modelo de sistema, no arcabouço de aplicação e nos manuais de utilização desse arcabouço. Um programador, ao utilizá-la para desenvolver um iMA, recebe o arcabouço e os manuais, e a partir desses recursos cria as funcionalidades do domínio escolhido com seu próprio código a partir do código-base e suas instruções de uso. Dessa forma, a técnica promove reúso de processo, com os manuais, de arquitetura e de código-fonte com o arcabouço.

Uma contribuição importante deste trabalho é a centralização do conhecimento que antes era distribuído entre os iMA existentes, sem uma documentação explícita para usuários ou desenvolvedores. Além disso, contribui para a consideração de fatores que eram implícitos nesses processos, como a organização do código e a utilização de um método sistemático de desenvolvimento. Esses fatores são importantes para a capitalização do conhecimento dos processos e produtos (Tchounikine, 2011).

4.1 Análise de Domínio

A análise de domínio foi realizada sobre os iMA existentes para a definição do escopo da LPS desenvolvida. Essa etapa resultou, basicamente, em três produtos: (i) uma lista de requisitos gerais para os iMA; (ii) a lista das funcionalidades para o usuário final, classificadas de acordo com a

presença nos aplicativos e possibilidade de reúso; e (iii) a modelagem das características da LPS, baseadas nesses requisitos e funcionalidades, representada por um diagrama de características.

4.1.1 Requisitos dos iMA

Os requisitos gerais dos iMA, sem considerar as funcionalidades para o usuário final (descritas na seção 4.1.2), refletem a abordagem da família de aplicativos. Dessa forma, a LPS deve fornecer todos esses requisitos de maneira obrigatória. De acordo com o descrito na seção 2.1.3, temos cinco principais: (i) funcionar em navegadores *Web*, uma forma mais geral do requisito de ser um *Applet*; (ii) possuir o protocolo de comunicação com SGC; (iii) ter ferramentas de autoria para professores; (iv) interatividade nas atividades; e (v) correção automática das soluções do aluno.

Em uma análise mais profunda, é possível destacar outro requisito dos iMA existentes, ser leve e exigir pouco processamento. O maior número possível de alunos deve poder ter acesso aos iMA. Assim, os aplicativos devem ser gratuitos, exigir pouco poder de processamento dos computadores para poderem ser usados em máquinas antigas, e ocuparem pouco espaço em disco, para facilitar sua obtenção em conexões lentas com a Internet. Além disso, como são *Applets Java*, devem ser compatíveis com versões antigas da máquina virtual para poderem ser usados em navegadores e sistemas operacionais antigos.

A formulação dos requisitos definidos para os iMA descritos nesta seção foi escolhida por ser suficiente para a modelagem das características nas etapas subsequentes do desenvolvimento da LPS. Essa formulação deve posteriorment eser realizada de maneira mais precisa, segundo a engenharia de requisitos (Sommerville, 2007). De acordo com a engenharia de requisitos, as definições descritas aqui podem ser consideradas como metas subjetivas para os aplicativos. Para melhor documentação e evolução da LPS, será incluido como trabalho futuro a melhor especificação desses requisitos.

Como a LPS tem como objetivo aprimorar o processo de desenvolvimento dos iMA, é possível também contribuir para os produtos finais com uma lista adicional de aspectos desejados com o objetivo de melhorar a contribuição dos iMA à comunidade. Esses aspectos também serão incluídos na LPS para os aplicativos de maneira obrigatória. A lista abaixo apresenta os principais.

- **Ter comportamento estável** - Para facilitar a utilização pelos alunos e revelar o menor número de defeitos possível, um requisito adicional para os iMA é ter um comportamento estável. Isso é importante também para que os alunos não se desmotivem por comportamentos indesejados dos aplicativos. Um comportamento estável pode ser melhor garantido com etapas de teste de software durante o desenvolvimento, aliado a uma boa documentação para agilizar a manutenção.

- **Ter manutenção fácil** - Em conjunto com o requisito anterior, um fator importante para aplicativos com ciclos de vida longo é a manutenção. Assim, a facilidade da manutenção e da adição de novas funcionalidades nos iMA será considerada como altamente desejável. Isso deve refletir nos processos de desenvolvimento e documentação dos aplicativos.

- **Ser código livre** - Para que o desenvolvimento dos aplicativos não dependa apenas de um grupo de pesquisa, consequentemente aumentando o número de programadores e agilidade no desenvolvimento, os aplicativos que não possuem fins lucrativos devem ter seu código livre. Porém, para que o projeto como código livre seja bem sucedido, a arquitetura e a documentação são essenciais.

Esses novos aspectos desejáveis têm caráter essencialmente computacional e são relacionados a qualidade de software e ao processo de desenvolvimento. Conforme descrito na seção 2.1, vários problemas com manutenção de código são encontrados, então começar a considerar isso como requisitos é uma atitude na direção de resolvê-los. O uso da LPS desenvolvida tem o objetivo de ajudar os iMA a atender a esses requisitos, o que traria diversos benefícios para desenvolvedores e usuários.

4.1.2 Funcionalidades dos iMA

As funcionalidades dos iMA para o usuário final são elementos essenciais para o desenvolvimento da LPS. O mapeamento das funcionalidades é necessário para a definição e classificação das características (seção 4.1.3), em que a modelagem de domínio se baseia. Além disso, esses são os elementos utilizados diretamente pelos usuários finais.

Dada a importância da determinação e classificação das funcionalidades para o usuário final, essa tarefa foi realizada sobre os iMA existentes. As funcionalidades foram classificadas em independentes e dependentes de domínio. As depentendes de domínio são específicas para o domínio de cada iMA e não podem ser aproveitadas pelos outros, que modelam domínios diferentes. As independentes são aquelas que não usam informações específicas do domínio para serem executadas, podendo ser aproveitadas em mais de um iMA. A Tabela 4.1 apresenta a ocorrência das funcionalidades independentes do domínio nos quatro iMA.

De acordo com a Tabela 4.1, o *iGeom* possui praticamente todas as funcionalidades independentes de domínio dos iMA, com exceção do "Histórico" de ações realizadas. Para a implementação dessas funcionalidades no núcleo da LPS, elas foram posteriormente agrupadas em características e classificadas de acordo com a variabilidade, o que é descrito na modelagem das características na próxima seção.

As funcionalidades dependentes do domínio também foram analisadas. Para facilitar a análise, elas foram agrupadas de acordo com o resultado obtido após sua execução no domínio em questão. Os grupos de funcionalidades dependentes do domínio são: (i) criação de objetos de domínio; e (ii) edição de objetos de domínio. Exemplos no caso do *iGeom* são a criação de pontos, circunferências ou reflexões em um eixo; como edição de objetos há mover ponto, atualizar e editar objetos. No caso do *iVProg*, há a criação de variáveis, expressões e estruturas de repetição, e mudar nome de variável e mudar uma linha de código de posição.

Todas as funcionalidades dependentes do domínio para o usuário final presentes nos iMA existentes podem ser classificadas nesses dois tipos: criação e edição de objetos de domínio. Essa análise serve de base para a modelagem genérica dessas funcionalidades desenvolvida no núcleo da LPS para iMA, apresentada na seção 4.2.

De maneira similar ao caso dos requisitos, a LPS também contribui para a família iMA com a adição de novas funcionalidades independentes de domínio, que podem ser fornecidas pelo arcabouço. Uma forma de prover isso é a generalização de funcionalidades encontradas em um ou dois iMA, o que ocorre em vários casos (vide tabela 4.1). Por exemplo, prover *scripts* para todos os iMA em que for relevante pedagogicamente é uma contribuição.

A nova funcionalidade para os iMA mais importante para o desenvolvimento da LPS é a de tutoria inteligente, conforme apresentado na seção 3.2.1. Essa adição motivou a criação da LPS de forma que as características mapeadas e o modelo de sistema criado incluem elementos que são generalizações das funcionalidades primeiramente relacionadas com tutoria inteligente. Outras

Tabela 4.1: *Funcionalidades independentes do domínio distribuídas nos iMA existentes*

Funcionalidade	*iGeom*	*iGraf*	*iComb*	*iVprog*
Applet e *desktop*	x	x	x	x
Autoria de exercícios	x	x	x	x
Salvar arquivo de exercício	x	x	x	x
Abrir arquivos de exercícios	x	x	x	x
Correção automática de exercícios	x	x	x	
Comunicação resultado a SGC	x	x	x	
Receber exercício do SGC	x	x	x	x
Desfazer/Refazer	x			x
Arquivo > Novo	x	x	x	x
Caixas de Texto/Anotações	x	x		
Enviar solução de exercício	x	x	x	x
Desfazer exercício	x			
Mostrar gabarito de exercício	x	x	x	
Salvar arquivo de solução	x		x	
Abrir arquivo de solução	x		x	
Executar *script*	x			
Iniciar geração de *script*	x			
Recorrência de *script*	x			
Executar *script* passo-a-passo	x			
Salvar *script* em botões	x			
Salvar arquivo de *script*	x			
Salvar arquivos html	x	x	x	x
Importar imagem	x			
Exportar imagem	x	x		
Exportar *Applet* exercício html	x	x	x	x
Exportar arquivo para curso *Web*	x			x
Nova aba	x	x		
Fechar aba	x	x		
Fechar todas outras abas	x			
Copiar	x	x		
Copiar tudo	x			
Colar	x			
Ajuda	x	x	x	x
Manual *Web*	x	x	x	x
Sobre	x	x	x	x
Lista de Objetos	x			
Parâmetro linha de comando tamanho	x			
Parâmetro linha de comando exemplo	x			
Abrir arquivo de língua	x	x	x	x
Histórico		x		

novas funcionalidades podem ser fornecidas pela LPS seguindo o modelo proposto, o que compõem outras características opcionais, separadas em funcionalidades sobre atividades e funcionalidades sobre operações de domínio, apresentadas na próxima seção.

4.1.3 Modelagem das características

Após a determinação dos requisitos dos iMA existentes e desejados, e do mapeamento das funcionalidades para o usuário final, essas informações são utilizadas para a definição das características presentes na LPS para iMA. As características agrupam funcionalidades com tema e variabilidade similares e servem de base para o modelo de sistema. A Figura 4.1 representa essas características e seus atributos na forma de um diagrama de características.

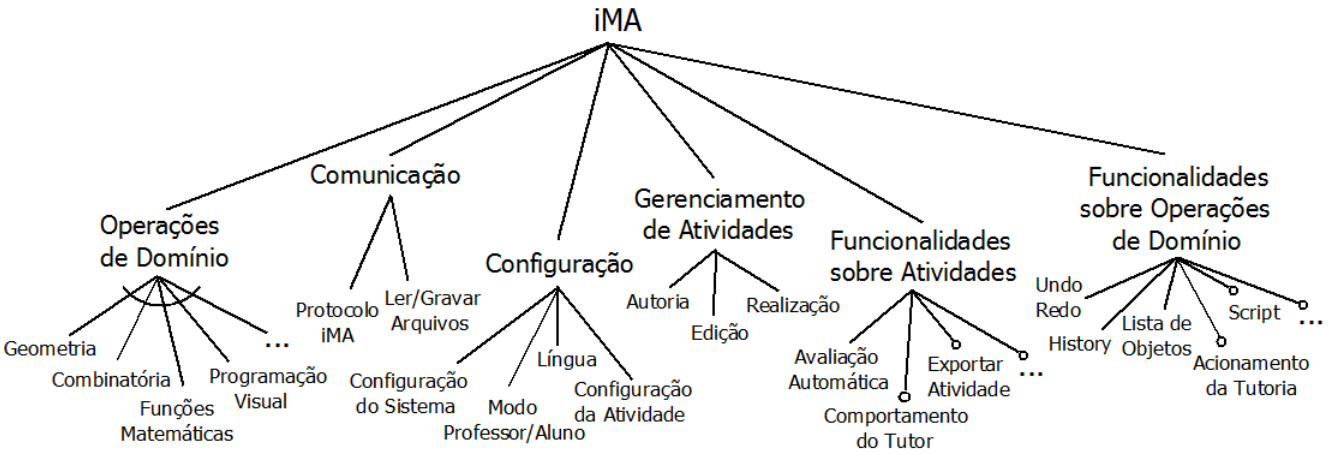

Figura 4.1: *Diagrama de características da LPS para iMA.*

As características representadas por um traço simples são comuns e obrigatórias. As marcadas com um arco são alternativas exclusivas, ou seja, cada iMA possui apenas uma delas, esse é o caso das operações de domínio. Aquelas com um círculo próximo ao seu nome são opcionais, ou seja, não estarão presentes em todos os iMA criados com a LPS, o que é o caso de algumas funcionalidades sobre atividade e sobre operações de domínio.

De um modo geral, esse diagrama representa as funcionalidades que o núcleo da linha de produto deve oferecer, e como elas podem ser escolhidas, no caso das opcionais e variantes, durante o desenvolvimento dos aplicativos. Por exemplo, para desenvolver um iMA, deve-se escolher um conjunto de operações de domínio e quais características opcionais farão parte do aplicativo. A seguir são detalhadas as características da figura.

- **Operações de Domínio** - Conjunto das funcionalidades específicas do domínio. Como são alternativas, os iMA possuem apenas um conjunto de operações de domínio. Cada um dos iMA existentes possui uma dessas sub-características: o *iGeom* possui geometria construtiva, *iGraf* gráficos de funções; *iComb* combinatória e *iVProg* programação visual.

- **Gerenciamento de Atividades** - Característica que trata das funcionalidades sobre gerenciamento de atividades, como autoria, gravação, representação, realização e edição desses recursos.

- **Comunicação** - Característica que reúne funcionalidades de comunicação do iMA com sistemas externos. Como indicado no diagrama, suas principais funcionalidades são abrir uma atividade a partir de um comando do SGC, enviar a solução do aluno (ou o resultado da avaliação automática, no caso em que essa funcionalidade está presente), e gravar ou abrir arquivos em geral (apenas na versão *desktop*).

- **Configuração** - Característica com as funcionalidades relacionadas aos parâmetros de configuração do aplicativo, sejam eles parâmetros de sistema, de atividade ou do idioma. Permite ao usuário final acessar a certas funcionalidades, verificar aspectos do aplicativo, ou modificar os valores desses parâmetros.

- **Funcionalidades sobre Atividades** - Representa as funcionalidades realizadas sobre uma atividade inteira, seja a que o professor crie ou a que o aluno realize. Exemplos são: (i) avaliação automática da solução do aluno; (ii) exportar a atividade a um outro formato, como imagens ou arquivos com especificações de objetos de aprendizagem; e (iii) a definição do comportamento do tutor inteligente.

- **Funcionalidades sobre Operações de Domínio** - Representa as funcionalidades realizadas sobre as ações que o professor ou aluno realizam durante uma atividade, em um nível de granularidade menor do que a característica anterior. Exemplos são: desfazer e refazer ações, histórico de ações, lista de objetos, *scripts*, execução das ações de tutoria inteligente, animações, etc.

As funcionalidades representadas como opcionais não são providas pela versão da LPS criada neste trabalho, principalmente pela falta de tempo para a implementação. Porém, os manuais fornecidos incluem considerações importantes para futuros programadores que podem inserir essas funcionalidades à LPS. No arcabouço elas são modeladas como *plug-ins*, como é descrito na seção 4.5.

Existem ainda considerações sobre essa etapa que devem ser feitas, como a interação entre as características, a granularidade e seu efeito no projeto. A interação entre as características é muito forte internamente nos iMA existentes, mas o arcabouço pretende reduzir esse fator, possuindo um código mais modularizado. Especificamente, há uma interação entre as características de atividades e comunicação, com a presença de avaliação automática. Com relação à granularidade, as características podem ser consideradas praticamente como casos de uso, que é a forma mais simples de um usuário entender uma característica.

O arcabouço deve ser projetado de maneira flexível, para que a dificuldade de adicionar novas funcionalidades não seja alta. A modelagem de características é a última atividade realizada na análise de domínio. As características nele definidas servem de base para o processo de projeto de software descrito nas próximas seções.

4.2 Modelo de Sistema

O modelo de sistema criado é uma forma de arquitetura de alto nível para iMA, definindo seus principais componentes, suas responsabilidades e a comunicação entre eles (Dalmon *et al.*, 2012c). O objetivo dessa abstração é principalmente garantir a compatibilidade entre as características definidas para os iMA e a independência do funcionamento do núcleo da LPS do domínio de cada aplicativo. Assim, seguindo esse modelo é possível criar um arcabouço que contenha todas as funcionalidades que os iMA requerem de maneira que possibilite a implementação de funcionalidades específicas de domínio durante sua instanciação.

Na seção anterior, seis características principais foram definidas. Para o modelo de sistema, simplificando a descrição, quatro componentes são usados: (i) estrutura, que compreende as carac-

terísticas de comunicação e configuração; (ii) domínio, para as operações de domínio; (iii) atividade, correspondendo ao gerenciamento de atividades; e (iv) extensões, que agrupa as características de funcionalidades sobre atividades e sobre operações de domínio. A figura 4.2 representa graficamente os quatro componentes e a comunicação entre eles.

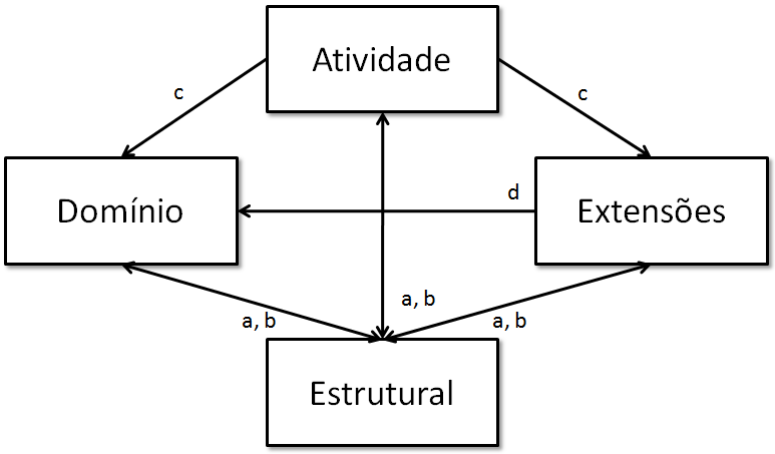

Figura 4.2: *Diagrama da estrutura do modelo de sistema para iMA.*

A comunicação entre os componentes, representada na figura pelas setas entre os blocos contém: (a) o componente estrutural inicializa os outros três; (b) estes últimos usam funcionalidades independentes do domínio fornecidas pelo componente estrutural; (c) o componente de atividade utiliza operações de domínio e as extensões para criar, executar e gerenciar as atividades; e (d) as extensões também usam operações de domínio para fornecer suas funcionalidades específicas. A seguir os componentes são descritos com mais detalhes, inclusive considerações particulares sobre a comunicação e a interdependência entre eles.

4.2.1 Componente Estrutural

O componente estrutural tem a responsabilidade de prover a estrutura do sistema, inicializando e estabelecendo a comunicação entre os demais componentes. Dessa forma, é o componente que define a arquitetura geral e o atendimento ao requisito de funcionar em navegadores *Web*. Além disso, deve fornecer as funcionalidades independentes de domínio que formam a abordagem dos iMA, principalmente relacionadas à comunicação com SGC e de configuração.

Apesar de ter muitas responsabilidades, a forma com que esse componente deve cumprí-las é definida pelas especificações dos outros componentes, para que sejam compatíveis. Assim, no nível de abstração do modelo de sistema, o componente estrutural é definido apenas como provedor de funcionalidades compatíveis com o funcionamento dos outros componentes.

No momento de implementação de um iMA, a instanciação do arcabouço, esse componente não precisa ser considerado além do fornecimento de uma interface com as funcionalidades que provém. Isso é possível porque todas as suas responsabilidades são independentes de domínio e não afetam o trabalho do programador que deve implementar as funcionalidades específicas.

4.2.2 Componente de Domínio

O componente de domínio é o responsável pelo fornecimento da base na qual as funcionalidades específicas de domínio são implementadas. Essa base deve ser tal que o funcionamento de todos os outros componentes do modelo sejam compatíveis e que seja possível criar interfaces com o usuário com interatividades pedagogicamente interessantes. Ou seja, seu papel é especificar minimamente a arquitetura das funcionalidades de domínio de forma que qualquer interface com o usuário possa ser compatível com o modelo de sistema.

O projeto desse componente foi baseado em dois conceitos: o funcionamento do paradigma de tutoria inteligente ETT (Aleven *et al.*, 2009) e o padrão de projeto orientado a objetos *Comando* (Gamma *et al.*, 1994). ETT são baseados em um grafo de estados que representa a atividade que o professor cria ou que o aluno realiza. O grafo é composto de vértices e arestas, no qual os vértices representam os possíveis estados da atividade, formados pelos objetos de domínio, e as arestas representam as transições entre estados, formadas por ações de domínio.

O padrão *Comando* desacopla uma operação em um objeto "comando", que funciona como um envelope para a mensagem-operação. A mensagem a ser passada é a chamada de um método da classe destinatária. Dessa forma, a modelagem para as operações de domínio que é independente do domínio em questão usada nesse componente aplica os conceitos de ações e objetos de domínio do paradigma ETT e define as ações como "comandos".

No caso dos iMA existentes, objetos de domínio têm como exemplos de geometria pontos, retas, circunferências; em cálculo há funções, variáveis, eixos; em combinatória há os elementos do universo e as restrições do problema; e em programação visual existem variáveis, linhas de código, estruturas de repetição, etc. De maneira análoga, ações de domínio em geometria têm como exemplos a criação ou movimentação de pontos, a criação de retas e circunferências. Na seção 4.1.2 as ações de domínio são descritas e classificadas de maneira genérica.

As ações como "comandos" são classes que recebem por parâmetros o estado da atividade sendo realizada e os objetos de domínio que manipula (por exemplo, o ponto que será movimentado no domínio da geometria) e ao serem executadas chamam um método do destinatário, considerado o "modelo de domínio", capaz de realizar efetivamente as operações. A figura 4.3 representa um diagrama de classes simplificado para essa modelagem.

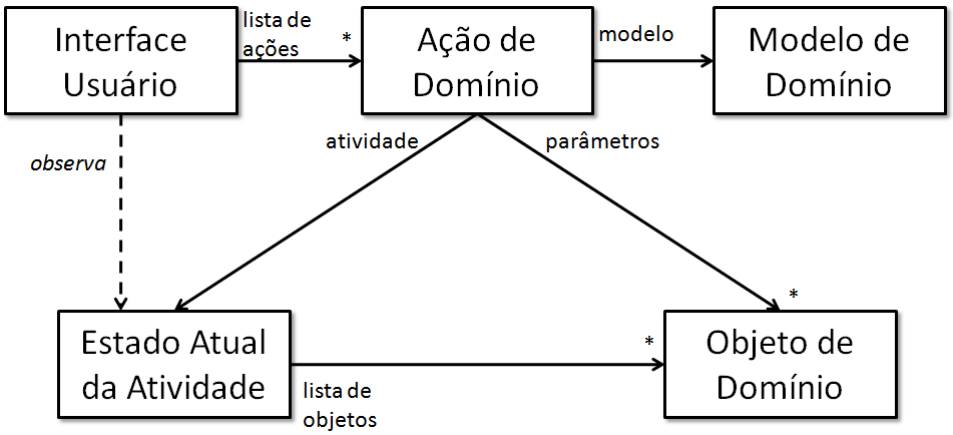

Figura 4.3: *Diagrama de classes simplificado para a modelagem das operações de domínio.*

A classe referente à interface com o usuário possui uma lista de ações de domínio, que são chamadas sempre que o usuário realiza uma ação específica. Essa classe também observa, usando o padrão de projeto *Observador* (Gamma *et al.*, 1994), o estado da atividade que é realizada no momento. A ação de domínio, a classe central dessa modelagem, possui uma referência ao estado da atividade corrente e a uma lista de objetos de domínio correspondente a seus parâmetros. Por fim, possui também uma referência ao modelo de domínio ao qual enviará a mensagem para realizar a operação.

