UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

ANDERSON BAIA MAIA

APSEE-Tail: um Modelo de Apoio à Adaptação de Processos de Software

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Prof. Dr. Daltro José Nunes
Orientador

Porto Alegre, dezembro de 2005

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Maia, Anderson Baia

APSEE-Tail: um Modelo de Apoio à Adaptação de Processos de Software / Anderson Baia Maia – Porto Alegre: Programa de Pós-Graduação em Computação, 2005.

249 f.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2005. Orientador: Daltro José Nunes.

1.Processos de Software 2.Reutilização de Processos de Software. 3.Adaptação de Processos de Software. I. Nunes, Daltro José. II. Título.

AGRADECIMENTOS

Em primeiro lugar a Deus, pois tenho certeza de que sempre esteve comigo durante esta caminhada, permitindo que eu conseguisse reunir forças dentro de mim para superar as adversidades e realizar as escolhas mais acertadas.

À minha família, por tudo o que significa para mim. Sem o apoio e suporte da minha família, em especial dos meus pais, Admilson e Cândida, sinceramente teria desistido no meio do caminho. Sempre serei grato pelos telefonemas, pelas conversas de incentivo e, por que não dizer, pelo apoio financeiro, sem o qual não teria condições de lutar por este objetivo.

À minha namorada e companheira, Tatiana, por ter suportado a distância e o tempo em que tivemos de ficar "separados". Ao meu stress e desânimo, ela sempre respondia com carinho e lealdade. Seu colo sempre foi a válvula de escape nos meus momentos mais difíceis. Como se não bastasse, devo a ela todo o agradecimento pela formatação deste trabalho.

À Ana, por ter divido comigo os bons e maus momentos desta caminhada. Sinto-me recompensado por ter conseguido a amizade de uma pessoa ímpar como ela. Sua ajuda foi fundamental na conclusão deste trabalho.

Ao Roberto e à Eliane, meus pais em POA, pela acolhida e período de convivência.

Ao pessoal da "Vila Robertão", em especial, ao Márcio (vulgo carioca), ao Ronaldo (vulgo mineiro) e ao Alex (vulgo Pará), pela amizade a mim concedida.

Ao pessoal da "Turma Mística", pelos momentos inesquecíveis.

Ao Rodrigo e à Carla, pelas conversas no início do mestrado (elas me deram um norte), bem como pela forma prestativa que sempre me trataram. Os dois são os exemplos de profissional que tento seguir.

Ao Lincoln, pelos comentários sempre pertinentes. À Duda, pelo auxílio prestado, em especial na submissão do projeto de cooperação aprovado na FAPERGS. Ao Guilherme, pela ajuda durante a época das disciplinas.

A todo o pessoal do grupo PROSOFT, pela ajuda prestada.

Ao Heribert e ao Tiago, pelo excelente trabalho na prototipação desta dissertação.

Por fim, ao meu orientador, professor Daltro, pela oportunidade oferecida e confiança depositada no meu trabalho. Foi difícil assimilar a sua filosofia, porém, olhando sob perspectiva, chego a conclusão de que a experiência e o aprendizado valeram a pena.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
ADS	Ambiente de Desenvolvimento de Software
APM	*Adaptable Process Model*
ATO	Ambiente de Tratamento de Objetos
CASE	*Computer-Aided Software Engineering*
CBR	*Case Based Reasoning*
SW-CMM	*Capability Maturity Model for Software*
FIPS	*Federal Information Processing Standarts*
GG	Gramática de Grafos
ICS	Interface de Comunicação do Sistema
IDEF0	*Integration Definition for Function Modeling*
JVM	*Java Virtual Machine*
NAC	*Negative Application Condition*
PML	*Process Modeling Language*
PSEE	*Process-Centered Software Engineering Environment*
RMI	*Remote Method Invocation*
RUP	*Rational Unified Process*
SADT	*Structured Analysis and Design Technique*
SEI	*Software Engineering Institute*
SPI	*Software Process Improvement*
SPICE	*Software Process Improvement and Capability dEtermination*
TAD	Tipo Abstrato de Dados
UML	*Unified Modeling Language*
XML	*eXtensible Markup Language*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O crescimento experimentado pela indústria do software nas últimas décadas, trouxe consigo o aumento das exigências do mercado. É exigido das organizações de software, que os sistemas sejam construídos de acordo com prazo e custos determinados, obedecendo-se certos padrões de qualidade. Para atender tais exigências e assim obter o diferencial competitivo, tornou-se necessário investir no processo de desenvolvimento de software, dada a relação cada vez mais evidente entre a qualidade do produto de software e a eficiência e eficácia do processo de desenvolvimento adotado. Uma estratégia na busca pela maturidade em termos de processos é a definição e adoção de um processo único a ser seguido em todos os projetos de uma organização, denominado processo padrão. Levando-se em consideração a singularidade de cada novo projeto de software, é natural presumir que o processo padrão tenha de ser adaptado para as necessidades específicas de cada situação, de forma a ser aceito, ter seu uso maximizado e garantir a qualidade do software a ser produzido.

Este trabalho apresenta, assim, o APSEE-Tail, um modelo para apoiar o engenheiro de processos na tarefa de adaptar o processo padrão de uma organização de software para as particularidades de cada um de seus projetos, possibilitando maior efetividade e eficiência no uso do mesmo. Sua abordagem de adaptação é livre, orientada à atividades e baseada no raciocínio, através da combinação das técnicas de interpretação de regras e CBR (*Case Based Reasoning*), sobre o conhecimento necessário. Tal conhecimento, neste trabalho, é agrupado em três categorias: diretrizes de adaptação do processo padrão, tipos de característica usados para definir os projetos de software e informações sobre adaptações realizadas anteriormente.

Os diferentes componentes envolvidos na definição do APSEE-Tail foram especificados algebricamente, o que constituiu uma base semântica de alto nível de abstração e possibilitou a construção de um protótipo, implementado no ADS (Ambiente de Desenvolvimento de Software) Prosoft-Java e fracamente acoplado ao APSEE, um ambiente de engenharia de software centrado no processo também prototipado no Prosoft-Java, tornando-se assim parte do meta-processo adotado pelo mesmo.

O texto apresenta ainda alguma fundamentação teórica sobre a área de Adaptação de Processos de Software, considerações sobre as abordagens pesquisadas, enfatizando as que mais influenciaram o APSEE-Tail, e um exemplo de aplicação do protótipo construído. Por fim, são apresentadas as contribuições e limitações da proposta, na visão do autor, bem como os trabalhos futuros vislumbrados.

Palavras-Chave: Processos de Software, Reutilização de Processos de Software, Adaptação de Processos de Software.

APSEE-Tail: A Model to Support Software Processes Tailoring

ABSTRACT

The growth experienced by software industry in the last decades, brought with itself the increase of market demands. Is demanded from the software organizations that systems be built in agreement with determined schedule and costs, obeying certain quality standards. To take care of such demands and, with this, obtain the competitive differential, became necessary to invest in the software process, given the relationship more and more evident between the quality of the software product and the efficiency and effectiveness of the adopted development process. A strategy in the search for maturity in terms of processes is the definition and adoption of one process to be followed in all organization's projects, denominated standard process. Taking in consideration the singularity of each new software project, is natural to suppose that the standard process has to be tailored for the specific needs of each situation, in way to be accepted, to have its use maximized and to guarantee the quality of the software to be produced.

So, this work presents APSEE-Tail, a model to support processes engineers in the task of tailoring the standard process of a software organization to the particularities of each one of its projects, making possible larger effectiveness and efficiency in the use of it. Its tailoring approach is free, guided to activities and based on the reasoning, through the combination of rules interpretation and CBR (Case Based Reasoning) techniques, on the necessary knowledge. Such knowledge, in this work, is classified in three categories: tailoring guidelines of the standard process, characteristic types used to define software projects and information of tailoring occurred previously.

The different components involved in the definition of APSEE-Tail were formally specified, what constituted a high abstraction level semantic base and made possible the construction of a prototype, implemented in SDE (Software Development Environment) Prosoft-Java and weakly coupled to APSEE, a process-centered software engineering environment prototyped in Prosoft-Java, becoming part of the meta-process adopted by the same.

The text still presents some concepts on the Software Processes Tailoring area, considerations about researched approaches, emphasizing those that have more influenced APSEE-Tail, and an application example of the built prototype. Finally, are made considerations concerning the contributions and limitations, in the author's view, of the proposal, as well the previewed future works.

Keywords: Software Processes, Software Processes Reuse, Software Processes Tailoring.

1 INTRODUÇÃO

Dentre as principais áreas que compõem a Ciência da Computação, uma das mais importantes é a Engenharia de Software, envolvida mais especificamente nos aspectos tecnológicos e gerenciais do processo de desenvolvimento de software[1], simplesmente chamado de processo de software. O software tornou-se a base de sustentação de inúmeras organizações dos mais diversos ramos de atuação, consistindo de um elemento estratégico na diferenciação de produtos e serviços atuais (PRESSMAN e REIS, 2002).

Neste cenário, a área de Processos de Software representou um importante passo em direção à melhoria da produtividade e da qualidade do software, principalmente através de mecanismos que proporcionam o gerenciamento automatizado do seu processo de desenvolvimento (FEILLER; HUMPHREY, 1993). Diversas teorias, conceitos, formalismos, metodologias e ferramentas surgiram nesta área, enfatizando a descrição de um modelo de processo[2] que é automatizado por um ambiente integrado de desenvolvimento[3] (REIS, 2002).

Recentemente, ganhou força a idéia de se reutilizar o processo de software aplicado em situações anteriores para auxiliar na tarefa de modelagem de novos processos. Através da reutilização de processos, as organizações de desenvolvimento de software, ou apenas organizações de software, podem obter expressivas economias, além de permitir um efetivo aumento na qualidade do software produzido (REIS, 2000). Estes ganhos se devem basicamente a dois motivos: as organizações de software não precisam definir os seus processos de desenvolvimento do zero; aumenta-se a possibilidade de reutilização de experiências anteriores que obtiveram êxito, podendo-se inclusive aperfeiçoá-las.

---

[1] Informalmente, o processo de software pode ser compreendido como o conjunto de todas as atividades necessárias para transformar os requisitos do usuário em software (HUMPHREY, 1989) (OSTERWEIL, 1987). Um processo de software é formado por um conjunto de atividades parcialmente ordenadas, relacionadas com conjuntos de artefatos, pessoas, recursos, ferramentas, estruturas organizacionais e restrições, tendo como objetivo final possibilitar o desenvolvimento dos produtos de software requeridos, com qualidade e obedecendo a prazo e orçamento determinados (COELHO, 2003) (DOWSON; NEJMEH; RIDDLE, 1991) (LONCHAMP, 1993).

[2] O modelo de um processo é uma descrição formal de seus elementos e da dinâmica entre os mesmos. Tal descrição deve agrupar vários tipos de informação, de modo a indicar quem, onde, como e por que as atividades são realizadas (REIS, 1999).

[3] PSEEs (*Process-Centered Software Engineering Environments*) são os ambientes que suportam a criação e exploração de modelos de processo, permitindo que os mesmos sejam definidos explicitamente pelo usuário através de uma PML (*Process Modeling Language*). Além disso, um PSEE deve auxiliar o usuário na monitoração do processo, automatizando algumas partes e direcionando a execução (GARG apud SOUSA, 2003).

Levando-se em consideração a singularidade das organizações de software e dos projetos de software conduzidos em tais organizações, é natural presumir que processos de software reutilizáveis[4] tenham de ser adaptados para as necessidades específicas de cada situação, de forma a serem aceitos, terem seu uso maximizado e garantirem a qualidade do software a ser produzido. Segundo Coelho (2003), adaptar processos de software para situações específicas, analisando os vários aspectos do processo e da situação, é uma tarefa complexa, exigindo grande esforço e conhecimento do engenheiro de processos[5]. Por isso, os modelos, métodos e ferramentas para adaptação de processos devem apresentar formas de lidar com essa complexidade, tornando a tarefa de adaptar processos mais simples e menos custosa.

É neste contexto, o da adaptação de processos de software reutilizáveis, que se enquadra o modelo proposto neste trabalho, denominado APSEE-Tail. Este capítulo apresenta uma visão geral do trabalho e está estruturado da seguinte forma:

- Na seção 1.1, é descrito o problema a ser abordado pelo trabalho;
- Na seção 1.2, são apresentadas as motivações do trabalho;
- Na seção 1.3, são apresentados os objetivos gerais do trabalho;
- Na seção 1.4, o trabalho é contextualizado no grupo de pesquisa do autor;
- Por fim, na seção 1.5, é descrita a organização do restante deste texto.

1.1 O Problema

A adaptação de processos de software (do inglês *software processes tailoring*[6]) é uma etapa importante da reutilização de processos e envolve todas as modificações necessárias para aplicar um processo de software reutilizável em determinada situação, tais como a modificação sintática do modelo de processo, o refinamento de tipos genéricos para tipos dependentes da organização-destino, entre outras (REIS, 2002). Esta atividade requer profissionais especializados, denominados engenheiros de processos, que são responsáveis também por fornecer o modelo de processo e adaptar o PSEE que apóia a execução deste modelo (DERNIAME; KABA; WASTELL, 1999). Por vezes, as organizações de software, visando aumentar o nível de maturidade de seus processos, costumam montar uma equipe dedicada exclusivamente a lidar com a definição, adaptação e melhoria dos mesmos.

---

[4] Processos de software reutilizáveis são processos suficientemente genéricos e abstratos, podendo ser utilizados em diferentes situações (por exemplo, em diferentes domínios de aplicação, organizações de software, projetos de software ou ambientes tecnológicos).

[5] As denominações projetistas de processos ou programadores de processos também são encontradas na literatura da área.

[6] Na literatura especializada, vários são os termos encontrados para designar adaptação de processos de software, tais como "adaptação", "customização", "especialização", "individualização", "configuração", entre outros. Neste trabalho, é utilizado o termo "adaptação" como correspondente do termo "*tailoring*", em inglês, de acordo com a recomendação do ABNT Software (Subcomitê de Software da Associação Brasileira de Normas Técnicas), através de sua Comissão de Estudos em Gerência do Ciclo de Vida do Software.

Durante a realização da adaptação, o engenheiro de processos deve-se valer de sua experiência e de seu conhecimento acerca do contexto de utilização[7], de modo a realizar as alterações necessárias e assim obter o processo adaptado adequado à situação corrente. A figura 1.1 ilustra esta situação, utilizando um diagrama SADT (*Structured Analysis and Design Technique*) (ROSS, 1985).

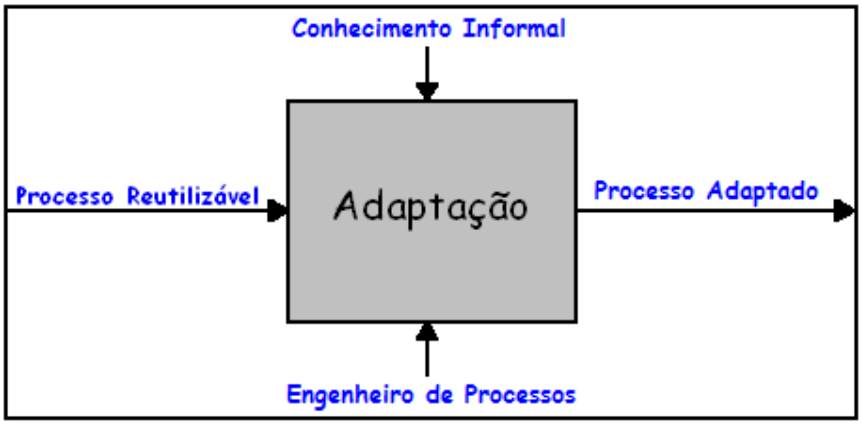


Figura 1.1: Cenário geral para adaptação de processos de software

Obter um processo de software especificamente adaptado para um determinado contexto de utilização não é uma tarefa trivial, em virtude de uma série de fatores, tais como (AHN et al, 2003) (REIS, 2002) (SCHMID; WIDEN, 2000) (XU et al, 2002): diversidade de informações / alterações que devem ser agregadas / realizadas no processo de software original; a adaptação de processos de software é uma atividade que requer uso intensivo de conhecimento (a experiência do engenheiro de processos, as características do contexto de utilização, as diretrizes que devem guiar a adaptação, repositórios de componentes de processo reutilizáveis, bem como informações sobre adaptações realizadas anteriormente e lições aprendidas, constituem exemplos do conhecimento utilizado durante esta atividade); como processos de software não podem ser totalmente automatizados, já que envolvem atividades complexas e criativas, desempenhadas por pessoas com as mais diversas capacidades, experiências e expectativas, algum nível de flexibilidade deve ser tolerado na adaptação dos mesmos.

Sem dispor do suporte metodológico e ferramental necessário para permitir o aumento de sua automatização, a adaptação de processos de software pode se tornar uma atividade consumidora de tempo, financeiramente custosa e sujeita a erros (AHN et al, 2003). Segundo opiniões dispersas em (COELHO, 2003), (MÜNCH; SCHMITZ; VERLAGE, 1997) e (REIS, 2002), soluções que busquem a automatização desta atividade devem atender aos seguintes requisitos:

- Exigir que o usuário forneça como entrada o processo reutilizável candidato à adaptação e a descrição rigorosa das características do contexto de utilização;
- Fornecer, como resultado da adaptação, um processo adaptado sintaticamente coerente[8], pronto para ser instanciado e executado, ou aperfeiçoado ainda mais

[7] O contexto de utilização representa o conjunto de informações que indicam onde e com que finalidade o processo de software reutilizável será aplicado.

[8] Um processo de software pode ser considerado sintaticamente coerente, quando seu modelo está em conformidade com o meta-modelo adotado, bem como não apresenta ciclos entre as atividades.

pelo engenheiro de processos. Portanto, em primeiro lugar, as modificações permitidas na adaptação de processos de software devem ser derivadas de regras gerais para a boa formação sintática de modelos de processos executáveis;

- Documentar as alterações realizadas no processo de software original, de forma adequada, tal que uma análise das variações deste processo seja possível;
- Tornar a tarefa de adaptar processos mais simples e menos custosa.

1.2 Motivações

A indústria do software vem experimentando um grande crescimento nas últimas décadas. Duas das principais conseqüências deste crescimento são o aumento da complexidade do software e as exigências cada vez maiores do mercado. É exigido das organizações de software que os sistemas sejam desenvolvidos com prazo e custo determinados e obedeçam a padrões de qualidade. Para atender essas exigências, tornou-se necessário investir no processo de software, já que é cada vez mais evidente a correlação entre a qualidade do produto de software desenvolvido e o processo de desenvolvimento adotado (MACHADO, 2000) (COELHO, 2003).

Visando permitir sua análise, compreensão, automatização e melhoria, processos de software são descritos formalmente através de modelos de processo, conforme já mencionado. A atividade de modelagem de processos de software, especialmente quando um bom nível de detalhamento é necessário, ou quando a automatização dos mesmos é requerida, é uma tarefa complexa e demorada, uma vez que diferentes aspectos tecnológicos, organizacionais e sociais devem ser inter-relacionados. Esta constatação, associada ao fato de que muitos processos de software similares são executados por diferentes organizações de software, motivou recentemente o surgimento de uma ampla área de pesquisa em reutilização de processos, envolvendo a definição de formalismos e ferramentas para facilitar a construção de modelos de processo a partir de processos e componentes já existentes. Assim, como em qualquer outra atividade custosa, a reutilização constitui uma contingência econômica, possibilitando que organizações de software possam obter um aumento na maturidade dos seus processos e na qualidade do software produzido (REIS, 2000; 2002).

