Componentes de um Modelo para Adaptação de Processos de Software

[1]Anderson Baia Maia, [1]Ana Vitoria Piaggio de Freitas, [1]Daltro José Nunes, [1]Igor Steinmacher

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – CEP 91.501-970 – Porto Alegre – RS – Brasil

{abmaia, avpfreitas, daltro, ifsteinmacher}@inf.ufrgs.br

Resumo: O processo padrão é o conjunto de elementos fundamentais que se deseja sejam incorporados em qualquer processo de desenvolvimento definido para os projetos de uma dada organização de software, garantindo assim conformidade com os padrões de qualidade e procedimentos adotados na mesma. Por ser genérico, o processo padrão precisa ser adaptado às necessidades específicas de cada projeto da organização de software, de forma a ser aceito, ter seu uso maximizado e garantir a qualidade do software a ser produzido. Este artigo apresenta, em linhas gerais, os componentes de um modelo para auxiliar a adaptação do processo padrão de uma organização de software para um projeto específico a ser conduzido na mesma, baseado nas características do projeto, nas diretrizes de adaptação do processo padrão e nas informações acerca de adaptações realizadas anteriormente. Visando permitir clareza e precisão de entendimento, é adotada uma notação formal de fácil compreensão, o Prosoft-Algébrico, para a apresentação destes componentes.

Palavras-Chave: Engenharia de Software, Processo de Software, Adaptação de Processos de Software, Processo Padrão e Prosoft-Algébrico.

Tema: Engenharia de Software

Evento: Workshop de Engenharia de Software e Banco de Dados (WISBD)

1 Introdução

Duas das principais conseqüências do grande crescimento experimentado pela indústria do software nas últimas décadas são o aumento da complexidade do software e as exigências cada vez maiores do mercado. É exigido das organizações desenvolvedoras de software (ou, de forma abreviada, organizações de software) que os sistemas de software sejam desenvolvidos com prazo e custo determinados e obedeçam a padrões de qualidade [1]. Para atender essas exigências, tornou-se necessário investir no processo de desenvolvimento de software (simplesmente chamado de processo de software), dada a clara relação entre a sua qualidade e a do software produzido [2].

Uma abordagem encontrada na literatura para o aumento da maturidade em termos de processos é a definição de um processo padrão para a organização. O processo padrão é o conjunto de elementos fundamentais que se deseja sejam incorporados em qualquer processo de desenvolvimento definido para os projetos de uma dada organização de software, garantindo assim conformidade com os padrões de qualidade e procedimentos adotados na mesma [3] [4]. Humphrey, em [5], define um conjunto de razões para a definição de um processo padrão:

- Redução dos problemas relacionados a treinamento, revisões e suporte a ferramentas;
- As experiências adquiridas nos projetos são incorporadas ao processo padrão e contribuem para melhorias em todos os processos de software definidos;
- Possibilidade de medições de processo e de avaliação da qualidade;

- Economia de tempo e esforço em definir novos processos de software adequados a projetos de desenvolvimento.

Por ser genérico, o processo padrão precisa ser adaptado às necessidades específicas de cada projeto de software, de forma a ser aceito, ter seu uso maximizado e garantir a qualidade do software a ser produzido. Organizações de software, que possuem um processo padrão definido e são capazes de adaptá-lo para cada um de seus projetos, obtêm uma avaliação positiva diante de modelos de maturidade como o SW-CMM [6] e o ISO/IEC TR 15504 (SPICE) [7], e conseqüentemente diante do mercado.

Segundo Ahn et al [8] e Xu et al [9], como a adaptação de processos de software é uma atividade que requer uso intensivo de vários tipos de conhecimento, boa parte de sua realização depende de profissionais especializados (denominados de engenheiros de processos) e de atividades manuais, o que a torna consumidora de tempo, custosa e sujeita a erros. Desta forma, o aumento no nível de automatização desta atividade traz consigo os seguintes benefícios:

- Redução de tempo e custo despendidos durante a definição de processos de software, o que tem impacto direto no prazo de entrega do produto de software final requerido;
- Simplificação do trabalho do engenheiro de processos, aumentando assim sua produtividade e a possibilidade de obtenção de processos de software mais adequados à situação corrente.

Tendo em vista a importância do processo padrão e a sua necessidade de adaptação, este artigo apresenta, em linhas gerais, um modelo para auxiliar o engenheiro de processos durante a adaptação do processo padrão de uma dada organização de software para um projeto específico da mesma, baseado nas características do projeto, nas diretrizes de adaptação do processo padrão e nas informações acerca de adaptações realizadas anteriormente. Especial ênfase é dada na apresentação dos componentes deste modelo, visando esclarecer o papel de cada um e destacar a influência dos mesmos na definição da estratégia de adaptação utilizada. Assim, buscando um entendimento claro e preciso, é adotada uma notação formal de fácil compreensão para a apresentação de tais componentes, o Prosoft-Algébrico.

A estrutura do artigo é como segue. Na seção 2, é apresentada uma visão geral da arquitetura e do funcionamento do modelo de adaptação proposto. A seção 3, centro do artigo, apresenta em detalhes os componentes deste modelo. A seção 4 compara o modelo proposto com alguns trabalhos correlatos, no que diz respeito à abordagem de adaptação e aos componentes utilizados. Por fim, a seção 5 apresenta as conclusões do artigo.

2 Visão Geral do Modelo de Adaptação

Nesta seção, é apresentado um modelo para auxiliar o engenheiro de processos durante a atividade de adaptar o processo padrão de uma organização de software para um projeto específico a ser conduzido na mesma. Partindo do fato de que a adaptação é uma atividade que utiliza intensivamente vários tipos de conhecimento, o modelo aqui proposto utiliza uma abordagem orientada a atividades e baseada:

- No suporte ao gerenciamento (representação, armazenamento e utilização) do conhecimento necessário à adaptação (características do projeto de software, diretrizes que guiam a adaptação do processo padrão e o conhecimento adquirido durante adaptações realizadas anteriormente);
- Em mecanismos capazes de raciocinar sobre este conhecimento.