Para a implementação de qualquer domínio usando esse modelo, basta criar classes que representam os objetos, ações e o modelo de domínio. Dessa forma, o arcabouço que implementa esse modelo tem como classes abstratas esses três elementos. Um exemplo de sequência de acontecimentos quando uma ação é executada pelo usuário nesse modelo é representado no diagrama de sequência da figura 4.4.

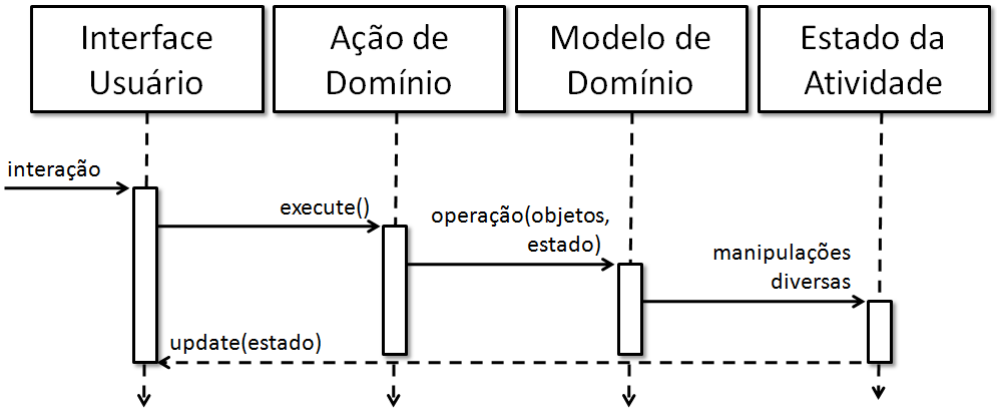

Figura 4.4: *Diagrama de sequência simplificado para a execução de uma ação de domínio.*

Nesse exemplo, o usuário interage com a interface usuário com o objetivo de realizar uma operação de domínio na atividade sendo feita. O objeto de interface chama a interface pública dos objetos "comando", que é o método *execute()* para que a ação de domínio envie a sua mensagem ao destinatário. Sua responsabilidade é, portanto, chamar um método do objeto de "modelo de domínio" correspondente à interação, informando os objetos-parâmetros e o estado da atividade corrente. Por fim, o modelo de domínio realiza manipulações diversas sobre o estado que informa a seus observadores que foi alterado pelo método *update(estado)*. A interface usuário recebe a atualização e mostra ao usuário o novo estado da atividade.

Essa modelagem abstrata das operações de domínio foi escolhida pelo fato de permitir qualquer interface usuário ser criada, para atender à possibilidade de implementação de interatividades relevantes pedagogicamente. Para desenvolver uma interface com o usuário nesse modelo basta que todas as interações do usuário ativem "comandos", ou seja, não há restrição quanto ao projeto da interação, apenas ao funcionamento interno do sistema. Além disso, o modelo garante a compatibilidade com o funcionamento dos componentes de atividade e de extensões, como são apresentados a seguir.

4.2.3 Componente de Atividades

O componente de atividades define a estrutura de dados utilizada para modelar uma atividade de maneira compatível com as funcionalidades de gerenciamento e as dos outros componentes. As funcionalidades de gerenciamento incluem a criação e edição de atividades por professores, e a realização das mesmas por alunos.

Usando o mesmo modelo do componente de domínio, uma atividade deve conter um estado atual, aquele no qual as ações são realizadas, que contém a lista de objetos de domínio sendo manipulados na atividade. Além desse estado, para permitir a autoria, uma atividade deve conter também um estado inicial, que o aluno visualiza quando começa a realizá-la, e um estado esperado, o gabarito da atividade, que será usado para a comparação com o estado atual quando a funcionalidade de avaliação automática for ativada.

Além desses três estados, uma atividade deve possuir informações complementares de configuração e dos módulos de extensão. As configurações definem o comportamento da atividade de acordo com a intenção do professor que a criou, e pode variar entre os domínios. Um exemplo é a limitação de acesso a funcionalidades que o aluno poderia usar e prejudicar sua aprendizagem. As informações dos módulos são dados usados pelas funcionalidades do componente de extensões para seu correto funcionamento, e variam entre elas. A figura 4.5 apresenta um diagrama de classes simplificado com os principais elementos desse componente.

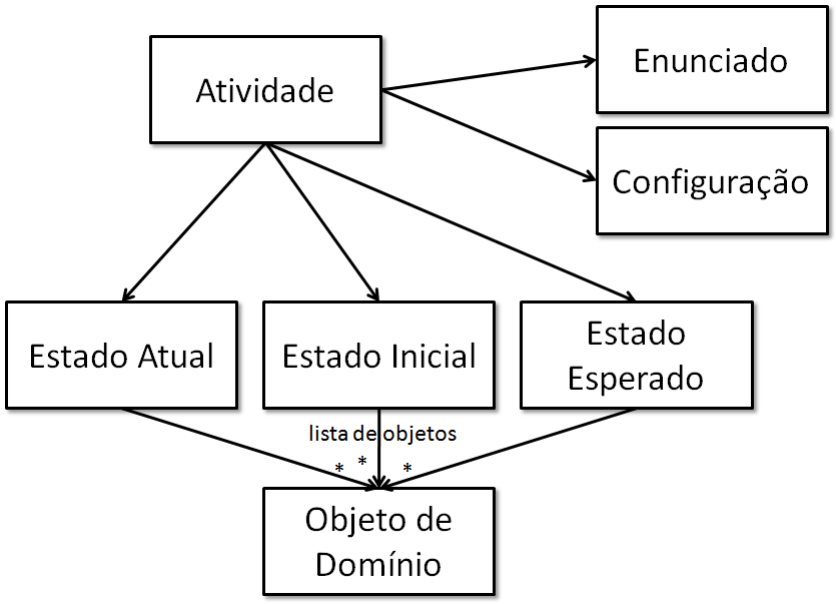

Figura 4.5: *Diagrama de classes simplificado com os elementos de uma atividade.*

Durante a implementação de um iMA, não é necessário extender as funcionalidades providas por esse componente. O único comportamento variável de domínio nesse caso é a interface com o usuário de autoria. A interface de autoria de atividades, segundo esse modelo, deve permitir ao professor definir os estados inicial e final, e as configurações da atividade. Como os estados, compostos por objetos de domínio, e as configurações são dependentes do domínio em questão, a interface com o usuário de autoria também o é. Esse procedimento é apresentado na seção 4.4.2.

4.2.4 Componente de Extensões

O componente de extensões é responsável pelas características de funcionalidades sobre atividades e sobre operações de domínio. Assim, é composto de duas especificações de módulos de extensões.

Funcionalidades sobre Atividades

As funcionalidades sobre atividades são aquelas que recebem uma atividade completa, definida na seção anterior, e realiza uma operação sobre ela. Esse tipo de operação foi inspirado pela avaliação automática dos iMA existentes e pela definição do comportamento do tutor inteligente, como funcionalidade a ser adicionada à família.

Dessa forma, de acordo com o definido anteriormente, o modelo para essa extensão é de um módulo que recebe uma atividade como parâmetro, e realiza sua operação. Para que seja possível a integração desse tipo de operação, os outros componentes devem permitir acesso a suas informações (da atividade, da configuração, por exemplo) através de uma interface pública. Mais detalhes sobre a implementação desses módulos no arcabouço são apresentados na seção 4.5.1.

Funcionalidades sobre Operações de Domínio

As funcionalidades sobre operações de domínio são aquelas que recebem uma ação de domínio (um "comando", como definido anteriormente) e realizam uma operação sobre ela. Esse tipo de operação foi inspirado pelos *scripts* do *iGeom* e pelo módulo atuador do tutor inteligente, que executa ações de ajuda para o aluno em resposta a ações de domínio realizadas por ele.

Assim, a arquitetura desse tipo de extensão deve ser de um módulo que recebe ações de domínio e realiza sua operação. Como o usuário interage diretamente com o sistema para realizar a ação de domínio, esses módulos não podem exigir que o usuário também ative diretamente suas funcionalidades. Dessa forma, os módulos de funcionalidades sobre ações de domínio devem usar o padrão de projeto *Observador* (Gamma *et al.*, 1994), para serem informados sempre que uma ação for executada.

Durante a implementação de um iMA, ou seja, a utilização do arcabouço para criar um sistema, as extensões podem cumprir papéis diferentes. Há módulos com funcionalidades sobre atividades e sobre operações de domínio que são genéricos o suficiente para serem usados por todos os iMA, como a avaliação automática e desfazer-refazer. Há também módulos independentes de domínio que eventualmente são relevantes pedagogicamente para o domínio em questão, como *scripts*, que podem ser importantes em assuntos relacionados a matemática mas não necessariamente quando o tema é biologia, por exemplo. Por fim, é possível inclusive a criação de módulos de extensão específicos para um domínio. Um exemplo desse último caso é a exportação de programas criados com o iMA *iVProg* para diferentes linguagens de programação, como *C* ou *Java*. O objetivo da definição desse componente é permitir esses três tipos de utilização pelos programadores.

A especificação desses quatro componentes do modelo de sistema abrange todas as funcionalidades requeridas para os iMA e cumpre os objetivos de manter a compatibilidade entre elas e sua independência do domínio dos sistemas. A compatibilidade entre o funcionamento dos componentes é garantida uma vez que suas definições são interrelacionadas. A independência do domínio é mostrada com suas definições que têm como exemplos instanciações em diferentes domínios. Ou-

tras informações sobre a implementação desse componente no arcabouço são apresentados na seção 4.5.2. A próxima seção apresenta o arcabouço de aplicação que foi implementado tendo como base o modelo de sistema apresentado.

4.3 O Arcabouço de Aplicação

Como materialização do modelo de sistema, da análise de domínio dos iMA existentes, e da lista de requisitos da LPS, um arcabouço de aplicação foi desenvolvido (Dalmon *et al.*, 2011a). Esse arcabouço é a peça concreta central da LPS, de forma que no cotidiano do andamento deste projeto, muitas vezes a LPS se confundia com ele. O arcabouço fornece código e arquitetura para serem reutilizados durante a implementação de novos iMA.

Esta seção apresenta em primeiro lugar o processo de desenvolvimento do arcabouço, destacando etapas intermediárias e as lições aprendidas com relação ao andamento do projeto. Em seguida, é apresentada a versão final do arcabouço em si, sua estrutura de componentes e seu funcionamento interno. Os métodos de utilização e evolução do arcabouço definidos são apresentados nas seções posteriores.

O número atual de linhas de código do arcabouço está em torno de 5.000, para a instanciação teste, esse número chega a 500. Comparado a iMA, esse número ainda é pequeno, *iGeom* tem cerca de 45.000, *iVProg* 37.000, *iGraf* 15.000 e *iComb* 8.000. O código do arcabouço está disponível como código livre na página do projeto (Dalmon e Brandão, 2012).

4.3.1 Desenvolvimento

O projeto e implementação do arcabouço foi iterativo, as primeiras versões incluíam poucas funcionalidades e aos poucos eram adicionadas em versões posteriores. A partir das características da LPS e do modelo de sistema, o desenvolvimento do arcabouço prosseguiu com o método PLUS, descrito na seção 3.2.2. As funcionalidades do arcabouço eram descritas por meios de casos de uso que consideram a variabilidade nos produtos da LPS, como mostra o diagrama da figura 4.6. Na figura, os casos de uso em **negrito** são comuns a todos iMA, enquanto que os em *itálico* são opcionais.

Esse diagrama descreve as funcionalidades associadas ao gerenciamento de atividades do arcabouço. Existem três tipos de usuários do sistema: o aluno, o professor e o SGC. O aluno realiza a atividade, e também ativa a avaliação automática. O SGC consulta as informações de avaliações das atividades dos alunos para armazenar e elaborar relatórios. O professor tem acesso às mesmas funcionalidades do aluno e também pode criar e editar atividades, que o SGC registra em arquivos em seu servidor. Além dessas funcionalidades obrigatórias para o arcabouço, os usuários podem ter acesso a operações opcionais em alguns iMA, como exportar a atividade para imagens ou objetos de aprendizagem.

A variabilidade considerada nos diagramas de caso de uso mostra categorias obrigatórias e opcionais. As obrigatórias estão presentes em todos os iMA, enquanto que as opcionais são escolhidas durante o desenvolvimento dos aplicativos. Casos de uso variantes, em que o comportamento é diferente em cada iMA, não ocorrem no nível do arcabouço. Os comportamentos variantes providos pelo arcabouço são aqueles que devem ser especificados durante sua instanciação, considerando particularidades do domínio. Um desses comportamentos é a interface com o usuário.

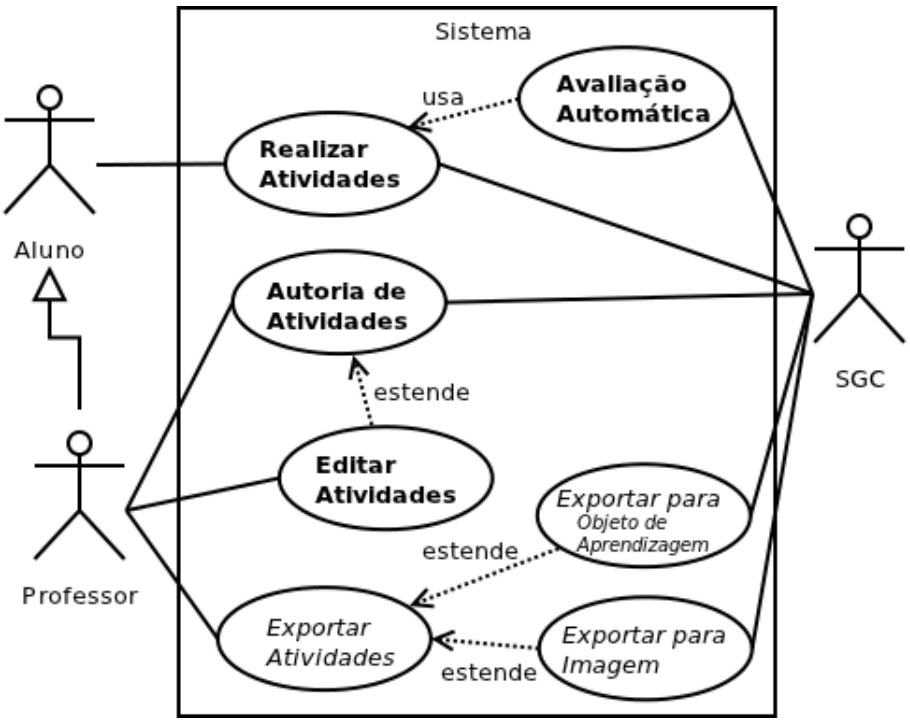

Figura 4.6: *Diagrama de casos de uso para a caracterítica de gerenciamento de atividades.*

A primeira iteração no desenvolvimento do arcabouço, que gerou sua primeira versão, incluía diagramas de casos de uso para as principais funcionalidades independentes de domínio e alguns diagramas de classe antes de iniciar a implementação. Um exemplo de diagrama de classes usado nas primeiras versões do arcabouço é mostrado na figura 4.7. Esse diagrama contém classes para a implementação das funcionalidades de gerenciamento das atividades, como atividade, estado de atividade, controlador das atividades, interface de atividades e interface de autoria.

A variabilidade considerada nesse diagrama classifica as classes de interface, de ações e de objetos de domínio como variantes, os módulos como opcionais e as restantes como obrigatórias. Essa divisão mantém o critério dos elementos variantes representarem os comportamentos dependentes de domínio. Ainda nas primeiras versões do arcabouço haviam outras categorias de classes com comportamentos variantes, que foram descartadas ou adaptadas para simplificar o projeto, a implementação e o uso do arcabouço.

Após as etapas de projeto de software, a implementação da primeira versão do arcabouço foi iniciada. Ela era composta de uma versão simplificada do modelo de sistema e um protótipo para a interface com o usuário. Ainda na primeira versão foi feita uma instanciação teste para o arcabouço. Essa instanciação teste permitia executar o arcabouço como um programa completo, modelando um domínio muito simples de iMA. Assim, era possível avaliar o processo de instanciação do arcabouço e simultaneamente testar as funcionalidades do arcabouço como se houvesse um iMA pronto. Mais informações sobre a instanciação teste são apresentadas na seção 4.4. Como a instanciação teste é mais dependente do modelo de sistema do que das especificidades e outras funcionalidades do arcabouço, seu código foi pouco alterado ao longo das versões.

Na segunda versão, um esforço maior foi despendido nas etapas de projeto de software. Nesse momento, diagramas para todos os componentes do arcabouço e a arquitetura detalhada foram definidos. Porém, durante a implementação, várias incompatibilidades entre os componentes proje-

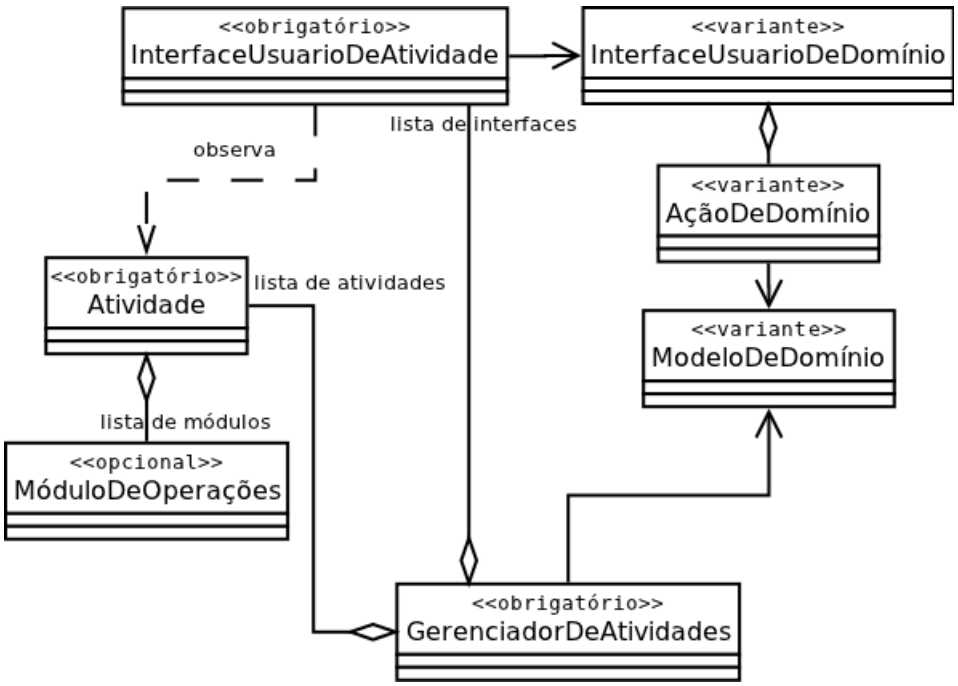

Figura 4.7: *Diagrama de classes para a caracterítica de gerenciamento de atividades.*

tados foram descobertas, o que levou ao projeto da terceira iteração. A partir da teceira iteração, o projeto de software e a arquitetura do sistema foram simplificados. Em seguida, na quarta versão do arcabouço, o projeto e a arquitetura estavam simples e coerentes o suficiente para a implementação de todas as funcionalidades.

Nesse processo, as fases de projeto (casos de uso, análise estática e dinâmica, e arquitetural) foram usadas para guiar a implementação. A partir da terceira iteração, apenas o projeto arquitetural foi usado com o objetivo de simplificar as definições e melhorar a compatibilidade entre os componentes. A implementação realizava o projeto e quando eram encontradas incoerências que não eram possíveis de resolver diretamente no código uma nova versão do arcabouço era iniciada. A etapa de testes foi usada a partir da terceira iteração, principalmente com o objetivo de verificar o funcionamento dos componentes separadamente e manter as interfaces de comunicação sem muitas alterações.

Como descrito anteriormente, o gerenciamento da variabilidade mudou ao longo do processo, com o objetivo de simplificar o desenvolvimento e o uso do arcabouço. No início, com o uso direto do método PLUS, várias formas de variabilidade foram identificadas e usadas no projeto. Com as novas versões, os tipos de variabilidade foram reduzidos para os comportamentos dependentes de domínio, os variáveis, e os opcionais, correspondentes às funcionalidades de extensões. Essa redução simplificou consideravelmente a implementação e permitiu que essa etapa fosse executada mais rapidamente.

O processo de desenvolvimento do arcabouço foi descrito com o objetivo de explicitar os desafios enfrentados e as decisões tomadas para que seja possível destacarmos algumas lições aprendidas. Quatro considerações principais podem ser feitas: (i) com relação aos métodos usados; (ii) ao processo iterativo; (iii) ao gerenciamento de variabilidade; e (iv) à documentação. Os métodos usados, FODA e PLUS, foram importantes para definir as etapas e as decisões que deveriam ser tomadas a cada momento, e satisfizeram as necessidades de modelagem. Com o tempo, a utilização desses

métodos foi simplificada e reduzida para concentrar os esforços nas tarefas de implementação e teste.

O processo iterativo foi essencial para permitir a revisão de problemas de etapas anteriores e aprimorar continuamente o projeto e o código do arcabouço. Apesar de consumir tempo a elaboração de várias versões de um sistema, o conhecimento obtido e acumulado com as iterações permite que a entrega final do sistema tenha sua qualidade fortalecida por seguidas revisões.

O gerenciamento de variabilidade permitiu, no início do processo, a explicitação e modelagem do domínio como um todo, mas apresentou dificuldades para a implementação por ser complexo demais. Com o andamento e a simplificação das variabilidades consideradas, a implementação se tornou também mais simples. Essa decisão tem também consequências para o uso do arcabouço para o desenvolvimento de iMA, como mostra a seção 5.2.

Por fim, o processo realizado não seria possível se não houvesse documentação do projeto de software e da implementação de cada iteração. Essa documentação acumulou informações durante o andamento do processo e facilitou a tomada de decisões por permitir o reconhecimento das falhas de projeto e o início de novas versões do arcabouço. O resultado final do arcabouço desenvolvido pelo processo relatado aqui é apresentado a seguir.

4.3.2 Estrutura de Componentes

A arquitetura atual do arcabouço é baseada em componentes que refletem aqueles definidos no modelo de sistema, além de outros com responsabilidades correspondentes a características com maior grau de detalhe. Especificamente no código-fonte, cada componente é representado por um pacote *Java*. A figura 4.8 apresenta a arquitetura do arcabouço desenvolvido.

