Desenvolvimento de Sistemas de Informação Flexíveis Utilizando *Frameworks* com Separação de Regras de Negócio

Sérgio E. C. Netto
Faculdade Professor Miguel Ângelo da Silva Santos (FeMASS)
sergioecnetto@gmail.com

Asterio K. Tanaka
Universidade Federal do Estado do Rio de Janeiro (UNIRIO), Departamento de Informática Aplicada
tanaka@uniriotec.br

Sahudy M. González **Alan C. Galante**
Universidade Cândido Mendes (UCAM), Campos RJ
Núcleo de Pesquisa em Desenvolvimento de Informática
sahudy@ucam-campos.br, alancarvalho@gmail.com

Resumo

O reuso é um objetivo e uma boa prática da Engenharia de Software. Frameworks e padrões de projeto têm sido usados para elevar o nível de abstração dos artefatos de software reusáveis. O propósito deste trabalho é incrementar o reuso de software, através da junção das técnicas de frameworks / padrões de projeto com a separação de regras de negócio em um processo de desenvolvimento de sistemas de informação. Uma linguagem de padrões de regras é usada para modelagem, especificação e mapeamento das regras de negócio. O processo de desenvolvimento de softwares reutilizáveis com o uso desta técnica traz um ganho substancial em flexibilidade e manutenibilidade, além de reduzir o tempo de desenvolvimento de aplicações do mesmo domínio.

1. Introdução

O presente artigo apresenta a possibilidade de desenvolvimento de projetos de sistemas de informação buscando alto grau de flexibilidade e manutenibilidade através da utilização conjunta de técnicas como *framework*, e separação de regras de negócio. A partir da obtenção de uma estrutura genérica, é possível alcançar maior flexibilidade, adotando-se ainda uma quarta camada responsável pela separação das regras de negócio na área de modelagem. Tal modelagem permite assim o desenvolvimento rápido e alterações facilitadas em aplicações do domínio estudado.

Inicialmente, a reutilização de software era feita através de cópia e alteração de código existente para o emprego em outros programas. Já na década de 70, quando a programação estruturada era o paradigma de desenvolvimento vigente, os módulos de programas eram vistos como partes que poderiam ser reutilizadas em novos sistemas. A reutilização de software é um propósito enunciado quase que simultaneamente com o surgimento da própria Engenharia de Software [2]. Com o surgimento da Orientação a Objetos na década de 80, a reutilização ganhou nova força através da utilização de herança como forma de aproveitamento de código.

Outra forma de implementar a reusabilidade é a utilização de padrões [6]. Estes são soluções consolidadas para um determinado problema de menor dimensão. Entre outras utilizações, os padrões de projeto apóiam a construção de *frameworks*. Pode-se dizer inclusive que um *framework* é composto por diversos padrões, sendo que estes podem estar presentes em diferentes *frameworks*.

O desenvolvimento de *frameworks* é mais uma forma de reutilização. Esta metodologia tem como diferencial o emprego da análise de domínios de aplicação através da identificação e organização do conhecimento a respeito de uma classe de problemas. *Framework* é um conjunto de classes abstratas e concretas que provê uma infra-estrutura genérica de soluções para um conjunto de problemas [9]. Assim, ao desenvolvê-lo, o objetivo principal da análise é capturar e modelar toda a funcionalidade essencial do domínio em questão, sendo possível sua reutilização no desenvolvimento de outros sistemas da mesma família de problemas.

Isto possibilita o reuso de todo o ciclo de desenvolvimento de software, pois como parte do conhecimento do domínio da aplicação, cada novo sistema desenvolvido permite a obtenção de um *framework* mais maduro. Quando desenvolvido, a modelagem é feita em um nível de abstração mais alto. O modelo concebido pode ser instanciado para qualquer realidade que tenha sido estudada dentro do domínio.