O diagrama superior da figura 1, descrito utilizando a notação SADT (*Structured Analysis and Design Technique*) [10], apresenta uma visão caixa-preta do modelo,. Nela, pode-se notar que: a adaptação é realizada em conjunto pelo engenheiro de processos e pelos mecanismos que realizam o

raciocínio sobre o conhecimento disponível; o modelo de adaptação é parametrizável, devendo ser configurado com o processo padrão da organização, as diretrizes (regras) que guiam a adaptação do processo padrão e o conjunto de características que poderão ser utilizadas para caracterizar um dado projeto de software. É bom ressaltar que, a qualquer momento, exceto durante uma adaptação, estes parâmetros podem ser alterados de acordo com as necessidades da organização de software.

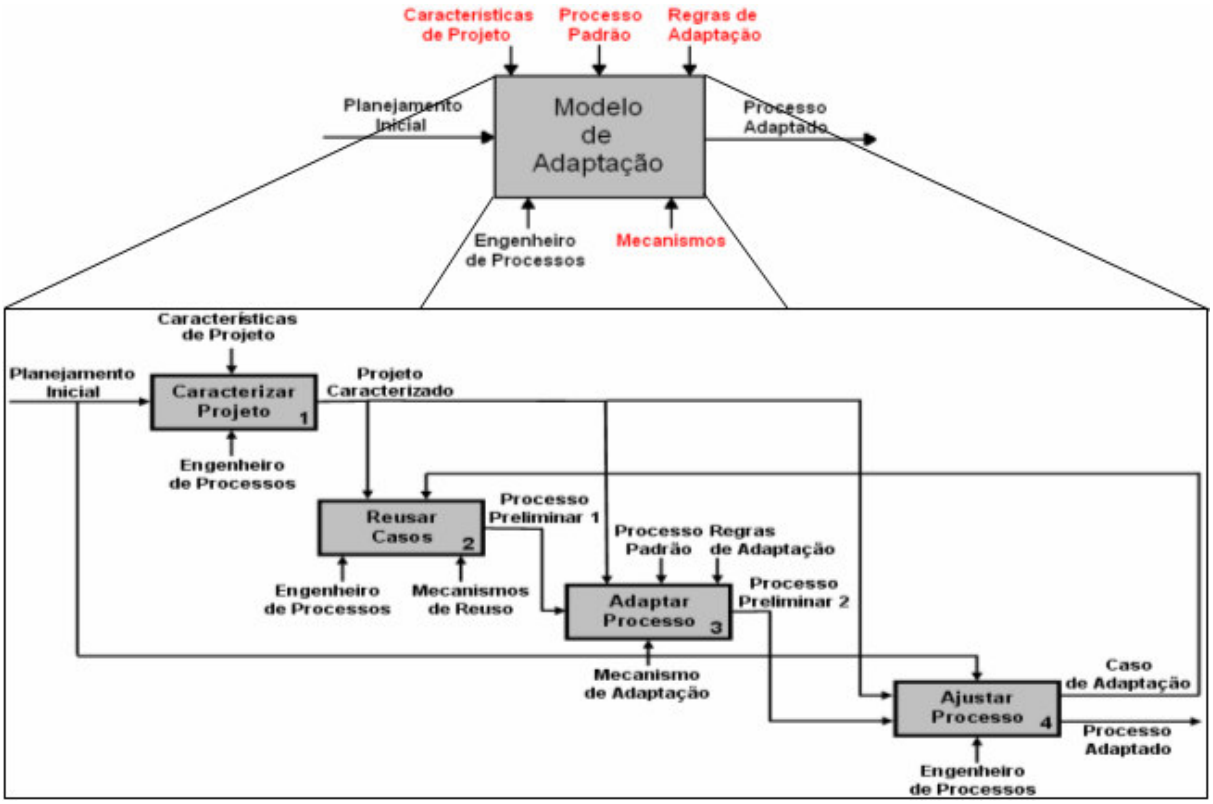

Fig. 1: Visão geral do modelo de adaptação

O diagrama inferior da figura 1 (também descrito utilizando a notação SADT) fornece uma visão geral das atividades realizadas durante a adaptação de processos de software, utilizando o modelo proposto e assumindo que o mesmo já tenha sido configurado com os parâmetros necessários. Na fase **Caracterizar Projeto**, baseado em um planejamento inicial e no conjunto de características disponíveis, o engenheiro de processos define as características do projeto de software em questão. Na fase seguinte, **Reusar Casos**, de posse do projeto caracterizado, o modelo proposto aplica um método de comparação para buscar, em um repositório de casos de adaptação, o projeto mais semelhante e reutilizar o processo adaptado associado, com alguma adaptação se necessário. A intervenção do engenheiro de processos pode ser necessária nesta fase, se mais de um caso de adaptação tiver sido recuperado ou em situações onde um elemento do processo padrão tiver sido incluído ou excluído no/do processo adaptado recuperado, em decisão contrária à do modelo. Na fase **Adaptar Processo**, é feita uma varredura nas atividades do processo padrão em busca daquelas que ainda não foram incluídas no processo gerado na fase anterior. Para cada uma dessas atividades, se não existirem restrições de uso (condições a serem testadas sobre as características do projeto corrente), as mesmas serão incluídos diretamente no processo adaptado. Caso contrário, essas condições serão primeiramente testadas. O resultado desta fase é um processo adaptado que pode ser livremente alterado pelo engenheiro de processos na fase **Ajustar Processo**, de acordo com as necessidades do projeto. Feitos os ajustes manuais necessários, é gerado um caso de adaptação e o processo adaptado pronto para ser instanciado (alocação de recursos e agentes, e

definição de cronograma) e utilizado no projeto de software real. Todavia, vale ressaltar que o suporte à instanciação e execução de processos de software está fora do escopo deste trabalho.