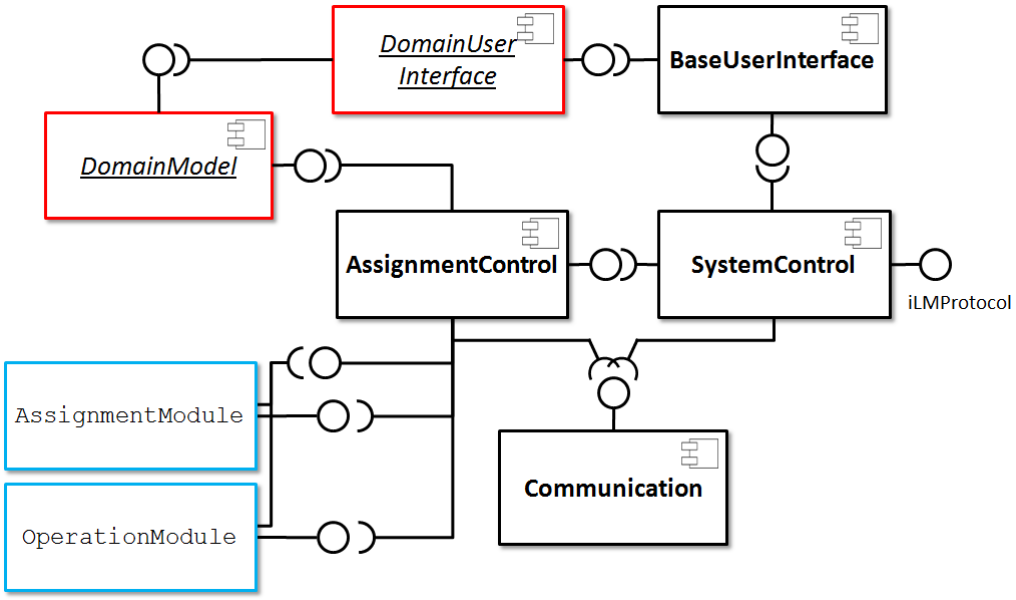

Figura 4.8: *Diagrama de componentes do arcabouço para desenvolvimento de iMA.*

Os componentes são divididos em três categorias: (i) obrigatórios, representados com **fonte em negrito**; (ii) específicos de domínio, em *itálico e sublinhado*; e (iii) opcionais em `fonte monoespaçada`. Essa classificação reflete a variabilidade de cada componente, os obrigatórios estão presentes em todos iMA sem sofrer alteração, os dependentes de domínio devem ser especializados durante

a instanciação do arcabouço para o domínio do iMA e os opcionais devem ser escolhidos se fazem parte do aplicativo. A seguir cada componente é apresentado em detalhe.

DomainUserInterface

O componente da interface com o usuário dependente do domínio é um dos comportamentos variantes do arcabouço, em que as principais classes devem ser estendidas para promover as funcionalidades específicas. O arcabouço fornece basicamente classes abstratas cujos métodos de comunicação com os outros componentes estão prontos e os métodos de construção dos elementos de interface podem ser implementados da maneira mais apropriada para cada domínio.

Existem duas interfaces com o usuário dependentes do domínio: (i) a de atividade e (ii) a de autoria. A de atividade é aquela em que o aluno utiliza para realizar uma tarefa, e a de autoria é a que o professor cria as atividades para seus alunos. Quando um programador usa o arcabouço, são essas duas interfaces que deve desenvolver.

BaseUserInterface

A base da interface do usuário dos iMA é fornecida como um comportamento invariável entre os aplicativos. Esse componente é composto por uma classe "JFrame" (Oracle, 2012e) que representa uma espécie de moldura para a interface usuário específica de domínio. Como mostra a figura 4.9.

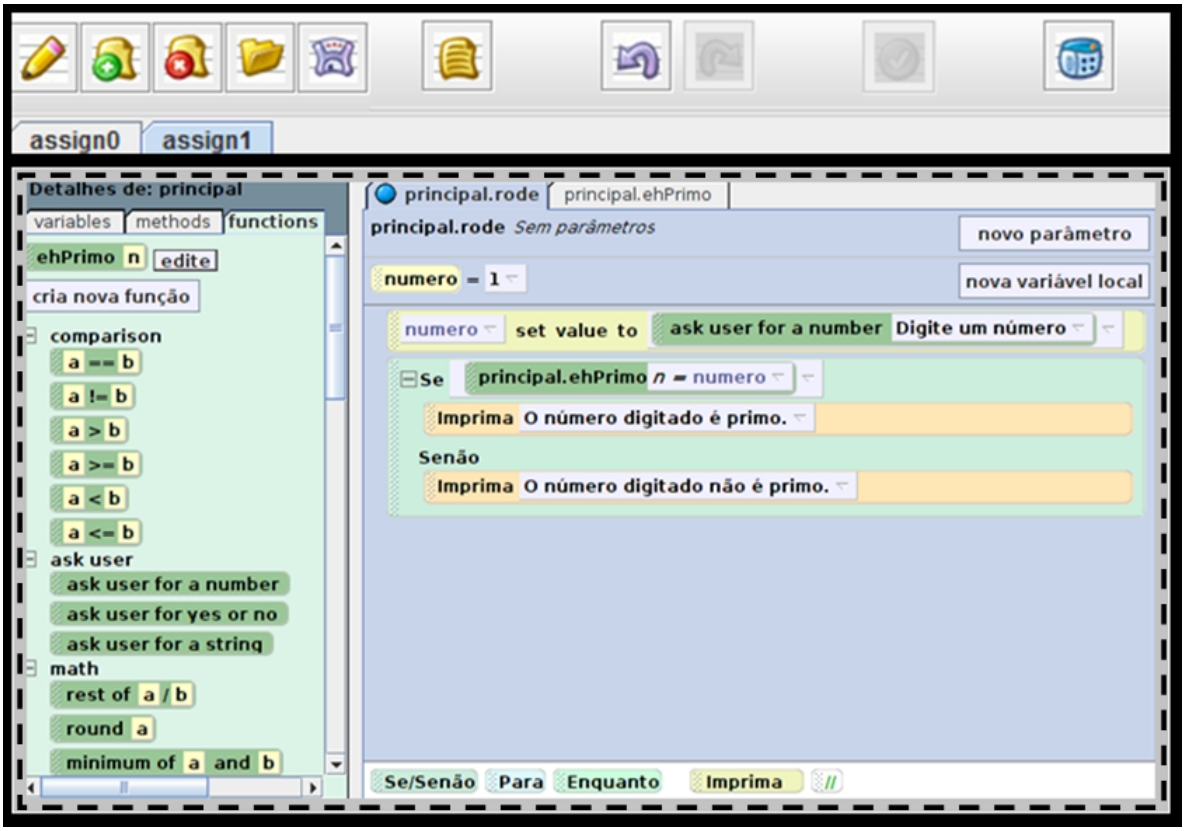

Figura 4.9: *Imagem da interface gráfica do arcabouço, formada pela base, com borda preta, e a específica de domínio, com borda tracejada, usando como exemplo o iMA iVProg.*

Essa figura mostra a interface do usuário base do arcabouço desenvolvido e a interface específica, destacada em preto, de domínio da instanciação teste, destacada com uma linha tracejada. A

instanciação teste é descrita em detalhes na seção 4.4. Um iMA teria essa aparência, substituindo a região com bordas vermelhas pela sua interface.

A interface base inicializa e provê a comunicação com o restante do arcabouço à interface específica. Além de fornecer essa "moldura" para a interface de domínio, o componente base tem como responsabilidade o gerenciamento dos botões e outras estruturas gráficas das funcionalidades independentes de domínio. Essas funcionalidades são apresentadas na imagem como a lista de botões na parte superior.

DomainModel

O componente do modelo de domínio é o responsável pela implementação do componente de domínio descrito no modelo de sistema para iMA. Ele é composto por classes abstratas referentes aos objetos e ações de domínio, assim como o modelo de domínio. Essas classes já possuem as funcionalidades que as comunicam com o restante do arcabouço, deixando alguns métodos abstratos para serem definidos especificamente em cada domínio.

Dessa forma, para criar um iMA usando o arcabouço, um programador deve criar todas as classes de objetos e ações de domínio, associar as ações à interface com o usuário específica e implementar as operações na classe do modelo de domínio.

Esse componente possui ainda uma classe abstrata para converter os objetos e ações de domínio para o formato que é gravado em arquivos das atividades. Como cada iMA existente possui um formato de arquivo, o arcabouço delega função de conversão de dados a essa classe de domínio. Futuramente isso será opcional, uma vez que o arcabouço possuirá um formato padrão para converter os dados e gravá-los em arquivos.

AssignmentControl

Esse é o componente de controle, ou gerenciamento, no nível das atividades, sendo parte do núcleo obrigatório do arcabouço. Ele é responsável por implementar as funcionalidades definidas pelo componente de atividades do modelo de sistema. Dessa forma, as funcionalidades de autoria, realização e gerenciamento das atividades são feitas por ele.

Além disso, é também responsável por interligar as atividades com o restante do arcabouço, os componentes *DomainModel*, *OperationModule*, *AssignmentModule*, e com as funcionalidades de comunicação fornecidas por *Communication*. Essas relações são necessárias pelo fato das atividades serem a característica fundamental dos iMA.

SystemControl

O componente de controle do sistema é o responsável pela inicialização de todos os outros componentes e por definir a comunicação entre eles. Além disso, representa o papel do componente estrutural do modelo de sistema para iMA em conjunto com o componente *Communication*.

Com relação às características definidas na seção 4.1.3, esse componente também implementa as funcionalidades de configuração, fornecendo as informações aos outros componentes para controlarem seus comportamentos. Por fim, ele fornece a interface pública *iLMProtocol* para os sistemas externos ao iMA, o que normalmente seria responsabilidade de comunicação. Essa decisão foi tomada pelo fato da comunicação entre a página e o aplicativo ocorrer pela classe que estende de

"JApplet" (Oracle, 2012b), que no caso do arcabouço está situada em nesse componente.

Communication

Esse componente é responsável pela comunicação com os sistemas externos, no caso da versão *desktop* do iMA é o sistema operacional, e no caso da versão *Applet* é o SGC. Essa comunicação consiste na leitura e gravação de arquivos em sistemas operacionais e em uma interface pública chamada *iLMProtocol* que contém os métodos do protocolo de comunicação dos iMA com as páginas *Web* de SGC.

OperationModule e *AssignmentModule*

O componente módulo de operações representa a arquitetura de *plug-ins* prevista pelo arcabouço para adicionar funcionalidades sobre operações de domínio. Esses *plug-ins* devem implementar uma interface provida pelo arcabouço e serem incluídos no código durante a implementação do iMA.

De maneira análoga ao componente anterior, o módulo de atividade representa os *plug-ins* que adicionam funcionalidades sobre atividades. Os módulos são gerenciados e relacionados com as atividades pelo componente *AssignmentControl*. Mais detalhes sobre o funcionamento desses componentes são apresentados na seção 4.5.

O funcionamento interno do arcabouço, assim como a sua documentação completa são apresentados no manual entitulado "Manual de Funcionamento e Estrutura Interna do Arcabouço iMA", que acompanha o código, disponível em (Dalmon e Brandão, 2012). A próxima seção descreve mais detalhes sobre como o arcabouço pode ser usado para desenvolver iMA.

4.4 Engenharia de Aplicação

De acordo com a descrição dos componentes, a Engenharia de Aplicação, ou seja, a utilização do arcabouço para a criação de iMA consiste na extensão de dois componentes: *DomainModel* e *DomainUserInterface*. Esta seção descreve com mais detalhes esses dois procedimentos.

O processo de utilização do arcabouço é detalhado e descrito de forma completa e seguindo um passo a passo em um manual preparado especificamente com esse objetivo. Esse manual, entitulado "Manual de Utilização do Arcabouço para o Desenvolvimento de iMA", é disponibilizado em conjunto com o código do arcabouço para ser utilizado como recurso por desenvolvedores na página desta pesquisa (Dalmon e Brandão, 2012). Os exemplos usados no manual para cada passo durante a utilização do arcabouço para o desenvolvimento de um iMA apresentam a instanciação teste usada durante a criação do arcabouço. Assim, nas descrições desta seção também são usados exemplos dessa mesma fonte.

A definição desse processo foi considerada durante o desenvolvimento do arcabouço de forma que seja simples e compatível com a experiência dos programadores que usualmente desenvolvem iMA. Assim, o trabalho para simplificar o gerenciamento da variabilidade também teve objetivo de reduzir a dificuldade de uso do arcabouço. Considerações sobre esse uso são descritas nas seções de resultados, 5.1 e 5.2.

Especificando alguns detalhes técnicos, além dos dois componentes descritos abaixo, o desenvolvedor criador de iMA deve ainda estender duas classes. Essas classes são mais simples do que as descritas para os componentes, e futuramente devem ser criadas por *plug-ins* de ambientes de

desenvolvimento. Existe uma classe chamada "fábrica de sistema", na qual o desenvolvedor deve implementar métodos abstratos que indicam quais classes específicas de domínio constituem o iMA sendo desenvolvido. A outra classe deve possuir o mesmo nome do iMA e precisa implementar os métodos *init* da versão *Applet* e *main* da versão *desktop*, inicializando a classe central do arcabouço com a fábrica de sistema específica do iMA.

Além da técnica orientada a objetos para a instanciação de arcabouços utilizada neste trabalho existem outras, como reflexão e metadados (Guerra, 2010). Essas técnicas não foram usadas por exigirem do desenvolvedor do arcabouço e eventualmente dos usuários que irão instanciá-lo, conhecimentos específicos dessas técnicas, o que pode aumentar a dificuldade de suas tarefas. Ainda assim, está previsto como trabalho futuro a adição de técnicas mais sofisticadas para simplificar no longo prazo o trabalho com o arcabouço.

4.4.1 Estender o Modelo de Domínio

O procedimento para estender o modelo de domínio fornecido pelo componente *DomainModel* é basicamente a instanciação do conceito abstrato apresentado no modelo de sistema. Existem quatro classes abstratas que devem ser estendidas: (i) objeto de domínio; (ii) ação de domínio; (iii) modelo de domínio; e (iv) conversor de domínio.

- **Objeto de Domínio** - classe mais simples, para a qual o desenvolvedor não é obrigado a implementar nenhum método. Essa classe pode representar qualquer objeto de domínio. Para definir um, a única exigência do arcabouço é a herança da classe fornecida por ele chamada "objeto de domínio". Uma classe deve ser criada para cada objeto de domínio existente no iMA.

- **Ação de Domínio** - classe um pouco mais complexa do que a dos objetos de domínio, uma vez que o desenvolvedor é obrigado a implementar dois métodos. Esses métodos são a execução da ação e sua operação inversa, para efeitos da funcionalidade desfazer. O código desses métodos deve ser somente a chamada de operações da classe do modelo de domínio. Outros dois métodos são necessários para definir os parâmetros que serão passados nessa mensagem. Uma classe deve ser criada para cada ação de domínio existente no iMA.

- **Modelo de Domínio** - classe em que as operações de domínio são realizadas, métodos que são chamados quando ações de domínio são executadas. Esses métodos não são especificados pelo arcabouço, suas responsabilidades obrigatórias são apenas iniciar um estado de atividade e o algoritmo de avaliação automática. O arcabouço exige que os métodos das operações de domínio recebam o estado da atividade em que ela é realizada e os parâmetros nos quais a operação é feita. Deve haver apenas uma classe modelo de domínio, e pelo menos um método para cada ação de domínio do iMA.

- **Conversor de Domínio** - a última classe do componente modelo de domínio é um detalhe técnico exigido pela abordagem dos iMA de gravar arquivos de atividade. Nesses arquivos são armazenados os dados da solução do aluno e da preparação da atividade pelo professor, que contém informações sobre os objetos e ações de domínio. Esse conversor deve implementar métodos de conversão de objetos e ações de domínio em texto (*string*) e a operação inversa.

O procedimento descrito aqui não é complexo, mas exige do desenvolvedor a capacidade de abstrair os conceitos envolvidos e separar as responsabilidades das estruturas de dados (objetos de domínio) e as operações (ações e modelo). Isso também obriga que o código do iMA seja separado dessa maneira, o que possivelmente facilita sua manutenção e evolução

Usando a instanciação teste criada durante o desenvolvimento do arcabouço como exemplo, temos o domínio simplório de adição e remoção de caracteres de uma palavra. Esse domínio não tem nenhum objetivo pedagógico, foi criado apenas para ser o exemplo mínimo que utilize as funcionalidades oferecidas pelo arcabouço.

Desse modo, há apenas um tipo de objeto de domínio, o caractere. Um estado de atividade é representado por uma palavra. Existem duas ações de domínio: adição e remoção de caracteres do estado da atividade, aumentando ou reduzindo a palavra. Assim, o modelo de domínio deve ser capaz de realizar essas duas operações. Por fim, o conversor deve ser capaz de criar esse objeto e os dois tipos de ação a partir de uma representação em texto e vice-versa. A interface do usuário, que ilustra melhor esse domínio, é descrita logo abaixo.

4.4.2 Estender a Interface Gráfica de Domínio

De maneira análoga ao componente de modelo, o componente de interface do usuário possui uma lista de classes abstratas que devem ser estendidas com o objetivo de adicionar o comportamento específico dependente de domínio. No caso da interface, há duas classes: (i) a interface de atividade; e (ii) a interface de autoria.

- **Interface com o Usuário de Atividade** - essa classe é a que mais representa o iMA, sendo utilizada por professores e alunos, e também usuários apenas conhecendo o aplicativo. Nessa interface os objetos de domínio são manipulados pelas ações de domínio e as atividades são realizadas. Para instanciar essa classe, o desenvolvedor de iMA deve implementar alguns métodos exigidos pelo arcabouço para inicializar a interface de maneira integrada com o restante do sistema e também métodos requeridos pela interface de autoria. Apenas uma classe de interface do usuário deve ser criada por iMA.

- **Interface com o Usuário de Autoria** - essa interface permite ao professor criar atividades, seguindo a estrutura definida no modelo de sistema. Assim, para instanciar a classe fornecida pelo arcabouço é necessário implementar métodos que preencham todos os campos existentes que definem uma atividade. Alguns desses métodos fazem referência à interface de atividade, uma vez que são usados objetos de domínios manipulados nela. Cada iMA deve possuir apenas uma interface de autoria.

No exemplo, a instanciação teste do arcabouço, no domínio da adição e remoção de caracteres de uma palavra, também possui uma interface de atividade e uma de autoria. As figuras 4.10 e 4.11 apresentam essas duas interfaces. O usuário escreveu “mensagem de teste” na entrada de texto e pressionou cinco vezes o botão “adicionar”. Na interface de autoria, os dois primeiros objetos “m1” e “e2” foram adicionados ao estado inicial.

Na interface de atividade, temos um campo em que o usuário escreve sua palavra, botões para adicionar e remover caracteres e a palavra do estado da atividade em uma etiqueta (*label*). Os botões, quando pressionados, disparam as ações de domínio sobre o estado da atividade, representado por uma lista de objetos de domínio que formam a palavra apresentada à direita.

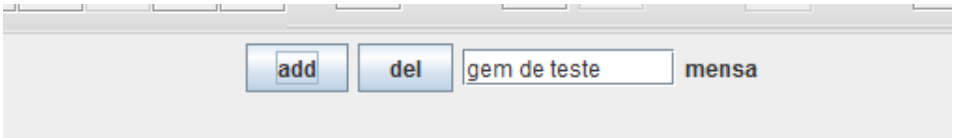

Figura 4.10: *Interface de atividade da instanciação teste do arcabouço.*

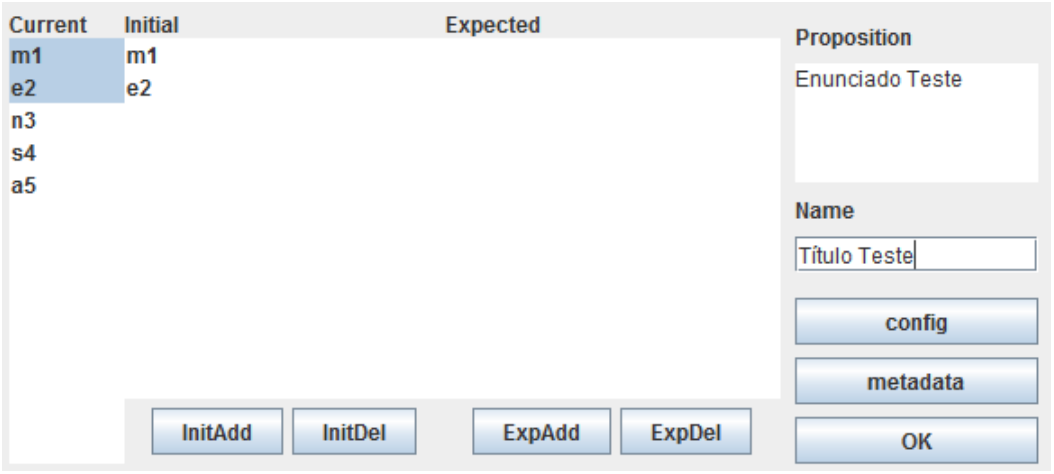

Figura 4.11: *Interface de autoria da instanciação teste do arcabouço.*

Na interface de autoria, temos três listas de objetos, uma para o estado atual, que contém exatamente os objetos correntemente mostrados pela interface de atividade, uma para o estado inicial e uma para o gabarito. Nesse domínio é suficiente haver botões para adicionar os objetos selecionados da lista do estado atual nos outros estados, ou remover algum de seus objetos. Além disso, há campos para o usuário definir o título da atividade, seu enunciado, a configuração e seus metadados.

Seguindo esses seis passos, ou seja, instanciando seis classes diferentes, é possível criar um iMA. O método definido para criar iMA usando o arcabouço permite ao desenvolvedor usar essa receita passo a passo de maneira a não se envolver com detalhes da implementação da estrutura do sistema ou de funcionalidades dependentes de domínio. Além dos passos definidos para a utilização do arcabouço, o desenvolvedor também pode criar funcionalidades para seu iMA usando os módulos de extensões, descritos na seção seguinte.

O método de utilização do arcabouço apresentado aqui é parte do processo fornecido pela LPS para ser reutilizado. O processo inclui também a organização das pessoas e as divisões de responsabilidade. Esses aspectos não são explorados significativamente nesta dissertação, porque foram mantidas as características anteriores do grupo de pesquisa. Mais considerações sobre o processo fornecido pela LPS são feitas no capítulo 5, sobre avaliação.

Uma consideração sobre o método proposto pode ser feita com relação aos mecanismos de orientação a objetos utilizados. Os mais utilizados foram a herança de classes abstratas e a sobrescrita de métodos. Esses mecanismos não são mais recomendados por desenvolvedores de arcabouços por provocarem dificuldades de extensão e manutenção no longo prazo (Guerra, 2010). Assim, estão incluídos nos trabalhos futuros a revisão dos mecanismos de instanciação do arcabouço e detalhes do método de criação de iMA.

4.5 Evolução da LPS

As últimas considerações sobre a LPS desenvolvida envolvem o planejamento de sua evolução. Essa evolução é prevista em dois aspectos: (i) manutenção de software sobre o código do arcabouço e consequentemente sobre os manuais que o descrevem; e (ii) a sua expansão pela adição de funcionalidades ao arcabouço com a utilização de módulos, ou *plug-ins*.

A manutenção do código do arcabouço foi considerada durante todo o seu desenvolvimento. O código criado tenta seguir as recomendações e boas práticas de programação orientada a objetos e possui uma documentação que inclui anotações *Javadocs* e um manual para descrever seu funcionamento interno, que acompanha o código-fonte. Dessa forma, espera-se que a manutenção e evolução do funcionamento do arcabouço não ofereça muitas dificuldades aos futuros programadores.

Para levantar indícios da facilidade de manutenção do código do arcabouço, ele foi analisado seguindo as mesmas métricas de código usadas no estudo de caso (Chidamber e Kemerer, 1994), descritas na seção de avaliação 3.3.3. A tabela 4.2 mostra os resultados dessa análise. Os valores obtidos com essa análise podem ser considerados entre bons e médios, de acordo com as análises do artigo e do manual da aplicação utilizada. Uma métrica não sugerida foi analisada pelo aplicativo e apontada como problemática, o número de parâmetros nos métodos, para a qual há alguns casos com valores muito altos.