Para se desenvolver uma aplicação a partir de um *framework,* o mesmo deve ser instanciado através de seus pontos de especialização, que representam a ligação da parte comum do sistema (núcleo) com a parte específica de cada caso. Estes pontos podem ser vistos, entre outras coisas, como regras de negócio. Estas definem a especificidade de cada realidade, ou seja, em cada aplicação de um domínio é possível que haja diferenças nas regras ou mesmo outras regras. Outro ponto importante a respeito das regras de negócio, é que estas podem ser alteradas com maior ou menor

freqüência em uma aplicação. Esta particularidade motiva uma atenção especial à modelagem das regras de negócio, inclusive sua separação do restante da aplicação.

Sendo assim, a junção de *frameworks* com separação de regras de negócio em um único processo de desenvolvimento, permite a concepção de aplicações únicas, de forma mais rápida, possibilitando ainda alterações de forma ágil e segura, evitando danos à mesma.

2. Regras de negócio e Programação Orientada a Aspectos

A abordagem de regras de negócio possui quatro pontos fundamentais: separar regras de negócio do núcleo da aplicação, seguir passo a passo a transição de regra de negócio para políticas e decisões de negócio, deixar as regras de negócio à mostra para os interessados e preparar as regras de negócio para mudanças.

Embora estes objetivos sejam suportados e amparados por várias abordagens, alguns autores sustentam que estes falham ao fazer a conexão das regras com o núcleo da aplicação, pois estas não têm suporte para encapsular o código com estas características, não o separando totalmente. A este fenômeno dá-se o nome de código *crosscutting,* na área de programação orientada a aspectos (AOP) [3].

O código das regras de negócio encontra-se normalmente espalhado nas classes da aplicação, não podendo ser encapsulado como funções ou objetos separadamente. Para resolver este problema, podem ser utilizados construtores adicionais fornecidos pelas abordagens que suportam a AOP encapsulando e tornando este código reutilizável. No entanto, a técnica proposta neste artigo mostra que é possível fazer a separação das regras de negócio sem estes construtores adicionais.

3. Desenvolvimento

Na sociedade de hoje, os sistemas podem estar funcionando vinte e quatro horas por dia, sete dias na semana, principalmente com o advento da internet. A busca de agilidade e flexibilidade é um ponto de suma importância perante esta realidade. Para se conseguir este objetivo, é fundamental prestar bastante atenção nas regras de negócio. Elas carregam o conhecimento a respeito do negócio que pode ter que ser alterado, sendo assim devem estar disponíveis para mudança a qualquer tempo.

O artigo em questão mostra a possibilidade de se desenvolver projeto de sistema de informação buscando alto grau de flexibilidade e manutenibilidade. Com o desenvolvimento do *framework* consegue-se uma grande flexibilidade, a partir da obtenção de uma estrutura genérica de classes que pode ser instanciada para várias realidades em um determinado domínio de aplicação.

É possível conseguir maior flexibilidade ainda, quando se pensa em algo como uma quarta camada na área de modelagem. Esta quarta camada surge da divisão da camada de negócio, que dá origem a uma camada de regras de negócio. Esta camada centraliza o conhecimento do negócio, separando suas regras em classes de regras. Esta abordagem é conseguida através do mapeamento, separação e posicionamento das regras. Desta forma, é possível realizar mudanças nas regras sem o comprometimento do restante da aplicação.

Quanto à aplicação, a forma de modelagem possibilita alterações fáceis tanto na camada de negócio quanto na camada de regras. Conclui-se que a técnica possibilita o desenvolvimento rápido de aplicações do domínio estudado.

Neste artigo é utilizado um *framework* do tipo caixa branca. Sob essa perspectiva são criadas subclasses provenientes de classes abstratas do *framework*. Os mecanismos de herança possibilitam a criação de aplicações específicas.

Quanto à classificação, este é um *framework* de aplicação empresarial, sendo parte fundamental do funcionamento da empresa [5].