Levando-se em consideração a singularidade das organizações de software e dos projetos de software conduzidos em tais organizações, é natural presumir que processos de software reutilizáveis tenham de ser adaptados para as necessidades específicas de cada situação, de modo a serem aceitos, terem seu uso maximizado e garantirem a qualidade do software a ser produzido. Além disso, organizações capazes de reutilizar seus processos em diferentes projetos obtêm uma avaliação positiva diante de modelos

de maturidade como o SW-CMM[9] (*Capability Maturity Model for Software*) (PAULK et al, 1997) e a norma ISO/IEC TR 15504[10] (1998).

Segundo Ahn et al (2003) e Xu et al (2002), 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 e de atividades manuais, o que a torna consumidora de tempo, custosa e sujeita a erros. Sendo assim, o aumento do nível de automatização desta atividade traz consigo os seguintes benefícios:

- Redução de tempo e custo despendidos durante a modelagem 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.

Por fim, segundo Reis (2002), o nível de flexibilidade exigido durante a adaptação dificulta a automatização da mesma e, portanto, a literatura da área apresenta poucos trabalhos especializados na adaptação automática de processos de software. Ainda segundo Reis (2002), idealmente a adaptação de processos pode se valer de soluções baseadas em conhecimento, com o objetivo de fornecer sugestões de adaptações em função de soluções semelhantes adotadas em situações similares, levando em consideração as características ambientais e organizacionais correntes.

1.3 Objetivos

Levando-se em consideração as motivações apresentadas na seção anterior, o objetivo deste trabalho é o desenvolvimento tecnológico-científico da área de Processos de Software, mais especificamente no que concerne à Adaptação de Processos. Portanto, são objetivos gerais deste trabalho:

- Definir e especificar formalmente um modelo, baseado em conhecimento, para apoiar o engenheiro de processos durante a adaptação do processo padrão de uma organização de software para os projetos conduzidos na mesma;
- Prototipar esse modelo, com base na sua especificação formal;

---

[9] O CMM é um modelo de avaliação da maturidade dos processos de uma organização de software, criado pelo SEI (*Software Engineering Institute*), estruturado em cinco níveis: inicial, repetível, definido, gerenciado e otimizado. Para que uma organização se enquadre no CMM nível 3, o seguinte cenário precisa ser constatado: o processo de software em relação às atividades de gerenciamento e desenvolvimento é documentado, padronizado e integrado em um processo de software padrão para a organização. **Todos os projetos utilizam uma versão aprovada e adaptada do processo padrão para desenvolvimento e manutenção de software**.

[10] A norma ISO/IEC 15504 é o padrão internacional para avaliação de processos de software. Estudos experimentais de seu uso têm sido realizados pelo projeto SPICE (*Software Process Improvement e Capability dEtermination*) (EMAN; DROUIN; MELO, 1998). Considera a existência de seis níveis de capacitação: incompleto, executado, gerenciado, estabelecido, previsível e otimizado. O quarto nível prevê que: o processo é definido, executado e gerenciado de acordo com princípios da engenharia de software. **Implementações individuais do processo utilizam versões aprovadas e customizadas de processos padrão documentados**.

- Apresentar um exemplo de aplicação do protótipo.

1.4 Contextualização

O trabalho apresentado nessa dissertação foi desenvolvido no contexto do grupo de pesquisa PROSOFT, da UFRGS (Universidade Federal do Rio Grande do Sul), sob a coordenação do Prof. Dr. Daltro José Nunes. O principal objetivo do grupo de pesquisa é a construção de um ambiente de desenvolvimento de software para apoiar o engenheiro de software desde a análise do problema até a construção do programa.

O ambiente desenvolvido pelo grupo, cuja implementação mais recente é denominada Prosoft-Java (SCHLEBBE, 1997), permite que ferramentas de apoio ao desenvolvimento de software sejam especificadas, implementadas e integradas ao ambiente, em um paradigma próprio (NUNES, 1992; 1994). Isto somente é possível devido à integração funcional[11], sintática[12] e semântica[13] que o ambiente provê para suas ferramentas.

Nos últimos anos, diversos trabalhos de mestrado e doutorado do Instituto de Informática da UFRGS foram desenvolvidos no contexto do grupo PROSOFT, gerando ferramentas incorporadas ao ambiente Prosoft-Java. Dentre os trabalhos desenvolvidos, pode-se citar: sistema especialista para o desenvolvimento de software (MORAES, 1997), suporte ao desenvolvimento cooperativo de software (REIS, 1998), abordagem para reutilização de especificações de requisitos (PIMENTA, 1998), modelo para decisões em grupo no desenvolvimento de software (ALVES, 2002), mecanismo para prototipação de software (RANGEL, 2003).

Além de suporte às etapas do desenvolvimento de software, uma das principais linhas de pesquisa do grupo PROSOFT tem como foco a construção de mecanismos que auxiliem na gerência de processos de software. Essa linha surgiu a partir do gerenciador de processos de software, proposto por Lima Reis (1998), e consiste de ferramentas que apóiam a reutilização e modelagem (REIS, 2002), a instanciação (LIMA REIS, 2003), a simulação (SILVA, 2001), a execução flexível (LIMA REIS, 2003) e, por fim, a visualização (SOUSA, 2003) de processos de software, constituindo uma infra-estrutura integrada para automação de processos de software, denominada APSEE[14], que também foi integrado ao ambiente Prosoft-Java.

Atualmente, as pesquisas acerca de processos de software no grupo PROSOFT envolvem estudantes e pesquisadores no desenvolvimento de ferramentas integradas ao APSEE, atuando sobre o seu meta-modelo. Ferramentas para gerência distribuída de processos de software (FREITAS, 2005) e para desenvolvimento e gerência de processos educacionais (DAHMER, 2005) estão sendo desenvolvidas dentro do contexto do APSEE.

---

[11] Por integração funcional, entende-se que cada ferramenta do ambiente deve ter uma função bem definida e interagir com outras através de sua interface.

[12] A integração sintática permite que dados gerados por uma ferramenta sejam lidos e usados por outras ferramentas.

[13] Graças à integração semântica, operações de uma ferramenta podem ser construídas a partir de outras ferramentas.

[14] APSEE é um acrônimo para "A Process-Centered Software Engineering Environment".

No tocante à reutilização de processos, o ambiente APSEE ainda não dispõe de um mecanismo que forneça apoio automatizado para a etapa de adaptação, embora suporte as demais etapas: modelagem e recuperação de *templates*, e generalização de processos concretos. Tendo em vista esta constatação e as motivações apresentadas anteriormente, pode-se dizer que este trabalho agrega valor[15] ao ambiente APSEE, oferecendo-lhe seus serviços de suporte automatizado à adaptação de processos de software reutilizáveis.

1.5 Organização do Texto

O presente trabalho está organizado da seguinte forma:

- O capítulo 2 apresenta a fundamentação teórica necessária sobre a adaptação de processos de software;
- O capítulo 3 apresenta uma visão geral do APSEE-Tail, seu escopo de atuação, suas características, componentes, funções e a estratégia de adaptação que utiliza;
- O capítulo 4 apresenta os formalismos utilizados na especificação APSEE-Tail e as justificativas para tal;
- O capítulo 5 apresenta o protótipo construído a partir da especificação formal do APSEE-Tail;
- O capítulo 6 apresenta um exemplo de aplicação do protótipo;
- O capítulo 7 apresenta as conclusões do trabalho;
- O anexo A apresenta os construtores de tipo do Prosoft-Algébrico;
- O anexo B apresenta o tipo *AbsProcesssModel*, que representa a estrutura de modelos de processo abstratos no ambiente APSEE;
- O apêndice A apresenta os ATOs algébricos do modelo;
- O apêndice B apresenta a semântica do mecanismo de adaptação;
- O apêndice C apresenta as regras para a garantia da coerência sintática dos processos adaptados gerados.

---

[15] De fato, o APSEE-Tail possui um acoplamento fraco com o ambiente APSEE. Por acoplamento fraco, entenda-se que o APSEE-Tail não está totalmente integrado ao APSEE, mas possui uma interface bem definida com ele: o processo padrão é modelado no APSEE e adaptado no APSEE-Tail; os processos adaptados são gerados e mantidos no APSEE-Tail, mas podem ser atualizados no APSEE. Esta abordagem de integração permite que cada ferramenta mantenha sua autonomia, mas tire proveito dos serviços já oferecidos pela outra. Isso só foi possível porque o APSEE-Tail: utiliza uma metodologia de adaptação autocontida; e deixa claro, como uma de suas premissas, que a modelagem de processos não é uma de suas responsabilidades.

2 ADAPTAÇÃO DE PROCESSOS DE SOFTWARE

A adaptação de processos de software pode ser encarada segundo duas visões complementares:

- **Área do conhecimento** dedicada ao estudo dos diversos aspectos relacionados com a adaptação de processos de software reutilizáveis em diferentes contextos. Como tal, pode-se hierarquizar as áreas do conhecimento que a englobam, segundo apresentado na figura 2.1;

Figura 2.1: Áreas do conhecimento que englobam a Adaptação de Processos

- **Etapa do meta-processo de software voltado à reutilização**, envolvendo todas as modificações necessárias para aplicar um processo de software reutilizável em determinada situação, tais como a modificação sintática do modelo de processo, o refinamento de tipos genéricos para tipos dependentes da organização-destino, entre outras.

Conforme dito no capítulo 1, a adaptação é uma atividade fortemente baseada em conhecimento, que requer profissionais especializados, denominados engenheiros de processos. Por vezes, as organizações de software, visando aumentar o nível de maturidade de seus processos, costumam montar uma equipe dedicada exclusivamente a lidar com a definição, adaptação e melhoria dos mesmos.

Neste capítulo, serão elucidados mais alguns aspectos relativos à adaptação de processos de software, sendo alguns deles de fundamental importância para a compreensão dos capítulos posteriores.

- Na seção 2.1, são apresentados os meta-processos de software: geral; voltado à reutilização; e voltado à adaptação. Esta seção tem por objetivo contextualizar a fase da adaptação dentro do meta-processo de software;

- Na seção 2.2, é apresentado o cenário ideal, na visão do autor, para a realização da adaptação de processos de software;
- Na seção 2.3, é apresentada a diferença entre adaptação e instanciação de processos, fundamental na compreensão do escopo deste trabalho;
- A seção 2.4 discorre sobre o processo padrão;
- A seção 2.5 apresenta os tipos de adaptação existentes, destacando aquele utilizado neste trabalho;
- Um resumo das abordagens de adaptação pesquisadas é apresentado na seção 2.6, seguido de um quadro comparativo entre as que mais influenciaram trabalho;
- Por fim, algumas considerações acerca do que é apresentado neste capítulo são realizadas na seção 2.7.

2.1 Contextualização no Meta-Processo de Software

2.1.1 Meta-Processo de Software

Um processo de software e seu modelo têm uma natureza evolucionária, devido à necessidade de melhoria e correção contínua e em virtude da instabilidade do ambiente operacional, ou seja, o modelo é primeiro estabelecido para representar o mundo real inicial, e então durante seu tempo de vida é exposto à mudanças causadas por eventos planejados e não-planejados de fora e de dentro da organização (NGUYEN apud LIMA REIS, 1998). Existe um ciclo de vida para processos de software análogo ao ciclo de vida de produtos de software. As atividades do ciclo de vida de processos de software são chamadas de meta-atividades, e o processo de desenvolvimento e evolução de processos de software é denominado de meta-processo. As fases do meta-processo são apresentadas em alto nível de abstração, de forma que cada fase pode ser decomposta em sub-fases de pouca granulosidade (LIMA REIS, 1998).

Ainda segundo Nguyen apud (LIMA REIS, 1998), meta-processos têm sido pouco estudados e seu potencial não tem sido bem explorado. Alguns PSEEs adotam uma parte do meta-processo, apenas como conseqüência de sua arquitetura, e se restringem às fases de modelagem e execução de processos. O meta-processo a ser considerado por este trabalho é aquele utilizado pelo ambiente APSEE, apresentado na figura 2.2 através da linguagem de modelagem do próprio ambiente. Este meta-processo possui fases adicionais que, em conjunto, não são encontradas em meta-processos da literatura e são apresentadas a seguir.

O ciclo de vida inicia-se com a fase de **análise de requisitos do processo** que produz requisitos para o processo a ser desenvolvido. A fase de **modelagem** resulta em um modelo de processo abstrato. Também é possível modelar *templates*[16] (processos abstratos) para reutilização. Essa fase pode ser auxiliada pela fase de **recuperação e adaptação de processos (reutilização)**, que pode fornecer *templates* de processo

[16] Embora os *templates* sejam apenas um dos possíveis tipos de processos reutilizáveis encontrados na literatura, este trabalho utiliza os mesmos como sinônimos de processos reutilizáveis.

adequados à situação corrente. A fase de **instanciação de processos** modifica o processo de software modelado para produzir um modelo de processo instanciado, ou seja, com recursos e agentes alocados e cronogramas definidos. Antes da execução, o processo instanciado pode passar por uma fase de **validação** que é auxiliada pela simulação de processos de software, a qual permite antever problemas de alocação e prazo no processo instanciado. Como resultado da fase de validação, pode ser necessário retornar a fases anteriores para refinamento do modelo, até que este esteja pronto para execução. Durante a **execução**, a máquina de processos coordena a interação com gerentes e desenvolvedores e obtém *feedback* sobre o andamento do processo. Neste caso, pode ser necessário modificar dinamicamente o modelo. Assim, as fases de modelagem, instanciação e validação podem ser realizadas em paralelo com a execução. Após a execução, o modelo de processo é enviado à fase de **avaliação do processo** para gerar novos requisitos e o registro de todas as ocorrências da execução. Processos executados também são enviados à fase de **generalização de processos** que alimenta a base de processos para reutilização.

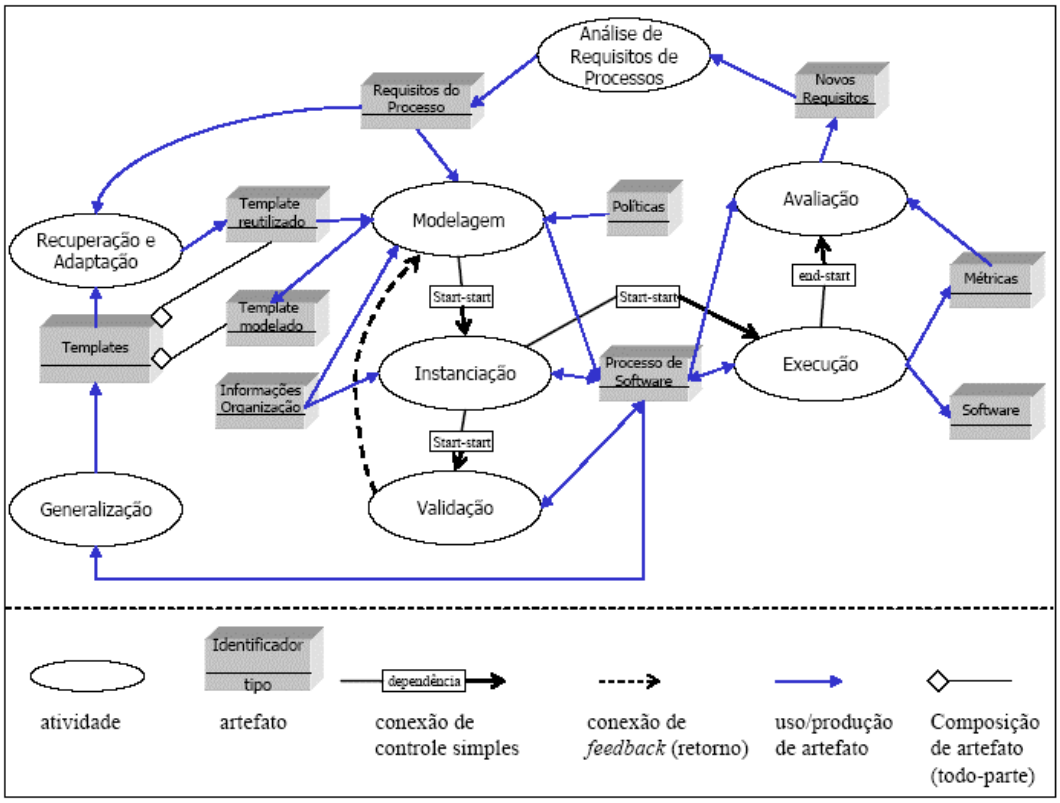


Figura 2.2: Meta-processo de software do ambiente APSEE (LIMA REIS, 2003)

2.1.2 Meta-Processo de Software Voltado à Reutilização

Conforme visto na subseção anterior, a reutilização de processos de software é uma etapa do meta-processo de software. O presente trabalho adota como etapas do meta-processo de software voltado à reutilização, aquelas apresentadas por Reis (2002) e ilustradas na figura 2.3. Assim, a etapa **modelagem de processos visando a reutilização** busca definir através de uma PML modelos de processos e componentes

genéricos e abstratos que possam ser, idealmente, reutilizados em diferentes contextos. A etapa **recuperação e adaptação de processos** define critérios, estratégias e mecanismos para auxiliar um engenheiro na seleção, adaptação e instanciação de processos genéricos que forneçam soluções para o problema sendo tratado. A etapa rotulada como **execução de processos** descreve todas as ocorrências em um modelo de processo desde o início de sua execução, ou seja, a partir do momento em que o software começa a ser produzido (ou parte do modelo iniciou a execução). Finalmente, a **generalização e avaliação de processos encerrados** fornece modelos de processos reutilizáveis a partir da experiência adquirida com processos bem sucedidos.

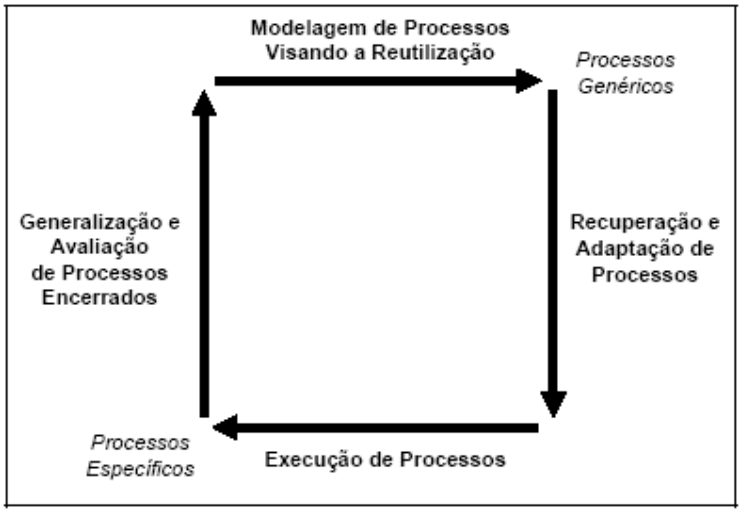


Figura 2.3: Meta-processo de software voltado à reutilização (REIS, 2002)

2.1.3 Meta-Processo de Software Voltado à Adaptação

Conforme visto anteriormente, a adaptação de processos de software é uma etapa da reutilização de processos de software, que por sua vez é uma das etapas do meta-processo de software. Entretanto, a adaptação de processos não constitui uma etapa atômica do meta-processo de software, podendo ainda se ser detalhada em etapas de menor granulosidade. O diagrama IDEF0[17] (*Integration Definition for Function Modeling*) da figura 2.4 foi adaptado de (BUDLONG et al, 1996), constituindo o que o autor convencionou chamar de "meta-processo de software voltado à adaptação".