Tabela 4.2: *Resultado da análise do código do arcabouço segundo métricas relevantes para sua manutenção.*

Métrica	Valor Médio	Valor Máximo
Métodos Ponderados por Classe	9,78	43,0
Profundidade da Árvore de Heranças	2,90	7,00
Número de Filhos	0,37	6,00
Acoplamento entre Classes	2,52	11,0
Falta de Coesão entre Métodos	0,32	1,00

A análise completa do aplicativo em forma de relatório está disponível na página da pesquisa (Dalmon e Brandão, 2012). Esses resultados mostram que mesmo com o esforço despendido para manter a qualidade do código alta, os valores das métricas foram apenas razoáveis. Assim, durante o desenvolvimento das próximas funcionalidades ou refatorações deve-se sempre avaliar as métricas com o objetivo de manter ou melhorar esses valores.

A outra forma planejada de evolução do arcabouço núcleo da LPS é a implementação de novas funcionalidades usando a arquitetura de *plug-ins*, ou módulos. De acordo com o descrito nas seções sobre o modelo de sistema e a arquitetura de componentes, existem dois tipos de módulos (*plug-ins*), de atividade e de operações, que são detalhados a seguir.

4.5.1 Módulos de Atividade

Os módulos de atividade são os componentes que fornecem funcionalidades sobre as atividades do iMA. Eles funcionam como *plug-ins*, ou seja, o arcabouço fornece uma interface com métodos que o módulo precisa implementar e outros métodos que dão acesso a informações internas necessárias para realizar as suas operações. Nesse caso, as informações que o módulo acessa são as atividades.

Para implementar um módulo de atividade, o arcabouço exige apenas que a classe principal do

módulo herde da classe abstrata fornecida e preencha um método que defina sua interface gráfica com o usuário. A herança da classe fornecida pelo arcabouço garante seu gerenciamento interno no sistema e dá acesso às informações necessárias para suas funcionalidades.

A interface do usuário permite que o usuário ative as funcionalidades fornecidas quando desejar. Nesse caso, o arcabouço permite que o módulo forneça uma lista de componentes *Java Swing* e os coloca na barra de botões e outras ferramentas na interface de atividade do iMA. Uma classe adicional para essa interface gráfica deve ser criada e instanciada no construtor do módulo.

As informações às quais o módulo de atividade tem acesso são a lista de atividades sendo realizadas pelo usuário e às funcionalidades de gravação e leitura de arquivos. Assim, o módulo aplica sua funcionalidade sobre a atividade que o usuário deseja e é capaz de armazenar os resultados de suas operações em arquivos. Apesar do arcabouço fornecer acesso a arquivos, os módulos podem ter outras funções não necessariamente relacionadas ao armazenamento.

Exemplos de módulos de atividade, já citados anteriormente, são: (i) a avaliação automática, que recebe a atividade, seleciona seu estado esperado e aplica um algoritmo específico de domínio para compará-lo ao estado atual, devolvendo um valor numérico; (ii) a exportação para imagens, em que um algoritmo (também dependente do domínio) recebe os objetos da atividade e cria um arquivo equivalente à interface com o usuário; ou (iii) a criação de objetos de aprendizagem, na qual o módulo recebe a atividade, demanda informações adicionais do usuário-autor relacionadas às especificações padrão e cria um pacote de arquivos; por fim, (iv) a definição do comportamento de um tutor inteligente associado à atividade, que adiciona as anotações desse comportamento ao arquivo da própria atividade. Desses exemplos, apenas a avaliação automática foi implementada na versão atual do arcabouço, sendo os outros sugestões para trabalhos futuros.

Em resumo, para implementar um módulo de atividade basta: (i) definir suas funcionalidades, (ii) considerar as informações disponíveis pelo arcabouço; e (iii) criar a interface do usuário para ser colocada na barra de ferramentas da interface de atividade do iMA. A seguir é descrito o outro tipo de módulo definido pelo arcabouço, o módulo de operações.

4.5.2 Módulos de Operações

De maneira similar aos módulos de atividade, os módulos de operações também são *plug-ins* e devem cumprir uma interface requerida pelo arcabouço para usufruir de métodos fornecidos. A diferença é que nesse caso, em vez de acessar diretamente as informações das atividades para realizar suas tarefas, os módulos de operação são apenas avisados quando uma ação de domínio ocorre, realizando então sua tarefa.

Além disso, outra diferença significativa é que os módulos de operação podem guardar dados específicos de cada atividade. Por exemplo, a funcionalidade desfazer-refazer exige que sejam armazenadas pilhas de ações de domínio específicas das atividades. Isso faz com que esses módulos tenham que ser gerenciados em conjunto com as atividades durante a utilização dos iMA.

Os módulos de operação têm uma arquitetura mais complexa para lidar com esses requisitos. A interface requerida pelo arcabouço inclui métodos para inicialização dos módulos, atualização de dados carregados de arquivos e para conversão de seu conteúdo para texto. Esses métodos são necessários para o funcionamento de módulos que armazenam dados e estes são guardados em arquivos. O gerenciamento e associação dos módulos às atividades é feito internamente pelo arcabouço. Além desses métodos, os módulos podem também fornecer as interfaces com o usuário,

para serem incluídas na barra de ferramentas do iMA.

As informações fornecidas pelo arcabouço aos módulos de operação são as ações de domínio e os objetos de domínio do estado atual da atividade. Sempre que uma ação é executada ou quando o estado da atividade é modificado, eles atualizam seus observadores, como os módulos desse tipo. Assim, a arquitetura interna de um módulo deve conter um Observador (*Observer*) de ações ou objetos de domínio, um Construtor (*Builder*) para construí-lo usando dados armazenados e inicializá-lo, ambos são padrões de projeto (Gamma *et al.*, 1994).

Exemplos de módulos de operações implementados na versão atual do arcabouço são o desfazer-refazer, o histórico de ações e a lista de objetos. O desfazer-refazer armazena uma pilha de ações para serem desfeitas e uma pilha para serem refeitas, atualizando sempre que uma ação é feita, desfeita ou refeita. O histórico de ações é similar, mas guarda apenas uma lista das ações realizadas. A lista de objetos, em vez de observar ações acompanha os objetos de domínio, e sempre que o estado de atividade tiver alterado seus objetos, essa lista se atualiza.

Exemplos de possíveis módulos de operações incluem *scripts*, animações e ativadores de ações de ajuda de um tutor inteligente. Os *scripts* nada mais são do que listas de ações que são executadas em conjunto, como uma adição ao módulo de histórico existente. As animações podem ser implementadas como *scripts* que possuem um temporizador para executar cada ação. E um ativador de ação de tutoria pode perceber sempre que uma ação for realizada e caso seja uma ação não esperada pelo professor, o tutor pode intervir e avisar o aluno.

Por fim, o processo para implementar esse tipo de módulo pode ser resumido da seguinte maneira: (i) definir as funcionalidades desejadas; (ii) implementar os métodos da interface exigida pelo arcabouço; (iii) criar as operações sobre ações ou objetos de domínio; e (iv) fornecer a interface com o usuário para ser adicionada à interface principal do iMA.

Assim se encerra a apresentação da LPS para iMA desenvolvida nesta pesquisa de mestrado. Foram abordadas todas as etapas de desenvolvimento e os resultados obtidos com relação à modelagem do domínio, à implementação do núcleo sob a forma do arcabouço de aplicação e os métodos sugeridos de utilização e evolução da LPS. O próximo capítulo apresenta os resultados obtidos relacionados à aplicação da LPS para o desenvolvimento de iMA.

Capítulo 5

Resultados e Discussões

Após serem apresentados os métodos e os resultados de desenvolvimento da LPS para iMA, este capítulo relata as avaliações das contribuições que este trabalho oferece à criação de iMA. Esses estudos consistem em entregar a LPS para programadores criarem seus aplicativos, analisando os processos de desenvolvimento subsequentes e comparando-os com os realizados sem o uso da LPS. Com os resultados das avaliações é possível verificar como o objetivo geral desta pesquisa foi alcançado, como apresentado na seção 1.3. O capítulo é dividido da seguinte maneira: em primeiro lugar é apresentada a prova de conceito de utilização da LPS, o novo iMA *iTangran*; e em seguida há o relatório do estudo de caso realizado considerando a refatoração do iMA existente *iVProg*.

Antes de apresentar os resultados vale ressaltar que a principal contribuição deste trabalho de mestrado é sobre a análise e o aprimoramento dos processos de desenvolvimento de iMA. Assim, apesar dos resultados de implementação utilizando a LPS não serem aplicativos em versões estáveis, as informações coletadas são suficientes para avaliar a influência da LPS sobre os desenvolvimentos na direção de seus objetivos.

5.1 Prova de Conceito - *iTangran*

A prova de conceito realizada neste trabalho teve o objetivo de verificar a possibilidade de utilizar a LPS criada para o desenvolvimento de um novo iMA, conforme o método descrito em 3.3.1. Dessa forma, uma versão em estágio avançado da LPS foi entregue a desenvolvedores em conjunto com uma série de requisitos que descreviam o aplicativo a ser criado, chamado *iTangran*. Ao longo do desenvolvimento, o autor intervia minimamente resolvendo dúvidas, corrigindo decisões realizadas com relação ao definido na LPS e eventualmente resolvendo os problemas do arcabouço encontrados para as próximas versões.

Os três desenvolvedores eram alunos de pós-graduação da Escola Politécnica da USP que cursavam uma disciplina do programa de pós-graduação em Ciência da Computação do IME-USP ministrada pelo orientador deste projeto de mestrado. Todos trabalham como professores de ensino superior em cursos relacionados à computação ou sistemas de informação e possuem interesses na área de pesquisa em Informática na Educação, particularmente no desenvolvimento de aplicativos educacionais. Além disso, possuem experiência profissional em desenvolvimento.

No início da disciplina os requisitos do iMA desejado, posteriormente batizado de *iTangran*, foram definidos pelo professor em conjunto com os programadores. Logo em seguida, foi disponibilizada a eles uma versão da LPS, com o arcabouço de aplicação, o manual de utilização para a

criação de iMA e um vídeo tutorial. O desenvolvimento era realizado em parte do período de aula da disciplina e, mais próximo do final, também em horários extra classe, quando a maior parte do trabalho foi realizado.

Ao longo da criação da prova de conceito várias considerações sobre a LPS foram feitas, gerando pequenas alterações na estrutura interna do arcabouço e em seu modo de utilização. Elas eram feitas sempre que uma dúvida maior ou problema no código surgia. As alterações foram realizadas principalmente para aumentar a clareza e a facilidade do processo sugerido no manual. A próxima subseção apresenta o resultado desse desenvolvimento.

5.1.1 Descrição do *iTangran*

Tangran é um brinquedo chinês milenar em formato de quebra-cabeça de sete peças: cinco triângulos, um quadrado e um paralelogramo, com as quais uma pessoa pode criar muitas formas. Esse jogo pode ser usado para ensinar alunos de matemática e outras disciplinas, conceitos sobre forma, posição, figuras compostas, entre outros (Pedrosa e dos Santos, 2004). Assim, ele constitui um domínio simples e apropriado para ser desenvolvido em um iMA.

Dessa forma, os requisitos funcionais específicos de domínio para o *iTangran* foram: (i) mostrar as peças do quebra-cabeça e (ii) permitir que o usuário as movimente com o *mouse* para formar as figuras. Isso torna o *iTangran* um iMA de simulação de um ambiente real do brinquedo, permitindo que o usuário brinque com as vantagens e desvantagens do computador. Não é necessário guardar as peças e a tarefa de organizá-las pode ser feita com um clique, ou então é possível permitir ao aprendiz usar a cor que desejar em qualquer momento para formar desenhos mais significativos. Porém, não há o toque, a sensação e a visualização espacial das peças reais.

Outra vantagem de usar o *iTangran* pode ser a criação de atividades e a avaliação automática das soluções dos alunos. Assim, um professor pode criar uma figura e pedir que os alunos tentem descobrir como fazer uma igual, entre outras atividades didáticas. O aplicativo pode automaticamente comparar a imagem do aluno com a previamente definida pelo professor de forma a informar o aluno se sua solução é aquela esperada ou não, de maneira mais interativa do que um livro com imagens montadas de Tangran.

A figura 5.1 mostra a tela principal do *iTangran*. Nessa imagem podemos ver a barra de botões das funcionalidades independentes de domínio, da esquerda para a direita: (i) autoria de atividades; (ii) nova aba; (iii) fechar aba; (iv) abrir arquivo; (v) gravar arquivo; (vi) histórico de ações; (vii) desfazer; (viii) refazer; (ix) avaliação automática; e (x) lista de objetos. Na região central encontramos a área do Tangran, a interface específica de domínio com os objetos de domínio que formam o quebra-cabeça.

As funcionalidades de domínio criadas até o momento da redação desta dissertação são o movimento e a rotação das peças. Assim, pelo uso do *mouse* é possível selecionar e arrastar as formas geométricas e com o clique direito é possível girá-las em sentido horário. Essas duas ações de domínio permitem a criação de qualquer desenho usando as peças do Tangran. Com relação às funcionalidades exigidas e fornecidas pelo arcabouço, ainda estão sendo implementadas as de autoria de atividade e de avaliação automática, os botões destinados a essas ações estão desativado.

O código do *iTangran* é formado por 13 classes, somando cerca de 700 linhas de código. As classes usadas e a relação delas com as fornecidas pelo arcabouço são mostradas no diagrama de classes da figura 5.2, na qual as classes do arcabouço estão situadas acima da linha tracejada, e as específicas

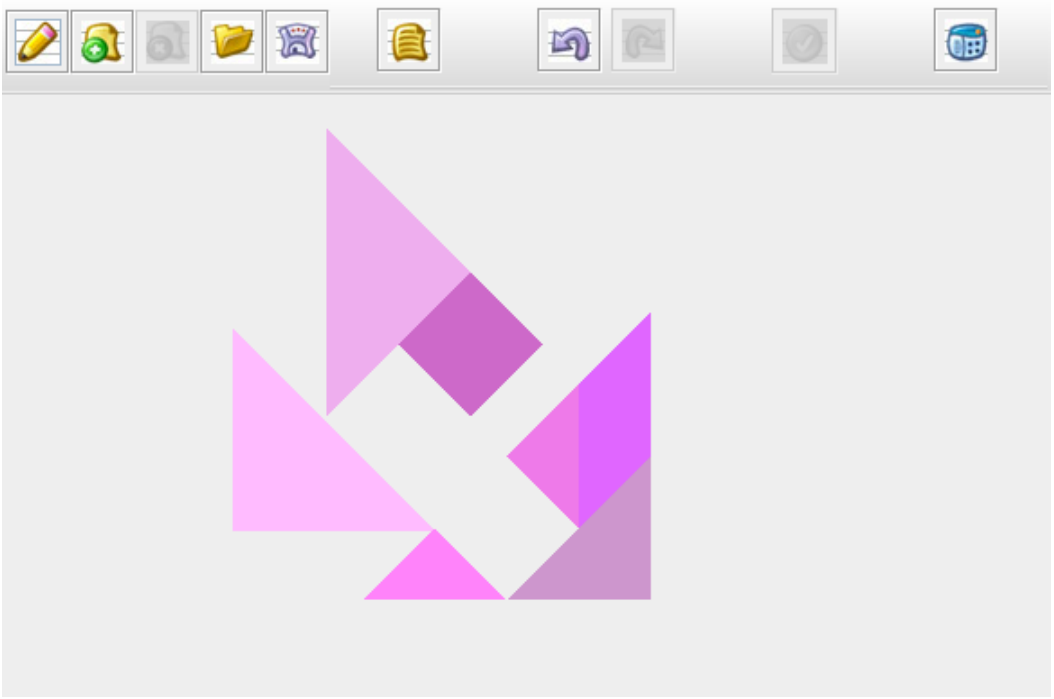

Figura 5.1: *Interface gráfica de atividade do iTangran com algumas peças movimentadas.*

abaixo. Foram usadas duas classes para as ações de domínio: "MoverForma" e "RotarForma" e oito classes para os objetos, sendo três de descrição ("Forma", "Cor" e "Direção") e cinco para os objetos em si (o losango, o paralelogramo e os três tipos de triângulo). Além dessas, foram criadas as classes únicas de interface com o usuário "TangranGUI", conversão "TangranConverter" e de modelo "TangranModel".

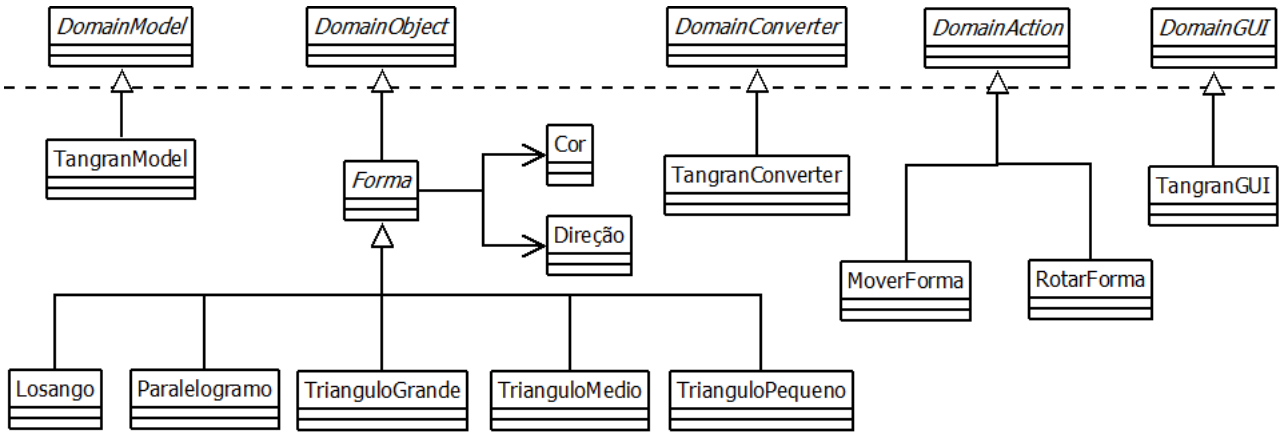

Figura 5.2: *Diagrama de classes do iTangran.*

O *iTangran* reflete o intuito da LPS para o desenvolvimento de iMA, a concentração dos esforços apenas em tarefas de modelagem e implementação de funcionalidades específicas de domínio. Assim, os programadores precisaram apenas criar as classes de domínio e seguiram a descrição do manual de utilização. O código do *iTangran*, assim como informações sobre as entrevistas conduzidas com os programadores, está disponível na página deste projeto (Dalmon e Brandão, 2012). A próxima subseção apresenta algumas discussões sobre o novo iMA desenvolvido.

5.1.2 Discussão

A prova de conceito *iTangran* mostra que é possível a criação de iMA baseando o desenvolvimento na LPS proposta nesta pesquisa. Aqui são analisados os resultados obtidos com relação ao código fonte e o processo executado. As informações discutidas aqui foram obtidas durante as aulas em que ocorreu a implementação e em entrevistas informais pouco antes do final da disciplina.

As funcionalidades implementadas satisfazem os requisitos de domínio, pois da forma em que foi modelado permite a criação de atividades interativas para alunos. Porém, o desenvolvimento do aplicativo continua com a criação das funcionalidades de autoria de atividades e avaliação automática. Com relação ao código, a partir de uma análise informal é possível afirmar que sua qualidade é satisfatória, devido a boas escolhas de nome e ao seu reduzido tamanho. Com relação à estrutura proposta pela LPS, houve um diferença na utilização da classe "TangranModel", que herda de "DomainModel". O processo da LPS (manual do arcabouço) descreve que a classe cujas operações são chamadas na execução das ações de domínio deve herdar de "DomainModel", mas no caso do *iTangran*, isso é feito diretamente pelos objetos de domínio. Isso não traz problemas para o funcionamento do sistema e mostra que a LPS pode ser utilizado de maneiras diversa daquela esperada, porém, pode influenciar a manutenção do sistema no futuro.

Os programadores desenvolveram o *iTangran* com a divisão de tarefas em: um programador, um auxiliar e um projetista, basicamente. Seus relatos apresentam um sentimento satisfatório com relação ao processo realizado e também sobre o arcabouço da LPS. Os principais obstáculos enfrentados foram com relação às especificidades da linguagem *Java* e do arcabouço gráfico *Swing* além da falta de conhecimento sobre padrões de projeto. A LPS gerou dúvidas apenas em questões pontuais, como na implementação das ações de domínio, sobre como decidir quais operações do usuário seriam consideradas ações de domínio. A decepção com a LPS foi a falta de algumas funcionalidades independentes de domínio implementadas no momento do desenvolvimento, como a comunicação com SGC.

As opiniões dos programadores convergem para duas situações. Considerando o projeto em isolamento, a LPS foi um obstáculo, atrasando o desenvolvimento pelo tempo necessário para aprender a utilizá-lo. Ao considerar o futuro do aplicativo, a LPS é considerada muito importante e com uma influência positiva, principalmente na organização do código e na adição das funcionalidades independentes de domínio. Várias sugestões foram feitas para a melhoria do arcabouço e do manual de utilização da LPS, que estão sendo realizadas atualmente ou estão presentes nas propostas de trabalhos futuros.

Por fim, a prova de conceito foi satisfatória. Mesmo com as considerações de algumas funcionalidades ainda não implementadas, um novo iMA foi desenvolvido tendo como base a LPS criada neste trabalho. As principais contribuições ao processo de desenvolvimento foram com relação à organização do código e oferencimento de funcionalidades independentes de domínio, o que, em conjunto com a definição das tarefas passo a passo a serem realizadas, são exatamente as contribuições almejadas pela LPS: promover reúso de código, arquitetura e processo. A próxima seção apresenta o estudo de caso que analisa, com maior profundidade, o processo de desenvolvimento do iMA *iVProg*.

5.2 Estudo de Caso - *iVProg*

O estudo de caso da refatoração do *iVProg* teve como objetivo investigar os efeitos da adoção da LPS de maneira a avaliar se houve aprimoramento do desenvolvimento em comparação ao desenvolvimento da versão anterior e à situação sem o seu uso. Assim, esse método foi aplicado para fornecer informações de forma a verificar se este trabalho atendeu aos objetivos firmados no início.

O caso estudado durou cerca de dez meses, dos quais houve cinco meses de desenvolvimento sem o uso da LPS e outros quatro após sua adoção, com um período sem desenvolvimento entre eles. Um programador foi o responsável pela tarefa ao longo de todo o período. As entrevistas ocorreram uma semana antes de iniciar o uso da LPS e próximo ao fim dos quatro meses de desenvolvimento. Os códigos analisados foram obtidos antes do início do uso da LPS e ao final do período analisado. Além do protocolo do estudo de caso, as informações coletadas ainda sem tratamento estão disponíveis na página do projeto (Dalmon e Brandão, 2012). O objetivo desse estudo de caso é levantar indícios do aprimoramento do processo de desenvolvimento de iMA a partir da análise do ocorrido na refatoração do *iVProg*.

De acordo com o descrito nos métodos de planejamento e execução, na seção 3.3.3, esta seção relata os resultados obtidos com a coleta de dados e em seguida os analisa para tirar algumas conclusões. Na próxima seção o aplicativo é apresentado de forma sucinta, sendo em seguida descrito um relato do seu desenvolvimento anterior ao uso da LPS. Na terceira subseção descrevemos a adoção da LPS e o desenvolvimento que se seguiu. Por fim, uma análise dos dados coletados e algumas discussões sobre os resultados são feitas.

5.2.1 Descrição do *iVProg*

O *iVProg* é o iMA mais recentemente desenvolvido, sendo que o projeto de mestrado responsável por ele terminou em 2010 (Kamiya, 2010). É um aplicativo de programação visual, que permite ao usuário construir programas de computador de forma similar ao paradigma de linguagem estruturada (ou procedimental), mas sem a necessidade de escrever código, apenas usando o *mouse* para selecionar e arrastar os componentes de código, como blocos "if-else" e "while". Esse tipo de aplicativo educacional possui um objetivo principal de reduzir a preocupação dos alunos de Introdução à Programação em detalhes das linguagens e permitir a eles focarem apenas nos algoritmos e soluções de problemas (Kamiya e Brandão, 2009).