A metodologia utilizada foi a de projeto orientado a pontos adaptáveis, onde se utiliza a identificação de pontos fixos e adaptáveis no *framework*. Foi empregada a modelagem da estrutura de classes genérica utilizando uma linguagem de modelagem, a "Unified Modeling Language" (UML). Em seguida, foi feita a identificação de seus pontos adaptáveis e o grau de flexibilidade desejado. O projeto do *framework* foi feito através da utilização de uma estrutura de classes e padrões visando alcançar a flexibilidade desejada.

Com a participação de especialistas do domínio, verificou-se a flexibilidade alcançada, e se esta atendia aos objetivos desejados, retornando à fase de identificação dos pontos adaptáveis até se alcançar à flexibilidade esperada.

4. Proposta de Trabalho

Quando se fala no desenvolvimento de *frameworks*, podem-se encontrar diversos trabalhos abordando esta metodologia. Não será citado nenhum trabalho específico, pois a técnica proposta aborda o desenvolvimento de *framework* de forma genérica, trazendo ainda um diferencial não apresentado em outros projetos, que é a separação de regras de negócio. Esta tecnologia também é encontrada em outros trabalhos, no entanto diferentemente da bibliografia estudada que trata estas, por exemplo, como aspectos [3], este trabalho faz a separação, modelando as regras através de uma linguagem de padrões própria. Toda a modelagem das regras é feita a partir de uma derivação do padrão *compound rule object* que faz parte de uma linguagem de padrões para o mapeamento de regras de negócio [1], que é instanciado para a realidade deste projeto.

4.1 A Construção de um *Framework*

A construção de um *framework* passa por uma série de etapas, que se repetem até que a estrutura genérica de

classes atenda aos requisitos de flexibilidade, extensibilidade e generalidade desejadas.

Embora estas atividades não necessitem de uma seqüência rígida, estas são descritas abaixo em uma ordem natural de entendimento.

- Identificação da estrutura de classes ou generalização: nesta etapa devem-se identificar as estruturas que se repetem nas realidades do domínio estudadas, sendo ainda identificadas as estruturas semelhantes de forma a obter-se uma que represente genericamente o domínio estudado. Tal generalização é obtida ao se confrontar estruturas implementadas ou não de diferentes aplicações. Podem-se buscar elementos prontos como padrões de projeto para serem reutilizados.
- Identificação dos pontos adaptáveis ou de flexibilização: nesta etapa identificam-se as partes que devem ser mantidas flexíveis (*hot spots*) na estrutura genérica. Desta forma, o *framework* pode ser especializado para possibilitar a geração de diferentes aplicações. Além da localização dos *hot spots* são determinadas as soluções de projeto que participam de sua modelagem.
- Projeto do *framework* e aplicação de metapadrões: após a identificação dos *hot spots*, faz-se necessária a comparação das necessidades que este apresenta com aspectos semelhantes tratados por padrões existentes. Quando um padrão é considerado adequado à solução de um problema, este deverá ser utilizado na solução do problema. A utilização de metapadrões se dá transformando um procedimento genérico em um método (*template*), ligado ao núcleo da aplicação através de métodos (*hook*).
- Aplicação de padrões de projeto: a utilização de padrões de projeto em *frameworks* consiste na inclusão da sua estrutura de classes no modelo em desenvolvimento, ou mesmo em utilizar as classes existentes assumindo as responsabilidades que correspondem às classes do padrão.
- Refinamento da estrutura do *framework*: esta etapa depende especialmente do especialista do domínio que irá verificar se o *framework* alcança o grau de flexibilidade desejado. Caso contrário é retomado o processo até que a flexibilidade desejada seja alcançada.

4.2 Separação de regras de negócio

Uma regra de negócio é: "Uma declaração que define alguns aspectos do negócio. Ela deve ser uma condição ou um fato construído a partir de uma afirmação, uma limitação construída a partir de uma ação ou uma derivação. Ela deve ser atômica não podendo ser quebrada ou decomposta em regras de negócio mais detalhadas" [8].