Conforme pode ser visto na figura, a adaptação de processos de software para contextos específicos compreende basicamente a duas etapas: a caracterização do contexto e a adaptação do processo de software para este contexto. Depois da etapa de adaptação, algumas outras etapas devem ser executadas para permitir, por exemplo, a documentação deste processo, bem como o armazenamento destas informações

[17] A notação IDEF0 Foi formalizada pelo FIPS (*Federal Information Processing Standarts*) (1993). A linguagem de modelagem de sistemas proposta pelo padrão IDEF0 é baseada no padrão SADT. Um modelo IDEF0 é composto por uma série hierárquica de diagramas que apresentam, gradativamente, um nível maior de detalhe, descrevendo funções e suas interfaces no contexto de um sistema. Por causa da uniformidade e iteração limitadas para facilitar o processo, este método é largamente utilizado em modelagem de processos industriais, gerenciais, executivos, etc., e é suportado por um grande número de ferramentas comerciais de modelagem de sistemas.

documentadas em um repositório para reutilização em adaptações futuras. Todas as etapas da adaptação de processos de software podem ser valiosamente auxiliadas por mecanismos de armazenamento e recuperação de conhecimento. Elas devem ser realizadas pelo engenheiro de processos, com ou sem apoio automatizado. Em boa parte destas etapas, os profissionais envolvidos diretamente com a construção de software (dentre eles, gerentes de projeto e engenheiros de software) podem ter valiosa participação, no sentido de fornecer informações e *feedback* para garantir maior adequabilidade do processo adaptado gerado.

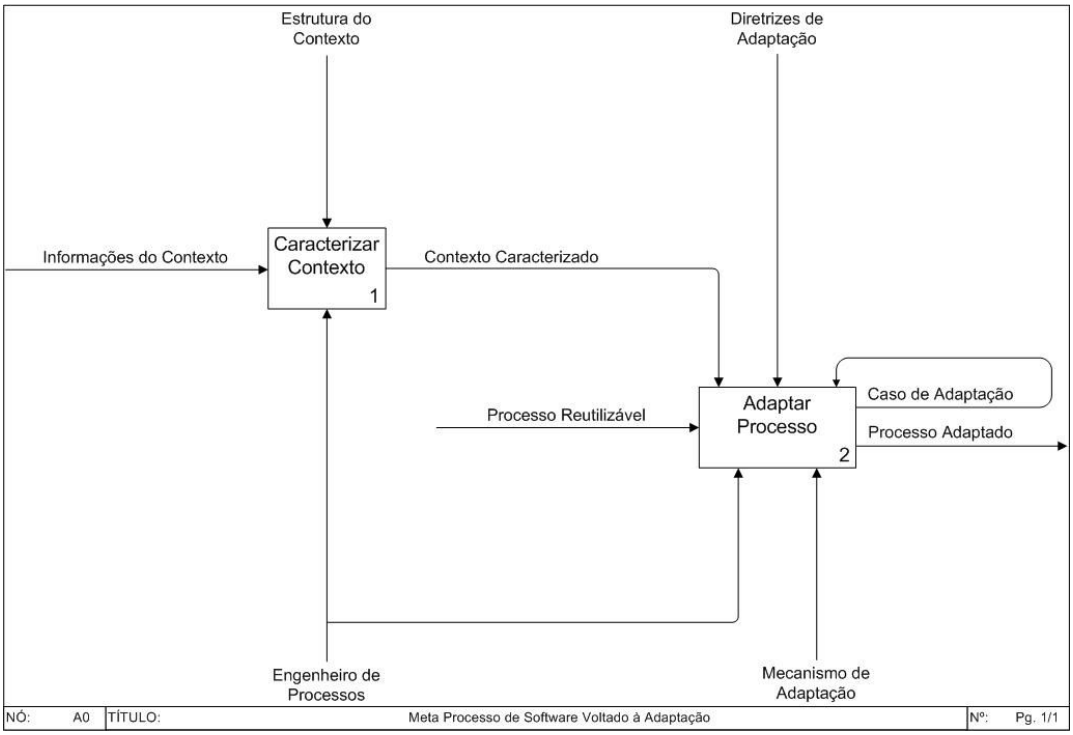


Figura 2.4: Meta-processo de software voltado à adaptação

O processo se inicia com a fase de **caracterização do contexto**. Esta fase tem por objetivo definir e quantificar quais as características do contexto para o qual deseja-se adaptar o processo de software. Este contexto pode ser um domínio de aplicação, uma organização desenvolvedora de software ou mesmo um projeto de software. As características do contexto também são chamadas de fatores de contexto e a representação das mesmas varia dentre as abordagens existentes para adaptação de processos de software. O resultado desta fase é um modelo contendo todas as características relevantes do contexto, as quais serão utilizadas pelas diretrizes de adaptação na fase seguinte. Esta etapa da adaptação não pode ser completamente automatizada, visto que a modelagem do contexto é um processo criativo e que passará obrigatoriamente pelas mãos do engenheiro de processos.

Na fase de **adaptação do processo**, o mecanismo de adaptação (o qual também varia dentre as abordagens encontradas na literatura, podendo ir desde simples guias de adaptação manual até mecanismos semi-automáticos, utilizando técnicas de inferência baseada em conhecimento, raciocínio baseado em casos, ontologias, dentre outros) receberá, como entrada, o processo reutilizável a ser adaptado. Este processo reutilizável pode ser um modelo de processo de uso genérico, o processo padrão de uma organização ou mesmo um processo abstrato já adaptado em um certo nível. O contexto

caracterizado gerado na fase anterior e as diretrizes de adaptação a serem avaliadas, conduzirão o processo de adaptação. O mecanismo de adaptação pode, por exemplo, se valer de uma combinação de raciocínio baseado em casos com um mecanismo de inferência para conduzir este processo. Assim, caso fosse encontrado algum contexto no repositório de conhecimento com características semelhantes a do contexto especificado, o mecanismo poderia escolher não executar a adaptação do processo reutilizável e, ao invés disso, devolver o processo adaptado que se encontra no repositório, com alguns ajustes se necessário. No caso de nada ser encontrado no repositório, então o mecanismo teria de realizar o processo normal de adaptação. Neste processo normal de adaptação, o mecanismo teria de lidar com o surgimento de possíveis incoerências no modelo de processo, bem como solicitar o auxílio do engenheiro de processos no caso de conflitos. O resultado desta etapa é um processo de software abstrato adaptado para o contexto específico e, conforme mencionado nesta mesma seção, deve ser armazenado em um repositório de conhecimento.

2.2 Cenário Ideal de Adaptação

A figura 2.5 apresenta o cenário ideal para a realização da adaptação de processos de software, na visão do autor, levando-se em consideração que a mesma esteja inserida em um contexto maior, o do meta-processo de software sendo suportado por um PSEE.

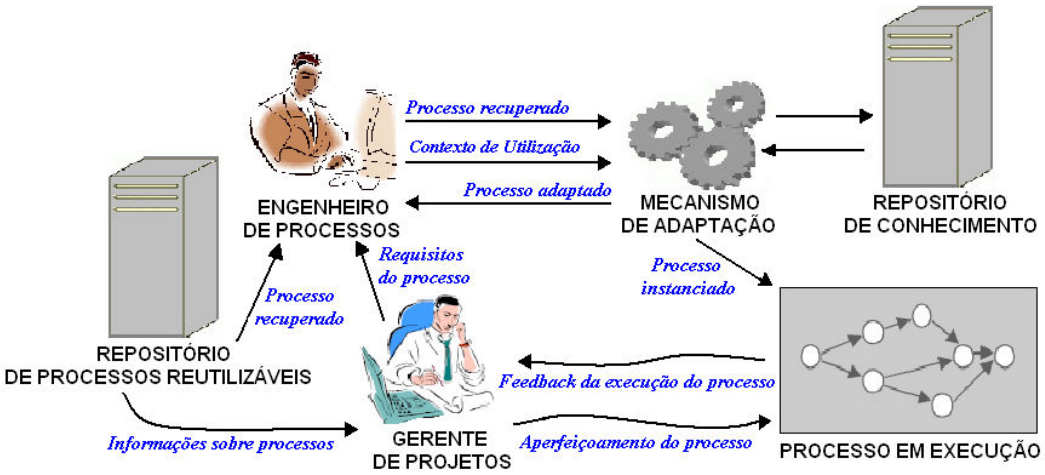


Figura 2.5: Cenário ideal para a adaptação de processos de software

Baseados em uma série de fatores (políticas organizacionais, características do projeto, da equipe de desenvolvimento e do produto a ser desenvolvido, dentre outros), o gerente do projeto apresenta os requisitos do processo de software necessário ao engenheiro de processos. Este, por sua vez, realiza duas tarefas: primeiramente, formaliza a definição do contexto de utilização partindo do zero ou de um contexto similar armazenado no repositório adequado; a seguir, seleciona e recupera um ou mais processos reutilizáveis, do repositório adequado, que possam servir de ponto de partida para a definição do novo processo de software. Esta seleção pode ser auxiliada através de mecanismos de raciocínio baseado em caso, por exemplo.

De posse do contexto de utilização e dos processos reutilizáveis recuperados, o engenheiro de processos, auxiliado pelo mecanismo de adaptação, inicia o processo de adaptação, o qual, de um modo geral, é realizado de forma interativa e evolutiva. O

mecanismo de adaptação utiliza-se das diretrizes de adaptação de cada processo reutilizável, bem como de informações relativas à adaptações realizadas anteriormente em contextos similares, podendo solicitar a intervenção do engenheiro de processos para solucionar conflitos, por exemplo. É de responsabilidade deste mecanismo, garantir a coerência sintática dos processos adaptados gerados.

O processo adaptado gerado pode passar então por ajustes manuais que se façam necessários, através da linguagem de modelagem sendo utilizada pelo PSEE. Caso este processo vá ser utilizado em um projeto de software, deve ser então instanciado antes de sua execução. Durante a execução do processo, o gerente de projetos recebe *feedbacks* do andamento da mesma e aperfeiçoa o processo, se necessário.

2.3 Adaptação x Instanciação

Várias são as definições encontradas na literatura para adaptação de processos de software, bem como os termos utilizados para designá-la. A literatura também é dúbia quanto à distinção entre adaptação e instanciação de processos de software e seus respectivos limites. Conforme pode ser visto na figura 2.6, o APSEE-Tail considera a adaptação e a instanciação de processos de software duas atividades distintas, na realidade, ortogonais.

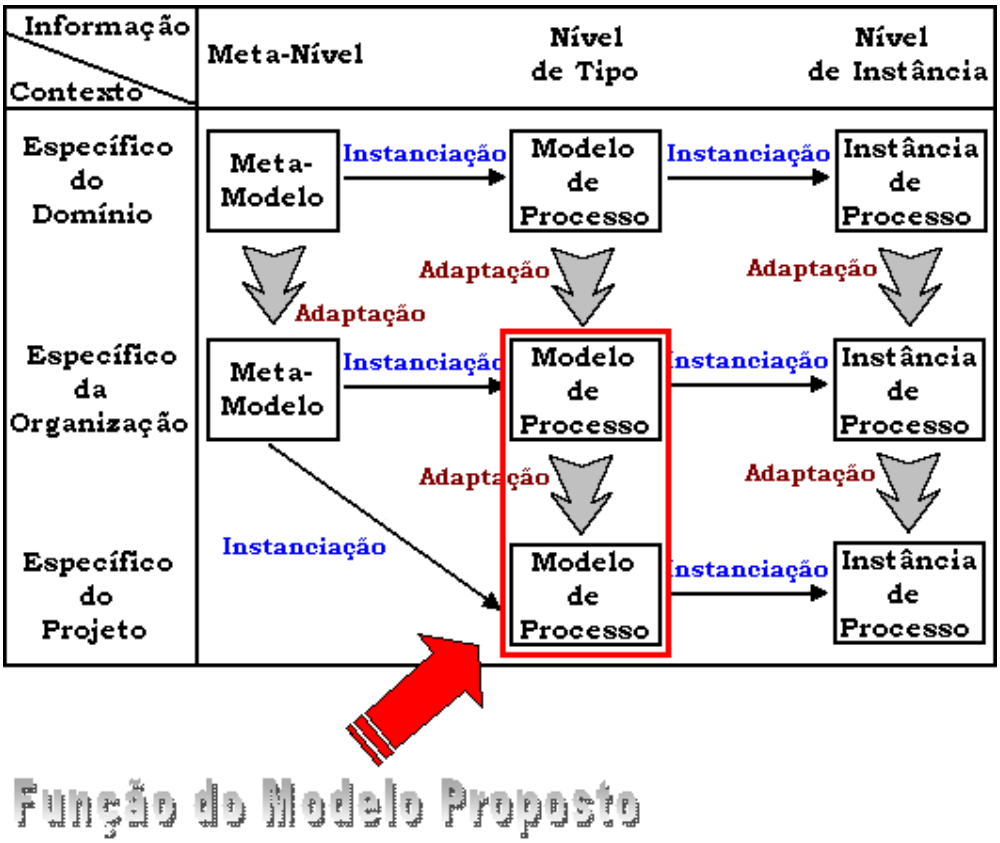


Figura 2.6: Adaptação x Instanciação - adaptado de (RUPPRECHT et al, 2000)

Nesta figura, as colunas representam os diferentes níveis de abstração de informação em que processos de software podem ser representados. O meta-nível informa os elementos que podem ser utilizados para construir modelos de processo. No nível de

tipos, encontram-se os modelos de processo propriamente ditos. Por sua vez, instâncias de processo de software são modelos de processo com o nível de informação necessário para serem executados. As linhas da figura correspondem aos diferentes contextos em que processos de software podem ser utilizados. Um processo de software, representado em determinado nível de abstração, pode ser utilizado em um domínio de aplicação específico (*template* ou *framework* de processo), em uma organização de software (processo padrão) ou em um projeto de software (processo adaptado).

Conforme apresentado nesta figura, a adaptação de processos de software pode gerar processos de software adequados a diferentes contextos. Trata-se, portanto, de uma transformação vertical (de processos de software mais gerais para outros mais específicos, no que diz respeito à aplicabilidade dos mesmos), que envolve basicamente alterações na estrutura do processo. A instanciação de processos de software, no entanto, atua no sentido de aumentar o nível de informação presente nos processos de software, podendo então ser considerada uma transformação horizontal (de processos mais abstratos para outros mais concretos). Conforme visto na figura, a função do APSEE-Tail é adaptar um processo abstrato, denominado processo padrão, pois é específico de uma organização, para o contexto de um projeto de software a ser conduzido na mesma. O processo de software resultante, denominado de processo adaptado, continua a ser abstrato, necessitando passar por instanciação para ser executado no projeto de software real. Isto é evidenciado através da figura 2.7.

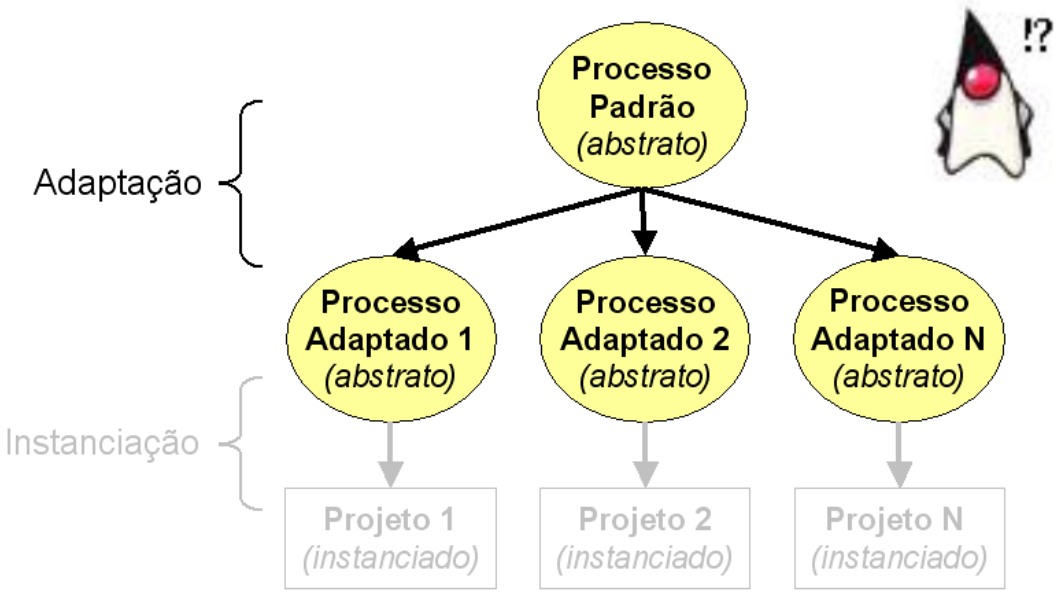


Figura 2.7: Passos para a utilização do processo padrão em projetos de software

Esta figura retrata exatamente os passos pelos quais o processo padrão deve passar, de modo a ser utilizado para guiar a execução de um projeto de software. Primeiramente, ele deve ser adaptado para a realidade do projeto, dando origem a um processo ainda abstrato. Este processo deve então ser instanciado, ou seja, devem ser alocados pessoas e recursos para as suas atividades, bem como deve ser estabelecido o cronograma a ser seguido. O objetivo do APSEE-Tail é fornecer suporte automatizado à etapa de adaptação, ficando a instanciação e demais fases do meta-processo a cargo do PSEE com o qual fará interface.

2.4 Processo Padrão

Não existe um processo de software que possa ser genericamente aplicado. Assim, na definição de um processo deve-se considerar a sua adequação às tecnologias envolvidas, ao tipo de software em questão, ao domínio de aplicação, ao grau de maturidade (ou capacitação) da equipe em engenharia de software, às características próprias da organização, ao projeto, à distribuição geográfica da equipe, entre outros fatores (ROCHA e MAIDANTCHIK apud MACHADO et al, 2000).

Em uma organização, diversos projetos podem coexistir, possuindo características específicas. Entretanto, existe um conjunto de elementos fundamentais que se deseja sejam incorporados em qualquer processo definido para os projetos. Emam apud (MACHADO et al, 2000), define o conjunto destes elementos fundamentais como o processo padrão, ou seja, o processo básico que guia o estabelecimento de um processo comum na organização. Desta forma, um processo padrão define uma estrutura única a ser seguida por todas as equipes envolvidas em projetos de software numa organização (MAIDANTCHIK apud MACHADO et al, 2000), independente das características do software que está sendo desenvolvido. Humphrey (1989) define um conjunto de razões para a definição de um processo padrão:

- Redução dos problemas relacionados a treinamento, revisões e suporte à ferramentas;
- As experiências adquiridas nos projetos são incorporadas ao processo padrão e contribuem para melhorias em todos os processos definidos;
- A utilização de padrões de processo, fornecendo as bases para medições de processos e qualidade;
- Economia de tempo e esforço em definir novos processos adequados a projetos.