Dessa forma as principais funcionalidades que o *iVProg* possui são: (i) o oferecimento de comandos para algoritmos como aqueles disponíveis em linguagens estruturadas: decisão, controle de fluxo, vetores, matrizes e outros; (ii) a execução dos comandos de maneira similar a um ambiente de desenvolvimento; e (iii) um compilador de linguagem procedimental, além das funcionalidades comuns aos iMA, como autoria de atividade e comunicação. O *iVProg* é o único iMA que ainda não possui a funcionalidade de avaliação automática, que está em desenvolvimento para assegurar o atendimento desse requisito da família.

Uma circunstância particular da primeira versão do *iVProg* é o fato dele ser o único iMA desenvolvido a partir de um outro aplicativo. Sua criação foi iniciada com a simplificação e remoção de código desnecessário para os requisitos considerados do sistema de código aberto Alice (Carnegie Mellon University, 2012). Alice é um aplicativo consideravelmente maior do que os iMA e oference muitas funcionalidades para o aprendizado de temas relacionados a algoritmos e anima-

ções. Dessa forma, a primeira versão do *iVProg* é uma modificação do Alice apenas com os recursos básicos de linguagem estruturada as funcionalidades requisitos da família iMA. Ela tem atualmente cerca de 37.000 linhas de código, das quais mais da metade são legado do sistema anterior.

A figura 5.3 apresenta a interface de atividade do *iVProg* na versão desenvolvida antes da adoção da LPS. Os principais elementos dessa interface gráfica são: (i) menu e barra de botões na parte superior; (ii) menu de estruturas de código à esquerda; e (iii) a área do programa sendo desenvolvido ao centro e direita. O menu e a barra de botões incluem funcionalidades independentes de domínio, como nova atividade, desfazer e refazer. Há também operações específicas do domínio, como o botão "Rode" que executa o programa sendo desenvolvido. O menu de estruturas de código dá acesso ao usuário aos elementos que podem fazer parte do programa, como as variáveis, operações matemáticas, estruturas de decisão e controle de fluxo, entre outras. Para inserir um desses elementos no programa basta arrastá-lo com o *mouse* para a região do programa. Por fim, a área do programa contém os elementos criados e possui aparência análoga a dos códigos de linguagem de programação procedimental, com a adição de bordas e cores para demarcar os comandos.

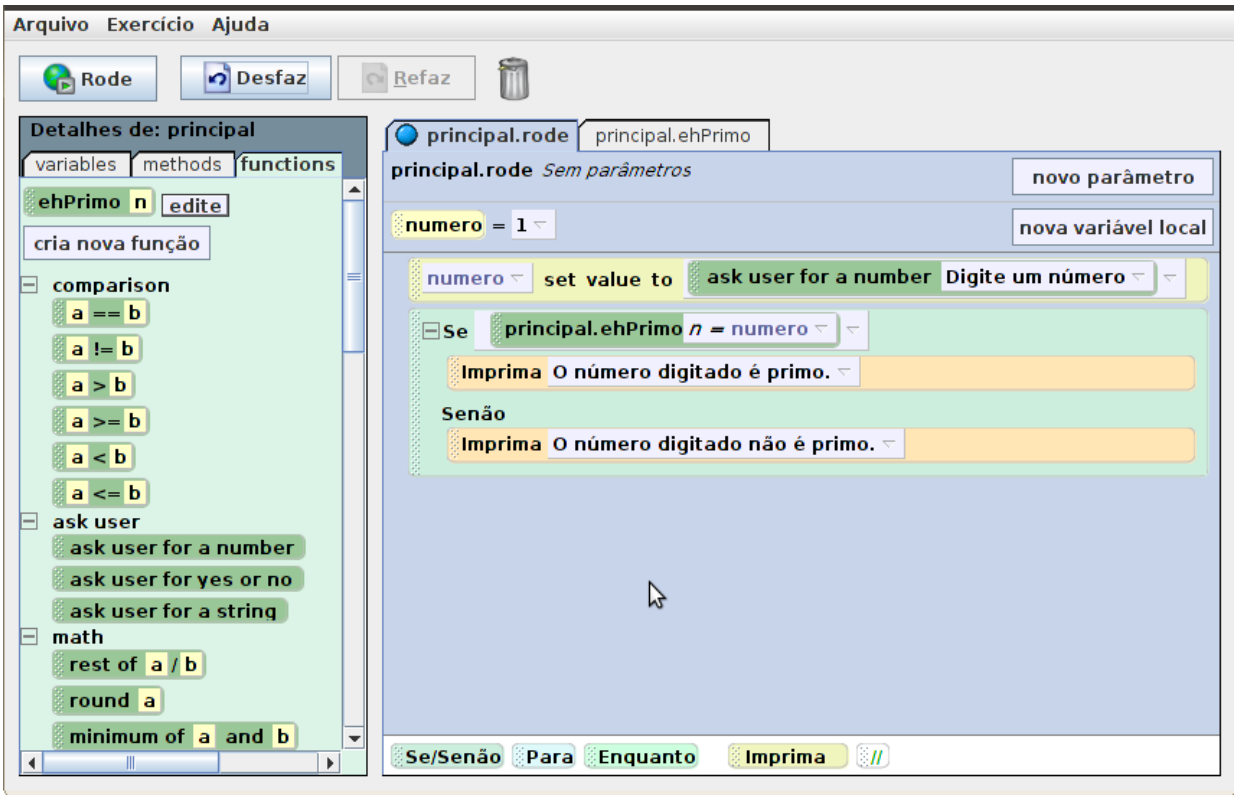

Figura 5.3: *Interface gráfica de atividade do iVProg.*

No exemplo da figura, podemos identificar na área de código duas abas, a primeira rotulada "principal.rode" e na segunda uma subrotina chamada "principal.ehPrimo". No código criado, temos uma variável que recebe o valor digitado pelo usuário e em seguida uma estrutura de decisão que usa o valor da subrotina para definir qual mensagem será exibida. Todos os elementos desse código foram criados a partir do menu de estruturas, do qual o usuário seleciona a arrasta para a área de código para formar seu programa.

Apesar de possuir um grande potencial como ferramenta educacional, como qualquer aplicativo, o *iVProg* pode ser continuamente expandido ou aprimorado, com mudanças de arquitetura ou adição de funcionalidades. A seguir é descrito como essa tarefa era realizada antes da adoção da LPS como método de desenvolvimento.

5.2.2 Desenvolvimento anterior à adoção da LPS

O desenvolvimento do *iVProg* sofreu um hiato após o término do projeto de mestrado que o originou, de forma que o novo programador começou a trabalhar em seu código na metade de 2011. O período analisado pelo estudo de caso inicia nessa data, caracterizando o caso estudado como o das tarefas de desenvolvimento do *iVProg* realizadas pelo novo programador apenas.

Para descrever como foi o desenvolvimento do *iVProg* antes da adoção da LPS como método de desenvolvimento, são abordados três informações principais: (i) a experiência do programador; (ii) os requisitos de implementação; e (iii) o método de desenvolvimento utilizado. Após essa descrição, os resultados desse desenvolvimento são apresentados. Essa divisão tem o objetivo de facilitar a compreensão do relato. Os dados considerados e resumidos para a análise dos resultados do estudo de caso são explicitados na seção 5.2.4.

Experiência do Programador

O programador responsável pelo desenvolvimento do *iVProg* durante o estudo de caso foi um aluno em tempo integral do programa de pós-graduação em Ciência da Computação do IME-USP sob tutela do mesmo orientador deste projeto. Ele iniciou seu mestrado logo após sua graduação em Licenciatura em Matemática, pelo mesmo instituto. Sua experiência profissional formal inclui alguns meses exercendo a função de assistente de produtor em uma editora de livros didáticos e como professor de matemática de ensino médio.

Sua experiência com relação a desenvolvimento de aplicativos antes de iniciar o projeto é restrita a disciplinas cursadas durante sua graduação e mestrado. Durante a graduação assistiu a quatro disciplinas em que trabalhou com linguagem procedimental, especificamente *C*, e uma sobre bancos de dados. Na última disciplina havia conceitos de qualidade de código relacionado a sua legibilidade. O trecho da entrevista destacado abaixo representa a descrição de sua experiência.

> Danilo - Qual a sua formação específica com relação a desenvolvimento de software?
>
> Programador - Em relação a desenvolvimento de software, zero. Eu fiz algumas disciplinas do bacharelado em ciência da computação aqui do IME. MAC110 que é Introdução à Computação, MAC122 que é Introdução a Algoritmos, MAC323 que é Estruturas de Dados, fiz Banco de Dados também, acho que só.

Pouco antes de receber o código, sabendo que trabalharia com a linguagem *Java*, começou a estudar de forma independente com livros e páginas *Web*. Logo em seguida cursou uma disciplina da pós-graduação sobre programação orientada a objetos, na qual teve mais contato com projeto orientado a objetos, qualidade de código, métricas, padrões de projeto e refatoração, conceitos importantes para o desenvolvimento de código de alta qualidade (Fowler *et al.*, 1999).

Em geral, o programador considera que sua formação em matemática o ajuda consideravelmente nas tarefas de programação, pelo caráter de resolução de problemas e elaboração de algoritmos.

Também considera que sua formação inicial em programação não era suficiente para realizar o projeto, mas que após as disciplinas de pós-graduação cursadas passou a se sentir um programador mais adequado ao projeto, mesmo sentindo falta de treinamento em métodos ágeis e trabalho em equipe.

Requisitos de Implementação

Ao retomar o desenvolvimento do *iVProg*, a meta do coordenador do projeto era reduzir alguns problemas encontrados durante seu uso e em seguida adicionar algumas funcionalidades. O principal problema do aplicativo era a utilização de processamento desnecessário do computador, enquanto que a primeira funcionalidade sob responsabilidade do programador era a opção de controlar os objetos com o *mouse* de maneira alternativa ao "arrastar e soltar", usando "clique-pega, clique-cola".

Ambas tarefas exigiam um profundo conhecimento do funcionamento interno do *iVProg*. Isso ocorreu pelo fato de possuir grande parte de código legado do aplicativo Alice e as funcionalidades em questão estarem fortemente relacionadas à estrutura interna do código. O alto consumo de memória estava associado principalmente aos vários processos (*threads*) criados durante a execução do *iVProg*. Assim, a tarefa do programador, quando iniciou o desenvolvimento, era deixar o aplicativo com apenas um processo, ou seja, eliminar a necessidade de *threads*. O trecho da entrevista abaixo exemplifica como foram entregues as atividades ao programador

> Programador - Bom, basicamente, até agora ele [o coordenador] deu [sic] o código, falou o que precisava ser feito. Eu tentei [sic] a princípio ele gostaria que o antigo *iVProg* fosse melhorado, que fossem excluídas as *threads*, que fosse melhorado o desempenho do aplicativo. No geral o aplicativo funciona bem. O problema dele são as *threads*.

Parte dos processos criados pelo *iVProg* era acoplada aos mecanismos de controle de interface gráfica com o usuário, especificamente com tratamento do arrastar e soltar dos objetos de programação visual. Isso fortalece a relação dos dois requisitos principais. A opção de poder controlar os objetos com "clique-pega, clique-cola" é usada em vários iMA e possui o objetivo de deixar o usuário escolher qual maneira prefere (Inkpen, 2001).

Método de Desenvolvimento

O método de desenvolvimento utilizado não era sistemático, assim, uma descrição mais informal será feita. Primeiro os requisitos eram informados ao programador em reuniões de projeto com o coordenador, que eram anotados. Além dos pedidos, o coordenador fornecia sugestões para as tarefas com relação às regiões do código do sistema que deveria ser modificada ou utilizada, para as possíveis estruturas da solução a ser aplicada e eventualmente para os problemas que poderiam ser encontrados na realização das tarefas.

O programador, de posse apenas do código do *iVProg*, uma vez que não havia nenhum tipo de documentação, passava a estudá-lo para planejar como implementar a solução para o requisito. Assim, o processo seguia iterativamente, de maneira que quando o programador encontrava um obstáculo cuja solução não estava disponível na *Web*, recorria ao coordenador para sanar suas dúvidas. Reuniões semanais entre os dois eram feitas para acompanhar o desenvolvimento. O trecho retirado da entrevista apresentado abaixo mostra como era o trabalho do programador.

> Danilo - E tinha documentação?
>
> Programador - Nada. Assim, você podia [sic] achar comentários no meio do código, mas era isso. De documentação era só isso.
>
> Danilo - Como eram esses comentários?
>
> Programador - Na verdade tinha assim: "para funcionar em Java4", "não funciona em tal coisa", [sic] comentários desse gênero.
>
> Danilo - Mas esses eram comentários do *iVProg*, nada do *Alice*?
>
> Programador - Isso. Do *Alice* as únicas informações que tinha [sic] eram em relação a direitos autorais. Que para distribuir precisava ter o nome do projeto do [sic], o nome do *Alice*.
>
> Danilo - E não tinha mais nada?
>
> Programador - De documentação nada.
>
> Danilo - Nada de projeto, de diagrama de classes, nada?
>
> Programador - Absolutamente nada. Para entender o código era na base [sic] da leitura mesmo.

Para analisar ou estudar o código foram usados dois procedimentos, com o objetivo de entender o funcionamento do aplicativo em geral, o programador lia o código partindo da classe principal ou de outra classe central e passava às outras classes que as referenciavam. No caso de problemas mais específicos, a funcionalidade de busca de termos dos ambientes de desenvolvimento era usada com o objetivo de encontrar a região do código responsável pelo termo para posterior estudo.

Resultados do Desenvolvimento

Ao final dos quatro meses de desenvolvimento do *iVProg* pelo programador, o resultado considerando código fonte criado pertencente a uma versão estável do aplicativo foi a correção de um defeito provocado com o uso de diferentes máquinas virtuais *Java* ou sistemas operacionais. Os demais objetivos não foram alcançados, ou seja, a solução dos problemas e requisitos não puderam ser implementadas nesse período. O trecho retirado da entrevista mostra o relato do programador.

> Danilo - Mas assim, o que você conseguiu fazer com o código?
>
> Programador - Eu encontrei um bug, porque ele lançava uma exceção e mais ou menos onde estava o problema [sic]. Eu corrigi esse bug, deixa eu ver o que mais... Eu mudei a parte dos parâmetros [sic] que o professor pediu. E foi só. Eu não fiz grandes coisas com aquele código.

Do tempo de desenvolvimento, de acordo com o programador, cerca de 90% foi utilizado com o estudo do código existente, em vez de implementação. Isso foi provocado pela dificuldade de leitura e compreensão do código, consequência de sua má qualidade. Exemplos de elementos com má qualidade relatados incluem nomes mal escolhidos, classes e métodos muito grandes, métodos com muitos argumentos, inclusive argumentos sem um significado claro, comentários apenas para anotações e não para descrição do código. O trecho abaixo mostra o relato do programador com relação a esse assunto.

> Danilo - Então o que te impedia de produzir mais, de programar mais?
>
> Programador - Acho que experiência. Se eu tivesse mais experiência, sei lá [sic], ou então se o código do *iVProg* fosse mais claro, estivesse documentado eu acho que iria perder muito menos tempo. Porque, assim, 90% do tempo era perdido para compreender, para entender, para buscar relação [sic] dentro do próprio código. Isso tomava muito tempo.

Apesar disso, a produtividade percebida pelo programador aumentou com o tempo de desenvolvimento, pelo fato dele entender gradualmente o código legado. Ele relata que se continuasse a trabalhar nessa versão do *iVProg* seria possível com o tempo resolver os problemas e adicionar as funcionalidades requeridas. Porém, a satisfação em programar nesse período foi baixa, de forma que a motivação descresceu com o tempo. Foi nesse momento em que o arcabouço núcleo da LPS estava em estágio avançado de desenvolvimento e foi decidido que uma nova versão do *iVProg* seria criada.

5.2.3 Desenvolvimento após a adoção da LPS

Após um período de discussão e levantamento de vantagens e desvatagens sobre iniciar a desenvolver uma nova versão do *iVProg*, o trabalho de manutenção da primeira versão do aplicativo foi descontinuado. Foram relevantes para essa decisão a satisfação e a produtividade do programador com a versão corrente, a dificuldade com que ela receberia manutenção e evolução no futuro e o fato da LPS para iMA estar em desenvolvimento. Assim, em meados do segundo semestre de 2011 a criação da nova versão do *iVProg* teve início.

O planejamento das tarefas de desenvolvimento considerou a arquitetura proposta pela LPS e os requisitos da versão existente do *iVProg*. Ele foi dividido em três etapas principais: (i) criação do núcleo do aplicativo, o que seriam os objetos de domínio; (ii) desenvolvimento da interface gráfica com o usuário; e (iii) integração com o arcabouço. As duas primeiras etapas foram realizadas sem a consulta direta ao código do arcabouço, apenas à sua arquitetura, uma vez que sua versão com todas as funcionalidades não estava pronta.

Novo Núcleo e Interface Gráfica

Dessa forma, o desenvolvimento da nova versão do *iVProg* começou com a criação do seu núcleo. O núcleo foi definido como a funcionalidade de representar o código criado pelo usuário e ser capaz de "executá-lo" de maneira similar a um compilador. Pelo fato do programador não ter conhecimentos na área de compiladores, um grande esforço de modelagem foi despendido. O curso de programação orientada a objetos e a comunicação com colegas programadores o ajudaram nessa tarefa. Após alguns meses, uma versão do núcleo foi criada utilizando padrões de projeto e desenvolvimento dirigido por testes (Beck, 2002), com o objetivo de manter a qualidade do código alta. Essa versão não possuía apenas a capacidade de executar funções recursivas. Um tempo equivalente ao da implementação de todas as outras funcionalidades foi usado para a recursividade. Ao fim, foram usados conceitos da teoria de compiladores sem o conhecimento explícito do programador. O trecho retirado da entrevista mostra um relato sobre o desenvolvimento do núcleo.

> Danilo - Como foi o método usado para fazer, para passar do *iVProg* velho para o novo?

> Programador - A gente tinha uma boa ideia do que precisava, do que o sistema tinha, do que a gente queria [sic] novo, aí a disciplina de Programação Orientada a Objetos, que a gente fez, deu uma boa ideia do que precisava. Do que precisava não [sic], de como a gente poderia fazer. A partir daí a gente começou a pensar nas funcionalidades do núcleo. Usando o padrão *Composite*, que eu acho que ajudou bastante o trabalho, a visão. E a gente fez o núcleo, e agora eu to [sic] trabalhando na parte da GUI [*Graphical User Interface*].

Em seguida a inteface gráfica começou a ser criada. Para essa tarefa o programador tinha a experiência acumulada na criação do núcleo e no trabalho com a versão anterior do *iVProg*. Assim, partindo dos problemas enfrentados com o arrastar e soltar, o programador passou a estudar os elementos oferecidos pelo *Java Swing* (Oracle, 2012e). Após algumas semanas sem conseguir produzir as funcionalidades desejadas, a decisão de criar diretamente operações antes delegadas ao *Swing* fez com que a implementação das formas de manipulação de objetos "arrastar e soltar" e "clique-pega, clique-cola" se tornasse simples. Assim, em pouco mais de dois meses a interface gráfica da nova versão do *iVProg* estava com grande parte de suas funcionalidades implementadas. Esse desenvolvimento usou um projeto em camadas para organizar o código e, em comparação com o núcleo, foi feito com menos considerações a qualidade de código.

Essas duas etapas puderam ser realizadas sem a utilização direta do código do arcabouço por se basearem em definições da arquitetura proposta por ele. Assim, foi possível criar os objetos de domínio considerando apenas suas características específicas de domínio, deixando para implementar o exigido pelo código do arcabouço apenas no momento da integração. O mesmo ocorreu com a interface com o usuário, o programador tinha conhecimento que esse componente era esperado pelo arcabouço apenas com a restrição de que na integração houvesse a herança da classe "DomainGUI". O fato de poder criar código específico de domínio de considerando apenas elementos arquiteturais num primeiro momento facilitou o trabalho do programador, pois ao mesmo tempo que possuía liberdade, havia também um guia para seu trabalho.

> Danilo - E no final das contas, qual a relação do arcabouço com o método que você usou para fazer o novo *iVProg*?
>
> Programador - Acho que foi mais na separação em objetos bem... assim por classes [sic]. Por exemplo, eu poderia reunir várias coisas em uma única classe mas eu acho que fica mais claro dado que existem lá [sic] os objetos de domínio, eu acho que ajudou colocar esses papéis [...]. Ele ajudou também a achar alguns pontos que estavam bem fracos no meu projeto. Quando eu comecei a integração eu enxerguei outras coisas, como por exemplo eu precisava de identificação única para cada objeto que eu ia [sic] adicionar no painel de domínio...

Uma alteração conceitual importante ocorrida nessa etapa foi a mudança do que é considerado objeto de domínio. Os objetos de domínio deixaram de ser aqueles usados para modelar o núcleo e passaram a consistir nos componentes de interface gráfica manipulados pelo usuário. Isso pode ser feito pelo fato dos objetos do núcleo serem criados somente no momento de "execução do código" e não durante a criação do programa pelo usuário, cujas ações têm significado didático para o aplicativo.

> Programador - No final das contas o núcleo acabou sendo [sic], eu acabei enxergando o núcleo como só uma funcionalidade de um botão, que não vai estar [sic] relacionado diretamente com o arcabouço. Os objetos que eu acreditava que seriam objetos de domínio do arcabouço antes eram os objetos que eu estava criando no núcleo mas, não. Não é.

Integração com o Arcabouço

Para realizar a integração do código do núcleo e da interface gráfica com o do arcabouço, o manual de utilização foi considerado em detalhes. Isso exigiu que pequenas alterações fossem feitas para os códigos se tornarem compatíveis com o esperado pelo arcabouço. Assim, o passo a passo descrito no manual foi seguido para realizar a integração. O trecho retirado da entrevisa apresentado abaixo ilustra como foi iniciada a etapa de integração.

> Programador - Mas esse processo de integração é a que eu disse "agora eu vou juntar" [sic] entendeu? Porque todo o processo, apesar de ter o arcabouço em vista, lá na frente, eu nunca peguei nada na frente e coloquei para fazer [sic]. Eu fiz agora. Eu peguei o que eu tinha da GUI e coloquei o que eu tinha dali [sic] e mais funcionalidades. O comecinho assim [sic], em termos de tempo, quanto tempo levou para pegar meu código, juntar com o arcabouço, e ver a GUI instanciando, acho que foi coisa [sic] de uma hora e meia. Então acho que foi rápido. [...]
>
> Programador - Acho que está bom, está ótimo, principalmente porque tem o manual e o exemplo, acho que foi mais do que suficiente para saber o que precisava ser feito, como eu já tinha lido o manual inteiro, eu fui fazendo pelo passo a passo. [...] Aí a primeira coisa que eu fiz para começar a fazer essa integração foi fazer com que meus objetos de domínio estendessem a classe que eles precisavam estender...

Todas as classes exigidas no método de utilização do arcabouço foram criadas, seja diretamente, seja adaptando as classes provenientes das duas etapas anteriores. Assim, os objetos de domínio referentes a linhas de código, blocos "if-else" e "while", foram definidos. Quatro ações de domínio foram suficientes para as operações principais de criar e movimentar linhas e blocos de código, mas algumas outras funcionalidades ainda devem ser modeladas como ações de domínio. Além disso, a classe do modelo de domínio foi criada para realizar as operações. Como o *iVProg* é um projeto maior em comparação com o *iTangran*, outros desenvolvedores estão envolvidos, particularmente com a gravação e leitura de arquivos. Assim, para completar a integração, faltam as tarefas de implementação do conversor de domínio, da interface de autoria e da avaliação automática.