A separação de regras de negócio é uma das partes principais deste artigo. Desta etapa constam a identificação e separação das regras, acompanhamento de suas mudanças, colocação desta à mostra e posicionamento das mesmas para mudanças.

Ao se identificar regras de negócio, deve-se separá-las e categorizá-las. Em seguida acompanhar suas mudanças objetivando descobrir quais são efetivamente políticas de decisão do negócio, definindo aquelas que realmente devem ser separadas do núcleo da aplicação, ou seja, as regras que fazem parte do conhecimento do negócio e podem ser alteradas com certa freqüência, devem ser separadas do restante da aplicação. Então devem ser colocadas à mostra, ou seja, estar em classes de regra separadas do restante da aplicação. Este mapeamento separado possibilita alterações fáceis e rápidas no código da regra, evitando assim que estas alterações causem problemas no restante da aplicação. Regras de Negócio são aplicadas nos casos onde estão bem definidos os pontos de execução das funcionalidades do núcleo da aplicação [4].

Nesta etapa, para a modelagem das regras, são utilizados padrões pertencentes a uma linguagem própria para o mapeamento de regras de negócio [1], garantindo assim uma solução já amadurecida de modelagem.

4.3 Exemplos da modelagem

Neste artigo é utilizado como estudo de caso, parte do processo de desenvolvimento de um *framework* para sistemas de controle acadêmico. Na seqüência, são apresentados alguns exemplos da modelagem alcançados no desenvolvimento do trabalho. Na Figura 1 pode-se observar um modelo do *framework* obtido da matrícula de alunos.

A separação das regras de negócio traz algumas particularidades ao modelo. Observando-se a Figura 2 atentamente, podem-se observar alguns pontos em que a granularidade muito fina de algumas classes pode trazer dúvidas quanto a sua validade. Há atributos representados por classes e valores representados como subclasses destes. Pode-se explicar tal abordagem no sentido em que há comportamentos associados a estados específicos e métodos pertencentes a eles que também podem apresentar comportamentos distintos. Assim, o comportamento e as regras associadas a estas classes estão acessíveis e flexíveis.

No caso deste artigo, no qual é modelado o processo de matrícula, pode-se observar o ponto onde através da classe *conectorPendencia*, faz-se a ligação com o diagrama de classes de regras de pendência mostrado na Figura 3. A classe *RegraPendenciaMatricula* se liga então à classe *RegraAbstrataMatricula*, fazendo assim a conexão do núcleo da aplicação com as regras separadas.

5. Estudo de caso

Neste tópico é apresentada a aplicação da técnica proposta, utilizando uma instituição de ensino de terceiro grau para a validação do *framework* com separação de regras de negócio. Cabendo ressaltar que é abordada a parte referente à matrícula do aluno em cursos.

5.1 Considerações iniciais

O modelo do *framework* foi desenvolvido com base no objeto do estudo e nas realidades de uma instituição de terceiro grau, além de cursos de pós-graduação, extensão e cursos livres.