3 Componentes do Modelo de Adaptação

Devido à diversidade de aspectos envolvidos na definição do modelo de adaptação proposto, optou-se por especificar formalmente os seus tipos de dados e algoritmos, de tal sorte que pudesse ser gerada uma especificação precisa e em alto nível de abstração, de modo a ser utilizada como base semântica para auxiliar no entendimento e na evolução futura da proposta. Além disso, segundo Wang e King, referenciados em [11], o campo de automação de processos de software é um terreno fértil para o desenvolvimento de soluções baseadas em métodos formais: a literatura especializada apresenta experiências com diferentes formalismos, incluindo LOTOS, Redes de Petri, CCS, Método Algébrico e Gramática de Grafos. Para a especificação dos componentes do modelo proposto, utilizou-se o Prosoft-Algébrico como formalismo, dentre outros motivos, pela possibilidade de derivação direta para implementação no Prosoft-Java[1], visto que há uma correspondência semântica entre os elementos usados na especificação e na implementação dos componentes de software descritos neste paradigma.

Nas próximas seções, são apresentadas uma visão geral do Prosoft-Algébrico e a descrição detalhada dos componentes do modelo proposto.

3.1 Prosoft-Algébrico

O Prosoft, descrito em [12], como método de especificação formal, apóia a reutilização e a modularização das especificações de TADs (Tipos Abstratos de Dados). A modularização torna possível quebrar o problema em partes menores e portanto mais fáceis de solucionar. A reutilização simplifica a especificação do TAD, tornado-a mais simples e mais rápida de ser construída.

Dado um desenvolvimento de software, os tipos de dados deste software podem ser especificados utilizando o Prosoft-Algébrico (estratégia *data-driven*). No Prosoft-Algébrico, cada ATO (Ambiente de Tratamento de Objetos) especifica somente um TAD. A construção de um ou mais ATOs representa uma solução do problema (software). Como visto na figura 2, lado esquerdo, os ATOs são integrados através da Interface de Comunicação do Sistema (ICS).

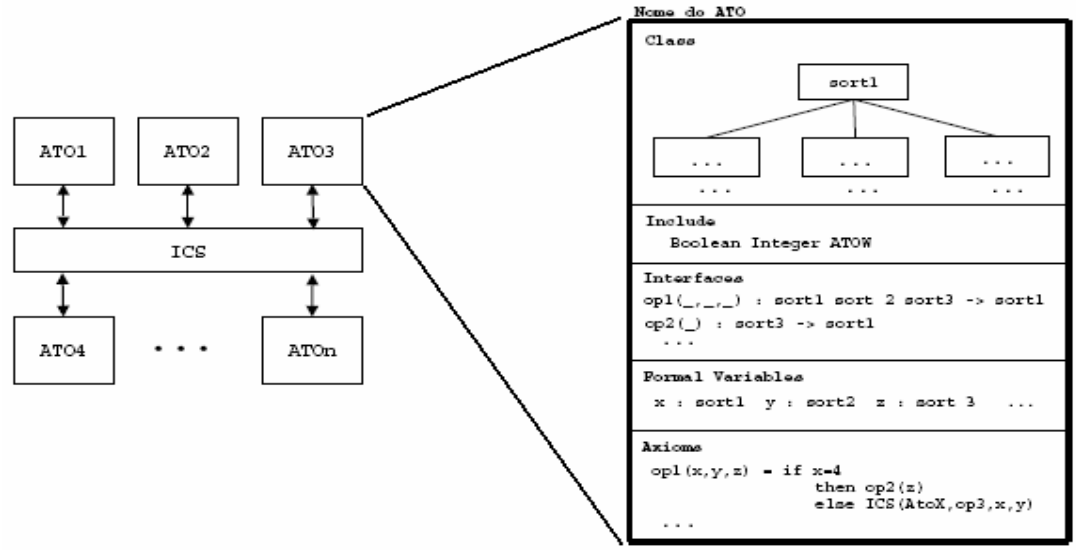

Fig. 2: Composição de ATOs algébricos na descrição de software no Prosoft

[1] Prosoft-Java é a denominação do ambiente escolhido para prototipação do modelo de adaptação proposto.

Ainda de acordo com a figura 2, lado direito, um ATO é composto de cinco partes. A classe define o TAD do ATO através da instanciação de especificações parametrizadas. A segunda parte define as importações. As partes seguintes especificam a funcionalidade (assinatura) das novas operações algébricas sobre o tipo de dado, as variáveis formais e a semântica das novas operações (axiomas). Como a apresentação detalhada da estrutura de um ATO foge ao escopo deste artigo, o leitor interessado pode buscar mais informações em [13]. Tendo em vista o cumprimento do limite de páginas do artigo, serão apresentadas apenas as classes dos ATOs, o que é suficiente para descrever os componentes do modelo proposto. Assim, entrar-se-á aqui em mais detalhes sobre a notação gráfica para construção de classes do Prosoft-Algébrico.

No Prosoft-Algébrico, existem dois grupos de tipos de dados: primitivos (Integer, Boolean, String, Date e Time) e compostos (Conjunto, Lista, Mapeamento, Registro e União Disjunta). A classe de um ATO é definida através da instanciação dos tipos compostos. Exemplos de classes são: lista de inteiros, lista de conjunto de boleanos, etc.

Para facilitar a utilização do método Prosoft-Algébrico, foi definida uma poderosa representação gráfica, inspirada nos diagramas de Jackson [14], para definição de classes. Cada tipo composto possui uma representação gráfica na forma de árvore. Na instanciação, a raiz da árvore descreve o *sort* definido pelo ATO, os nodos são tipos de dados compostos e as folhas são referências a outros tipos de dados. Os tipos compostos e suas representações gráficas são apresentados na figura 3, através de exemplos. Para cada tipo composto e primitivo do Prosoft-Algébrico, existem operações geradoras, modificadoras e observadoras.