A figura 5.4 apresenta a interface do usuário da nova versão do *iVProg* criada com o auxílio da LPS, que possui uma estrutura de interface com o usuário similar à da versão anterior do *iVProg*, com a barra de botões, o menu de comandos e a área de código. Porém, nesse caso, a barra de botões é fornecida pelo arcabouço e inclui as funcionalidades independentes de domínio. Será nessa barra que serão incluídas também funcionalidades específicas de domínio como exportação do código criado para linguagens de programação, como *C* e *Java*. As outras regiões da interface não diferem significativamente da versão anterior.

A integração com o arcabouço que gerou a nova versão do *iVProg* durou cerca de duas semanas, sendo que houve três dias de dedicação exclusiva à tarefa. De acordo com seu relato, ela foi rea-

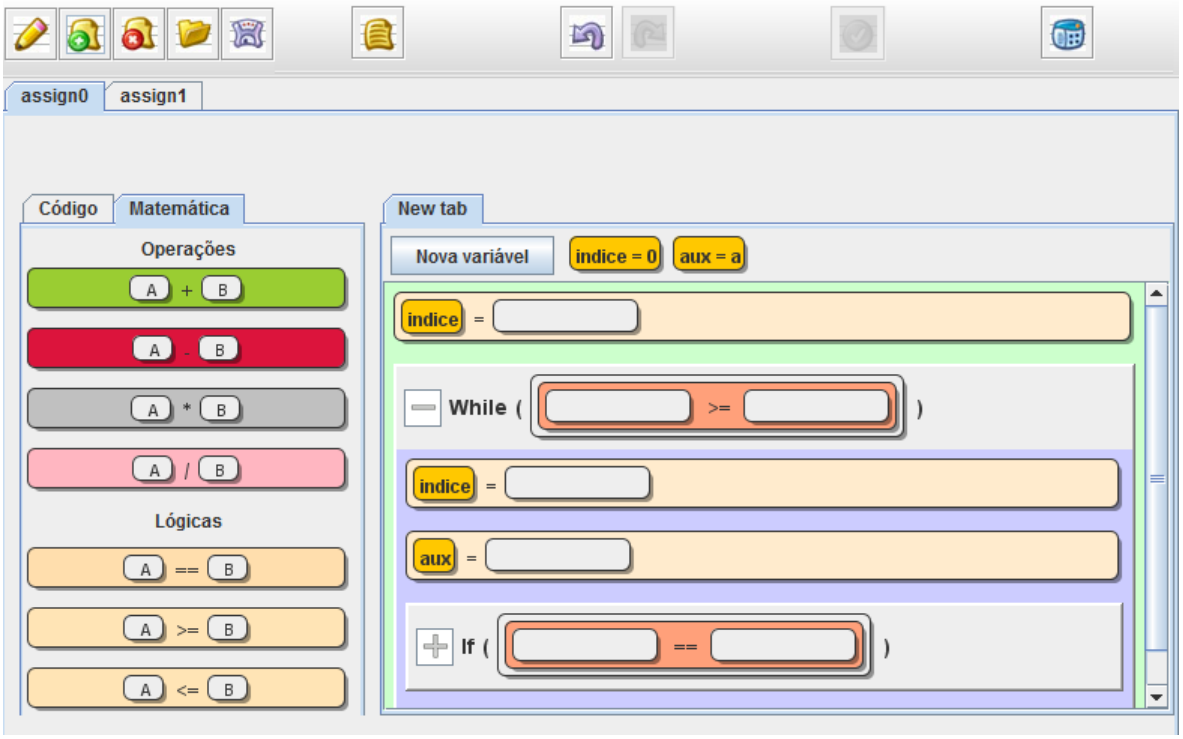

Figura 5.4: *Interface gráfica de atividade da nova versão do iVProg criada com o auxílio da LPS.*

lizada facilmente com apenas algumas pequenas dificuldades, resolvidas com a leitura do manual ou com a revisão do código. Além disso, foi satisfatório usar código de outro programador sem muitos problemas. Uma dificuldade importante para relatar foi a necessidade de modificar a classe "objeto de domínio" do arcabouço para permitir que seus objetos herdem simultaneamente dela e do componente de interface gráfica. Não foi usada herança múltipla, pois "objeto de domínio" anteriormente não herdava de nenhuma classe. Durante o desenvolvimento do arcabouço, não foi considerado que objetos de domínio pudessem ser componentes gráficos.

Impressões do Programador e do Coordenador

Para o programador, o manual de utilização foi essencial durante a etapa de integração, considerando esse recurso claro e sucinto. De maneira similar à experiência relatada com o *iTangran*, houve diversas sugestões de melhoria para o manual, como criar tutoriais em vídeo.

No caso do arcabouço, o programador destacou em seu relato algumas influências mais significativas. A arquitetura pré-definida proposta pela LPS foi essencial para a organização do código e do método de desenvolvimento. Sem a qual o programador acredita que a nova versão do *iVprog* poderia se encontrar rapidamente em situação similar àquela da versão anterior quando começou o desenvolvimento.

> Programador - Foi bom para melhorar o código, e achar falha [sic] no meu código. Na parte que eu fiz, eu encontrei um monte de problemas e eu dizia "nossa, é verdade tinha que ser assim", e foi importante também [sic]. Eu não iria conseguir fazer assim, se não tinha prestado atenção naquilo [sic]. Não que eu diria que é importante para prestar atenção naquilo, para o arcabouço, seria importante mesmo que se eu não estivesse

> usando o arcabouço.
>
> Programador - Acho que se não houvesse o arcabouço, o código ficaria mais parecido com o antigo [*iVprog*], eu acho que por conta da direção que o arcabouço dá, é bem diferente [sic].

Além disso, a flexibilidade permitida ao programador foi considerada alta, pelo método usado para criar a nova versão do aplicativo, principalmente na etapa da interface gráfica. Há inclusive a contribuição das várias funcionalidades independentes de domínio às quais o programador não precisa se preocupar, como o fato do arcabouço criar e gerenciar as atividades no domínio específico. Uma influência não planejada durante a criação do arcabouço relatada foi o aprimoramente do código criado anteriormente durante a integração, quando houve momentos em que o código das etapas anteriores foi ligeiramente modificado, aumentando sua qualidade, de acordo com o programador.

Em relato, o coordenador do projeto ressaltou o papel fundamental do arcabouço no desenvolvimento da nova versão do *iVprog*, fornecendo estruturas importantes para facilitar essa implementação e deixando as maiores dificuldades no domínio específico. Além disso, o método dividido em etapas, com versões protótipo feitas antes de considerar todos os requisitos foi importante para o andamento do processo.

> Danilo - Qual a relação da LPS/arcabouço com esse desenvolvimento?
>
> Coordenador - O fundamental foi o estudo sobre a estrutura do sistema relacionada ao código de programação (daí a adoção de classes que refletissem a estrutura da linguagem de programação desejada) e o modelo espiral de "desenho-implementação-teste". [...] não parece ter havido dificuldade em seguir o modelo do arcabouço. As dificuldades sempre concentraram-se no domínio específico, sendo as mais difíceis a estrutura de classes que permitisse implementação de todos os requisitos de programação (como funções recursivas e variáveis locais) e a linguagem de representação.
>
> Danilo - Qual a principal diferença para esse desenvolvimento em comparação ao *iVProg* anterior?
>
> Coordenador - Total, haja vista que agora é possível desenhar um novo sistema, limpo e eficiente. No anterior era sempre uma batalha para entender e melhor um código intrincado.

O desenvolvimento da nova versão do *iVprog* pode ser analisado e gerou as informações necessárias para a análise da influência do arcabouço e da LPS no processo. O projeto continua em seguida com a implementação das funcionalidades restantes e da adição de novas, como a avaliação automática das soluções dos alunos. A seguir a análise sistemática dos dados apresentados aqui é feita, concluindo o estudo de caso.

5.2.4 Análise dos Dados e Discussão

Conforme descrito na seção 3.3.3, a coleta e análise de dados foram categorizadas em quatro elementos: (i) o método de desenvolvimento; (ii) características do código fonte produzido; (iii) a percepção de produtividade durante o período analisado; e (iv) as impressões gerais do programador.

Nesta seção são apresentadas as análises em cada uma dessas categorias. Todos os dados analisados aqui estão disponíveis na página do projeto (Dalmon e Brandão, 2012).

Em seguida, é feita a triangulação dos dados, uma análise em conjunto das conclusões feitas considerando cada categoria. Essa análise, acompanhada de algumas discussões, apresenta a conclusão do estudo de caso e o resultado da avaliação da influência da LPS para o aprimoramento do desenvolvimento de iMA.

Método de Desenvolvimento

O método de desenvolvimento utilizado, conforme o relato do programador e do coordenador do projeto, pode ser dividido em duas partes: (i) influenciado pela arquitetura; e (ii) pelo código do arcabouço. Nas etapas de desenvolvimento do núcleo e da interface com o usuário o programador seguiu um método mais voltado à arquitetura do arcabouço e ao passar para a integração, ele adaptou seu método às especificidades do código e do passo a passo descrito no manual.

Mesmo que o desenvolvedor tenha mais liberdade para definir seu método antes da etapa de integração, há evidências da influência positiva da LPS nos relatos. Exemplos incluem a organização do código e a divisão das etapas, em que o próprio programador acredita ter sido bem guiado, mas indiretamente, pela LPS. Essas características são importantes para manter a qualidade do código alta e facilitar sua manutenção.

Durante a integração, a LPS tem uma influência maior no método usado, uma vez que o programador tem à disposição o processo descrito passo a passo. Assim, as evidências dos relatos apontam que esse método foi seguido estritamente até o estágio de desenvolvimento considerado (sem as funcionalidades ainda não implementadas). O método foi inclusive associado a verificações de defeitos no código antes não detectados. Isso pode ser visto também pela estrutura detalhada do código, descrito a seguir.

Código Fonte produzido

O código fonte da nova versão do *iVprog* foi analisado com métricas para prever a facilidade de realizar manutenção, da mesma maneira que o apresentado para a LPS na seção 4.5. O total de linhas de código atualmente se aproxima de 3.200. Os valores obtidos para as métricas analisadas são mostrados na tabela 5.1.

Tabela 5.1: *Resultado da análise do código da nova versão do iVprog.*

Métrica	Valor Médio	Valor Máximo
Métodos Ponderados por Classe	7,54	30,0
Profundidade da Árvore de Heranças	4,90	8,00
Número de Filhos	0,62	18,0
Acoplamento entre Classes	3,15	9,00
Falta de Coesão entre Métodos	0,34	1,00

Os valores encontrados para o código do *iVprog* são para a maioria das métricas ligeiramente maiores do que os encontrados para o arcabouço, com a exceção dos métodos ponderados por classe, cujo valor é um pouco menor. Isso indica que sua capacidade de receber manutenção também pode

ser avaliada entre boa e regular, comparável à do arcabouço. De maneira similar, o desenvolvimento futuro deve incluir esforços para melhorar os resultados dessa análise. Em comparação à primeira versão do *iVprog* apresentados na tabela 5.2, os resultados da tabela 5.1 são no máximo a metade dos valores obtidos, o que representa a dificuldade de realizar manutenção nessa outra versão do aplicativo.

Tabela 5.2: *Resultado da análise do código da versão anterior do iVprog.*

Métrica	Valor Médio	Valor Máximo
Métodos Ponderados por Classe	13,5	383,0
Profundidade da Árvore de Heranças	4,07	10,0
Número de Filhos	0,69	17,0
Acoplamento entre Classes	7,35	44,0
Falta de Coesão entre Métodos	0,19	1,57

Não é possível afirmar apenas com esses valores que a LPS influenciou o desenvolvimento de maneira a melhorar os resultados obtidos. Mas como os resultados foram próximos aos do arcabouço, pode-se estimar que desenvolvedores um pouco experientes, como é o caso do autor deste texto e do programador do *iVprog*, quando preocupados com qualidade de código, conseguem resultados razoáveis para facilitar a manutenção do aplicativo que desenvolvem.

Percepção de Produtividade

A produtividade do programador percebida por ele ao longo do desenvolvimento utilizando a LPS foi, em geral, significativamente mais alta do que enquanto trabalhava sobre a versão inicial do *iVprog*. Isso é evidenciado em vários trechos de seu depoimento.

Houve momentos, porém, em que a sua produtividade caiu, principalmente enquanto desenvolvia a funcionalidade de executar funções recursivas e no início da etapa da interface com o usuário. As dificuldades de modelagem provocadas pela falta de conhecimento teórico foram as causas da baixa produtividade da primeira situação. A falta de conhecimento da tecnologia *Java Swing* e sua complexidade foram as causas da segunda. Mesmo nessas situações, o programador acredita que sua produtividade era maior do que antes de adotar a LPS.

Hipoteticamente, podemos comparar sua produtividade com uma situação em que desenvolveria a nova versão do *iVprog* sem a base da LPS. Mesmo que para as mesmas funcionalidades desenvolvidas a produtividade fosse maior, dificilmente seria maior o suficiente para compensar o ganho de produtividade relacionado às funcionalidades independentes de domínio providas pela LPS. Mesmo assim, o programador acredita que sem o uso da LPS, não teria uma produtividade maior. Essa opinião contrasta com a dos desenvolvedores do *iTangran*. Isso pode ter sido provocado pelo desenvolvedor do *iVprog* ter maior contato com o desenvolvimento da LPS do que os da prova de conceito.

Impressão geral do Programador

A impressão geral que o programador tem do seu desenvolvimento utilizando a LPS para criar a nova versão do *iVprog* é satisfatória. Isso é evidenciado por trechos com opiniões positivas sobre

a LPS e o método proposto ao longo das entrevistas. Por exemplo, ao final do relato é citado que a LPS cumpre o que promete (servir de base e aprimorar o desenvolvimento de iMA) e ajudou bastante nesse caso, principalmente organizando e melhorando o código produzido.

Além disso, o programador tem várias críticas ao processo definido para o uso da LPS e também ao código produzido por ele. Acredita que o arcabouço deveria ter a responsabilidade de ser executado sem a adição de código dependente de domínio informando quais componentes são necessários, para guiar ativamente a implementação. Outra sugestão é com relação à sua estrutura interna, sobre o acesso das ações de domínio pela interface gráfica. Essas informações isoladamente não permitem tirar conclusões sobre a influência da LPS no desenvolvimento do *iVprog*, mas servem de argumento na análise geral dos resultados, apresentada a seguir.

Triangulação dos Dados e Discussão

Analisando os resultados obtidos no estudo de caso com relação às quatro categorias de dados, temos o seguinte. Em resumo, podemos dizer que o método utilizado foi basicamente aquele definido pela LPS, o código fonte produzido possui uma qualidade entre razoável e boa, a produtividade percebida para as funcionalidades implementadas foi equivalente a desenvolver um aplicativo desde o início sem restrições e a impressão geral do programador com a LPS foi satisfatória.

As informações relacionadas ao método utilizado permitem associar os resultados das outras categorias à aplicação da LPS no desenvolvimento. Assim, os valores das métricas obtidas na análise do código fonte e a percepção de produtividade são influenciados pelo uso da LPS. É preciso verificar qual a extensão dessa influência, pois podem existir outros fatores mais influentes. O relato do programador, no qual diz que o arcabouço oferece funcionalidades prontas ao aplicativo em desenvolvimento sem prejudicar a produtividade e que é ele o motivo de ter organizado o código daquela maneira, mostra indícios da extensão da influência da LPS. O código fonte, cuja estrutura é a sugerida pelo arcabouço, foi fortemente influenciado pela adoção da LPS. Além disso, a impressão geral do programador mostra que foi agradável desenvolver o *iVprog* com o uso da LPS, o que contrasta com a insatisfação das tarefas anteriores com o código legado.

Todas essas são evidências do aprimoramento do desenvolvimento do *iVprog* provocados pela LPS. Em resumo, o desenvolvimento pode ser considerado: (i) mais produtivo, pois a LPS oferece funcionalidades reutilizáveis; (ii) que produz código com melhor qualidade, pois o arcabouço e o processo definido ajudam a estruturar o código e remover defeitos; e (iii) deixa as tarefas menos desgastantes para o programador, pela qualidade do código e pelo guia descrito passo a passo. Essas são as principais contribuições da LPS para o aprimoramento do processo de desenvolvimento de iMA.

Essa análise responde à pergunta "A LPS aprimorou o desenvolvimento do *iVprog*?", mas não necessariamente atende ao objetivo geral desta pesquisa. Para poder responder à pergunta referente a qualquer iMA, é preciso analisar outros fatores de transferabilidade de resultados. Considerando as três principais contribuições provocadas pela adoção da LPS no desenvolvimento do *iVprog*, analisamos cada uma com relação ao caso da criação de outros iMA.

Primeiro, as funcionalidades independentes de domínio fornecidas pelo arcabouço não são alteradas no desenvolvimento de cada iMA, a menos que seja escolhido deliberadamente que o aplicativo não as utilizará. O código a ser reutilizado pode oferecer obstáculos para o desenvolvimento pelo tempo de aprendizado, o que pode reduzir o efeito positivo do reúso. Espera-se que esses obstáculos

sejam reduzidos com o aprimoramento contínuo do código e dos manuais de utilização, além de diminuir também com a quantidade de iMA e documentação criados. Assim, ao longo do tempo a quantidade de funcionalidades para serem reutilizadas aumentará e os obstáculos de aprendizagem serão reduzidos, permitindo que o desenvolvimento de qualquer iMA aproveite dessa contribuição.

Em segundo lugar, consideramos a qualidade do código produzido no desenvolvimento de diferentes iMA. Essa contribuição é sensível à experiência do programador, sendo que programadores mais experientes são menos influenciados. Assim, a LPS pode ter uma contribuição mais significativa quando os programadores têm menos experiência com desenvolvimento de software.

Por fim, a terceira contribuição é sobre a satisfação do desenvolvedor. No caso do *iVprog*, as tarefas se tornaram menos desgastantes em comparação ao desenvolvimento anterior usando código legado com má qualidade. Assim, os desenvolvimentos de iMA sem código legado não podem receber a contribuição dessa maneira. Porém, as instruções do método passo a passo para o desenvolvedor, principalmente com pouca experiência, podem servir de guia para o programador e reduzir a sensação de falta de planejamento.

Concluindo, a LPS desenvolvida neste trabalho de mestrado contribuiu para aprimorar o desenvolvimento do *iVprog* e pode contribuir no caso de outros iMA. O desenvolvimento de iMA em geral pode receber mais contribuições da LPS nos casos em que o aplicativo utiliza todas as funcionalidades fornecidas pela LPS e quando o programador possui pouca experiência em desenvolvimento de software.

5.2.5 Limitações e Ameaças à Validade

Os resultados apresentados neste capítulo estão sujeitos a limitações e ameaças à validade, que serão analisados aqui. Primeiro apresentamos as principais limitações dos estudos de validade realizados, analisando as implicações para estudos futuros, e em seguida as ameaças à validade dos resultados relatados, analisando os fatores que poderiam invalidar as conclusões obtidas.

Conforme já descrito a prova de conceito possui uma contribuição limitada a um exemplo de possibilidade da criação de um iMA simples utilizando a LPS desenvolvida. Além disso, esse desenvolvimento não incluiu algumas funcionalidades que seriam providas pelo arcabouço e que devem ser adicionadas ao *iTangran*, ou seja, não considera o desenvolvimento de todas as funcionalidades requeridas do iMA.

O estudo de caso relatado aqui não abrange todo o processo de desenvolvimento e refatoração do *iVProg*, pois esse é maior do que o período de execução deste projeto de mestrado. Dessa forma, os resultados de implementação apresentados não atingiram todas as funcionalidades existentes na versão anterior do aplicativo.

Apesar dos resultados apresentados serem limitados por esses aspectos, eles podem ser utilizados para derivar conclusões sobre o processo de desenvolvimento de iMA e outros aplicativos educacionais em contexto acadêmico e servir de base para outros projetos de pesquisa ou desenvolvimento. Características relatadas aqui podem ser similares a outros projetos e permitir a utilização das conclusões elaboradas. Exemplos incluem o contexto de desenvolvimento, a experiência do programador, qualidade do código legado, documentação. Assim, os resultados podem superar os limites restritos dos casos relatados e contribuir para projetos futuros similares.

Com relação às ameaças à validade dos resultados podemos citar: (i) as funcionalidades não implementadas; (ii) o tamanho em linhas de código dos aplicativos desenvolvidos; (iii) o processo

executado ser diferente daquele definido pela LPS; (iv) apenas o fato do código ser reescrito ter contribuído para o aprimoramento do desenvolvimento; (v) a experiência do programador ter mudado durante o processo; e (vi) a qualidade do código produzido ser baixa.

As funcionalidades que não foram consideradas durante os processos relatados dificilmente mudariam as conclusões tiradas dos resultados porque suas características se mostram mais simples para o desenvolvedor do que aquelas já implementadas. A maior diferença seria no tempo para a implementação dessas funcionalidades adicionais, e não na contribuição da LPS para o desenvolvimento. Além disso, com o tempo a LPS possuirá um número maior de funcionalidades a serem reutilizadas, aumentando sua contribuição ao desenvolvimento de iMA.

O tamanho em linhas de código dos aplicativos desenvolvidos neste trabalho é relativamente menor do que os iMA existentes. Isso pode indicar que os problemas com relação a qualidade de código não são aparentes nesse estágio de desenvolvimento. Porém, a documentação gerada nesse estágio e a preocupação com a qualidade do código já é desde o início maior do que o realizado durante o desenvolvimento dos iMA existentes, de acordo com o relato dos desenvolvedores. As conclusões feitas com relação às contribuições da LPS assumem que essas atitudes para manter o código com alta qualidade e documentado são realizadas continuamente.

Outra limitação é o fato do processo promovido pela LPS não ter sido seguido corretamente, o que poderia desqualificar as conclusões sobre a influência da LPS no desenvolvimento. É fato que se o processo utilizado não for aquele descrito pela LPS sua influência é menor, mas não inexistente. A LPS contribui ainda com a reutilização da arquitetura e de código, retirando a responsabilidade do programador de definir a arquitetura do aplicativo e com as funcionalidades reaproveitáveis.

O aprimoramento do trabalho de desenvolvimento considerado quando comparado antes e depois da adoção da LPS pode ter sido provocado principalmente pelo fato do código ser reescrito, e não pela LPS. A influência da reescrita do código para o andamento do desenvolvimento pode ser maior do que a da LPS quando considerado o caso do *iVProg*. Porém, a reescrita do código isoladamente incluiria como responsabilidade do programador várias tarefas que a LPS realiza, como a especificação das funcionalidades e a arquitetura. Dessa forma, a remoção dos obstáculos provocados pela qualidade do código legado não foi o único aprimoramento do desenvolvimento com o uso da LPS.

Durante o estudo de caso o programador continuou estudando e cursando disciplinas relacionadas a seu trabalho de desenvolvimento. Isso fez com que sua experiência como programador fosse diferente entre os períodos anterior e posterior à adoção da LPS. Além disso, pode invalidar a comparação feita entre os processos de desenvolvimento entre esses períodos, porque o programador poderia ter realizado as considerações de qualidade de software sem as contribuições da LPS. Nesse caso, a LPS, mesmo com o programador mais experiente, continua contribuindo com a redução de seu esforço, que deixa de programar diversas funcionalidades, deixa de criar a arquitetura e de precisar definir as etapas do desenvolvimento. Essa contribuição é inversamente proporcional à experiência do desenvolvedor. Então, é possível que o programador atinja um nível de maturidade que o permita afirmar que o desenvolvimento seja melhor sem o uso da LPS, mas esse não foi a situação do caso analisado.