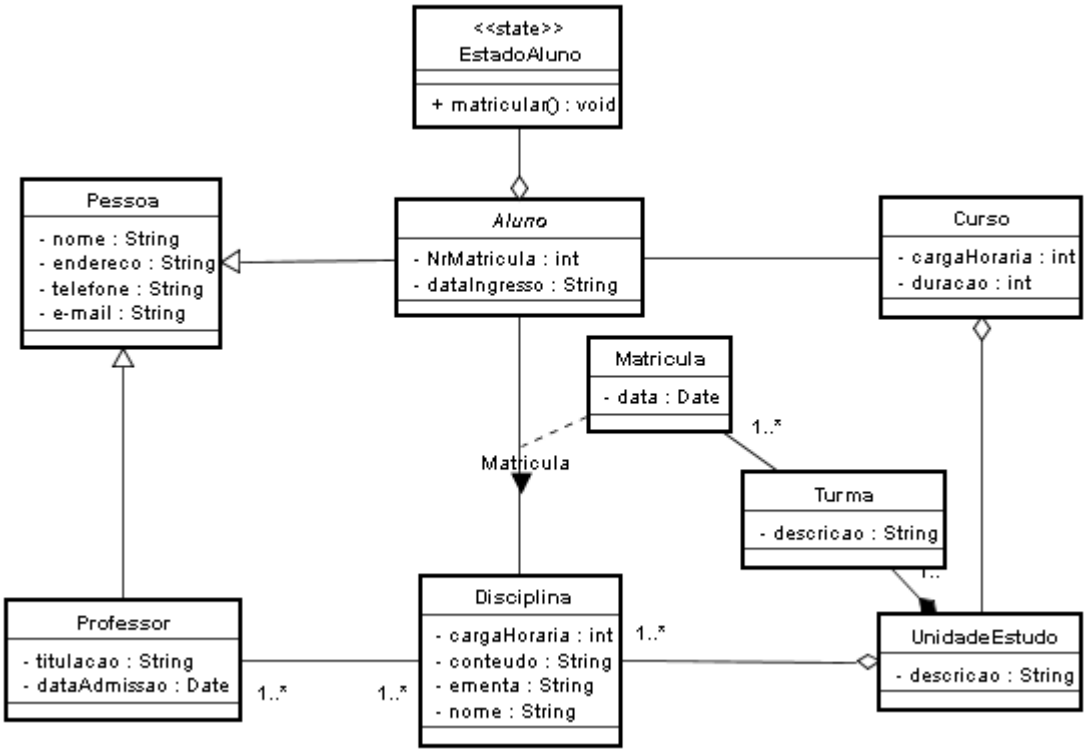

Figura 1. Modelo do *Framework*

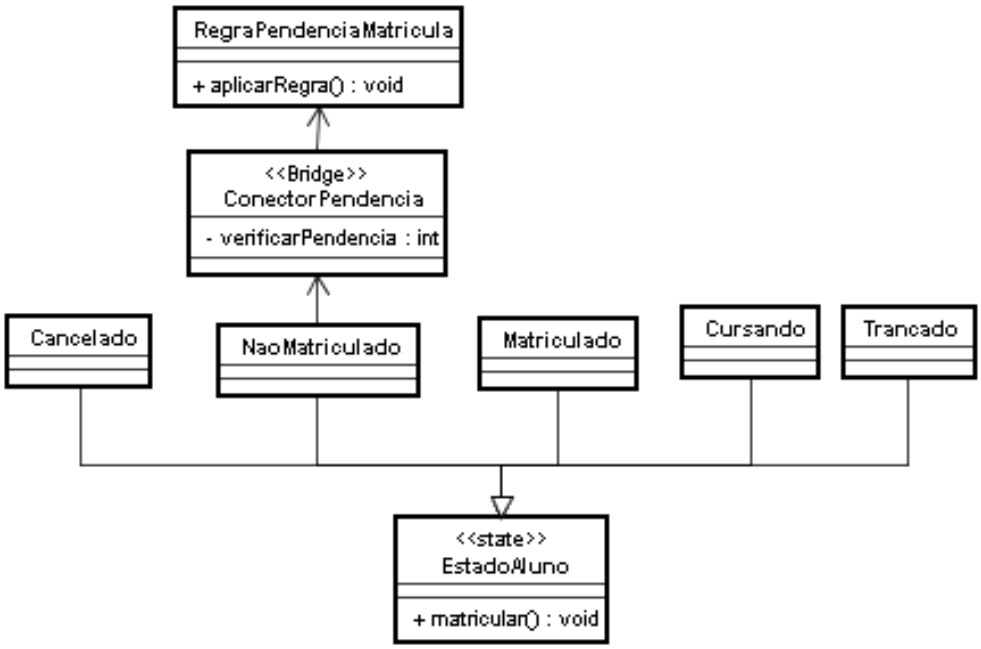

Figura 2. Estado do Aluno (Padrão *State*)