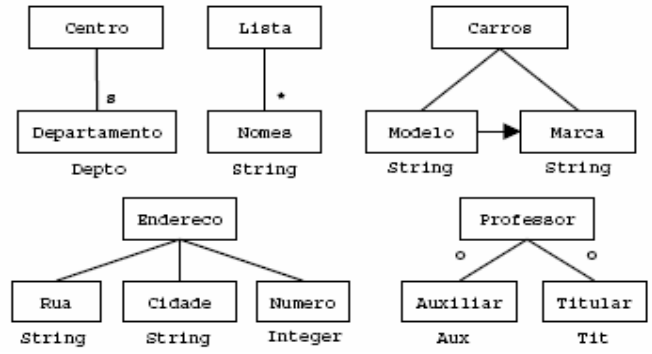

Fig. 3: Representação gráfica dos tipos compostos do Prosoft-Algébrico

3.2 Modelo de Adaptação

Conforme visto na figura 4, o modelo de adaptação proposto (classe *AdaptationModel*) é composto basicamente: pelo processo padrão (atributo *Standard*) da organização de software; pelas regras (atributo *Rules*) que guiam a adaptação do processo padrão; pelos projetos de software (atributo *Projects*) conduzidos na organização de software; pelas características (atributo *CharacteristicsTypes*) utilizadas para definir os projetos de software; pelos processos de software (atributo *Processes*) originados a partir da adaptação do processo padrão para um projeto de software específico da organização; e por fim, pelos casos de adaptação (atributo *Cases*), os quais armazenam informações acerca de adaptações realizadas anteriormente. O atributo *Configuration* armazena informações sobre o armazenamento físico dos componentes do modelo.

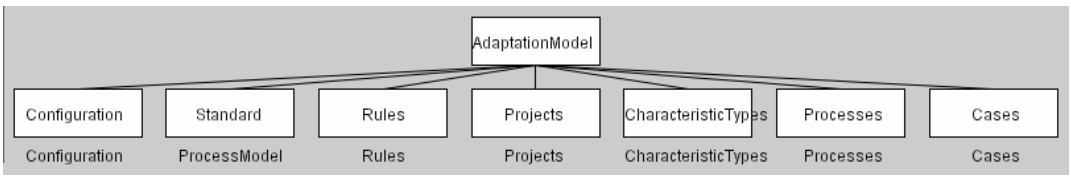

Fig. 4: A classe *AdaptationModel*

3.3 Processo Padrão e Processos Adaptados

Conforme pode ser visto na figura 5, tanto o processo padrão quanto os processos adaptados originados a partir deste, possuem sua estrutura básica representada pela mesma classe, denominada *ProcessModel*. A diferença básica é que como o modelo proposto trata da adaptação de um único processo padrão, não há a necessidade de um identificador único para o mesmo. O mesmo não se pode dizer sobre os processos adaptados. A justificativa para o uso da mesma classe para representar a estrutura de ambos os tipos de processos é que, para o modelo proposto, eles estão no mesmo nível de abstração, ou seja, ambos são modelos de processo abstratos. Este fato justifica a necessidade de instanciação do processo adaptado para sua aplicação em um projeto de software real. Mais uma vez, é bom relembrar que a instanciação está fora do escopo deste trabalho.

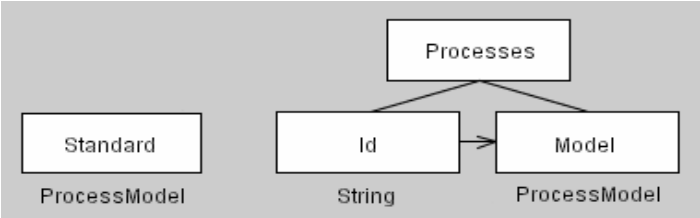

Fig. 5: As classes *Standard* e *Processes*

Em virtude de a classe *ProcessModel* ser relativamente grande (formada por 13 sub-classes), ela não será apresentada na íntegra neste artigo, sendo, no entanto, exemplificada. Basicamente, um modelo de processo é formado pelo conjunto de atividades a serem realizadas e pelas conexões entre as mesmas. Para cada atividade, podem ser indicados, dentre outros, os papéis dos agentes que a desempenham, os seus artefatos de entrada e saída e os recursos necessários. É importante ressaltar que a modelagem de processos de software com a estrutura proposta deve ser suportada em uma ferramenta externa e a seguir convertida para um formato XML (este formato já está definido, no entanto não será apresentado neste artigo) reconhecível pelo modelo proposto. A figura 6 mostra um modelo de processo representado visualmente pela linguagem APSEE-PML[2], de acordo com a estrutura da classe *ProcessModel*. Este modelo representa as etapas para desenvolvimento de software, utilizando o paradigma Prosoft. Nesta figura, elipses representam atividades, setas pontilhas e contínuas com retângulo representam conexões e caixas representam artefatos.

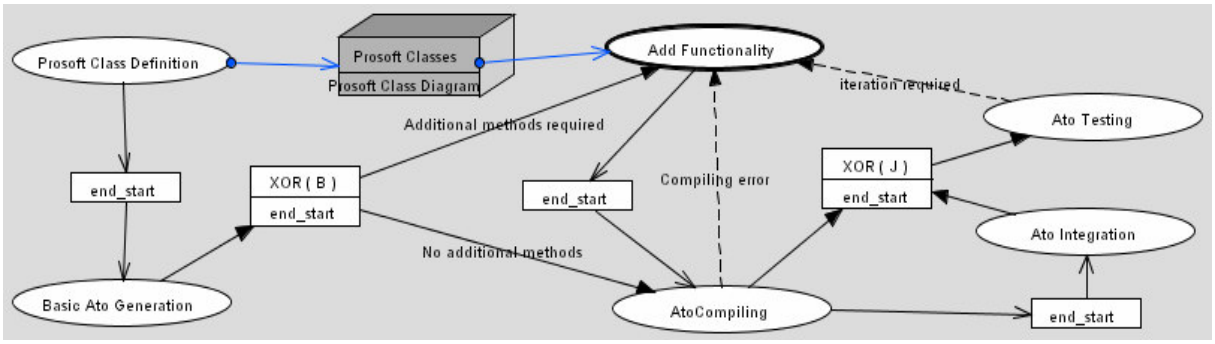

Fig. 6: Um objeto da classe *ProcessModel*

3.4 Regras de Adaptação

As regras são as diretrizes que guiam a adaptação, sendo específicas de cada processo padrão. Basicamente, uma regra de adaptação indica sob que condições determinada atividade do processo padrão deve ser incluída ou não no processo adaptado, daí dizer-se que a abordagem utilizada é