Por fim, a última ameaça à validade considerada é a avaliação da qualidade do código produzido. As métricas utilizadas, apesar de serem sugeridas como indicadores da facilidade de realizar manutenção, podem ter sido mal utilizadas ou podem, nesse caso, não serem precisas. Essa ameaça

pode ser importante caso a preocupação em manter ou aumentar a qualidade do código durante o desenvolvimento futuro do aplicativo diminua. Caso os programadores se esforcem em seguir o processo descrito pela LPS e, além disso, em documentar e manter a qualidade do código alta, os problemas de manutenção serão menos graves do que os enfrentados atualmente.

Capítulo 6

Conclusões

Neste trabalho, foi criada uma LPS com o objetivo de aprimorar o desenvolvimento de iMA, consistindo de um arcabouço de aplicação e seus manuais de utilização. Para avaliar o aprimoramento pretendido, um novo iMA chamado *iTangran* e uma nova versão do iMA existente *iVProg* foram desenvolvidos usando como base a LPS criada.

As principais contribuições para o processo de desenvolvimento de iMA oferecidos pela LPS são: (i) aumento da produtividade pelo fornecimento de arquitetura e funcionalidades para serem reutilizadas; (ii) aumento da qualidade do código produzido pela estrutura exigida para o aplicativo e pelas boas práticas de programação estimuladas pelos manuais; e (iii) redução da dificuldade das tarefas para o programador, principalmente quando possui pouca experiência em desenvolvimento de software, provocado pelo reúso de processo sobre como utilizar a LPS apresentado passo a passo.

A partir dessas contribuições é possível afirmar que a LPS criada neste trabalho aprimora o desenvolvimento de iMA. Como contribuições adicionais deste trabalho, podemos listar: (i) a centralização do conhecimento sobre os aplicativos da família iMA na modelagem de domínio feita para a LPS; (ii) a apresentação e aplicação de métodos sistemáticos de desenvolvimento de software e sua avaliação, como exemplo para outros trabalhos na área de Informática na Educação; (iii) os recursos concretos da LPS, o arcabouço de aplicação e os manuais produzidos; e (iv) a criação de um novo iMA e da nova versão de um iMA existente, como contribuições para professores e alunos.

Um dos maiores desafios enfrentados no decorrer deste trabalho foi a falta de prática do grupo de pesquisa, que reflete em artigos da área de Informática na Educação, na utilização e avaliação de métodos sistemáticos de desenvolvimento de software. Outro grande desafio foi a dificuldade de planejamento e coordenação dos desenvolvimentos, uma vez que o trabalho realizado envolveu diversos programadores e alunos de pós-graduação.

Dessa forma, a principal mensagem desta dissertação de mestrado é que mesmo em contextos em que aparentemente métodos sistemáticos de desenvolvimento não são apropriados ou não contribuem significativamente, como naquele em que são criados os iMA e outros aplicativos educacionais, esforços no sentido de aprimorar os processos de criação de sistemas podem trazer muitos benefícios. Assim, este trabalho se apresenta como exemplo de aprimoramento do desenvolvimento de aplicativos educacionais.

Analisamos a seguir a relação entre os resultados obtidos e o cumprimento dos objetivos específicos definidos no início do trabalho, seção 1.3.

- *Levantar e explicitar características e problemas do processo de desenvolvimento de aplicati-*

vos educacionais em contexto acadêmico. Esse objetivo foi cumprido em dois momentos do trabalho. O primeiro foi durante o levantamento de evidências da problemática considerada, em que foram relatados problemas encontrados e características do desenvolvimento em nosso grupo de pesquisa e em outros grupos. O segundo momento foi o relato mais cuidadoso das características e do andamento do desenvolvimento do *iVProg*, feito no estudo de caso da seção 5.2. As características levantadas nesses momentos serviram de justificativa para este trabalho e podem ser usadas como um ponto de partida para outros estudos sobre esses contextos de desenvolvimento.

- *Adotar e divulgar o uso de métodos sistemáticos para o desenvolvimento de aplicativos educacionais*. A adoção de métodos sistemáticos se deu ao longo da execução do trabalho, conforme relatado em vários pontos desta dissertação. Com relação à divulgação do uso desses métodos, isso foi realizado pela publicação de artigos acadêmicos em conferências e revistas especializadas. A lista de artigos publicados durante a realização do projeto é: (i) (Dalmon *et al.*, 2012a), (ii) (Dalmon *et al.*, 2012c), (iii) (Dalmon *et al.*, 2012b); (iv) (Dalmon *et al.*, 2011a); (v) (Dalmon *et al.*, 2011b); (vi) (Dalmon *et al.*, 2011c); (vii) (Dalmon *et al.*, 2010a); (viii) (Dalmon *et al.*, 2010c); e (ix) (Dalmon *et al.*, 2010b).

- *Reunir e documentar aspectos relacionados à família de sistemas iMA*. Esse objetivo foi considerado em dois momentos da realização do trabalho. O primeiro foi no levantamento e relato do histórico de desenvolvimento dos sistemas, feito na seção 2.1. O segundo foi durante todo o desenvolvimento da LPS, em que as características da família como um todo foram usadas nessa tarefa e relatadas nesta dissertação no capítulo 4.

- *Fornecer recursos concretos para serem reutilizados no desenvolvimento de novos iMA ou na refatoração dos existentes* e *Promover a reorganização, a refatoração e a adição de novas funcionalidades aos iMA existentes*. Esses objetivos foram cumpridos considerando os resultados do desenvolvimento da LPS e os estudos de avaliação realizados. Todos os recursos criados estão disponíveis na página do projeto (Dalmon e Brandão, 2012).

Nas próximas seções são feitas algumas considerações finais sobre os resultados obtidos e as ideias geradas durante o desenvolvimento deste trabalho e por fim sugestões de trabalhos futuros a partir do que foi criado neste.

6.1 Considerações Finais

Após o relato e a conclusão desta dissertação de mestrado, vale ressaltar algumas considerações finais sobre assuntos levantados durante o seu desenvolvimento e alguns desdobramentos das atividades realizadas.

Projeto de Pesquisa ou de Desenvolvimento

Durante parte da execução deste projeto ele foi considerado como um projeto de desenvolvimento, cuja principal contribuição seriam os aplicativos desenvolvidos com ele. Porém, ao incluir o aspecto de pesquisa, foi necessário decidir qual era o objetivo do trabalho e elaborar as atividades de avaliação. Como essa transição foi realizada com o projeto em andamento, houve dificuldades

para definir e executar aspectos do método científico. Em geral, essas dificuldades foram superadas e o trabalho gerou resultados significativos.

Uma referência importante para essa transição é o livro publicado por Tchounikine (2011), na qual são definidos projetos de desenvolvimento e projetos de pesquisa. Além disso, nesse mesmo livro, o autor diz que a área de pesquisa em desenvolvimento de aplicativos educacionais tem falta de projetos de pesquisa e falta de iniciativas para capitalização do conhecimento.

Após a definição das questões de pesquisa e dos objetivos deste trabalho, foi um esforço contínuo realizar a capitalização do conhecimento gerado ou analisado neste trabalho. Acreditamos que com os artigos publicados e documentos elaborados, além da comunicação com membros do grupo de pesquisa, esse esforço se propague.

Estudos de Avaliação Incompletos

Além dos estudos de avaliação apresentados no capítulo 5, outros estudos foram feitos mas não geraram resultados com os quais era possível tecer conclusões. Como citado na seção 3.3.3, estudos de caso do desenvolvimento de outros iMA entram nessa descrição. As informações coletadas sobre o desenvolvimento anterior à adoção da LPS nos estudos de caso para os outros iMA apresentaram situações de desenvolvimento diferentes, mas nenhum deles produziu resultados sobre o período posterior.

Outro estudo realizado foi uma prova de conceito para a expansão do arcabouço pela implementação de funcionalidades de tutoria inteligente usando a arquitetura de módulos de atividade e operação. Esse estudo passou por algumas iterações de projeto e implementação mas não alcançou um código funcional. A continuação desses estudos estão incluídos nas sugestões de trabalhos futuros.

Esses estudos tinham como objetivo reduzir as limitações dos resultados obtidos com o estudo de caso relatado na seção 5.2. Além disso, deveriam também gerar resultados com relação a outros aspectos que não podem ser analisados apenas com o estudo do desenvolvimento do *iVProg*, como a influência da experiência do programador, a produtividade relativa entre os projetos, entre outros.

Durante a realização desses estudos, outras questões foram levantadas. Exemplos são o planejamento da pesquisa e a revisão crítica do código criado por outro programador. O planejamento da pesquisa deve levar em consideração a avaliação e possivelmente um período posterior, com o objetivo de realizar algumas atividades sugeridas nessa ocasião. Além disso, a avaliação deve ser realizada de forma sistemática, com considerações explícitas sobre o que se deseja avaliar para gerar resultados relevantes para a sociedade. O planejamento desta pesquisa não reservou tempo suficiente para vários estudos de caso e para revisar o que foi desenvolvido após a avaliação.

A revisão do código criado por outro programador foi levantada durante a prova de conceito de expansão do arcabouço. O código gerado na implementação de um aplicativo não pode ser considerado de qualidade até que outro programador com experiência o analise, como ocorreu com partes do código do arcabouço, cujas falhas de projeto foram percebidas quando o código foi avaliado por outro programador. Isso não foi praticado nos trabalhos analisados em (Dalmon *et al.*, 2012b) mas deve começar a ser feito no grupo de pesquisa deste trabalho. Além dessas considerações, outras lições foram aprendidas, apresentadas abaixo.

Lições Aprendidas

Ao longo do desenvolvimento da LPS criada neste trabalho, algumas considerações sobre esse processo foram percebidas. Algumas já foram descritas, ao longo da implementação do arcabouço apresentamos a importância do processo iterativo de implementação da revisão contínua dos métodos utilizados. Nas atividades de avaliação destacamos o planejamento do projeto como um todo e considerações metodológicas de pesquisa, para que os resultados obtidos tenham validade acadêmica. Ao longo dos estudos de utilização percebemos que a avaliação e revisão dos aplicativos durante o desenvolvimento aprimora os resultados. Por fim, nas conclusões gerais, afirmamos a importância da utilização de métodos sistemáticos de desevolvimento para aplicativos educacionais e da centralização e explicitação do conhecimento em projetos de desenvolvimento ou de pesquisa.

Algumas lições adicionais podem ser destacadas, referentes principalmente às tarefas de implementação. A menos que o programador seja experiente, a comunicação com colegas é essencial para o bom andamento dos trabalhos de implementação. Assim, os programadores compartilham suas ideias, que são avaliadas por outras pessoas, resolvendo problemas mais rapidamente e usando melhores soluções de projeto.

A sistematização do processo de desenvolvimento, ou ao menos a explicitação de todas as atividades conduzidas pelo programador, também contribui para o bom andamento do trabalho. Isso deve ser aliado à valorização do planejamento e dos futuros usuários do aplicativo sendo implementado. Sem essa consideração, código de má qualidade continuará a ser criado e problemas de manutenção não serão reduzidos.

6.2 Sugestões para Trabalhos Futuros

Pelo fato deste trabalho ter realizado um esforço com o objetivo de centralizar as pesquisas e desenvolvimentos de iMA, como um sub-produto do aprimoramento desses processos várias propostas e sugestões de trabalhos futuros podem ser feitas. Especificamente, essas sugestões estão organizadas em três categorias: (i) para melhorar; (ii) expandir; ou (iii) investigar a LPS desenvolvida e a família de aplicativos iMA, apresentadas a seguir.

Por melhorar a LPS desenvolvida, pretende-se aprimorar seus recursos de forma a aumentar sua contribuição positiva para os processos de desenvolvimento de iMA. Nessa categoria destacamos três frentes: (a) processo; (b) arquitetura; e (c) código. O aspecto processo foi o menos explorado durante o desenvolvimento da LPS para iMA, atuando principalmente no método de utilização do arcabouço, não considerando em profundidade a organização das pessoas envolvidas e a divisão das responsabilidades. Dessa forma, a LPS pode ser aprimorada com um processo mais elaborado e detalhado. De acordo com as sugestões dos usuários do arcabouço, várias alterações no método de utilização sugerido, além da elaboração de recursos adicionais, podem ser feitos visando facilitar o recebimento da LPS por novos programadores.

De maneira similar, a arquitetura e o código fornecido no arcabouço também podem ser aprimorados em trabalhos futuros. A arquitetura pode sofrer melhorias para facilitar mais o trabalho dos desenvolvedores ou se adaptar a funcionalidades adicionais. O código, por sua vez, deve ser constantemente atualizado para manter ou aumentar sua qualidade, prevenindo problemas de manutenção e evolução de software, conforme apresentado na seção 4.5, cujos valores das métricas são apresentados. A utilização de técnicas mais sofisticadas para o desenvolvimento do arcabouço, como

reflexão e metadados (Guerra, 2010), pode ser aplicada com o objetivo principal de facilitar o uso do arcabouço pelos programadores, o que pode motivar outros projetos de mestrado.

A frente considerada mais direta quando menciona-se a expansão da LPS para iMA é o desenvolvimento de novos iMA, aumentando, assim, a família de aplicativos e a contribuição dos desenvolvedores para a sociedade. Cada iMA novo, desenvolvido com o auxílio da LPS, adiciona um novo domínio aos iMA. A melhoria da educação é o objetivo último do desenvolvimento da LPS e de cada sistema da família. Esse provavelmente é o conjunto de trabalhos futuros com maiores contribuições diretas aos usuários finais professores e alunos.

Outras formas de expandir a LPS criada podem representar outros trabalhos futuros são relacionados a aumentar o número de funcionalidades independentes de domínio fornecidas e aumentar o escopo dos aplicativos que podem ser criados com a linha. Caso sejam realizados trabalhos para a continuação da engenharia de domínio, além das iniciativas de melhoria, novas responsabilidades podem ser adicionadas ao arcabouço. Exemplos dessas funcionalidades são os módulos de atividade e de operações apresentados nas seções 4.5.1 e 4.5.2, e ainda outras, como modelos de interfaces com usuário (*templates*), tutores inteligentes de diferentes paradigmas e funções de comunicação em rede para promover aprendizagem colaborativa durante atividades nos iMA.

A terceira possibilidade de expansão da LPS é sobre seu escopo, ou seja, sobre a definição de iMA. Isso pode ser feito com a redução ou o aumento das especificações de um aplicativo para ser considerado iMA, ou pela alteração da LPS de forma a auxiliar o desenvolvimento de outros tipos de aplicativos educacionais. Uma sugestão relevante feita pelos programadores usuários da LPS deste trabalho, que vem sendo estudada há tempos pelo grupo de pesquisa, é a elaboração de manuais para a adaptação de sistemas existentes para a família iMA. Dessa forma, em vez de auxiliar o desenvolvimento do aplicativo, o arcabouço ofereceria apenas algumas funcionalidades, principalmente as de ser *Applet* e de comunicação, para que um número maior de aplicativos possam oferecer atividades interativas integradas a SGC.

A última categoria de trabalhos futuros sugeridos a partir do relatado aqui é a de investigações sobre a LPS. Por investigações nos referimos a procurar respostas a questões de pesquisa, sendo que as sugestões anteriores dizem respeito mais a projetos de desenvolvimento. Reconhecemos três principais perguntas que nascem deste trabalho, relacionadas a: (i) o processo de desenvolvimento de iMA; (ii) a relação com o desenvolvimento de outras famílias e outros aplicativos educacionais; (iii) a abordagem pedagógica dos iMA; e (iv) as contribuições para a área de pesquisa em linha de produto deste trabalho.

O estudo de avaliação da influência da LPS proposta no desenvolvimento dos aplicativos é apenas uma tentativa de analisar um período curto de tempo, de forma que os resultados encontrados são preliminares considerando a expectativa de vida da linha criada. Assim, uma avaliação mais completa e sobre um período maior de tempo é necessário para descobrir respostas mais robustas e de relevância acadêmica.

Outro estudo possível trataria da relação dos resultados deste trabalho com processos de desenvolvimento de outros aplicativos educacionais ou da aplicação de outras técnicas de Engenharia de Software, investigando a abrangência das descobertas e os fatores que provocam as limitações. Por fim, duas perguntas de pesquisa que urgem são sobre a influência da abordagem definida para os iMA nos processos de ensino e aprendizagem, e quais as influências que a qualidade do código de um aplicativo pode ter sobre esses processos. Uma resposta para essas perguntas esclareceria

quais fatores e funcionalidades são mais importantes desses sistemas, quais as menos importantes, e possivelmente guiaria novos desenvolvimentos.

Por fim, uma última pergunta para projetos de pesquisa é: quais as contribuições do trabalho relatado nesta dissertação para a área de pesquisa em LPS? Nesse caso seria necessário estudar o desenvolvimento de outras LPS e verificar quais as diferenças do processo executado aqui, e que lições aprendidas podem ser transferidas a outros projetos de desenvolvimento de LPS, independentemente do domínio a ser aplicado.

Referências Bibliográficas

Ahmed e Zualkernan (2011) Faheem Ahmed e Imran Zualkernan. A software product line methodology for development of e-learning system. *International Journal of Computer Science and Emerging Technologies*, 2:285–295. Citado na pág. 2, 23, 24

Aleven *et al.* **(2009)** Vincent Aleven, Bruce M. Mclaren, Jonathan Sewall e Kenneth R. Koedinger. A new paradigm for intelligent tutoring systems: Example-tracing tutors. *International Journal of Artificial Intelligence in Education*, 19:105–154. Citado na pág. 29, 48

Amory *et al.* **(1999)** Alan Amory, Kevin Naicker, Jacky Vincent e Claudia Adams. The use of computer games as an educational tool: identification of appropriate game types and game elements. *British Journal of Educational Technology*, 30(4):311–321. Citado na pág. 1

Ateyeh e Lockemann (2006) K. Ateyeh e P. C. Lockemann. Reuse- and aspect-oriented courseware development. *Educational Technology and Society*, 9:95–113. Citado na pág. 17, 22, 24

Basili *et al.* **(1986)** Victor R. Basili, Richard W. Selby e David H. Hutchens. Experimentation in software engineering. *IEEE Transactions on Software Engineering*, 12. Citado na pág. 35

Batista *et al.* **(2011)** Silvia Cristina F. Batista, Patricia Alejandra Behar e Liliana Maria Passerino. M-learnmat: Aplicação de um modelo pedagógico para atividades de m-learning em matemática. Em *Anais do XXII Simpósio Brasileiro de Informática na Educação*. Citado na pág. 1

Beck (2002) Kent Beck. *Test Driven Development: By Example*. Addison-Wesley Professional. ISBN 0321146530. Citado na pág. 74

Ben-Menachem e Marliss (1997) Mordechai Ben-Menachem e Gary Marliss. *Slaying the Software Dragon: A Practical Guide to Software Quality*. International Thomson Computer Press, Boston, MA, USA, 1st ed. ISBN 1850323267. Citado na pág. 21, 34

Berggren *et al.* **(2005)** Anders Berggren, Daniel Burgos, Josep M. Fontana, Don Hinkelman, Vu Hung, Anthony Hursh e Ger Tielemans. Practical and pedagogical issues for teacher adoption of ims learning design standards in moodle lms. *Journal of Interactive Media in Education*. Citado na pág. 22

Bloom (1984) Benjamin S. Bloom. The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher*, 13:4–16. Citado na pág. 1

Boehm (1986) B. Boehm. A spiral model of software development and enhancement. *ACM SIGSOFT Software Engineering Notes*, 11:14–24. Citado na pág. 32

Boehm *et al.* **(1976)** B. W. Boehm, J. R. Brown e M. Lipow. Quantitative evaluation of software quality. Em *Proceedings of the 2nd international conference on Software engineering*, páginas 592–605. Citado na pág. 4

Bote-Lorenzo *et al.* **(2004)** M. L. Bote-Lorenzo, D. Hernández-Leo, Y. A. Dimitriadis, J. I. Asensio-Pérez, E. Gómez-Sánchez, G. Vega-Gorgojo e L. M. Vaquero-González. Towards reusability and tailorability in collaborative learning systems using ims-ld and grid services. *International Journal on Advanced Technology for Learning*, 1:129–138. Citado na pág. 22, 24

Boyle (2003) Tom Boyle. Design principles for authoring dynamic, reusable learning objects. *Australian Journal of Educational Technology*, 19:46–58. Citado na pág. 20, 21, 22, 24

Brandão (2012a) Leônidas O. Brandão. Emulador/compilador "computador a gavetas". http://www.matematica.br/programas/icg/, 2012a. Último acesso em 05/06/2012. Citado na pág. 11

Brandão (2012b) Leônidas O. Brandão. Torres de hanói. http://www.matematica.br/ihanoi/, 2012b. Último acesso em 05/06/2012. Citado na pág. 11

Brandão e Isotani (2003) Leônidas O. Brandão e Seiji Isotani. Uma ferramenta para ensino de geometria dinâmica na internet: *iGeom*. *Anais do IX Workshop de Informática na Educação*. Citado na pág. 9, 10, 11

Brandão *et al.* (2004) Leônidas O. Brandão, Seiji Isotani e Janine G. Moura. Geometria dinâmica com o *iGeom*: Algoritmos geométricos, autoria e avaliação automática de exercícios. *Anais do XV Simpósio Brasileiro de Informática na Educação*. Citado na pág. 10, 11

Carnegie Mellon University(2012) Carnegie Mellon University. Alice. http://www.alice.org, 2012. Último acesso em 05/06/2012. Citado na pág. 69

Cavaye e Cragg (1995) Angèle L. M. Cavaye e Paul B. Cragg. Factors contributing to the success of customer oriented interorganizational systems. *Journal of Strategic Information Systems*, 4. Citado na pág. 36, 37

Centro de Competência em Software Livre(2012) CCSL-IME-USP Centro de Competência em Software Livre. Redmine. http://ccsl.ime.usp.br/redmine/, 2012. Último acesso em 05/06/2012. Citado na pág. 33

Chan e Lee (1996) Patrick Chan e Rosanna Lee. *The Java Class Libraries: An Annotated Reference*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st ed. ISBN 0201634589. Citado na pág. 17

Chidamber e Kemerer (1994) Shyam R. Chidamber e Chris F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20. Citado na pág. 39, 62

Choquet e Corbière (2006) C. Choquet e A. Corbière. Reengineering framework for systems in education. *Educational Technology and Society*, 9:228–241. Citado na pág. 2, 17, 22, 24

Chou *et al.* (2010) Chien Chou, Hsinyi Peng e Chun-Yu Chang. The technical framework of interactive functions for course-management systems: Students' perceptions, uses, and evaluations. *Computers and Education*, 55:1004–1017. Citado na pág. 14

Clayberg e Rubel (2008) Eric Clayberg e Dan Rubel. *Eclipse Plug-ins*. Addison-Wesley Professional, 3rd ed. ISBN 0321553462. Citado na pág. 34

Clements e Northrop (2001) Paul Clements e Linda Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley Professional. ISBN 0201703327. Citado na pág. 2, 6, 15, 17, 25, 27

Clements *et al.* (2006) Paul C. Clements, Lawrence G. Jones, John D. McGregor e Linda M. Northrop. Getting there from here: a roadmap for software product line adoption. *Communications of the ACM*, 49:33–36. Citado na pág. 29

Conlan *et al.* (2002) Owen Conlan, Cord Hockemeyer, Vincent Wade e Dietrich Albert. Metadata driven approaches to facilitate adaptivity in personalized elearning systems. *Journal of Information and Systems in Education*, 1:38–44. Citado na pág. 2, 22, 24