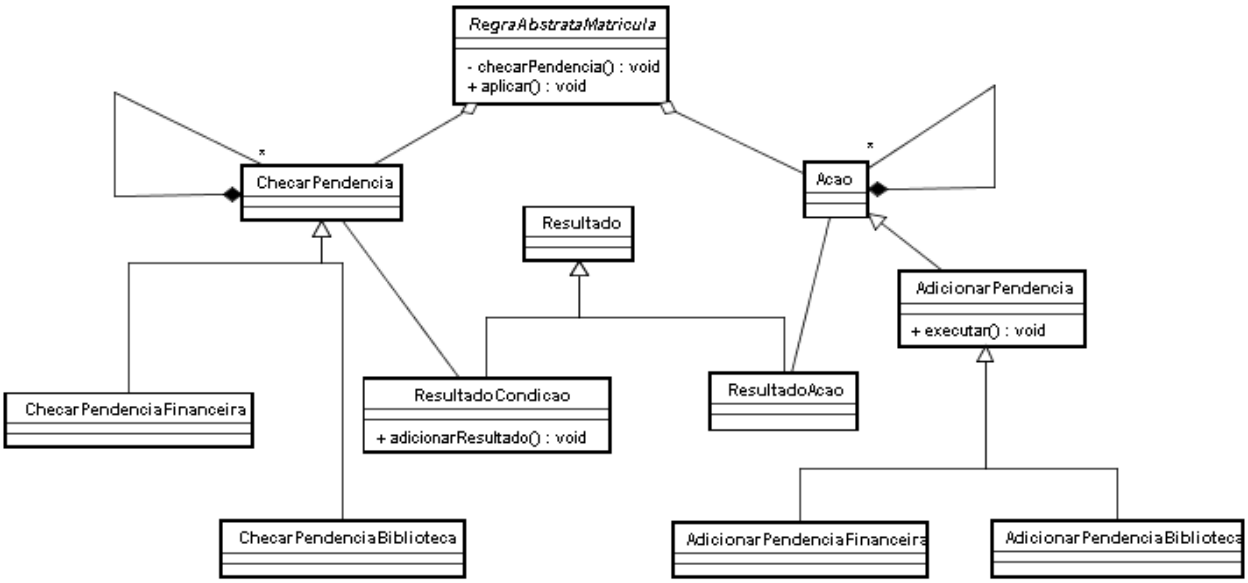

Figura 3. Diagrama de Classe de Regra de Pendência

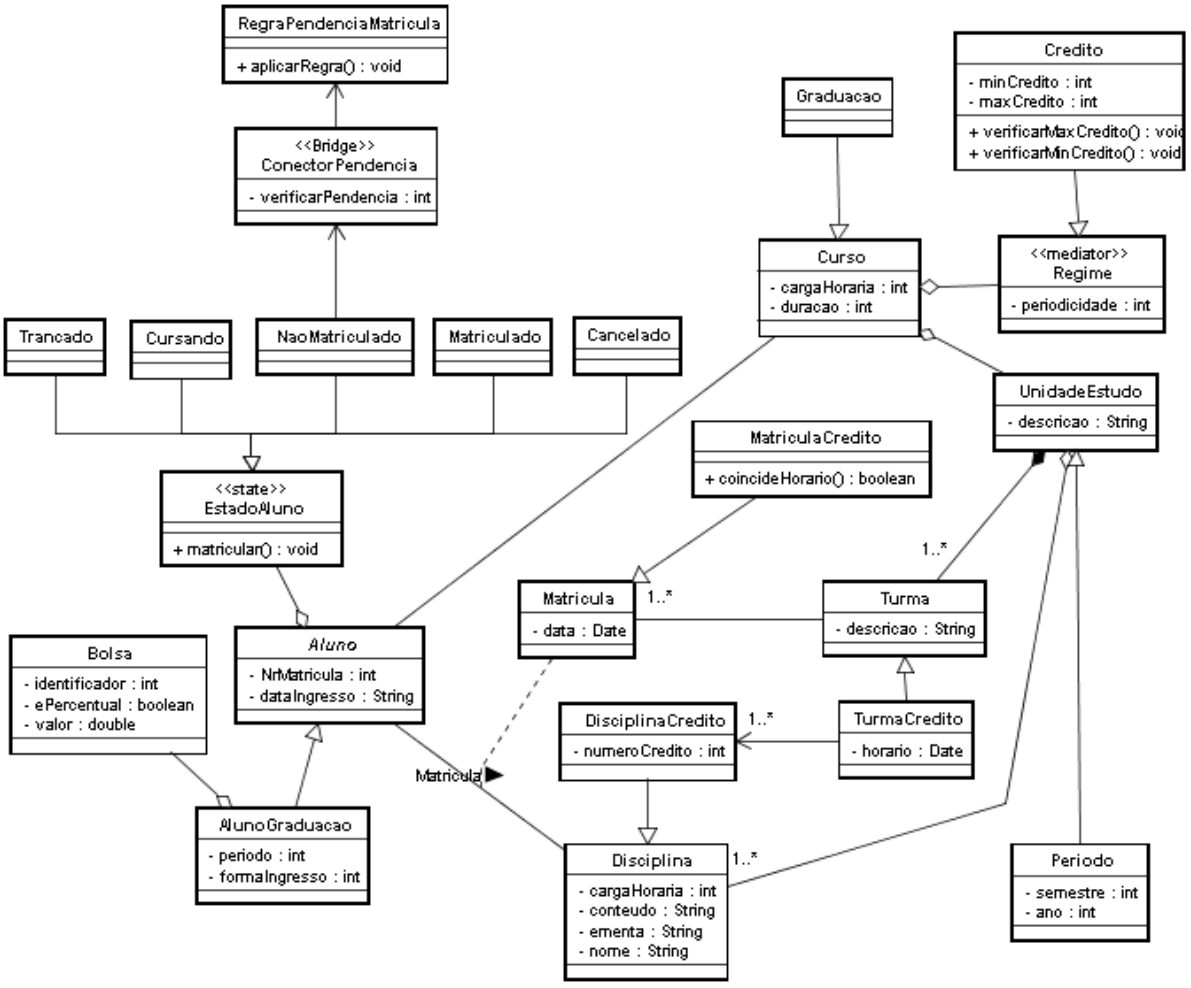

Figura 4. *Framework* Instanciado

5.2 Regras de Matrícula

Será tratada a matrícula em determinado semestre e ano. Verificam-se então a existência de pendências e a possibilidade de se efetivar a matrícula de um aluno de acordo com possíveis impeditivos que possam existir.

Estes impeditivos podem ser neste estudo de caso, pendências oriundas de documentação, da biblioteca ou financeiras.

Como objeto de estudo, é abordada a parte referente à matrícula, fazendo a verificação de pendências para a execução da mesma.

5.3 Modelagem e Desenvolvimento

Com o estudo das realidades dos sistemas de controle acadêmico apoiado na idéia deste artigo, foi feito um trabalho de modelagem até se alcançar os modelos do núcleo da aplicação e das regras.

Após a obtenção do modelo do framework, passa-se a instanciação do mesmo, usando a realidade descrita de uma instituição de 3º grau por regime de créditos, como se vê na Figura 4.

Pode-se observar claramente a existência de uma estrutura pouco usual em modelagem, na parte referente ao estado do aluno. O aluno possui uma subclasse *estado*, que normalmente poderia ser um atributo. No entanto, este estado possui ainda subclasses ligadas a ele, que poderiam ser os valores deste "atributo".

Esta construção é inspirada no padrão *state* [6], onde o comportamento de determinado objeto dependerá de seu estado quando solicitado. Esta parte é extremamente importante neste projeto, pois a validação da separação das regras de negócio passa por aí.

Quando da solicitação de matrícula de um aluno, esta só será executada caso o estado do mesmo se encontre como não matriculado. Embora todos os demais estados possuam o mesmo método herdado da classe pai, a implementação que efetivamente procede a mesma está apenas na classe não matriculado.