[2] A APSEE-PML, descrita em [15], é a linguagem visual de modelagem utilizada pelo ambiente APSEE, um ambiente de desenvolvimento de software orientado ao processo, que foi especificado e prototipado seguindo o paradigma Prosoft.

orientada a atividades. A parte condicional de uma regra de adaptação é formada por testes sobre os valores das características de projeto para o projeto de software corrente. No modelo proposto, as regras de adaptação são representadas pela classe *Rules*, a qual é apresentada na figura 7. Conforme mostrado nesta classe, uma regra de adaptação pode estar associada a uma e somente uma atividade (atributo *ActivityId*), pode possuir comentários associados (atributo *Comments*), possui um tipo (atributo *Type*, que indica se a regra expressa as condições para inclusão – *Positive* – ou para não inclusão – *Negative* – da atividade no processo adaptado) e, por fim, as condições propriamente ditas (atributo *Condition*).

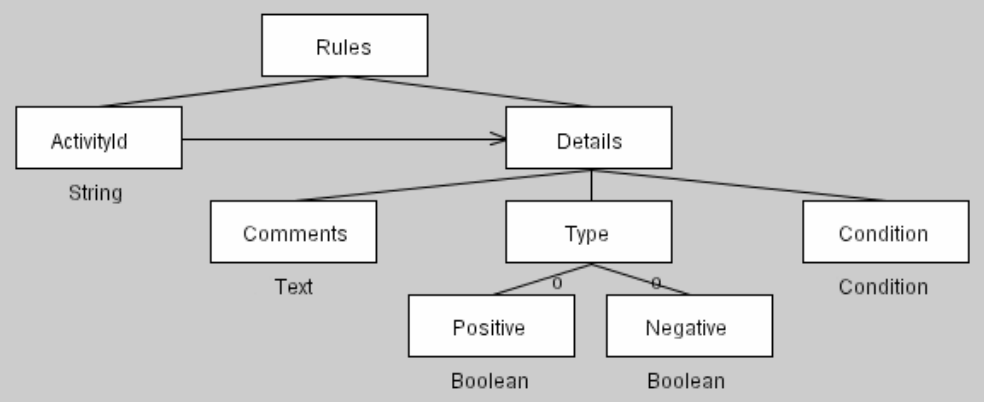

Fig. 7: A classe *Rules*

A gramática expressa na figura 8 especifica o formato da parte condicional de uma regra de adaptação.

```
<conditon>               → <expression> [<optional_connection>]
<expression>             → <characteristic_type_id> <comparison_type> <value>
<optional_connection>    → <connection_type> <condition>
<characteristic_type_id> → <string>
<comparison_type>        → = | <> | > | >= | < | <= | contains | not_contains
<value>                  → <string> | <set_of_string>
<connection_type>        → and | or
```

Fig. 8: Gramática para especificação da parte condicional de uma regra de adaptação

Considerando, por exemplo, um processo padrão que possuísse uma atividade chamada Gerenciamento de Versões, a qual dependesse das características Criticidade do Projeto (cujos possíveis valores são "pequena", "média" e "grande") e Tamanho da Equipe (cujos possíveis valores são "muito pequeno", "pequeno", "médio", "grande" e "muito grande") para ser aplicada, poder-se-ia associar a esta atividade a regra de adaptação apresentada na figura 9.

Atividade: "Gerenciamento de Versões"
Comentários: "Esta atividade está em conformidade com a política da organização para controle e versionamento do software produzido em projetos críticos e com equipes relativamente grandes"
Tipo: Positiva
Condição: CriticidadeProjeto = "alta" **and** TamanhoEquipe >= "médio"

Fig. 9: Exemplo de regra de adaptação

É importante ressaltar que nem toda atividade do processo padrão precisa ter uma regra de adaptação associada, entretanto, se não o tiver, será automaticamente incluída no processo adaptado durante a adaptação do processo padrão.

3.5 Projetos de Software

A classe *Projects*, apresentada na figura 10[3], representa os projetos de desenvolvimento de software conduzidos no âmbito da organização que está utilizando o modelo proposto. Conforme pode ser visto, um projeto de software possui um identificador único (atributo *Id*), um nome (atributo *Name*), uma descrição (atributo *Description*), um gerente (atributo *Manager*) responsável pela sua condução, uma data de início (atributo *Start*), uma data de término esperada (*ExpectedEnd*) e um conjunto de características (atributo *Characteristics*), cada uma possuindo um valor (atributo *Value*) e um peso representando sua importância (atributo *Weight*). Para cada projeto de software, deve ser gerado, através do modelo proposto, um processo de software adaptado às características específicas do projeto, a partir do processo padrão da organização.