Na literatura atual, observa-se uma tendência à utilização de processos padrões na definição de processos. A norma ISO/IEC 12207, o ISO/IEC TR 15504 e o SW-CMM definem um processo padrão como um ponto base a partir do qual um processo especializado (adaptado) poderá ser obtido de acordo com as características de um projeto de software específico. Desta forma, ser adaptável e configurável torna-se um importante requisito a ser atingido na definição de um processo padrão (MACHADO et al, 2000).

Este tipo de esforço de padronização sofre, portanto, com um problema inerente: para acomodar todos os tipos de iniciativas de desenvolvimento em uma organização, o padrão vai inevitavelmente estar em um nível de abstração que atenda às necessidades de todos os projetos, mas não vai ser capaz de fornecer apoio específico às atividades individuais do projeto. A diversidade de projetos de tecnologia da informação frustra qualquer tentativa direta de sistematizar os processos usados para seu desenvolvimento (MACHADO et al, 2000). Não existe um único formato que seja adequado para efetivamente apoiar a realização de todos os projetos, conforme dito anteriormente.

A figura 2.8 ilustra a especialização do processo padrão de uma organização até o nível de processo instanciado. Com o intuito de aplicar o processo padrão de uma organização nos diferentes projetos conduzidos pela mesma, é necessário adaptá-lo a cada um desses contextos específicos, levando em consideração diversos fatores (características do contexto), alguns deles citados na figura. Neste nível de especialização, o processo de software adaptado ainda não pode ser executado; é

necessário que o mesmo seja instanciado, através do acréscimo de informações às suas atividades, tais como: artefatos produzidos e consumidos, datas de início e fim, ferramentas e recursos utilizados, agentes responsáveis, além de um cronograma e orçamento para o projeto como um todo.

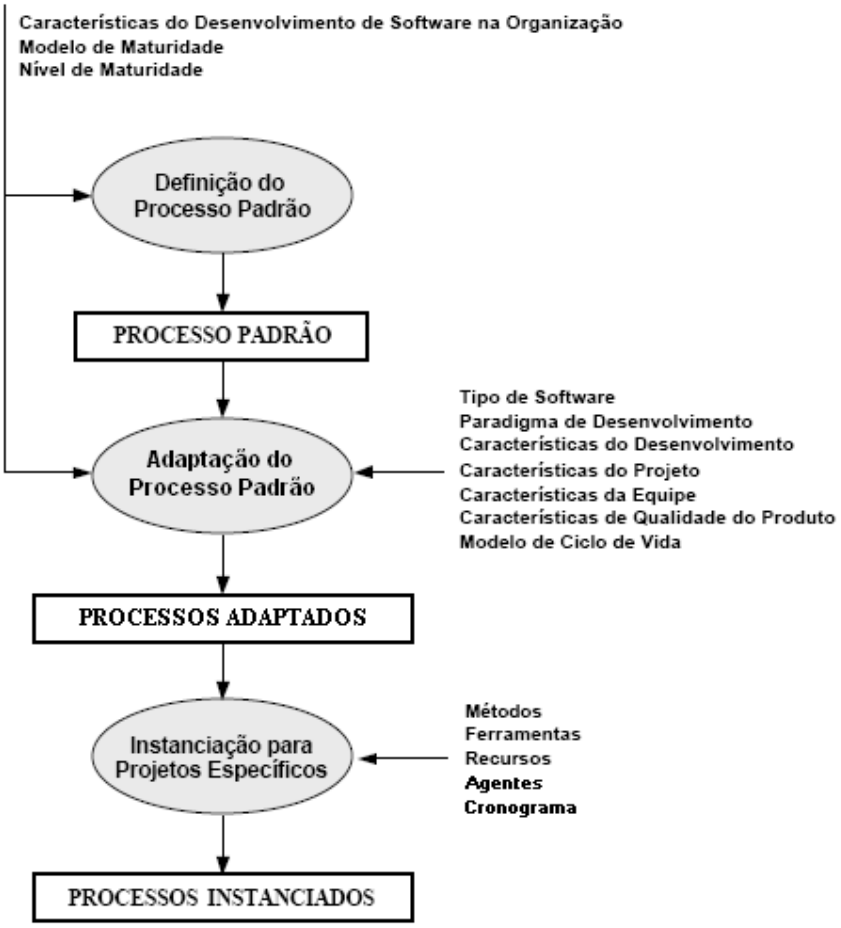


Figura 2.8: Especialização do processo padrão (MACHADO et al, 2000)

2.5 Tipos de Adaptação

Durante a modelagem de processos é comum enfrentar situações em que um ou mais *templates* (ou sub-processos) precisam ser recuperados, adaptados e combinados para a composição de um novo processo. A adaptação é necessária por diferentes razões, tais como: aspectos tecnológicos específicos do problema tratado, a falta de experiência da organização no uso de uma determinada técnica proposta por um *template*, ou a natureza inerentemente específica do software sob desenvolvimento. Assim, um protocolo de composição é necessário para guiar a adaptação de modelos de processos gerados a partir de *templates* reutilizáveis (REIS, 2002).

Esse tópico, sob a ótica da reutilização de software, lida com requisitos contraditórios: enquanto de um lado a descrição de protocolos para componentes de software deve ser precisa e rígida, por outro lado alguma flexibilidade é necessária para permitir a adaptação de componentes em um maior número de contextos. Quanto à

adaptação de *templates* de processos de software, não existe uma abordagem universalmente aceita. De fato, este é um tópico pouco explorado na literatura especializada (REIS, 2002). Visto que - ao contrário de componentes de software, executados por máquinas - os processos de software são modelados e executados por humanos, tem-se um consenso de que modificações em componentes recuperados devem ser toleradas (PERRY, 1996) (JORGENSEN apud REIS, 2002).

O projeto atual de protocolos de composição e adaptação de processos deve levar em consideração a expectativa do surgimento, em futuro próximo, de ferramentas que automatizem a recuperação e adaptação automática de templates e seus componentes. Assim, partindo-se das características dos protótipos existentes atualmente, tais como a ferramenta ProTail descrita em (MÜNCH; SCHMITZ; VERLAGE, 1997), deve-se levar em conta que tais ferramentas exigem que o usuário forneça como entrada o(s) *template*(s) candidato(s) à adaptação e a descrição rigorosa das características do contexto destino, obtendo como resultado um processo adaptado sintaticamente válido, pronto para ser instanciado e executado, ou aperfeiçoado ainda mais pelo engenheiro. Portanto, em primeiro lugar, as modificações permitidas na adaptação e composição de *templates* devem ser derivadas das regras gerais para boa formação sintática de modelos de processos executáveis (REIS, 2002).

O meta-modelo proposto por Reis (2002), para construção de *templates* de processos de software, oferece três tipos de restrições a serem escolhidas pela organização-usuária, para guiar o processo de adaptação:

- A **adaptação livre** (*free adaptation*) não apresenta qualquer restrição na adaptação de um *template* recuperado. Desta forma, na construção de um processo derivado a partir de um *template*, o projetista é livre para remover ou adicionar quaisquer elementos, desde que o resultado final seja um processo sintaticamente válido. Este é o tipo de adaptação adotado pelo APSEE-Tail;
- A **adaptação guiada por políticas** (*policy-based adaptation*) está baseada no fato de que políticas expressam propriedades gerais e ortogonais aos *templates*, devendo ser respeitadas mesmo na utilização dos *templates* em novos contextos. Assim, na adaptação guiada por políticas, as políticas habilitadas no *template* original não podem ser removidas nos processos derivados e, por conseguinte, o processo resultante será restrito ao expresso pelas políticas originais;
- A **adaptação restrita** que preserva os elementos sintáticos originais (*restricted adaptation*) segue uma linha que ainda privilegia a flexibilidade, mas impede que os principais componentes inseridos por um engenheiro de processos sejam removidos nas adaptações posteriores. Desta forma, tal abordagem restringe a adaptação de *templates* e componentes recuperados, enquanto que os elementos sintáticos originais são preservados.

Para exemplificar a diferença entre estes tipos de adaptação, será utilizado como exemplo aquele fornecido em (REIS, 2002). A figura 2.9 apresenta um *template* exemplo. Esse exemplo é composto por duas atividades: Modelagem de Dados (atividade normal do tipo *DataModeling*) e Projeto de Bases de Dados (tipo *DatabaseDesign*). Os artefatos envolvidos são: Modelo de Dados (produzido pela atividade Modelagem de Dados e fornecido à atividade de Projeto), Requisitos (artefato fornecido como entrada para as duas atividades do modelo), e Projeto de Bases de Dados (produzido pela atividade de Projeto). Finalmente, a Política Estática *Development Produce Artifacts*, apresentada na figura 2.10, está habilitada no *template*.

Por fim, a atividade Modelagem de Dados possui uma conexão simples do tipo *end-start*, tendo como destino a atividade Projeto de Bases de Dados.

A figura 2.11 apresenta seis processos candidatos (com rótulos numéricos), representando adaptações realizadas a partir do *template* original. No caso da adaptação livre, somente a adaptação de número quatro (4) não é válida, por incluir uma conexão simples que introduz um ciclo no processo (conexão *end-start* entre as atividades Revisão e Modelagem de Dados). No caso da adaptação guiada por políticas, considera-se que a adaptação dois (2) constitui um exemplo inválido (por não satisfazer a política habilitada), além do item 4 (por não satisfazer a adaptação livre). Considerando os candidatos apresentados na figura 2.11, para a adaptação restrita, somente os itens 5 e 6 são considerados adaptações válidas, pois os itens de 1 a 3 removem elementos sintáticos descritos no *template* original. Como nos outros tipos de adaptação fornecidos, a adaptação número 4 foi excluída por constituir um exemplo de processo sintaticamente incorreto.

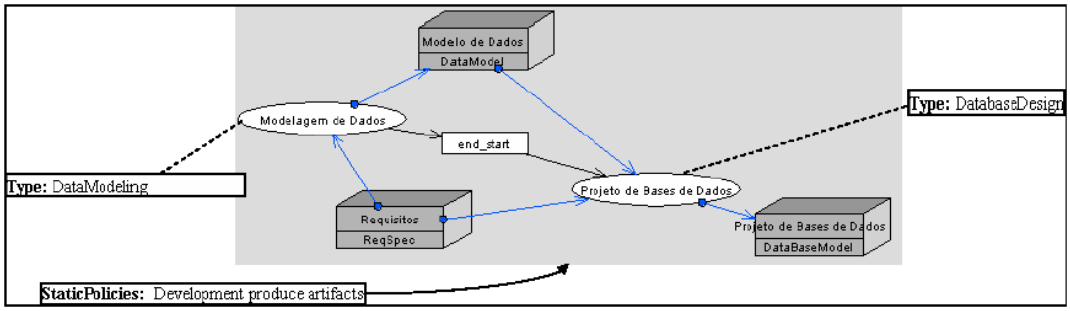


Figura 2.9: Exemplo de *template* fornecido para adaptação (REIS, 2002)

ID: "Development produce artifacts"
Description: "A Development activity that does not have output artifacts is forbidden".
Mandatory: True
Interface: a: Activity;
Enabling Properties: a.get_type() **sub_type_of** "Development"
Properties: a.get_output_artifacts().get_size() >= 1

Figura 2.10: Exemplo de política estática (REIS, 2002)

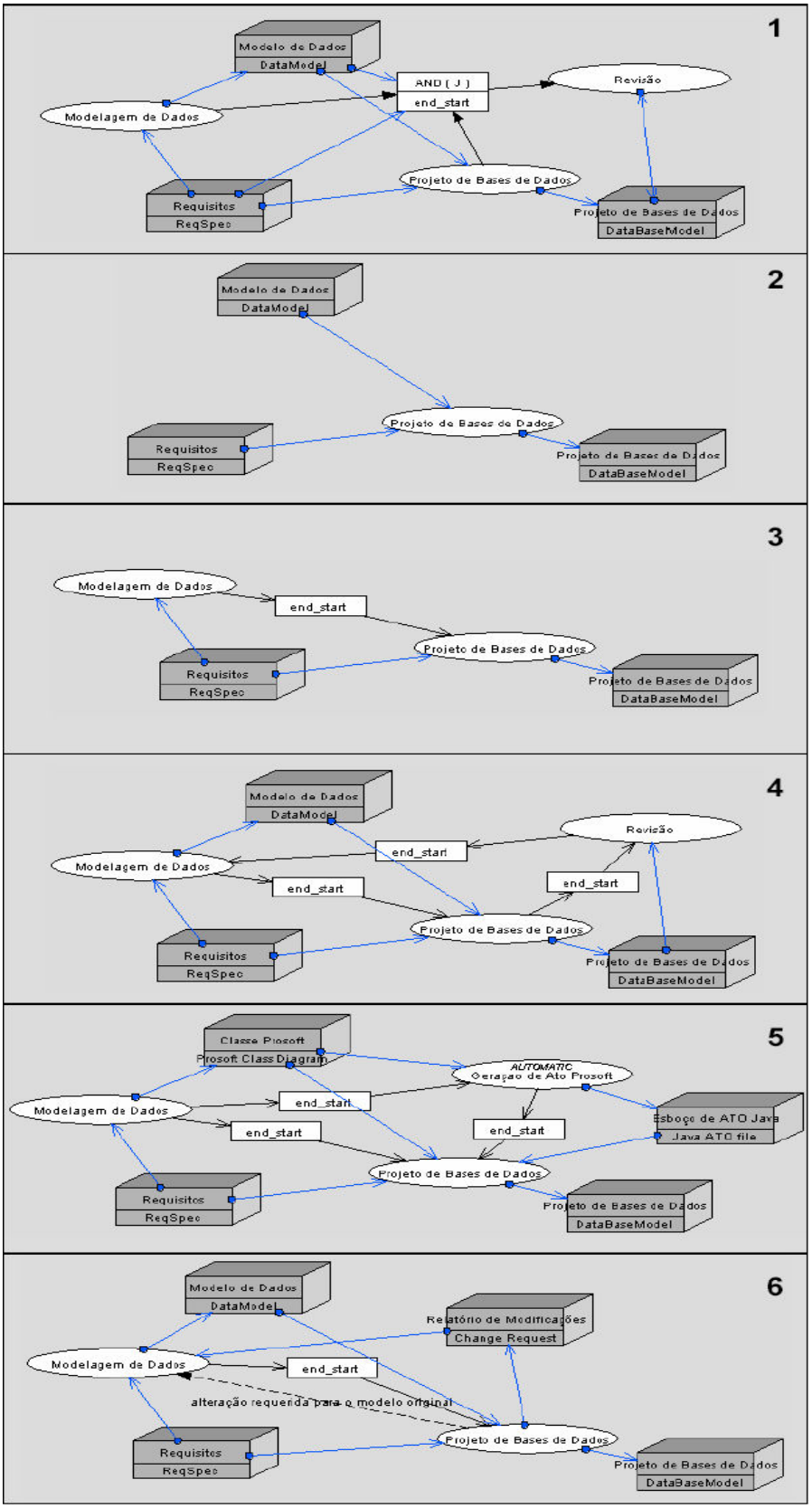


Figura 2.11: Exemplos de adaptações do *template* apresentado (REIS, 2002)

2.6 Abordagens Pesquisadas

O fato de que processos de software devem ser adaptados para os ambientes específicos nos quais serão aplicados, de forma a serem aceitos e terem seu uso maximizado, é bem conhecido. Basili et al (1987), apresentam uma metodologia para melhoria do processo de software, através de sua adaptação para as metas de um ambiente e projeto específicos. A metodologia proposta é suportada pela ferramenta denominada TAME (*Tailoring A Measurement Environment*).

Posteriormente, trabalhos foram feitos no sentido de determinar o grau de adequação entre um processo de software e o ambiente no qual ele é usado. Pérez, Emam e Madhavji (1996), descrevem um sistema para avaliar tal adequação, baseado em um modelo de contingência, derivado empiricamente e consistindo dos atributos de um modelo de processo de software e de seu ambiente, e dos relacionamentos entre eles.

O desenvolvimento de uma instância de processo de software que é especificamente adaptada para o seu ambiente é uma tarefa difícil, em virtude dos diferentes tipos de informação que devem ser agregados e das alterações que devem ser realizadas no processo de software original. Conseqüentemente, este problema raramente tem sido verdadeiramente resolvido. Ao invés disso, grande parte das descrições de processos de software é representada em um alto nível de abstração, de forma a poderem assumir diferentes variantes. Isto é freqüentemente referido como um *framework* de processos. Neste sentido, pode-se destacar a proposta de Baldassarre et al (2002), que propõe o ProMisE, um *framework* baseado em padrões de processo, capaz de capitalizar as experiências obtidas pelo uso de um modelo de processo em diversos ambientes.

Enquanto esta é a abordagem geral, algumas notáveis exceções existem. Por exemplo, o processo padrão do governo alemão "*Das Vorgehensmodell*" (ou V-Model), oferece meios explícitos para ser adaptado para projetos específicos. Entretanto, a adaptação é limitada a deleção de elementos do processo padrão, o que restringe demais as possibilidades de customização. A ferramenta ProcePT (*Process Programming and Testing*), apresentada em (HAUSEN; WELZEL, 1997), automatiza a adaptação do V-Model para diferentes projetos, através das regras de deleção de atividades e produtos do processo. Assim como o V-model, a norma ISO/IEC 12207 e o *framework* RUP (*Rational Unified Process*) também são adaptáveis para diferentes organizações e projetos de software, sendo que a adaptação do RUP é suportada pela ferramenta RUP-Builder (2002). Uma metodologia de adaptação para projetos específicos do processo de manutenção do ISO/IEC 12207 é apresentada em (POLO, 1999), sendo suportada pela ferramenta automática MANTOOL.

Nos ambientes de pesquisa, métodos mais sofisticados para a adaptação de processos de software têm sido desenvolvidos, tais como a ferramenta ProTail, que suporta adaptação semi-automática de modelos de processo usando regras de adaptação (MÜNCH; SCHMITZ; VERLAGE, 1997). Fortemente relacionada com a adaptação explícita da ferramenta ProTail, aparece a abordagem *PuLSE Baselining and Customization* (PuLSE-BC), uma técnica para customização de processos, a qual também descreve o que necessita ser provido em adição a um processo, de tal forma que a customização possa ser realizada baseada nas características ambientais (SCHMID; WIDEN, 2000). Entretanto, ao contrário da ferramenta ProTail, a abordagem PuLSE-BC não foca na adaptação técnica de modelos de processos e sim na forma mais apropriada de adaptar o processo através dos aspectos de sua linha de produção.

Abordagens mais recentes têm dado bastante destaque ao papel do conhecimento no processo de adaptação, utilizando técnicas de Inteligência Artificial para representar, armazenar, recuperar e utilizar corretamente esse conhecimento. Berger (2003), define um método de instanciação de processos de software apoiado pela ferramenta AdaptPro, através da qual é disponibilizado ao gerente de projetos o conhecimento sobre instanciação de processos de software acumulado pela organização de software em projetos anteriores. Esta abordagem fundamenta-se nos conceitos de gerência de conhecimento e de ambientes de desenvolvimento de software orientados à organização, tendo sido acoplada à Estação TABA (TRAVASSOS, 1994). Rupprecht et al (2000), apresentam uma abordagem fundamentada em um kit de construção de processos de software, que consiste de um repositório para o gerenciamento de blocos básicos para construção de processos de software e um vetor de operadores para adaptar tais blocos. Este kit foca no projeto conceitual de processos de software em termos de ontologias, enquanto ao mesmo tempo oferece suporte automatizado para adaptação de processos de software. Henninger (1998) apresenta uma abordagem que utiliza um sistema baseado em regras, organizadas sob a forma de árvores de decisão, para adaptar processos de software para as necessidades específicas de projetos individuais e usa técnicas de aprendizagem organizacional para modificar processos de software, de forma a atender às necessidades da organização de software. Esta abordagem é suportada pela ferramenta GUIDE, uma aplicação Web que utiliza o repositório baseado em casos BORE para capturar experiências em projetos anteriores. Vale destacar ainda a abordagem, apresentada em (AHN, 2003), que defende a adaptação de processos de software baseada em características do projeto a ser conduzido, experiências anteriores de adaptação e iniciativas de melhoria. Esta abordagem utiliza a combinação de inferência baseada em conhecimento com raciocínio baseado em casos para adaptar processos de software, em virtude dos diferentes tipos de conhecimento envolvidos na adaptação.