A solicitação da matrícula é feita pela secretaria que busca um aluno para tal. Ao efetuar o pedido de matrícula para um aluno não matriculado, passa-se a outra parte fundamental deste trabalho que é a separação das regras de negócio.

A classe não matriculado, utiliza o método *verificaregra()* pertencente à classe *RegraPendenciaMatrícula*. Esta, por sua vez, retornará uma resposta do tipo booleana, habilitando ou não a efetivação da matrícula.

As regras implementadas neste projeto dizem respeito as possíveis pendências existentes para um aluno. Desta forma, a classe *RegraPendenciaMatricula* utiliza o método *checarPendencia()* que irá verificar em todos os tipos de pendências possíveis, oriundas de sistemas externos (financeiro, biblioteca, etc.), retornando um valor também booleano. Caso não haja pendências, a matricula será efetivada.

Quando há pendência para um aluno, a classe que checa tal pendência, por exemplo, *ChecarPendenciaFinanceira*, irá adicionar uma pendência do mesmo tipo, no caso, o método adicionar pendência estará direcionado à classe *AdicionarPendenciaFinanceira,* não permitindo então a efetivação da matrícula.

6. Conclusões e Trabalhos Futuros

O objetivo deste artigo foi o estudo de técnicas de reutilização de software baseadas em *framework*, com a separação de regras de negócio, com o objetivo de desenvolver uma abordagem diferenciada para construção de softwares flexíveis e reutilizáveis. Estas vantagens são conseguidas, inclusive no que diz respeito às regras de negócio dentro de um domínio de aplicação.

A técnica proposta traz benefícios quando da necessidade do desenvolvimento de aplicações de domínio, permitindo o desenvolvimento de diferentes aplicações a partir do *framework* e ainda na facilidade de manutenção, principalmente por tornar as regras de negócio visíveis e alteráveis com grande agilidade.

A proposta de continuação nesta linha de desenvolvimento, é a de inicialmente instanciar por completo dois sistemas de controle acadêmico estudados. Em paralelo, desenvolver formas de inclusão e manipulaçao de regras sem a participação de especialistas, bem como estudar a criação de uma camada de conectividade composta por padrões tornando o projeto ainda mais modular, flexível e extensível.

7. Referências

[1] Arsanjani, A. "Rule Object 2001:A Pattern Language for Adaptive and Scalable Business Rule Construction" PLoP'2000 - Pattern Languages of Programming Conference, 2001.

[2] Bosch, J., Molin, P., Mattsson, M., Bengtsson, P., Fayad, M. "Framework problem and experiences in M. Fayad, R. Johnson, D. Schmidt. Building Application Frameworks: Object-Oriented Foundations of Framework Design", John Willey and Sons, p. 55–82, 1999.

[3] Cibran, M. "Aspect-Oriented For Connecting Business Rules", International Conference on Business Information Systems (BIS'03). Colorado Springs, USA, 2003.

[4] Cibran, M. "Dynamic Business Rules for Web Service Composition", International Conference on Aspect-Oriented Software Development, 2005.

[5] Fayad, M. E., Schmidt, D. C. "Object-oriented application frameworks". Communications of the ACM, v.40 n.10, p. 32-38, Oct. 1997.

[6] Gamma, E, Helm, R., Johnson, R., Vlissides, J. "Design Patterns: Elements of Reusable Object-Oriented Software". Addison-Wesley, Reading, MA, 1995.

[7] Halle B. Von. "Business Rules Applied. Building Better Systems". New York: Wiley, 2001, 464 p.

[8] Hay, D., Healy, K. A. "Defining business rules - what are they really? Guide project by business rules", 1995. Disponível em: <www.guide.org/ap/apbrules.htm>.

[9] Johnson, R.; Foote, B. Designing reusable classes. Journal of Object Oriented Programming, SIGS, June/July 1988, pp. 22–35.