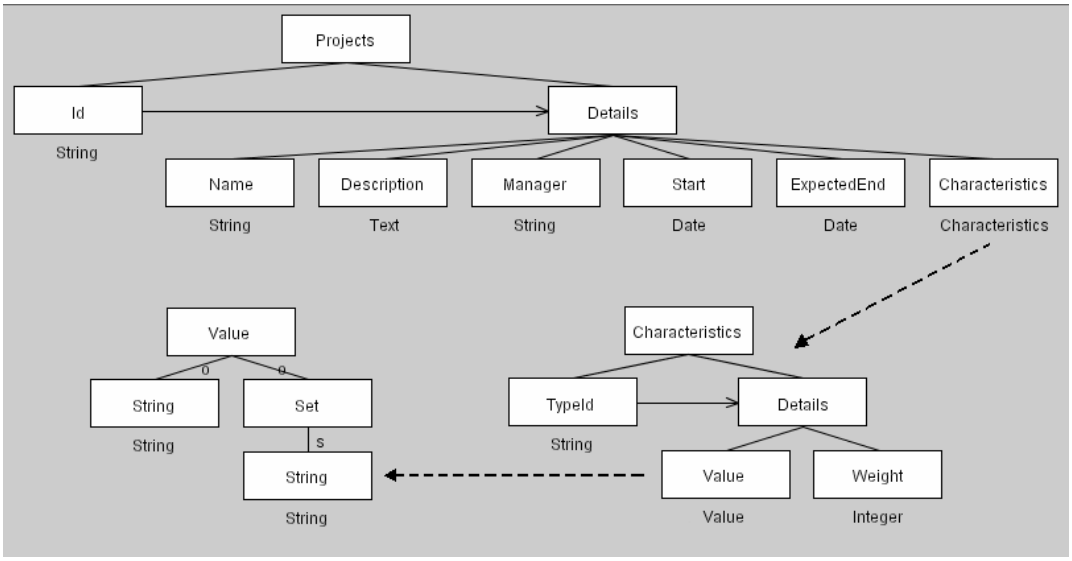

Fig. 10: As classes *Projects*, *Characteristics* e *Value*

Na figura 11, é apresentado um exemplo de objeto desta classe, com o objetivo de tornar mais claro o que é um projeto de software para o modelo proposto.

Id: "ProjetoExemplo"
Name: "Projeto de desenvolvimento Web"
Description: "Aqui o projeto é descrito, seus objetivos e suas descrições são apresentados"
Manager: Anderson Baia Maia
Start: 99/99/9999
ExpectedEnd: 99/99/9999
Characteristics: TipoSoftware = "Aplicação Web", CriticidadeProjeto = "Média", RiscosTecnicos = {"restrições de H/W e S/W existentes", "interface com sistema legado"}

Fig. 11: Exemplo de projeto de software

3.6 Características de Projeto

A classe *CharacteristicTypes* representa o conjunto de características definidas pela organização de software e que podem ser utilizadas para caracterizar um determinado projeto de software conduzido na mesma. Conforme apresentado na figura 12, cada característica de projeto possui um identificador único (atributo *Id*), um nome (atributo *Name*), uma descrição (atributo

[3] Em nível de esclarecimento, vale mencionar que as setas pontilhadas, presentes na figura, não fazem parte da notação de classes do Prosoft-Algébrico, e são incluídas em várias figuras deste texto como o objetivo de facilitar o entendimento dos relacionamentos entre as classes apresentadas.

Description), um tipo e um conjunto de valores que pode assumir (atributo *Kind*). É interessante notar que, para ser utilizada em determinado projeto de software, uma característica deve ser instanciada, ou seja, deve ser associada a um ou mais de um (dependendo do seu tipo) valor dentre os possíveis, bem como lhe deve ser atribuído um peso (importância) dentro do projeto em questão.

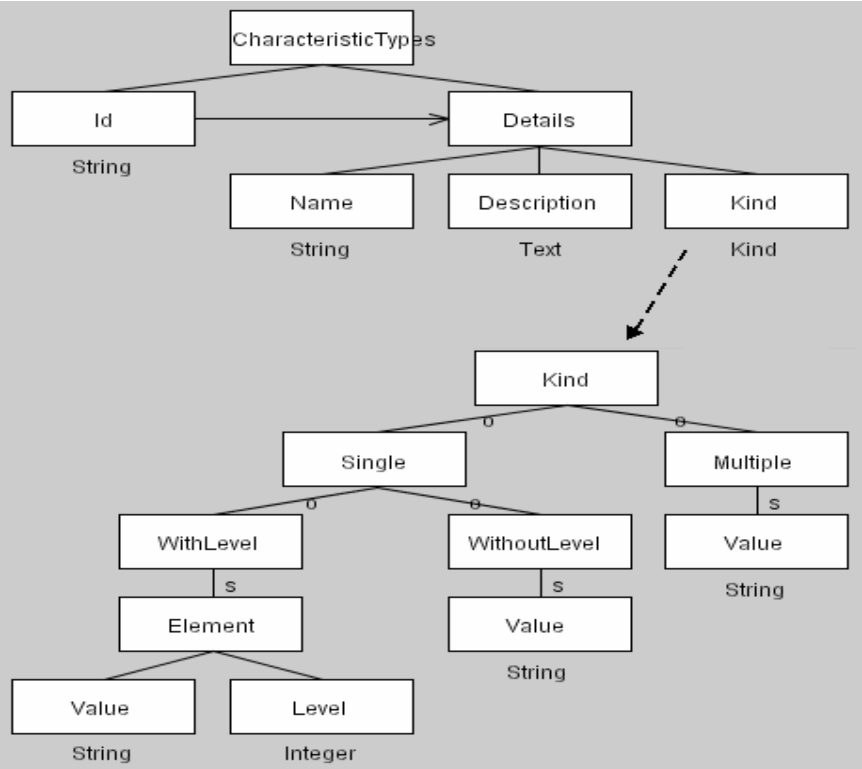

Fig. 12: As classes *CharacteristicTypes* e *Kind*

No modelo de adaptação proposto, as características de projeto são agrupadas em três categorias:

- Características simples e sem nivelamento (*Single* → *WithoutLevel*): suas instâncias podem assumir um único valor dentre os possíveis, não havendo relação de ordem entre os mesmos. Por exemplo, uma instância da característica Tipo de Software (cujos possíveis valores são "Aplicação Web", "Aplicação Desktop", "Sistema de Tempo Real" e "Sistema Embarcado") poderia assumir apenas um destes valores em determinado projeto de software;
- Características simples e com nivelamento (*Single* → *WithLevel*): suas instâncias podem assumir um único valor dentre os possíveis, havendo uma relação de ordem entre os mesmos. Cada possível valor de uma característica deste tipo está associado a um nível, que indica a sua ordem. Por exemplo, uma instância da característica Tamanho do Projeto, cujos possíveis valores são "pequeno" (nível 1), "médio" (nível 2) e "grande" (nível 3), poderia assumir apenas um destes valores em determinado projeto de software;
- Características múltiplas (*Multiple*): suas instâncias podem assumir um subconjunto de valores dentre os possíveis, não havendo relação de ordem entre os mesmos. Por exemplo, uma instância da característica Risco Técnico (cujos possíveis valores são "tecnologia imatura", "interface de sistema complexa", "restrições de H/W e S/W existentes", "ausência de metodologia de desenvolvimento", "requisitos de performance e segurança", "interface com sistema legado") poderia assumir um subconjunto destes valores em um determinado projeto de software.