Seguindo a tendência atual da área à utilização de processos padrão na definição de processos especializados, Machado et al (2000) apresentam a ferramenta DefPro, construída para auxiliar na definição de um processo padrão, de acordo com a norma ISO/IEC 12207, com modelos de maturidade tais como SW-CMM e ISO/IEC 15504 e com as características da organização de software. Coelho (2003), por sua vez, apresenta o MAPS (Modelo de Adaptação de Processos de Software), o qual tem por objetivo adaptar o processo padrão de uma organização para os projetos conduzidos na mesma, baseado nas características desses projetos e em adaptações anteriores, utilizando uma abordagem orientada a artefatos.

As abordagens apresentadas neste parágrafo seguem um caminho um pouco diferente daquelas apresentadas anteriormente, possuindo em sua grande maioria um domínio de aplicação mais restrito. Soligen et al (1999) apresentam uma metodologia para adaptar processos de software, através da identificação e melhoria das áreas do processo que não atendem aos requisitos de qualidade pretendidos para o produto de software a ser construído pelo mesmo. Em (SEO, 2000), é apresentada uma técnica de adaptação de processos de teste, a qual utiliza o paradigma de desenvolvimento baseado em componentes e é suportada pela ferramenta AutoTP (*Automatic Tailoring Test Process Tool*). Yoon et al (2001) propõem um método sistemático para formalizar um processo padrão através de módulos de processos reutilizáveis e encapsulados, para adaptar esses módulos de processos e para a verificação desses processos adaptados. Este método é suportado pelo protótipo SoftPM. Soluções baseadas em políticas de

instanciação, tal como descrito no modelo APSEE (LIMA REIS, 2003), adotam uma postura diferente por fornecer solução automatizada para a escolha de agentes e recursos a partir de critérios genéricos definidos a *priori*.

Na tabela 2.1, é apresentado um quadro comparativo entre as abordagens de adaptação pesquisadas que mais influenciaram o trabalho. Os critérios de comparação foram os seguintes: tipo de adaptação (L para livre, P para guiada por políticas e R para restrita); paradigma de adaptação (At para orientada à atividades, Ar para orientada a artefatos e Bc orientada a blocos de construção); tipo de conhecimento usado na adaptação (Cc para caracterísicas do contexto, Ca para casos de adaptação, Da para diretrizes de adaptação e Pb para processo base); estratégia de adaptação utilizada (RBC para raciocinio baseado em casos e RBR para raciocínio baseado em regras); nível de automatização da adaptação (M para manual, S para semi-automática e A para automática).

Tabela 2.1: Quadro comparativo das abordagens que mais influenciaram o trabalho

Critério	*Münch (1997)*	*Henninger (2001)*	*RUP-Builder (2002)*	*Ahn (2003)*	*Coelho (2003)*
Tipo	L	L	L	L	L
Paradigma	At	At	Bc	At	Ar
Estratégia	RBR	RBR		RBC e|ou RBR	RBC e RBR
Automatização	S	S	S	M	M
Conhecimento	Cc, Da	Cc, Da	Pb	Cc, Ca e Da	Cc, Ca, Da e Pb

Segundo Reis (2003), idealmente a adaptação de processos de software pode se valer de soluções baseadas em conhecimento, com o objetivo de fornecer sugestões de adaptações em função de soluções semelhantes adotadas em situações similares, levando em consideração as características ambientais e organizacionais correntes.

2.7 Considerações

Este capítulo apresentou alguns dos principais conceitos e aspectos, relacionados à adaptação de processos de software, necessários para o melhor entendimento do restante do texto. Foram apresentadas as diferentes visões acerca da adaptação, sua contextualização dentro do meta-processo de software, o cenário ideal para sua aplicação, a sua diferença em relação à instanciação, os tipos existentes e, por fim, as abordagens pesquisadas. Um parêntese foi aberto para uma explanação mais detalhada sobre o processo padrão.

Segundo Coelho (2003), a relevância da adaptação de processos está ligada ao fato de que a qualidade do software produzido e a eficiência e eficácia do desenvolvimento estão diretamente ligadas à qualidade do processo adotado. Assim, utilizando o processo mais adequado ao projeto, pode-se conseguir melhorias no desenvolvimento e na qualidade do produto.

Porém, ainda segundo Coelho (2003), apesar da importância da adaptação de processos, essa é uma área que ainda é negligenciada ou tratada de forma apenas superficial por grande parte das metodologias de desenvolvimento de software. A falta

de uma melhor definição, nas metodologias, sobre a forma como a adaptação de processos deve ser realizada acaba dificultando seu uso nas organizações desenvolvedoras. Por conta disso, muitas organizações acabam tomando como base um modelo mais genérico, como o SW-CMM, e sendo obrigadas a desenvolver métodos de adaptação próprios que possam ser utilizados para seus processos específicos.

Tendo sido então apresentada a fundamentação teórica do trabalho, o próximo capítulo apresenta a visão geral do APSEE-Tail, seu escopo de atuação, suas principais características, seus componentes, suas funções e a estratégia de adaptação que utiliza.

3 VISÃO GERAL DO APSEE-TAIL

Neste capítulo, é apresentada uma visão geral do APSEE-Tail: seu escopo; suas principais características, seus componentes, seu funcionamento e a estratégia de adaptação adotada. Conforme visto no capítulo 1, seção 1.3 - Objetivos, o APSEE-Tail visa auxiliar o engenheiro de processos durante a adaptação do processo padrão de uma organização de software para os projetos a serem conduzidos na mesma. Partindo do fato de que a adaptação é uma atividade que utiliza intensivamente vários tipos de conhecimento, o APSEE-Tail adota uma estratégia de adaptação livre[18], orientada à atividades[19] e baseada:

- No suporte ao 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 APSEE-Tail tem como principais características: possui caráter genérico quanto à sua aplicação, ou seja, não é restrito a um único *framework* ou família de processos; 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 a realização de uma adaptação, estes parâmetros podem ser alterados de acordo com as necessidades da organização de software.

O restante deste capítulo é como segue.

- Na seção 3.1, faz-se necessário deixar claro o escopo de atuação do APSEE-Tail;
- A seção 3.2 apresenta a arquitetura do APSEE-Tail;
- A seção 3.3 apresenta informalmente os componentes do APSEE-Tail;
- Na seção 3.4, é apresentado o funcionamento do APSEE-Tail, em termos das funções suportadas pelo mesmo;

---

[18] Conforme mencionado no capítulo anterior, a adaptação livre permite a inclusão, exclusão e alteração dos elementos de um processo de software.

[19] Define-se sob que condições as atividades do processo padrão serão aplicadas nos projetos da organização de software.

- Na seção 3.5, a estratégia de adaptação adotada pelo APSEE-Tail é apresentada em detalhes, com os algoritmos pertinentes e pequenos exemplos;
- Por fim, a seção 3.6 traz algumas considerações acerca deste capítulo.

3.1 Delimitação do Escopo

A adaptação é apenas uma etapa do meta-processo de software, porém os limites entre ela e suas etapas vizinhas, em especial modelagem e instanciação, é muito tênue. Cabe aqui então a adoção de algumas premissas, de modo a esclarecer e delimitar o escopo de atuação do APSEE-Tail:

- A função do APSEETail é gerar um processo adaptado sintaticamente coerente[20] a partir de outro processo também coerente. Portanto, a garantia da coerência de qualquer processo importado pelo APSEE-Tail, é responsabilidade do ambiente externo onde tal processo foi definido (processo padrão) ou atualizado (processos adaptados);
- O APSEE-Tail não provê suporte à modelagem, concretização, instanciação ou execução de processos de software, sendo papel de um ambiente externo prover suporte à estas etapas do meta-processo, tal qual apresentado na figura.

3.2 Arquitetura

Conforme pode ser observado na figura 3.1, a arquitetura do APSEE-Tail é dividida em três camadas. Na camada de **interação com usuário**, se encontram as funções que o APSEE-Tail oferece à organização usuária, agrupadas nas duas categorias apresentadas. A camada **mecanismos para adaptação de processos** inclui os três componentes lógicos que, conjuntamente, implementam a estratégia de adaptação adotada, ou seja, provêem a semântica (comportamento) do mecanismo de adaptação do APSEE-Tail. Por fim, a camada **repositório de conhecimentos** é responsável pela persistência de todo o conhecimento gerado e utilizado nas / pelas camadas superiores.

[20] Processos de software podem apresentar coerência sintática, conforme descrito no capítulo 1, e semântica. Um processo de software está semanticamente coerente quando os elementos de seu modelo estão dispostos de tal maneira que realmente colaborem para a construção do produto de software final requerido. Este tipo de coerência está fora do escopo deste trabalho, pois é baseada principalmente em critérios subjetivos, ou seja, depende muito das visões do gerente de projeto e do engenheiro de processos. Logo, é correto afirmar que um processo adaptado gerado pelo APSEE-Tail estará coerente sintaticamente, mas não necessariamente do ponto de vista semântico.

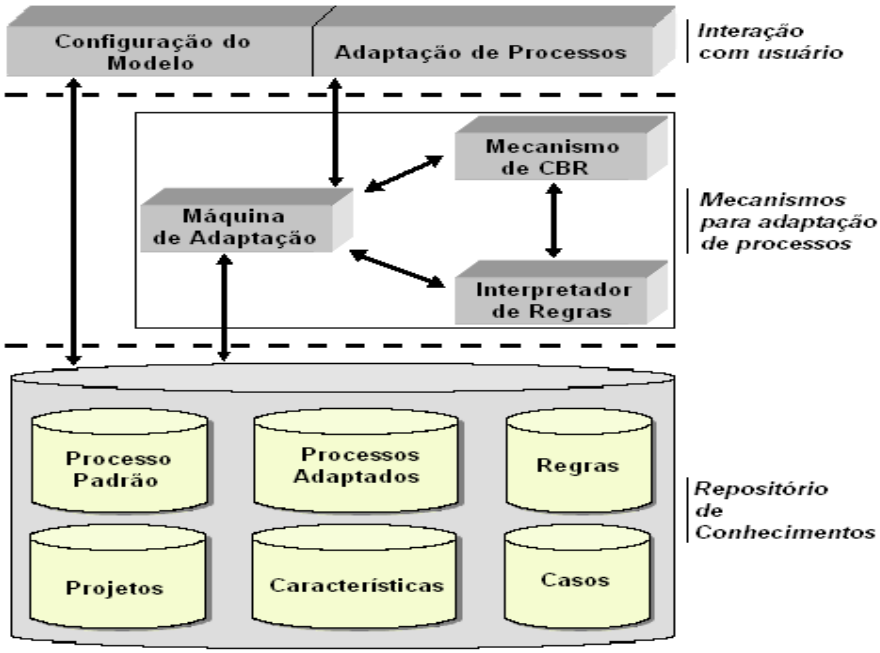


Figura 3.1: Arquitetura do APSEE-Tail

3.3 Componentes

Conforme pode ser visto no diagrama de classes UML da figura 3.2[21], o APSEE-Tail é composto basicamente: pelo processo padrão da organização de software (classe *Standard*), o qual possui um modelo de processo (classe *AbsProcessModel*) e um conjunto de regras que guiam a adaptação do mesmo (classe *Rule*); pelos projetos de software conduzidos na organização de software (classe *Project*); pelos tipos de característica utilizados para definir os projetos de software (classe *CharacteristicType*); pelos processos de software originados a partir da adaptação do processo padrão para projetos da organização (classe *AdaptedProcess*); e por fim, pelos casos de adaptação, os quais armazenam informações acerca de adaptações realizadas anteriormente (classe *Case*).

[21] Em virtude do APSEE-Tail, conforme mencionado no capítulo 1, fazer parte de um projeto de cooperação internacional entre Brasil e Alemanha, toda a sua especificação (diagramas UML, IDEF0, ATOs algébricos e Regras em Gramática de Grafos) está em inglês. Há uma pequena diferença de nomenclatura entre os nomes das classes em UML e no Prosoft-Algébrico. Enquanto no primeiro, os nomes sempre aparecem no singular, no segundo, dependendo da estrutura de dados que os representem, seus nomes podem aparecer no plural (no caso de listas, conjuntos e mapeamentos) ou no singular (no caso de registros).

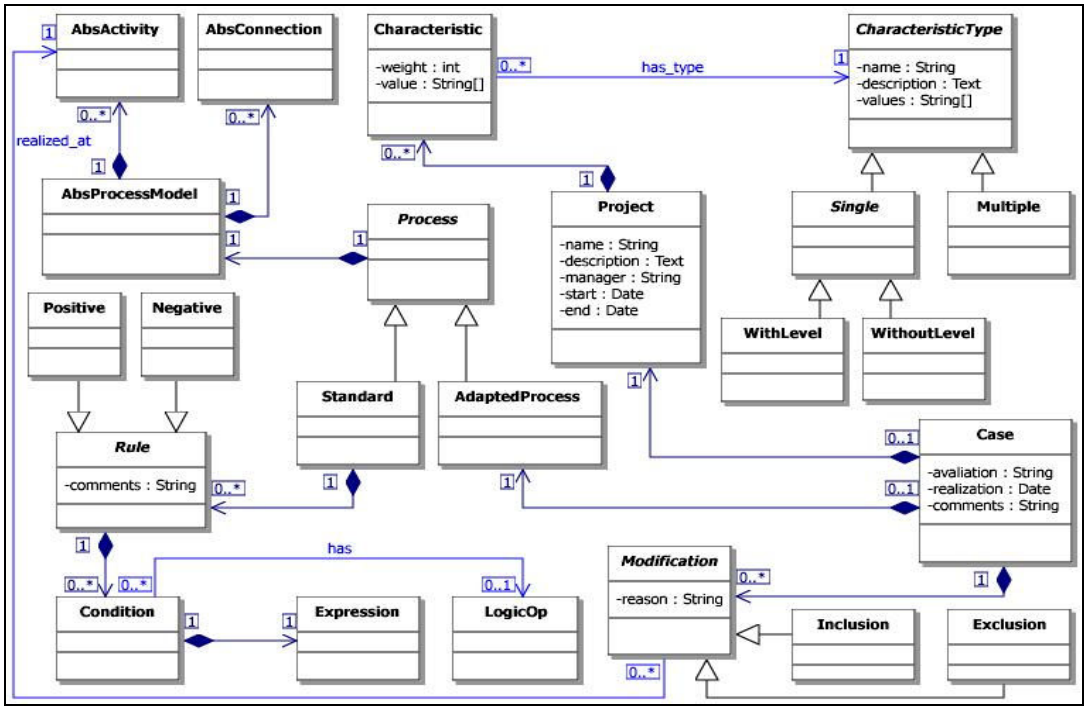


Figura 3.2: Componentes do APSEE-Tail

A seguir, cada um destes componentes será brevemente descrito. Uma descrição mais detalhada de cada componente, seus atributos e comportamento, pode ser encontrada no capítulo 4, devidamente especificada em Prosoft-Algébrico.

3.3.1 Processo Padrão e Adaptados

Tanto o processo padrão quanto os processos adaptados originados a partir deste utilizam a mesma estrutura para representar o seu modelo de processo. A diferença básica entre ambos é que como o APSEE-Tail trata da adaptação de um único processo padrão, não há a necessidade de um identificador para o mesmo. Além disso, o processo padrão possui um conjunto de regras de adaptação associadas. A justificativa para o uso da mesma classe para representar a estrutura de ambos os tipos de processos é que, para o APSEE-Tail, 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.

A estrutura utilizada para representar os modelos do processo padrão e dos processos adaptados é a mesma utilizada por Reis (2002), para representar seus *templates*. A justificativa para tal é simples: embora de caráter genérico, o APSEE-Tail foi prototipado no ADS Prosoft-Java e fracamente acoplado ao APSEE, cujos modelos de processo abstratos são na realidade *templates*. As classes que compõe um modelo de processo abstrato podem ser vistas, na íntegra, no anexo 2. Basicamente, um modelo de processo abstrato 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 utilizada deve ser suportada em uma ferramenta externa e a seguir convertida para o formato reconhecível pelo APSEE-Tail. A figura 3.3 mostra um modelo de processo representado visualmente pela linguagem APSEE-PML[22]. 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 de software.

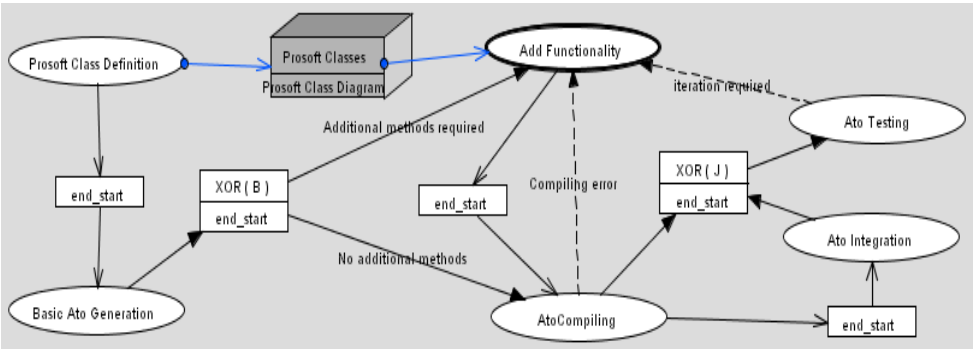


Figura 3.3: Exemplo de modelo de processo

3.3.2 Tipos de Característica

Representam o conjunto de características definidas pela organização de software e que podem ser utilizadas para caracterizar um determinado projeto de software conduzido na mesma. Cada tipo de característica possui um identificador único, um nome, uma descrição, uma espécie e um conjunto de valores que pode assumir. É interessante ressaltar que, para ser utilizado em determinado projeto de software, um tipo de característica deve ser instanciado, ou seja, deve ser associado a um ou mais de um (dependendo da sua espécie) valor dentre os possíveis, bem como lhe deve ser atribuído um peso (importância) dentro do projeto em questão. No APSEE-Tail, os tipos de característica são agrupados nas três categorias apresentadas a seguir.

3.3.2.1 Tipo de característica simples e sem nivelamento

Suas instâncias podem assumir um único valor dentre os possíveis, não havendo relação de ordem entre os mesmos. Por exemplo, uma instância do tipo de característica Tipo de Software (cujos possíveis valores são “Aplicação Web”, “Aplicação Desktop” e “Sistema Embarcado”) poderia assumir apenas um destes valores em determinado projeto de software.