Dalmon e Brandão (2012) Danilo L. Dalmon e Leônidas O. Brandão. Arcabouço para módulos de aprendizagem interativos. http://ccsl.ime.usp.br/redmine/projects/ima, 2012. Último acesso em 05/06/2012. Citado na pág. 14, 34, 39, 40, 52, 58, 62, 67, 69, 79, 86

Dalmon *et al.* **(2010a)** Danilo L. Dalmon, Seiji Isotani e Leônidas O. Brandão. Sistemas de geometria interativa e tutores inteligentes: desafios e perspectivas de uma abordagem conjunta. *Anais do XXI Simpósio Brasileiro de Informática na Educação*. Citado na pág. 28, 86

Dalmon *et al.* **(2010b)** Danilo L. Dalmon, Seiji Isotani e Leônidas O. Brandão. Melhorando a geometria interativa com o uso de tutores rastreadores de padrões: *iGeom*e ctat. *Anais do XVI Workshop de Informática na Escola*. Citado na pág. 28, 86

Dalmon *et al.* **(2010c)** Danilo L. Dalmon, Maria José G. S. Tanbellini, Seiji Isotani e Leônidas O. Brandão. Desafios e novas perspectivas de aplicação de sistemas de geometria interativa: novas funcionalidades do *iGeom*. *Anais do V Simpósio de Pós-graduação e Iniciação Científica do IME-USP*. Citado na pág. 86

Dalmon *et al.* **(2011a)** Danilo L. Dalmon, Seiji Isotani, Anarosa A. F. Brandão e Leônidas O. Brandão. Work in progress - a framework for building interactive learning modules. *Proceedings of 41st Frontiers in Education Conference*. Citado na pág. 52, 86

Dalmon *et al.* **(2011b)** Danilo L. Dalmon, Seiji Isotani, Anarosa A. F. Brandão e Leônidas O. Brandão. Work in progress - enhancing interactive geometry systems with intelligent tutoring features. *Proceedings of 41st Frontiers in Education Conference*. Citado na pág. 28, 86

Dalmon *et al.* **(2011c)** Danilo L. Dalmon, M. J. G. S. Tanbellini, A. Eisenmann, M. Nascimento, P. A. Rodrigues, S. Isotani, A. A. F. Brandão e L. O. Brandão. Interactive learning modules in engineering education and as a motivational tool for middle and high school students. *Proceedings of International Symposium on Engineering Education - IGIP*. Citado na pág. 2, 9, 28, 86

Dalmon *et al.* **(2012a)** Danilo L. Dalmon, Anarosa A. F. Brandao, Seiji Isotani e Leônidas O. Brandão. A domain engineering for interactive learning modules. *Journal of Research and Practice in Information Technology - aceito para publicação*. Citado na pág. 86

Dalmon *et al.* **(2012b)** Danilo L. Dalmon, Anarosa A. F. Brandão e Leônidas O. Brandão. Uso de métodos e técnicas para desenvolvimento de software educacional em universidades brasileiras. Em *Anais do I Workshop de Desafios da Computação Aplicada à Educação - DesafIE*. Citado na pág. 2, 3, 24, 86, 87

Dalmon *et al.* **(2012c)** Danilo L. Dalmon, Anarosa A. F. Brandão, Seiji Isotani, Leônidas O. Brandão e Guilherme Gomes. Work in progress - a generic model for interactivity-intense intelligent tutor authoring tools. *Proceedings of 42nd Frontiers in Education Conference*. Citado na pág. 46, 86

do Prado (2008) Reginaldo do Prado. *iGraf*: Uma proposta de sistema para ensino de função via web. Dissertação de Mestrado, Instituto de Matemática e Estatística - USP. Citado na pág. 11

do Prado e Brandão (2006) Reginaldo do Prado e Leônidas O. Brandão. *iGraf*: Módulo de aprendizagem para ensino de função na web. *Anais do XIV Simpósio Brasileiro de Informática na Educação*. Citado na pág. 28

Dodero *et al.* **(2007)** Juan Manuel Dodero, Telmo Zarraonandia, Camino Fernandez e David Diez. Generative adaptation and reuse of competence development programmes. *Journal of Interactive Media in Education*, 4. Citado na pág. 2, 17, 22, 24

Dotta (2011) Sílvia Dotta. Uso de uma mídia social como ambiente virtual de aprendizagem. Em *Anais do XXII Simpósio Brasileiro de Informática na Educação*. Citado na pág. 1

Douglas (2001) Ian Douglas. Instructional design based on reusable learning objects: Applying lessons of object-oriented software engineering to learning systems design. *Proceedings of 31st Frontiers in Education Conference*. Citado na pág. 20, 22, 24

Downes e McMillan (2000) Edward J. Downes e Sally J. McMillan. Defining interactivity a qualitative identification of key dimensions. *New Media and Society*, 2. Citado na pág. 14

The Eclipse Foundation(2012) The Eclipse Foundation. Eclipse. http://www.eclipse.org/, 2012. Último acesso em 05/06/2012. Citado na pág. 33

Eisenmann e Brandão (2009) Alexandre L. K. Eisenmann e Leônidas O. Brandão. *iComb*: um sistema para o ensino e aprendizagem de combinatória em ambiente web. *Anais do XX Simpósio Brasileiro de Informática na Educação*. Citado na pág. 28

Eisenmann (2009) Alexandre Luís Kundrát Eisenmann. *iComb*: Um sistema para o ensino e aprendizagem de combinatória em ambiente web. Dissertação de Mestrado, Instituto de Matemática e Estatística - USP. Citado na pág. 12

Fayad e Schmidt (1997) Mohamed Fayad e Douglas C. Schmidt. Object-oriented application frameworks. *Communications of the ACM*, 40:32–38. Citado na pág. 18

Fayad e Johnson (2000) Mohamed E. Fayad e Ralph E. Johnson, editors. *Domain-Specific Application Frameworks: Frameworks Experience by Industry*. Wiley, New York. ISBN 978-0-471-33280-0. Citado na pág. 18

Fayad *et al.* **(1999)** Mohamed E. Fayad, Douglas C. Schmidt e Ralph E. Johnson. *Implementing application frameworks: object-oriented frameworks at work*. John Wiley & Sons, Inc., New York, NY, USA. ISBN 0-471-25201-8. Citado na pág. 4, 18

Fowler *et al.* **(1999)** Martin J. Fowler, Kent Beck, John Brant, William Opdyke e Don Roberts. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional. ISBN 0201485672. Citado na pág. 34, 71

Gadelha *et al.* **(2010)** B. Gadelha, E. Cirilo, H. Fuks, M. A. Gerosa, A. Castro e C. J. P. Lucena. Uma abordagem para o desenvolvimento de linhas de produto de groupware baseados em componentes utilizando o groupware workbench. *VII Simpósio Brasileiro de Sistemas Colaborativos*, páginas 32–38. Citado na pág. 23, 24

Gamma *et al.* **(1994)** Erich Gamma, Richard Helm, Ralph Johnson e John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional. ISBN 0201633612. Citado na pág. 29, 34, 48, 49, 51, 64

Geller *et al.* **(2009)** Marla T. B. Geller, Marialina Corrêa Sobrinho e Carlos A. P. Araújo. Proposta de customização de um processo para desenvolvimento de software educativo. Em *Anais do XX Simpósio Brasileiro de Informática na Educação*. Citado na pág. 20

Gerring (2006) John Gerring. *Case Study Research: Principles and Practices*. Cambridge University Press. ISBN 0521676568. Citado na pág. 6, 35, 36, 37

Software Freedom Conservancy(2012) Software Freedom Conservancy. Git. http://git-scm.com/, 2012. Último acesso em 05/06/2012. Citado na pág. 33

Gomaa (2004) Hassan Gomaa. *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA. ISBN 0201775956. Citado na pág. 29, 30

Google (2012) Google. Google acadêmico. http://scholar.google.com.br/, 2012. Último acesso em 05/06/2012. Citado na pág. 19

Green (2010) Kenneth C. Green. The 2010 national survey of information technology in u.s. higher education. Relatório técnico, Campus Computing Project. Citado na pág. 1, 12

Guerra (2010) Eduardo Martins Guerra. *A Conceptual Model for Metadata-based Frameworks*. Tese de Doutorado, Instituto Tecnológico de Aeronáutica. Citado na pág. 18, 59, 61, 89

Hailpern e Tarr (2006) B. Hailpern e P. Tarr. Model-driven development: the good, the bad, and the ugly. *IBM Syst. J.*, 45:451–461. Citado na pág. 22

Hinostroza *et al.* **(2000)** Enrique Hinostroza, Lucio E. Rehbein, Harvey Mellar e Christina Preston. Developing educational software: a professional tool perspective. *Education and Information Technologies*, 5:103–117. Citado na pág. 1, 2, 21

Inkpen (2001) Kori M. Inkpen. Drag-and-drop versus point-and-click mouse interaction styles for children. *ACM Transactions on Human Computer Interface*, 8. Citado na pág. 72

Isotani (2005) Seiji Isotani. Desenvolvimento de ferramentas no *iGeom*: utilizando a geometria dinâmica no ensino presencial e a distância. Dissertação de Mestrado, Instituto de Matemática e Estatística - USP. Citado na pág. 11

Isotani e Brandão (2008) Seiji Isotani e Leônidas O. Brandão. An algorithm for automatic checking of exercises in a dynamic geometry system: *iGeom*. *Computers and Education*, 51: 1283–1303. Citado na pág. 10, 14, 28

Kamiya (2010) Reginaldo Rideaki Kamiya. *iVProg*: um sistema visual para ensino-aprendizagem de programação via web. Dissertação de Mestrado, Instituto de Matemática e Estatística - USP. Citado na pág. 12, 69

Kamiya e Brandão (2009) Reginaldo Rideaki Kamiya e Leônidas O. Brandão. *iVProg*- um sistema para introdução à programação através de um modelo visual na internet. *Anais do XX Simpósio Brasileiro de Informática na Educação*. Citado na pág. 3, 28, 69

Kang *et al.* **(1990)** K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak e A. S. Peterson. Feature-oriented domain analysis (*FODA*) feasibility study. Relatório técnico, Carnegie-Mellon University Software Engineering Institute. Citado na pág. xi, 25, 26, 27

Kiczales e Hilsdale (2001) Gregor Kiczales e Erik Hilsdale. Aspect-oriented programming. *SIGSOFT Software Engineering Notes*, 26. Citado na pág. 22

Koedinger e Aleven (2007) K. R. Koedinger e V. Aleven. Exploring the assistance dilemma in experiments with cognitive tutors. *Education Psychological Review*, páginas 239–264. Citado na pág. 14

Kortenkamp (1999) Ulrich Kortenkamp. *Foundations of Dynamic Geometry*. Tese de Doutorado, Swiss Federal Institute of Technology. Citado na pág. 10

Li e Henry (1993) Wei Li e Sallie Henry. Object-oriented metrics that predict maintainability. *Journal of Systems and Software*, 23. Citado na pág. 39

Matos *et al.* **(2010)** Cícero Jorge Rodrigues Matos, Alisandra Cavalcante Fernandes, Maria de Fátima de Sousa, Lavina Lúcia Vieira Lima, Raquel Santiago Freire, Mauro Cavalcante Pequeno e José Aires de Castro Filho. Avaliando objetos de aprendizagem a partir de testes pedagógicos. *Anais do XXI Simpósio Brasileiro de Informática na Educação*. Citado na pág. 21

McAndrew *et al.* **(2006)** Patrick McAndrew, Peter Goodyear e James Dalziel. Patterns, designs and activities: unifying descriptions of learning structures. *International Journal of Learning Technology*, 2:126–242. Citado na pág. 2, 20

Merril (1984) M. D. Merril. *Don't bother me with instructional design: I'm busy programming!* CSE report. Center for the Study of Evaluation, Graduate School of Education, UCLA. Citado na pág. 1, 21

Meziane *et al.* (1999) Farid Meziane, Chan Weng Onn, Jane Labadin e Roger Harris. Towards a framework for developing educational software. *Proceedings of the International Conference on Information Technology in Asia*. Citado na pág. 2, 21

Miles e Hamilton (2006) Russ Miles e Kim Hamilton. *Learning UML 2.0*. O'Reilly Media, Inc. ISBN 0596009828. Citado na pág. 27, 29

Moodle.org (2012) Moodle.org. Moodle statistics. http://moodle.org/stats, 2012. Último acesso em 05/06/2012. Citado na pág. 12

Mor e Winters (2007) Y. Mor e N. Winters. Design approaches in technology-enhanced learning. *Interactive Learning Environments*, 15:61–75. Citado na pág. 2, 21

Moundridou e Virvou (2003) Maria Moundridou e Maria Virvou. Analysis and design of a web-based authoring tool generating intelligent tutoring systems. *Computers and Education*, 40: 157–181. Citado na pág. 10, 14

Moura (2007) Janine Gomes Moura. *SAW*- sistema de aprendizagem pela web: motivações e desenvolvimento. Dissertação de Mestrado, Instituto de Matemática e Estatística - USP. Citado na pág. 12

Mpaka (2010) Nlandu Mpaka. O ensino e aprendizagem do gráfico da função quadrática com e sem auxílio do software winplot. Dissertação de Mestrado, Instituto de Ciências da Educação Universidade Lusófona de Humanidades e Tecnologias. Citado na pág. 11

Nash (2005) Susan Smith Nash. Learning objects, learning object repositories, and learning theory: Preliminary best practices for online courses. *Interdisciplinary Journal of Knowledge and Learning Objects*, 1:217–228. Citado na pág. 20, 21

Nicolson e Scott (1986) R. I. Nicolson e P. J. Scott. Computers and education: the software production problem. *British Journal of Educational Technology*, 17(1):26–35. Citado na pág. 1, 2, 21, 24

Oberweis *et al.* (2007) Andreas Oberweis, Victor Pankratius e Wolffried Stucky. Product lines for digital information products. *Information Systems*, 32:909–939. Citado na pág. 2, 16, 23, 24

Ohl (2001) T. M. Ohl. An interaction-centric learning model. *Journal of Educational Multimedia and Hypermedia*, 10:311–332. Citado na pág. 14

Oliveira e Gerosa (2011) Lucas Santos Oliveira e Marco Aurélio Gerosa. Collaborative features in content sharing web 2.0 social networks: A domain engineering based on the 3c collaboration model. *17th CRIWG Conference on Collaboration and Technology*, páginas 142–157. Citado na pág. 23, 24

Oracle (2012a) Oracle. Code samples and apps - applets. http://java.sun.com/applets/, 2012a. Último acesso em 05/06/2012. Citado na pág. 13

Oracle (2012b) Oracle. Java Class JAppletdocumentation. http://docs.oracle.com/javase/6/docs/api/javax/swing/JApplet.html, 2012b. Último acesso em 05/06/2012. Citado na pág. 58

Oracle (2012c) Oracle. Java SE Desktop Technologies. http://www.oracle.com/technetwork/java/javase/tech/index-jsp-138795.html, 2012c. Último acesso em 05/06/2012. Citado na pág. 22

Oracle (2012d) Oracle. Javadoc tool. http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html, 2012d. Último acesso em 05/06/2012. Citado na pág. 34

Oracle (2012e) Oracle. Java Class JFramedocumentation. http://docs.oracle.com/javase/6/docs/api/javax/swing/JFrame.html, 2012e. Último acesso em 05/06/2012. Citado na pág. 56, 75

Pankratius (2005) Victor Pankratius. Aspect-oriented learning objects. *Proceedings of the IASTED International Conference on Web-based Education*. Citado na pág. 17, 22, 24

Pedrosa e dos Santos (2004) Christiane Pastich Pedrosa e Maria Helena Burle Pio dos Santos. Reconstruindo a geometria com o tangram. Em *Anais do VIII ENEM - Encontro Nacional de Ensino de Matemática*. Citado na pág. 66

The Pennsylvania State University(2012) The Pennsylvania State University. Cite seer x beta. http://citeseerx.ist.psu.edu/index, 2012. Último acesso em 05/06/2012. Citado na pág. 19

Pohl *et al.* **(2005)** Klaus Pohl, Günter Böckle e Frank van der Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer, Berlin Heidelberg New York. ISBN 3540243720. Citado na pág. 6, 15, 16, 27, 29

Polsani (2003) Pithamber R. Polsani. Use and abuse of reusable learning objects. *Journal of Digital Information*, 3. Citado na pág. 10, 22, 24

Quintana *et al.* **(2003)** Chris Quintana, Joseph Krajcik, Elliot Soloway e Cathleen Norris. A framework for understanding the development of educational software. Em Julie A. Jacko e Andrew Sears, editors, *The human-computer interaction handbook*, páginas 823–834. CRC Press. Citado na pág. 1, 20

Raines e Clark (2011) Joan M Raines e Linda M. Clark. A brief overview on using technology to engage students in mathematics. *Current Issues in Education*, 14. Citado na pág. 1, 10

Reis (2007) Rosa M. Reis. Development of educational software. *International Journal of Education and Information Technologies*, 1. Citado na pág. 2, 23, 24

Repenning *et al.* **(2001)** Alexander Repenning, Andri Ioannidou, Michele Payton, Wenming Ye e Jeremy Roschelle. Using components for rapid distributed software development. *IEEE Software*, 18:38–45. Citado na pág. 22

Richards *et al.* **(2002)** Griff Richards, Rory McGreal, Marek Hatala e Norm Friesen. The evolution of learning object repository technologies: Portals for on-line objects for learning. *Journal of Distance Education*, 17:67–79. Citado na pág. 10

Rodrigues *et al.* **(2010)** P. A. Rodrigues, L. O. Brandão e A. A. F. Brandão. Interactive assignment: a moodle component to enrich the learning process. *Proceedings of the 40th Frontiers in Education*. Citado na pág. 10, 14

Rodrigues *et al.* **(2011)** Patrícia A. Rodrigues, Márcia Roberta S. P. Silva, Danilo L. Dalmon e Leônidas O. Brandão. O uso de geometria interativa em cursos a distância para motivar e aprofundar conhecimentos de matemática: um estudo com alunos do ensino fundamental ii. Em *Anais do XVII Workshop de Informática na Escola*. Citado na pág. 13

Rodrigues (2011) Patrícia Alves Rodrigues. *iTarefa*: componente moodle para incorporar módulos de aprendizagem interativa em cursos web. Dissertação de Mestrado, Instituto de Matemática e Estatística - USP. Citado na pág. 9, 13

Rosa e Brandão (2010) Eliane R. Marion Santa Rosa e Leônidas O. Brandão. *iRCD*: Proposta de um novo recurso de repositório de conteúdos digitais interativos para o moodle. *Anais do XXI Simpósio Brasileiro de Informática na Educação*. Citado na pág. 10

Roschelle *et al.* **(1998)** Jeremy Roschelle, Jim Kaput, Walter Stroup e Ted M. Kahn. Scaleable integration of educational software: Exploring the promise of component architectures. *Journal of Interactive Media in Education*, 6:1–31. Citado na pág. 2, 22, 24

Roschelle *et al.* **(1999)** Jeremy Roschelle, Chris DiGiano, Manolis Koutlis, Alexander Repenning, Jonathan Phillips, Nicholas Jackiw e Dan Suthers. Developing educational software components. *Computer*, 32:50–58. Citado na pág. 22

Roschelle *et al.* **(2000)** Jeremy Roschelle, Chris DiGiano e Mark Chung. Reusability and interoperability of tools for mathematics learning: Lessons from the escot project. *Proceedings of Intelligent Systems and Applications*, páginas 664–669. Citado na pág. 17, 22

Ruthven *et al.* **(2008)** Kenneth Ruthven, Sara Hennessy e Rosemary Deaney. Constructions of dynamic geometry: A study of the interpretative flexibility of educational software in classroom practice. *Computers and Education*, 51:297–317. Citado na pág. 11

Sampson (2005) Demetrios Sampson. Towards next generation activity-based web-based educational systems. Em *Proceedings of the Fifth IEEE International Conference on Advanced Learning Technologies*, páginas 868–872. Citado na pág. 10, 20

Schleyer e Johnson (2003) T. K. Schleyer e L. A. Johnson. Evaluation of educational software. *Journal of Dental Education*, 67:1221–1229. Citado na pág. 21

Schmidt *et al.* **(2000)** Douglas C. Schmidt, Hans Rohnert, Michael Stal e Dieter Schultz. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*. John Wiley & Sons, Inc., New York, NY, USA, 2nd ed. ISBN 0471606952. Citado na pág. 22

Sentas *et al.* **(2005)** Panagiotis Sentas, Lefteris Angelis, Ioannis Stamelos e George Bleris. Software productivity and effort prediction with ordinal regression. *Information and Software Technology*, 47. Citado na pág. 39

Shanks (1997) Graeme Shanks. The challenges of strategic data planning in practice: an interpretive case study. *Journal of Strategic Information Systems*, 6. Citado na pág. 36, 37, 39

Sims (1997) Rod Sims. Interactivity: A forgotten art? *Computers in Human Behavior*, 13(2): 157–180. Citado na pág. 10, 14

Sommerville (2007) I. Sommerville. *Software engineering*. Addison-Wesley. ISBN 9780321313799. Citado na pág. 2, 4, 42

Sourceforge (2012) Sourceforge. Metrics plugin for eclipse. http://metrics.sourceforge.net/, 2012. Último acesso em 05/06/2012. Citado na pág. 39

Spalter e van Dam (2003) Anne Morgan Spalter e Andries van Dam. Problems with using components in educational software. *Computers and Graphics*, 27:329–337. Citado na pág. 2, 4, 20, 22

Squires e Preece (1999a) David Squires e Jenny Preece. Experiences with reusable learning objects. *Interacting with Computers*, 11:467–483. Citado na pág. 20, 21

Squires e Preece (1999b) David Squires e Jenny Preece. Predicting quality in educational software: - evaluating for learning, usability and the synergy between them. *Interacting with Computers*, 11(5):467–483. Citado na pág. 21

Stuikys e Damasevicius (2008) Vytautas Stuikys e Robertas Damasevicius. Development of generative learning objects using feature diagrams and generative techniques. *Informatics in Education*, 7. Citado na pág. 23, 24

Tang (2005) B. C. Tang. Interactive e-learning activities to engage learners - a simple classification. *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications*, páginas 4092–4097. Citado na pág. 1, 10, 14

Tchounikine (2011) Pierre Tchounikine. *Computer Science and Educational Software Design: A Resource for Multidisciplinary Work in Technology Enhanced Learning*. Springer. ISBN 3642200028. Citado na pág. 1, 20, 24, 41, 87

van Dam *et al.* (2005) Andries van Dam, Sascha Becker e Rosemary Michelle Simpson. Next-generation education software - why we need it and a research agenda for getting it. *Educause Review*, páginas 26–43. Citado na pág. 1, 17, 22, 24

van der Linden *et al.* (2007) Frank van der Linden, Klaus Schmid e Eelco Rommes. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer-Verlag New York, Inc., Secaucus, NJ, USA. ISBN 3540714367. Citado na pág. 15, 27, 38

Wazlawick (2009) Raul Sidnei Wazlawick. *Metodologia de Pesquisa em Ciencia da Computaçao Metodologia de Pesquisa em Ciencia da Computaçao*. Campus. ISBN 9788535235227. Citado na pág. 6, 35

Wiley (2000) D. A. Wiley. Connecting learning objects to instructional design theory: A definition, a metaphor, and a taxonomy. Citado na pág. 19

Winters e Mor (2008) Niall Winters e Yishay Mor. Idr: A participatory methodology for interdisciplinary design in technology enhanced learning. *Computers and Educaction*, 50:579–600. Citado na pág. 2, 20

Yin (2008) Robert K. Yin. *Case Study Research: Design and Methods*. Sage Publications, 4th ed. ISBN 1412960991. Citado na pág. 6, 36, 39