3.7 Casos de Adaptação

São registros de adaptações já realizadas, de modo a armazenar o conhecimento necessário para ser utilizado em adaptações futuras, através da técnica de Raciocínio Baseado em Casos. A utilização de casos de adaptação possibilita o reuso de experiências bem sucedidas anteriormente, bem como reduz o esforço necessário para realizar novas adaptações. Para o modelo proposto, conforme apresentado na figura 13, um caso de adaptação é formado basicamente por três partes: o problema (projeto de software conduzido na organização, representado pelo atributo *ProjectId*), a solução (processo de software adaptado às características deste projeto, representado pelo atributo *ProcessId*) e a avaliação da aplicação do processo adaptado no projeto de software real (atributo *Avaliation*), a qual servirá como informação de apoio à decisão do engenheiro de processos pela escolha de um caso de adaptação, no caso de dois ou mais projetos serem similares ao projeto corrente. Além destes atributos, um caso de adaptação possui comentários associados (atributo *Comments*), uma data de realização (atributo *Realization*) e o conjunto de alterações realizadas manualmente no processo adaptado, a contragosto do modelo de adaptação (atributo *Modifications*).

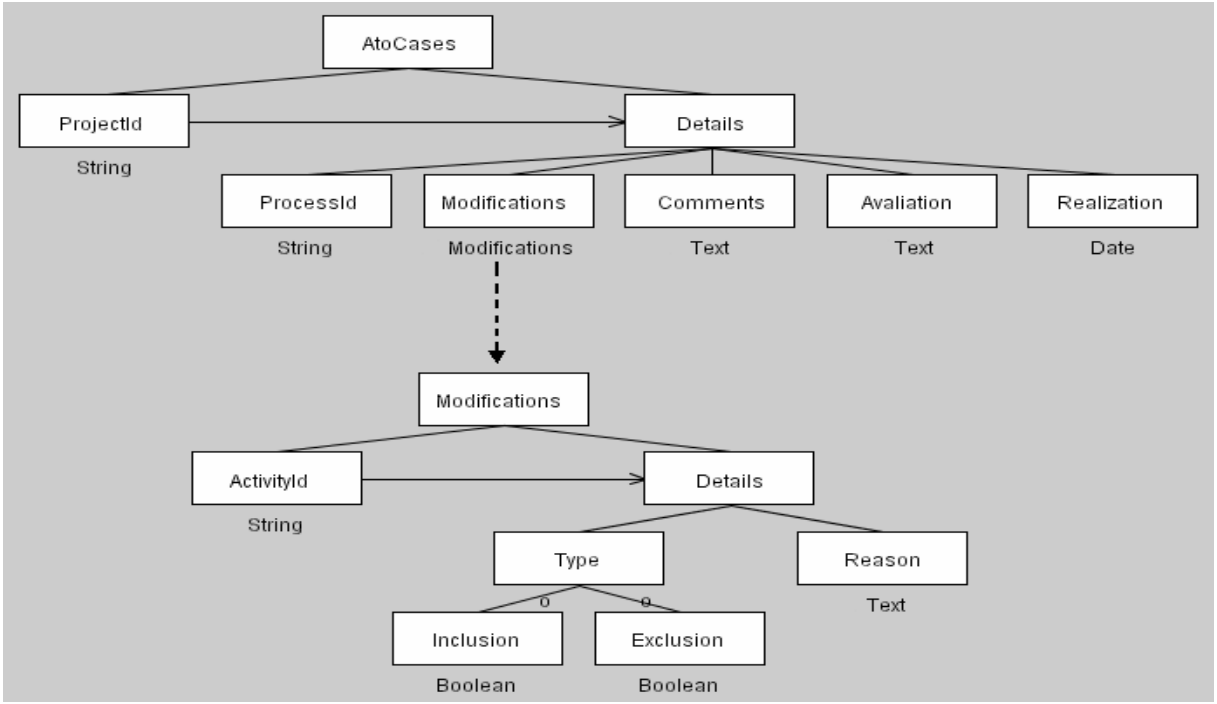

Fig. 13: As classes *Cases* e *Modifications*

4 Trabalhos Correlatos

O modelo de adaptação proposto está fortemente embasado nas idéias presentes nas propostas de Ahn et al e Coelho, cabendo aqui uma rápida comparação, adotando-se como critério os componentes presentes em cada modelo e a abordagem de adaptação utilizada.

Ahn et al apresentam, em [8], um modelo de adaptação baseado nas características do projeto de software, na experiência adquirida em adaptações anteriores (armazenada na forma de casos, como no modelo proposto) e em iniciativas de melhorias (uma iniciativa de melhoria indica o nível de determinado modelo de maturidade ao qual o processo adaptado deve se adequar). A abordagem de adaptação utilizada combina raciocínio baseado em casos com inferência baseada em conhecimento e, de forma similar ao modelo aqui proposto, também é orientada a atividades. Um

projeto de software é caracterizado através de fatores de domínio (idêntico ao conceito de característica de projeto), entretanto, ao contrário do modelo aqui proposto, existe um conjunto fixo destes fatores e o conceito de nivelamento não é suportado (o que reduz a precisão no momento do cálculo da similaridade entre dois projetos). Nesta proposta, regras de adaptação são substituídas pelo conceito de *drives*. Um *driver* associa um fator de domínio ou uma iniciativa de melhoria a um conjunto de atividades que devem ou não ser suportadas no processo adaptado. A desvantagem da utilização de *drives* é que eles permitem a ocorrência de conflitos, por exemplo, quando um *driver* A solicita a inclusão da atividade X e outro *driver* B solicita exclusão desta mesma atividade. CBR e inferência baseada em conhecimento podem ser utilizados separadamente ou juntos, sendo que no último caso conflitos podem ocorrer havendo a necessidade da intervenção do engenheiro de processos para resolução dos mesmos.