3.3.2.2 Tipo de característica simples e com nivelamento

Suas instâncias podem assumir um único valor dentre os possíveis, havendo uma relação de ordem entre os mesmos. Cada possível valor de um tipo de característica dessa espécie está associado a um nível, que indica a sua ordem. Por exemplo, uma

[22] A APSEE-PML, descrita em (LIMA REIS, 2003), é a linguagem visual de modelagem utilizada pelo ambiente APSEE.

instância do tipo de 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.

3.3.2.3 Tipo de característica Múltiplo

Suas instâncias podem assumir um subconjunto de valores dentre os possíveis, não havendo relação de ordem entre os mesmos. Por exemplo, uma instância do tipo de característica Risco Técnico (cujos possíveis valores são "tecnologia imatura", "restrições de H/W e S/W existentes", "requisitos de performance e segurança", "interface com sistema legado") poderia assumir um subconjunto destes valores em um determinado projeto de software.

3.3.3 Projetos de Software

Representam os projetos de desenvolvimento de software conduzidos no âmbito da organização que está utilizando o APSEE-Tail. Um projeto de software possui um identificador único, um nome, uma descrição, um gerente responsável pela sua condução, uma data de início, uma data de término esperada e um conjunto de características, cada uma possuindo um tipo, um valor e um peso representando sua importância. A partir do processo padrão, o APSEE-Tail deve gerar para cada projeto de software um processo adaptado às suas características. Na figura 3.4, é apresentado um exemplo com o objetivo de tornar mais claro como é definido um projeto de software no APSEE-Tail.

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"}

Figura 3.4: Exemplo de projeto de software

3.3.4 Regras de Adaptação

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. A parte condicional de uma regra de adaptação é formada por testes sobre os valores das instâncias dos tipos de característica para o projeto de software corrente. Uma regra de adaptação deve ser associada a uma e somente uma atividade, possui comentários associados, um tipo (que indica se a regra expressa as condições para inclusão – Positiva – ou para não inclusão – Negativa – da atividade no/do processo adaptado) e, por fim, as condições propriamente ditas. A gramática da figura 3.4 especifica o formato da parte condicional de uma regra.

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

Figura 3.5: Gramática para a parte condicional de uma regra de adaptação

Considerando, por exemplo, um processo padrão que possuísse uma atividade chamada Gerenciamento de Versões, a qual dependesse dos tipos de característica 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 3.6. 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.

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"

Figura 3.6: Exemplo de regra de adaptação

3.3.5 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[23]. 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 APSEE-Tail, um caso de adaptação é formado basicamente por quatro partes: o problema (projeto de software conduzido na organização), a solução (processo de software adaptado às características deste projeto), as justificativas para a solução encontrada (o conjunto de modificações realizadas no processo adaptado e a descrição dos passos realizados pelo mecanismo de adaptação) e a avaliação da aplicação do processo adaptado no projeto de software real, 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, uma data de realização e o conjunto de alterações realizadas manualmente no processo adaptado, em decisão contrária a do modelo de adaptação.

[23] CBR (*Case Based Reasoning*) em inglês (ABEL, 1996).

3.4 Funções

Todas as funções[24] oferecidas pelo APSEE-Tail para a organização de software que pretende utilizá-lo estão listadas no diagrama IDEF0 da figura 3.7.

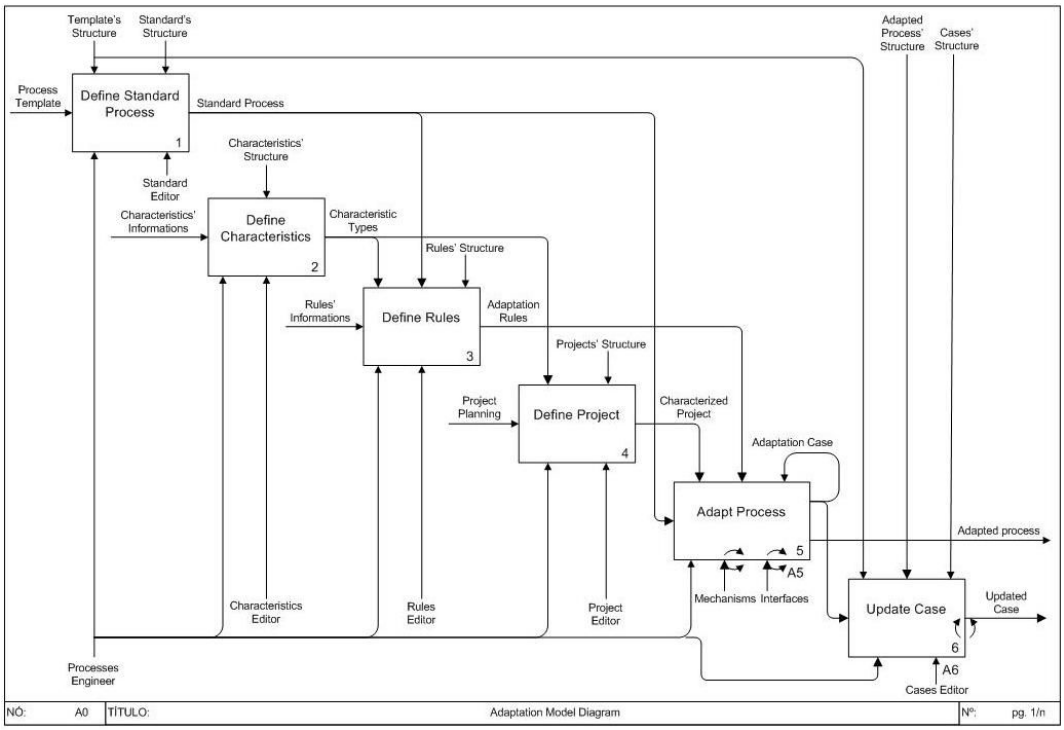


Figura 3.7: Funções do APSEE-Tail

O APSEE-Tail tem como função básica adaptar o processo padrão de uma organização de software para um projeto específico a ser conduzido na mesma. No entanto, para que isso seja possível, ele deve ser configurado com os parâmetros necessários, quais sejam: o processo padrão da organização de software, as regras que guiam a adaptação deste processo e o conjunto de tipos de característica que podem ser utilizados para caracterizar projetos de software. Sendo assim, as suas funções podem ser agrupadas em duas categorias: funções de configuração (funções 1, 2 e 3); e funções de auxílio à adaptação (funções 4, 5 e 6). Abaixo, cada uma destas funções será descrita em detalhes, salvo a de número 5, que será vista separadamente na próxima seção. Para cada função, serão discutidas suas entradas e saídas, controles e mecanismos, bem como suas subfunções[25], caso existam.

3.4.1 Definir Processo Padrão

Esta função tem por objetivo configurar o APSEE-Tail com o processo padrão (mais especificamente com o seu modelo de processo) da organização de software que o está utilizando. Conforme já mencionado anteriormente, a modelagem do processo padrão e a garantia de sua coerência sintática e semântica devem ser suportadas por um ambiente

[24] Os termos funcionalidades ou operações também podem ser usados com o mesmo significado.

[25] Os termos etapas ou passos também podem ser usados com o mesmo significado.

externo. Como o meta-modelo (estrutura) para construção de modelos de processo abstratos adotado é aquele utilizado pelos *templates* do APSEE, proposto no trabalho de Reis (2002), esta função nada mais faz do que sobrescrever o modelo do processo padrão por aquele extraído de um *template* modelado no APSEE e fornecido ao APSEE-Tail. Após esta etapa, duas situações são possíveis: em se tratando da definição de um novo processo padrão, todos os componentes do APSSE- Tail com que ele se relaciona mais fortemente (regras, casos e processos adaptados) terão suas instâncias excluídas; no caso de tratar-se apenas de uma atualização no modelo do processo padrão, haverá a necessidade de atualizar apenas o conjunto de regras de adaptação, uma vez que algumas podem estar apontando para atividades que não existam mais. Esta função é suportada pelo Editor do Processo Padrão.

3.4.2 Definir Tipos de Características

O objetivo desta função é permitir que seja definido o conjunto de tipos de característica que será utilizado pela organização de software para caracterizar seus projetos. Somente a experiência da organização na condução de projetos de software pode indicar quais são os tipos de característica que realmente influenciam os seus projetos e, portanto, devem ser definidos. Esta função é suportada pelo Editor de Tipos de Característica, o qual permite que o engenheiro de processos informe os dados necessários para o cadastro de um tipo de característica. O conjunto de informações necessárias para a definição de um tipo de característica de projeto foi representado no diagrama IDEF0 da figura 3.7 pelo que se convencionou chamar de estrutura do tipo de característica e guia a realização desta função.

3.4.3 Definir Regras de Adaptação

O objetivo desta função é permitir que seja definido o conjunto de regras que guiarão a adaptação do processo padrão. Como estas regras devem estabelecer a relação que existe entre as atividades do processo padrão e os tipos de característica existentes, ambos fazem parte do controle desta função juntamente com a estrutura da regra (indica o conjunto de informações necessárias para a definição de uma regra de adaptação). Ela é suportada pelo Editor de Regras, o qual permite que o engenheiro de processos informe os dados necessários para o cadastro de uma regra de adaptação. É importante notar que esta função depende dos resultados das funções Definir Processo Padrão e Definir Tipos de Característica, ou seja, o processo padrão deve possuir um modelo de processo não vazio-, assim como o conjunto de tipos de característica não deve ser vazio também.

3.4.4 Definir Projeto de Software

Esta função tem por objetivo permitir que se defina o projeto de software para o qual se pretende adaptar o processo padrão. Ela está dividida em duas outras subfunções, conforme pode ser visto no diagrama IDEF0[26] da figura 3.8, quais sejam Criar Projeto e

[26] Neste texto, são utilizados códigos ICOM (Input, Control, Output, Mechanism) em todos os diagramas IDEF0, a partir do primeiro nível de detalhamento.

Caracterizar Projeto. Primeiramente, o engenheiro de processos, com o auxílio do Editor de Projetos, informa os dados requeridos pela estrutura do projeto (indica o conjunto de informações necessárias para a definição de um projeto de software). Criado o projeto de software, o engenheiro de processos, através do mesmo editor, seleciona e instancia (atribui um valor e um peso à) aos tipos de característica que influenciam o mesmo. É interessante notar que esta função deve receber como entrada um planejamento formal ou informal do projeto, para que o engenheiro de processos, através da análise do mesmo, possa extrair as informações e características pertinentes. O auxílio do gerente de projetos é extremamente aconselhável.

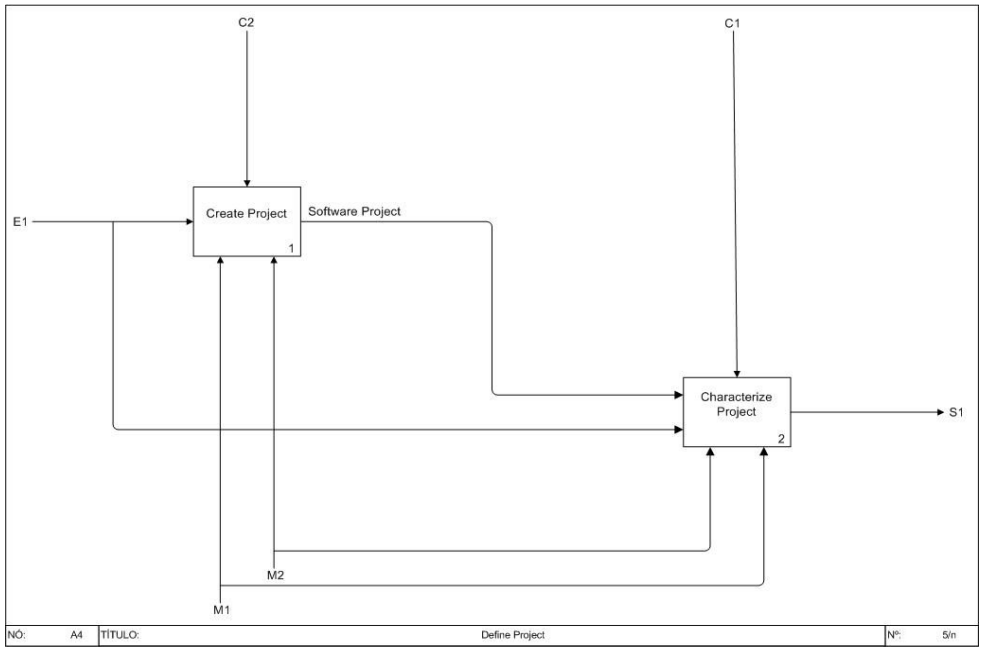


Figura 3.8: Etapas para a definição de um projeto de software

3.4.5 Atualizar Caso de Adaptação

Esta função tem como objetivos, permitir que o processo adaptado seja alterado livremente em uma ferramenta de modelagem externa de acordo com as necessidades do projeto de software em questão, bem como permitir que a aplicação de tal processo no projeto de software real seja avaliada. Após a sua avaliação, o processo adaptado não poderá mais ser alterado. Vale mencionar que esta avaliação será mais uma informação a auxiliar o engenheiro de processos na escolha de um caso de adaptação similar ao atual e que mecanismos mais elaborados para avaliação da aplicabilidade do processo de software adaptado ainda precisam ser pesquisados. A figura 3.9 apresenta o diagrama IDEF0 desta função. É interessante notar que o engenheiro de processos utiliza o Editor de Casos para comentar e avaliar os casos de adaptação, e o Editor de Processos Adaptados para exportar/importar os processos gerados para/de a ferramenta de modelagem externa.

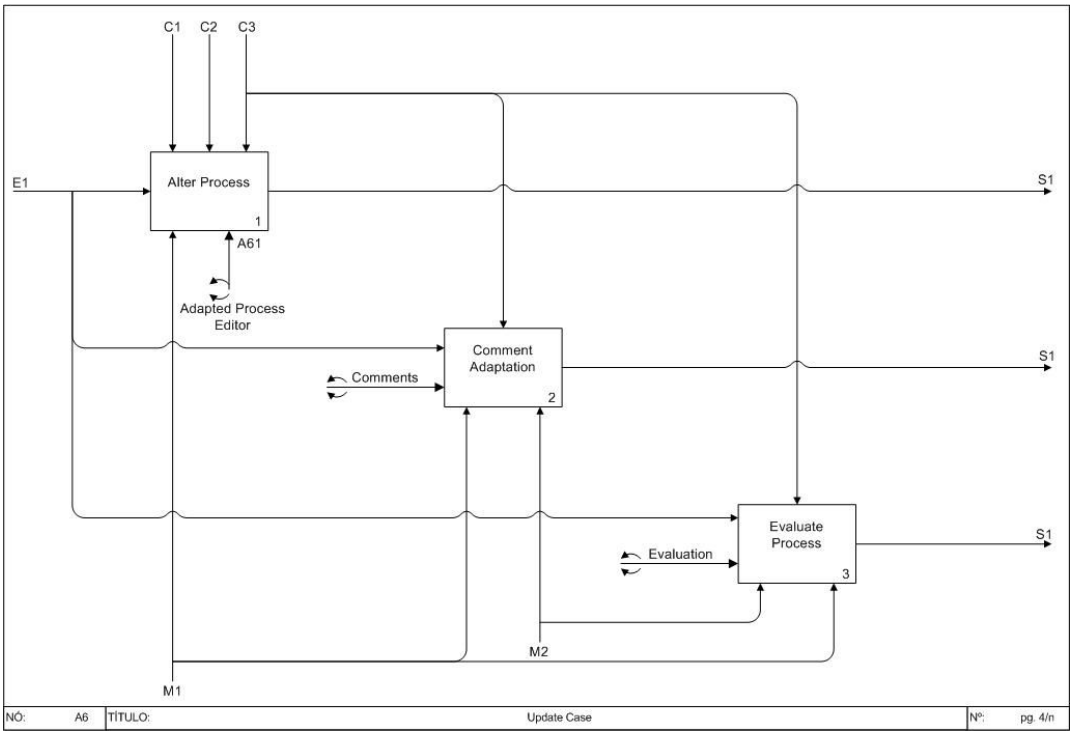


Figura 3.9: Etapas para a atualização de um caso de adaptação

O diagrama IDEF0 da figura 3.10 mostra o detalhamento da subfunção Alterar Processo Adaptado. Primeiramente, o processo adaptado deve ser convertido em um *template* e exportado para o APSEE. Com o auxílio deste ambiente o engenheiro de processos realiza as alterações necessárias no processo adaptado e, a seguir, importa o mesmo. Na realidade, não se trata apenas de uma simples substituição de modelos de processo. O APSEE-Tail irá sobrescrever o modelo do processo antigo pelo do processo novo, comparar a estrutura de ambos os processos e atualizar o conjunto de modificações do caso de adaptação associado. O pseudocódigo[27] abaixo mostra o algoritmo para atualização do processo adaptado antigo com as modificações realizadas externamente.

```
Sejam:

pa = processo adaptado antigo
pn = processo adaptado novo
pp = processo padrão
ca = conjunto de atividades de "pa" e "pn"

Então:
```

[27] Durante a aplicação de alguns dos algoritmos, descritos neste capítulo em pseudocódigo, mudanças irão ocorrer no modelo do processo adaptado. Para garantir a coerência deste processo, regras, descritas em gramática de grafos no capítulo 4, serão aplicadas. Os pontos de aplicação destas regras no pseudocódigos aparecem devidamente destacados.

```
1. Verificar se "pa" existe
2. Verificar se "pa" está associado a um caso avaliado
3. Sobrescrever o modelo de processo de "pa" pelo de "pn"
4. Para cada atividade de "ca" faça
Se atividade existir apenas em "pn"
Então //→ Trata-se de uma inclusão
      Se atividade não existir em "pp"
      Então Se atividade possuir modificação
            Então excluir modificação
            Senão incluir modificação de inclusão
      Senão Se não possuir regra associada
            Então Início
                  Se atividade possuir modificação
                  Então excluir modificação
                  Se tipo da atividade = fragmento
                  Então reaplicar algoritmo ao fragmento
                  Fim
            Senão Início
                        avaliação ← avaliação da regra associada
                        incluir ← ( ( avaliação = verdadeiro )
                                    e ( tipo da regra = positiva ) )
                                    ou
                                    ( ( avaliação = falso )
                                    e ( tipo da regra = negativa ) )
                        Se incluir = verdadeiro
                        Então Se atividade possuir modificação
                              Então excluir modificação
                        Senão incluir modificação de inclusão
                        Fim
    Senão Se atividade existir apenas "pa"
          Então //→ Trata-se de uma exclusão
                Se atividade não existir em "pp"
                Então Se atividade possuir modificação
                      Então excluir modificação
                      Senão incluir modificação de exclusão
                Senão Se não possuir regra associada
                      Então Início
                            Se atividade possuir modificação
```

```
                    Então excluir modificação
                    incluir modificação de exclusão
                    Fim
               Senão Início
                        aval ← avaliação da regra
                        incluir ← ( ( aval = verdadeiro )
                                  e ( regra = positiva ))
                                  ou
                                  ( ( aval = falso )
                                  e ( regra = negativa ))
                        Se incluir = verdadeiro
                        Então Se atividade possuir mod
                              Então Início
                                    excluir modificação
                                    incluir modificação
                                    Fim
                              Senão incluir modificação
                        Fim
Senão //→ Trata-se de uma manutenção
  Se atividade não existir em "pp"
  Então Se atividade possuir modificação
        Então excluir modificação
  Senão Se não possuir regra associada
        Então Início
              Se atividade possuir modificação
              Então excluir modificação
              Se tipo da atividade = fragmento
              Então reaplicar algoritmo ao fragmento
              Fim
        Senão Início
                    aval ← avaliação da regra associada
                    incluir ← ( ( aval = verdadeiro )
                              e ( regra = positiva  ) )
                              ou
                              ( ( aval = falso     )
                              e ( regra = negativa ) )
                    Se incluir = verdadeiro
                    Então Se atividade possuir modificação
                          Então excluir modificação
```

```
                    Senão incluir modificação de inclusão
                Fim
```

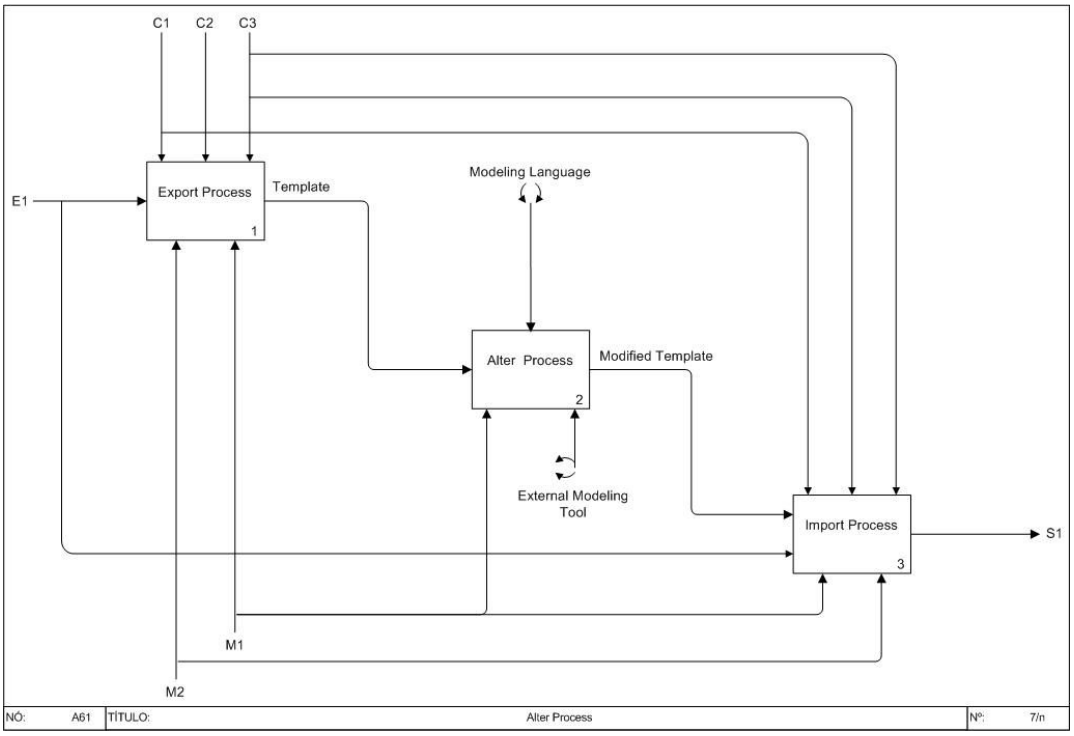


Figura 3.10: Etapas para alteração do processo adaptado

3.5 Estratégia de Adaptação

Nesta seção, é apresentada a estratégia de adaptação utilizada pelo APSEE-Tail, ou seja, o detalhamento da função Adaptar Processo, vista na figura 3.7. É interessante notar que esta função recebe como entrada o processo padrão da organização de software, produzido pela função Definir Processo Padrão, e produz como saída o processo adaptado, adequado às características do projeto de software definido na fase Definir Projeto, bem como um caso de adaptação com as informações acerca da adaptação corrente, o qual realimentará a função em adaptações posteriores. Apesar de servir como entrada para a função, o processo padrão não é alterado durante a adaptação, tendo tão somente as suas atividades utilizadas para a criação do processo adaptado. Também se pode notar que a adaptação é semi-automática, ou seja, realizada em conjunto pelo engenheiro de processos e o APSSE-Tail (seus editores e mecanismos de raciocínio) e guiada pelas regras de adaptação, pelos casos de adaptação gerados anteriormente e pelas características do projeto de software corrente. Trata-se de uma abordagem de adaptação orientada à atividades (defini-se quais atividades do processo padrão e do processo adaptado recuperado serão incluídas ou não no processo adaptado a ser gerado) que combina raciocínio baseado em casos com interpretação de regras. A figura 3.11 apresenta o diagrama IDEF0 que detalha esta função.

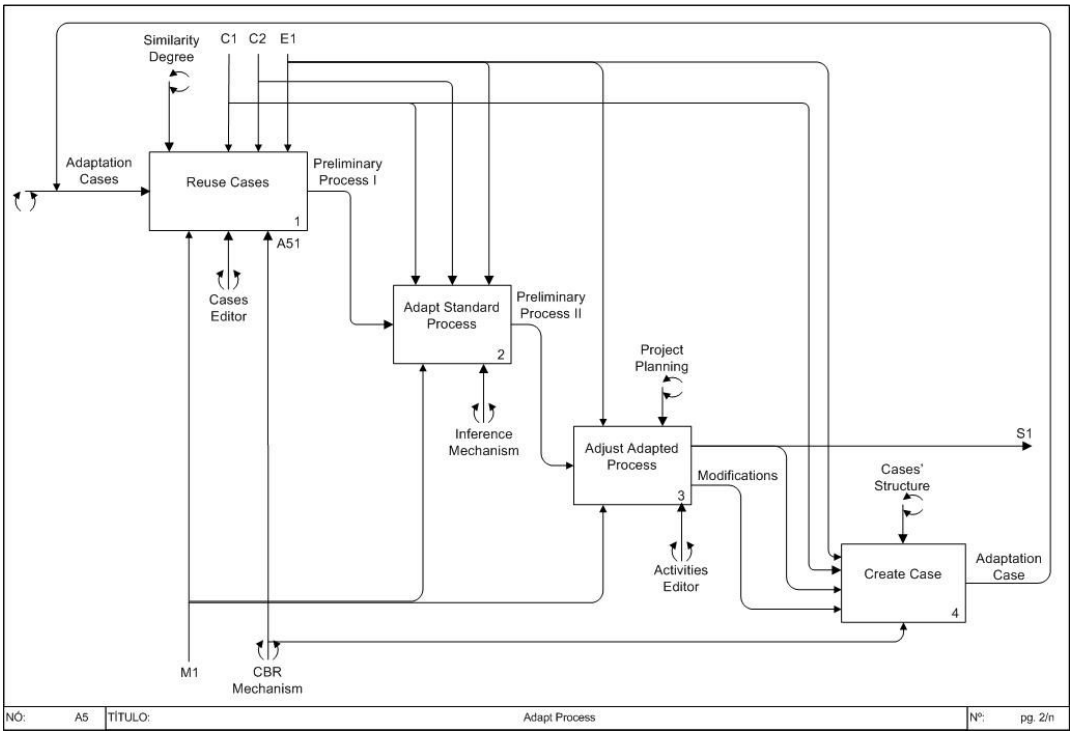


Figura 3.11: Etapas para adaptação de um processo de software

O processo de adaptação inicia com a função Reusar Casos, a qual recebe como entrada o conjunto de casos de adaptação existentes e retorna o processo associado ao projeto de software mais semelhante ao projeto corrente, com as adaptações necessárias. 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 uma atividade do processo padrão tiver sido incluída ou excluída no/do processo adaptado recuperado, em discordância à decisão do modelo. O diagrama IDEF0 da figura 3.12 apresenta esta função em detalhes.

Inicialmente, a função Comparar Casos retorna, dentre todos os casos de adaptação existentes, aqueles cujo grau de similaridade em relação ao caso de adaptação corrente seja superior ou igual ao especificado. É interessante que este grau de similaridade não seja nem muito baixo (o que retornaria casos cujo processo adaptado não seria muito adequado ao projeto corrente) e nem muito alto (o que poderia inclusive não retornar um único caso sequer). Para Coelho (2003), dois projetos de software podem ser classificados quanto ao grau de semelhança entre ambos, como:

- Totalmente semelhantes, quando o grau de semelhança for de 100%;
- Muito semelhantes, quando o grau de semelhança estiver entre 50 e 99%;
- Pouco semelhantes, quando o grau de semelhança estiver entre 0 e 49%.

Com base nesta classificação, pode-se dizer que algo entre 80 e 90% é considerado um bom valor para o grau de similaridade, considerando que exista um número razoável de casos no repositório de conhecimentos. O método de comparação é aplicado sobre as características dos projetos de software e baseia-se no algoritmo da vizinhança (WATSON E MARIR apud ABEL, 1996). Dados dois projetos de software, o corrente e o recuperado do repositório de conhecimentos, calcula-se a similaridade de ambos em

relação a cada uma de suas características e então é obtida a média ponderada destes valores. O algoritmo, descrito no pseudocódigo abaixo, mostra o método de comparação de projetos utilizado pelo APSEE-Tail.

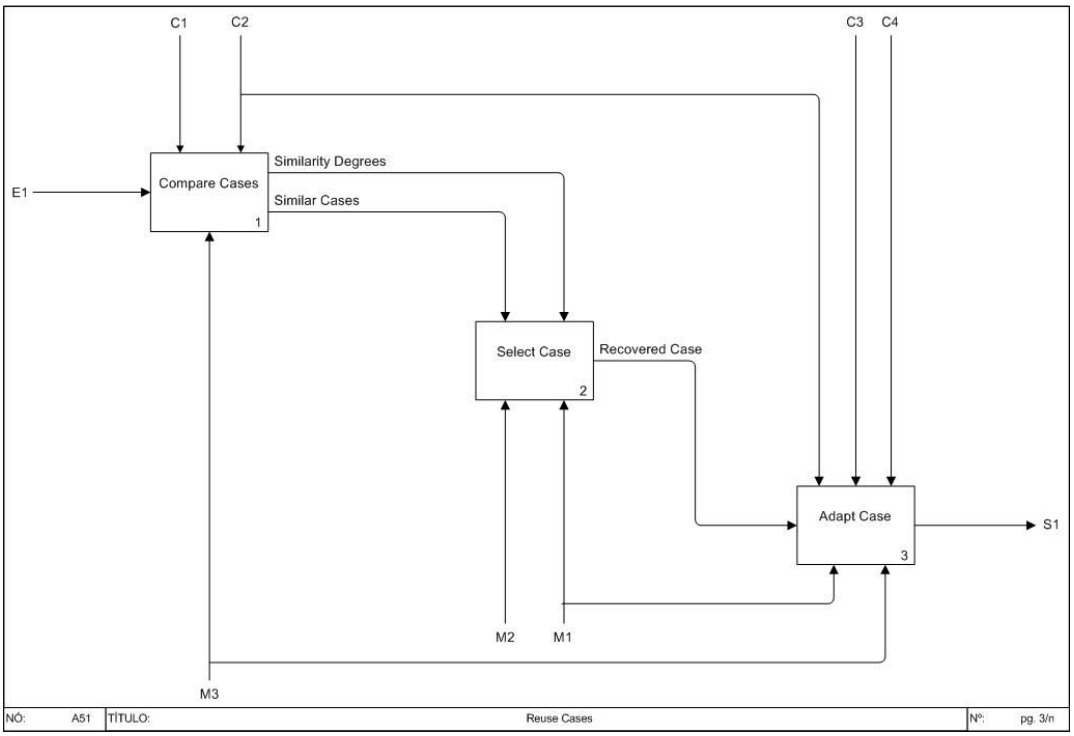


Figura 3.12: Etapas para a reutilização de casos de adaptação

```
Sejam:

pp = projeto passado
pc = projeto corrente
cc = conjunto formado pela união das características dos
     projetos passado e corrente
n  = número de elementos de cc
i  = i-ésima característica de um conjunto
w  = importância de uma característica em um projeto
p_i = valor da característica i no projeto p. Se a
     característica for nivelada, então o valor equivale ao
     seu nível
w_i = importância da característica i no projeto corrente.
     Caso a característica só exista no projeto passado,
     seu valor no projeto corrente será 1
f  = função de similaridade de uma característica em dois
     projetos de software
```

Então a função que calcula o grau de similaridade entre dois projetos é dada por:

$$Similaridade = ((\sum_{i=1}^{n} f(pc_i, pp_i) * w_i) / (\sum_{i=1}^{n} w_i))$$

Sendo que:

$f(pc_i, pp_i)$
= Se i existir em apenas um dos projetos
  Então retorne 0
  Senão Se i for uma característica simples
        Então Se i não tiver nivelamento
              Então Se $pc_i = pp_i$
                    Então retorne 1
                    Senão retorne 0
              Senão retorne $1 - (| pc_i - pp_i | / Max(i))$
        Senão retorne $N(pc_i \cap pp_i) / N(pc_i \cup pp_i)$

Como exemplo de aplicação do algoritmo acima, considere o conjunto de tipos de características, extraído e adaptado de (AHN, 2003) e (COELHO, 2003), apresentadas na tabela 3.1, bem como os valores destas características nos projetos hipotéticos A (projeto corrente) e B (projeto passado recuperado do repositório), apresentados na tabela 3.2. O Projeto A trata da construção de uma nova aplicação Web, complexa e com requisitos de performance e tolerância à falhas. Apesar da criticidade do projeto para a organização desenvolvedora, devido ao potencial do cliente para futuras contratações, o mesmo possui sérias restrições de hardware e software para hospedagem da aplicação. O projeto B, por outro lado, trata da reengenharia de uma aplicação desktop para a Web, a qual deve manter interface com sistemas legados e ter sua performance consideravelmente melhorada. A equipe é distribuída.

Tabela 3.1: Tipos de característica considerados na definição dos projetos A e B

Característica	*Tipo*	*Valores Possíveis*
Tipo de Desenvolvimento	Simples sem Nivelamento	Novo Desenvolvimento, Configuração / Modificação de Pacote, Prototipação, Reengenharia, Reuso.
Tipo de Sistema	Simples sem Nivelamento	Aplicação Web, Sistema Embarcado, Sistema Desktop.
Complexidade do Software	Simples com Nivelamento	Baixa, Média, Alta.
Criticidade do Projeto	Simples com Nivelamento	Baixa, Média, Alta.
Tamanho da Equipe	Simples com Nivelamento	1 a 10 pessoas, 11 a 50 pessoas, 51 a 100 pessoas, acima de 100 pessoas.
Distribuição da Equipe	Simples com Nivelamento	Mesma Sala, Mesmo Prédio, Mesma cidade, Cidades Diferentes.

Equipe	Nivelamento	
Riscos Técnicos	Múltiplo	Tecnologia imatura, Interface com sistema imatura, Restrições de hardware e software existentes, Necessidade de repositório / ferramentas CASE, Requisitos de performance, Interface com sistemas legados, Requisitos de tolerância a falhas.
Políticas Organizacionais	Múltiplo	Políticas de Segurança, Gerência de Riscos, Processo de Reengenharia de Negócios, Terceirização, Conformidade com Normas e Padrões de Qualidade.

Tabela 3.2: Projetos hipotéticos A e B

Característica	*Projeto A*	*Projeto B*
Tipo de Desenvolvimento	Novo Desenvolvimento	Reengenharia
Tipo de Sistema	Aplicação Web	Aplicação Web
Complexidade do Software	Alta	Alta
Criticidade do Projeto	Alta	Média
Tamanho da Equipe	11 a 50 pessoas	1 a 10 pessoas
Distribuição da Equipe	Mesmo Prédio	Cidades Diferentes.
Riscos Técnicos	Interface com sistema imatura, Restrições de hardware e software existentes, Necessidade de repositório / ferramentas CASE, Requisitos de performance, Requisitos de tolerância a falhas	Necessidade de repositório / ferramentas CASE, Requisitos de performance, Interface com sistemas legados
Políticas Organizacionais		Políticas de Segurança, Gerência de Riscos, Conformidade com Normas e Padrões de Qualidade

Levando-se em consideração as características dos projetos A e B, bem como o algoritmo para cálculo de similaridade entre dois projetos, a tabela 3.3 mostra a similaridade de ambos os projetos em relação a cada tipo de característica. A coluna Importância se refere ao peso de uma característica no projeto A. Se esta característica existir apenas no projeto B, então seu peso no projeto A será 1, o que significa dizer que ela contará como um ponto diferenciador entre os projetos. Aplicando-se a média ponderada nos valores obtidos na tabela abaixo, obtém-se que os projetos A e B são 56% semelhantes, ou seja, seria viável reaproveitar o processo de software associado ao projeto B no projeto A, mas não sem alguma adaptação.

Tabela 3.3: Similaridade entre os projetos A e B

Característica	*Similaridade entre A e B*	*Importância em A*
Tipo de Desenvolvimento	0	4
Tipo de Sistema	1	4
Complexidade do Software	1	3

Criticidade do Projeto	0,67	3
Tamanho da Equipe	0,75	2
Distribuição da Equipe	0,5	2
Riscos Técnicos	0,33	4
Políticas Organizacionais	0	1

Os casos de adaptação recuperados pela função Comparar Casos, bem como seus respectivos graus de similaridade em relação ao projeto de software corrente, servem de entrada para a função Selecionar Caso. Nesta função, o engenheiro de processos, baseado nos graus de similaridade, nos comentários e na avaliação dos casos de adaptação, escolhe aquele que lhe convier. É aconselhável que o mesmo priorize os graus de similaridade como critério de escolha e, em caso de empate, analise primeiro a avaliação do caso, e só então os comentários deixados nos mesmos. A saída desta função pode ser o caso de adaptação mais semelhante ao projeto corrente ou vazia, no caso de sua entrada ter sido também vazia.