Coelho, em [1], apresenta o MAPS (Modelo de Adaptação de Processos de Software), que tem por objetivo auxiliar a adaptação do processo padrão de uma organização de software para um projeto a ser conduzido na mesma, baseado nas características deste projeto e em experiências adquiridas em adaptações anteriores. Ao contrário do modelo aqui proposto, o conjunto de características de projeto é fixo e não são suportadas as categorias múltipla e simples sem nivelamento, o processo padrão é fixo (o modelo trabalha em cima do RUP – *Rational Unified Process*), a abordagem de adaptação é orientada a artefatos (verificam-se sob que condições os artefatos do processo padrão devem ser incluídos ou não no processo adaptado, e só então as atividades necessárias para produzi-los ou consumi-los) e os conceitos de regras e casos de adaptação não estão formalizados. O ponto positivo desta proposta é que ela define um conjunto claro e coerente de passos a ser seguido para a adaptação do processo padrão, além de suportar mecanismos que permitam a melhoria do mesmo.

5 Conclusão

A adaptação de processos de software é uma atividade fortemente baseada em conhecimento e, sem o suporte metodológico e ferramental necessário, torna-se custosa, demorada e sujeita a erros. Soluções que busquem aumentar o seu nível de automatização ocasionam uma diminuição dos custos e do tempo despendidos no processo de desenvolvimento de software das organizações de software, além de garantirem uma maior aceitação e otimização no uso de tais processos de software.

Neste artigo, foram apresentadas as características, os componentes e o funcionamento de um modelo para a adaptação de processos de software. Este modelo utiliza uma abordagem orientada a atividades, embasada no gerenciamento e no raciocínio do/sobre o conhecimento necessário à adaptação. Ele possui como pontos fortes o seu caráter genérico (quanto à aplicação) e a sua flexibilidade (pelo fato de ser parametrizável). Especial ênfase foi dada na apresentação dos componentes deste modelo, utilizando-se para isso uma notação formal, o Prosoft-Algébrico, de forma a permitir clareza e precisão na compreensão dos mesmos, quais sejam: processo padrão e processos adaptados, regras de adaptação, projetos de software, características de projeto e casos de adaptação. Por fim, foi feita uma comparação do modelo em questão com trabalhos correlatos, novamente dando-se ênfase nos componentes utilizados.

Este modelo de adaptação ainda está em fase de desenvolvimento. Neste contexto, vislumbram-se como trabalhos futuros a sua prototipação e a elaboração de um estudo de caso, para fins de experimentação e validação.

Referências

[1] C. C. Coelho, “MAPS: Um Modelo de Adaptação de Processos de Software”, Dissertação de Mestrado, Centro de Informática, Universidade Federal de Pernambuco, 2003.

[2] A. Fuggeta, “Software Process: A Roadmap”, In *Proc. 2000 International Conference on Software Engineering*, pp. 25-34.

[3] P. M. Berger, “Instanciação de Processos de Software em Ambientes Configurados na Estação TABA”, Dissertação de Mestrado, COPPE, Universidade Federal do Rio de Janeiro, 2003.

[4] L. F. C. Machado, G. Santos, K. M. Oliveira and A. R. Rocha, “Def-Pro: Apoio Automatizado para Definição de Processos de Software”, In *Proc. 2000 Simpósio Brasileiro de Engenharia de Software (Caderno de Ferramentas)*, pp. 359-362.

[5] W. S. Humphrey, *Managing the Software Process*, New York: Addison-Wesley, 1989.

[6] M. Paulk, The Capability Maturity Model: Guidelines for Improving the Software Process, [s.l.]: Addison-Wesley, 1994.

[7] K. E. Emam, J. N. Drouin and W. Melo, *SPICE – The Theory and Practice of Software Process Improvement and Capability Determination,* [s.l.]: Edwards Brothers Inc., 1998.

[8] Y. W. Ahn, H. J. Ahn and S. J. Park, “Knowledge and Case-Based Reasoning for Customization of Software Processes - A Hybrid Approach”, *International Journal of Software Engineering and Knowledge Engineering*, vol. 13, n. 3, pp. 293-312, Jun. 2003.

[9] P. Xu and B. Ramesh, “A Tool for the Capture and Use of Process Knowledge in Process Tailoring”, In *Proc. 2002 Hawaii International Conference on Systems Sciences*.

[10] D. Ross, “Applications and Extensions of SADT”, *IEEE Computer*, New York, v. 18, n.4, p.25-35, Apr. 1985.

[11] R. Q. Reis, “*APSEE*-Reuse: Um Meta-Modelo para Apoiar a Reutilização de Processos de Software”, Tese de Doutorado, Instituto de Informática, Universidade Federal do Rio Grande do Sul, 2003.

[12] D. J. Nunes, “Estratégia Data-driven no Desenvolvimento de Software”. In *Proc. 1992 Simpósio Brasileiro de Engenharia de Software*, pp. 81-95.

[13] G. S. Rangel, “ProTool: uma ferramenta de prototipação de software para o ambiente PROSOFT”, Dissertação de Mestrado, Instituto de Informática, Universidade Federal do Rio Grande do Sul, 2003.

[14] M. Jackson, *System Design,* [s.l.]: Prentice-Hall International, 1985.

[15] C. A. L. Reis, "Uma Abordagem Flexível para Execução de Processos de Software Evolutivos", Tese de Doutorado, Instituto de Informática, Universidade Federal do Rio Grande do Sul, 2003.