O caso de adaptação selecionado, caso exista, servirá de entrada para a função Adaptar Caso. Esta função, baseada nas características do projeto de software corrente, nas regras de adaptação e no processo padrão, realizará as adaptações necessárias no processo de software associado ao caso de adaptação recuperado, retornando tal processo na saída. É realizada em conjunto pelo engenheiro de processos e pelo mecanismo de CBR. Basicamente, para cada atividade do processo recuperado, é verificado: se possui uma modificação associada (neste caso, questiona-se o engenheiro de processos pela sua permanência ou não); se pertence ou não ao processo padrão (caso pertença, executa-se a regra associada com as características do projeto corrente para decidir pela sua permanência). O algoritmo, escrito em pseudocódigo e apresentado abaixo, ilustra o processo de adaptação do caso recuperado.

```
Sejam:

pr = processo recuperado
pp = processo padrão

Então:

1.Para cada atividade do modelo de processo de "pr" faça
  Se a atividade não existir em "pp"
  Então Se a atividade não possuir modificação
        Então manter a atividade em "pr"
        Senão questionar o engenheiro de processos pela
              manutenção da atividade
              {Regras 1.a, 1.b e 1.c}
  Senão Se a atividade não possuir uma regra associada
        Então Início
```

```
                Se a atividade não possuir modificação
                Então manter a atividade em "pr"
                Senão questionar o engenheiro de processos
                      pela manutenção da atividade
                      {Regras 1.a, 1.b e 1.c}
                Se a atividade tiver sido mantida e for um
                   fragmento
                Então reaplicar o algoritmo, onde:
                      "pr" <- fragmento da atividade de "pr"
                      "pp" <- fragmento da atividade de "pp"
                Fim
          Senão Início
                avaliação ← avaliação da regra associada
                Se tipo da regra = positivo
                Então Se avaliação = verdadeiro
                      Então manter a atividade em "pr"
                      Senão Se a atividade possuir
                               modificação
                            Então questionar o engenheiro de
                                  processos pela sua
                                  manutenção
                                  {Regras 1.a, 1.b e 1.c}
                            Senão excluir a atividade de "pr"
                                  {Regras 1.a, 1.b e 1.c}
                Senão Se avaliação = verdadeiro
                      Então Se a atividade possuir
                               modificação
                            Então questionar o engenheiro de
                                  processos pela sua
                                  manutenção
                                  {Regras 1.a, 1.b e 1.c}
                            Senão excluir a atividade de "pr"
                                  {Regras 1.a, 1.b e 1.c}
                      Senão manter a atividade em "pr"
                Fim
Obs: Após a exclusão de qualquer atividade do modelo de
     processos, aplicar regras 2.a, 2.b, 3.a, 3.b, 4.c,
     4.d, 5.a, 5.b e 6, para garantir a consistência das
     conexões.
```

O resultado da função Reusar Casos é o processo de software associado ao projeto mais semelhante ao projeto corrente, com as devidas adaptações. A este processo de software convencionou-se chamar de processo preliminar I. Ele será entrada para a função seguinte, denominada Adaptar Processo Padrão. Basicamente, nesta função, o interpretador de regras percorre um modelo do processo padrão em busca de todas aquelas atividades que ainda não existam no processo preliminar I. Para cada uma destas atividades, duas situações são possíveis: a atividade não possui regra e, portanto, será incluída diretamente no processo preliminar I; a atividade possui uma regra associada e, portanto, esta regra deve ser avaliada em relação às características do projeto corrente para determinar a sua inclusão ao não no processo preliminar I. O processo de software resultante desta função é chamado de processo preliminar II, sendo que regras terão de ser aplicadas para checar a coerência de suas conexões. O algoritmo aplicado por esta função é mostrado em pseudocódigo, a seguir.

```
Considerando-se que:

pp = processo padrão
pr = processo preliminar
ca = conjunto de atividades com modificações de exclusão

Tem-se:

1. Habilita as políticas de "pp" em "pr" {regra 15.g}.
2. Para cada atividade de "pp" faça
   Se atividade não estiver em "pr"
   Então Se atividade não possuir regra
         Então Início
               Se atividade pertencer a "ca"
               Então questionar sua inclusão
               Senão incluí-la em "pr"
                     { 1° regras de 14.a a 14.f
                       2° regras de 15.a a 15.h }
               Se atividade for um fragmento
               Então reaplicar algoritmo, onde:
                     "pp" <- modelo do fragmento
                     "pr" <- modelo vazio
               Fim
         Senão Início
               avaliação <- avaliação da regra associada
               Se tipo da regra = positivo
               Então Se avaliação = verdadeiro
```

Então Se atividade pertencer a “ca”
Então questionar sua inclusão
{ 1° regras de 14.a a 14.f
2° regras de 15.a a 15.h }
Senão incluí-la em “pr”
{ 1° regras de 14.a a 14.f
2° regras de 15.a a 15.h }
Senão não incluí-la em “pr”
Senão Se avaliação = falso
Então Se atividade pertencer a “ca”
Então questionar sua inclusão
{ 1° regras de 14.a a 14.f
2° regras de 15.a a 15.h }
Senão incluí-la em “pr”
{ 1° regras de 14.a a 14.f
2° regras de 15.a a 15.h }
Senão não incluí-la em “pr”
Fim

3. Aplicar as regras 2.a, 2.b, 3.a, 3.b, 4.a, 4.b, 4.c, 4.d, 5.a, 5.b e 6 nas conexões de “pp” levando-se em consideração as atividades de ”pr”, ou seja, as origens e destinos das conexões de “pp” devem ser checados em “pr”. Como resultado desta etapa, é gerado um conjunto de conexões, contendo apenas aquelas que serão consistentes em “pr”, aqui denominado “cons”.
4. Fazer o merge das conexões do conjunto “cons” com aquelas de “pr”, de modo a evitar a inclusão de ciclos e a duplicação de conexões em “pr”. As regras para inclusão de conexões e redefinição de suas entradas e saídas variam com o tipo da conexão e são listadas abaixo:
    - ⇨ Conexões Simples
        - o Inclusão: regra 7
    - ⇨ Conexões de Feedback
        - o Inclusão: regra 8
    - ⇨ Conexões Múltiplas
        - o Inclusão: regras 9.a, 9.b, 9.c, 9.d, 9.e e 9.f
        - o Redefinição de IO: regras 10.a, 10.b, 10.c, 10.d, 10.e, 10.f, 10.g, 10.h, 10.i, 10.j, 10.k, 10.l, 10.m, 10.n, 10.o, 10.p, 10.q e 10.r
    - ⇨ Conexões de Artefato
        - o Inclusão: regra 11

```
        o Redefinição de IO: regras 12.a a 12.e
Obs: para a redefinição das origens e destinos de uma conexão
     Múltipla ou de artefato, incluída em "pr", deve-se incluir na
     mesma cada origem ou destino da conexão equivalente em "cons"
     e que exista em "pr", de acordo com as regras 10.a a 10.r e
     12.a a 12.e.
```

Por fim, o processo preliminar II, o projeto caracterizado e o conjunto de modificações gerado até então servirão de entrada para a última função do processo de adaptação, a função Ajustar Processo Adaptado. Nela, o engenheiro de processos poderá realizar ajustes manuais no processo preliminar II com o auxílio do Editor de Processos Adaptados (cujas únicas funcionalidades são a visualização, inclusão e exclusão de atividades). Atividades do processo padrão que não foram incluídas poderão ser e aquelas que foram poderão ser retiradas, em ambos os casos mediante uma justificativa. Para realizar estes ajustes o engenheiro irá se basear no planejamento do projeto e na sua experiência. Somente então é que o mecanismo de CBR juntará o processo adaptado, suas modificações e o projeto de software caracterizado para formar um caso de adaptação.

3.6 Considerações

Este capítulo apresentou a visão geral do APSEE-Tail, descrevendo suas principais características, escopo de atuação, componentes, funções e a estratégia de adaptação utilizada.

Conforme visto, o APSEE-Tail objetiva auxiliar o engenheiro de processos na adaptação do processo padrão de uma organização de software para os projetos serem conduzidos na mesma, baseando-se na regras de adaptação do processo padrão, nas características do projeto de software em questão e nas informações a respeito de adaptações realizadas anteriormente. O APSEE-Tail é parametrizável, de caráter genérico quanto à sua aplicação e utiliza uma abordagem de adaptação orientada a atividades, baseada na manutenção e no raciocínio do/sobre o conhecimento necessário, que combina raciocínio baseado em casos com interpretação de regras.

Dentre seus principais componentes, pode-se destacar: processo padrão; regras de adaptação; tipos de característica; projetos de software; processos adaptados; e casos de adaptação.

Foi visto também que as funções do APSEE-Tail podem ser agrupadas em duas categorias: funções de configuração do modelo (Definir Processo Padrão, Definir Tipos de Característica e Definir Regras); e funções de apoio à adaptação (Definir Projeto, Adaptar Processo e Atualizar Caso).

A definição do APSEE-Tail foi fortemente embasada nas idéias presentes nos trabalhos de Ahn (2003) e Coelho (2003), apresentando ainda pontos que podem ser melhorados. Neste capítulo, destacou-se a carência no suporte a fase de avaliação do processo adaptado, cujo *feedback* é de grande valia para a realimentação do modelo. O próximo capítulo apresenta os formalismos utilizados na especificação APSEE-Tail e as justificativas para tal.

4 ESPECIFICAÇÃO FORMAL DO APSEE-TAIL

O desenvolvimento de um sistema de software complexo demanda a escolha de métodos de Engenharia de Software que sejam gerenciáveis, permitindo a especificação precisa e viável dos componentes do sistema. Atualmente, a literatura descreve um amplo leque de métodos que apóiam o desenvolvimento sistemático de software em diferentes níveis de abstração e usando notações específicas (REIS, 2002).

Devido à diversidade de aspectos envolvidos na definição do APSEE-Tail, 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, para ser utilizada como base semântica para auxiliar no entendimento e na evolução futura da proposta. Além disso, segundo Wang e King apud (REIS, 2002), 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 do APSEE-Tail, utilizou-se a combinação dos métodos formais Prosoft-Algébrico e Gramática de Grafos (abordagem algébrica).

Tendo em vista a especificação formal do APSEE-Tail, a qual é apresentada em detalhes nos apêndices deste trabalho, este capítulo está organizado como segue.

- Na seção 4.1, fala-se rapidamente do uso da especificação formal no desenvolvimento de software;
- Na seção 4.2, é descrito o processo de formalização do APSEE-Tail, ou seja, quais os passos que o autor seguiu para formalizá-lo;
- Na seção 4.3, é apresentada a visão geral do Prosoft-Algébrico, justificando-se a sua utilização. A hierarquia de classes do modelo é apresentada logo em seguida;
- Na seção 4.4, é apresentada a visão geral de Gramática de Grafos, justificando-se a sua utilização. O grafo tipo e a descrição informal das regras para coerência sintática de processos de software são apresentados na seqüência;
- Por fim, na seção 4.5, são feitas as considerações sobre o capítulo.

4.1 Especificação Formal no Desenvolvimento de Software

Uma questão tradicional na Engenharia de Software é se o sistema proposto é realmente uma solução para o problema considerado. Uma maneira de responder esta pergunta é através do uso de métodos formais, cuja idéia é focar-se na modelagem formal do sistema em construção, abstraindo seus comportamentos menos importantes. Esse modelo formal pode ser empregado para analisar o comportamento do sistema, a

fim de garantir que o modelo construído possui o comportamento correto e as propriedades desejadas. Entre os métodos formais mais conhecidos, citam-se: VDM, Z, OBJ, Larch, CASL, Álgebra de Processos, Gramática de Grafos, Redes de Petri e CCS.

Segundo Cohen apud (RANGEL, 2003), os métodos formais podem ser totalmente integrados no ciclo de vida clássico do desenvolvimento de software. Na fase de análise, eles podem ser usados juntamente com outras técnicas semiformais. Na fase de projeto, a especificação pode sofrer uma série de refinamentos e iterações até chegar à implementação. O uso de raciocínio formal e refinamento ajuda em assegurar a corretude de cada versão do software em desenvolvimento.

Em alguma etapa do desenvolvimento a descrição informal dos requisitos precisa ser formalizada. É essencial que a formalização capture os requisitos contidos na especificação informal de modo mais correto e completo possível. Infelizmente, este processo é inerentemente tendencioso a erros e não pode ser completamente automatizado e provado (COHEN apud RANGEL, 2003). Outra dificuldade é o fato dos requisitos do usuário serem freqüentemente mal definidos e usualmente evoluírem através do ciclo de desenvolvimento. Portanto, não existe uma especificação "completa".

Na maioria dos projetos em que especificações formais são usadas, especificações informais e formais são combinadas. Alguns componentes e passos são formalizados enquanto outros não. A decisão de usar especificações formais depende principalmente do grau de criticidade do componente de software, em termos de conseqüências de uma falha (vidas humanas, custo, etc), da complexidade de seus requisitos e seu desenvolvimento.

As linguagens de especificação concentram-se na funcionalidade e no projeto das estruturas, deixando algoritmos e detalhes de implementação para uma posterior fase de codificação. Linguagens de especificação declarativas, assim como o OBJ, permitem que o desenvolvimento de uma especificação siga uma abordagem incremental, onde o primeiro estágio de especificação pode consistir simplesmente em nomear as entidades que irão compor o sistema, deixando para estágios posteriores o refinamento das funcionalidades de tais entidades.

Os métodos de especificação algébrica fornecem um conjunto de técnicas para abstração de dados e para especificação, validação e análise das estruturas de dados construídas (LEEUWEN apud RANGEL, 2003). Estes métodos podem ser considerados, neste contexto, uma linha de especificação formal mais evoluída, tanto pela teoria subjacente quanto pelas suas eficientes implementações existentes.

4.2 O Processo de Formalização Seguido

Conforme mencionado na seção introdutória deste capítulo, optou-se por especificar formalmente o APSEE-Tail em virtude de uma série de razões: complexidade e diversidade de aspectos envolvidos na definição do modelo; possibilidade de verificações do modelo quanto à sua corretude; geração de uma especificação precisa e em alto nível de abstração, para ser utilizada como base semântica para auxiliar no entendimento e na evolução futura da proposta.

Vale ainda ressaltar que, como era objetivo do trabalho a prototipação do APSEE-Tail, a especificação tornou-se aliada no projeto do sistema, permitindo uma descrição

abstrata, porém precisa do seu comportamento. Embora o uso de métodos formais possibilite a realização de provas e verificações matemáticas acerca de propriedades do modelo, tal característica não constitui objetivo do trabalho aqui apresentado sendo, entretanto, um recurso importante que pode ser aproveitado em trabalhos futuros.

Foram utilizados na especificação do APSEE-Tail, de maneira complementar, os métodos formais Prosoft-Algébrico e Gramática de Grafos. Os componentes do modelo, apresentados no apêndice A, e a semântica do mecanismo de adaptação, apresentada no apêndice B, foram especificados usando-se o Prosoft-Algébrico, enquanto que as regras para garantia da coerência sintática dos processos adaptados gerados, apresentadas no apêndice C, foram especificadas usando-se Gramática de Grafos. Para expressar as macro-funções do modelo de adaptação, ou seja, suas funcionalidades, foi utilizada a notação IDEF0. Nesta seção, a mesma notação IDEF0 é utilizada para expor o processo de formalização do APSEE-Tail.

Conforme pode ser visto na figura 4.1, a utilização dos formalismos foi realizada em paralelo. Na prática, somente com as atividades 1 e 3 bem definidas é que foi possível elicitar as regras que seriam necessárias à coerência sintática do processo adaptado, pois foi neste ponto que se definiu em que etapas estas regras deveriam se aplicadas.

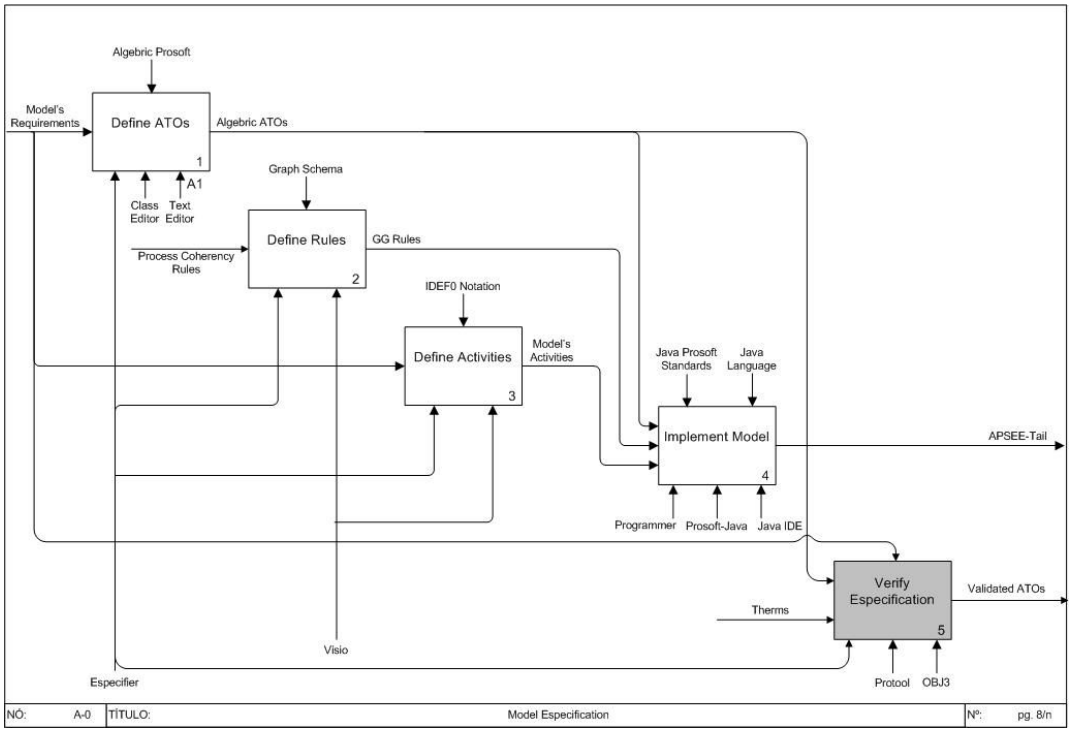


Figura 4.1: Processo de formalização do APSEE-Tail

Na atividade 1, Definir ATOs, o especificador, baseado nos requisitos iniciais do APSEE-Tail, pôde, através da utilização do editor de classes do Prosoft-Java e de um editor de textos comum, especificar os ATOs algébricos que representariam os componentes do modelo. Esta atividade é expandida no diagrama da figura 4.2.

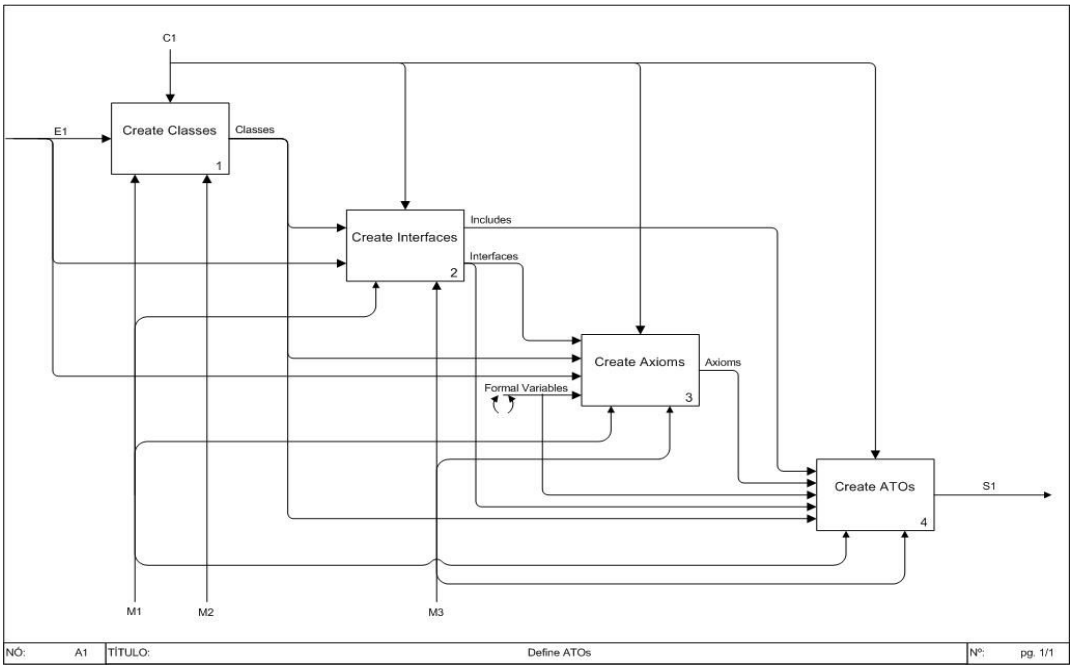


Figura 4.2: Definição dos ATOs do APSEE-Tail

Seguindo a estratégia *data-driven* do Prosoft-Algébrico, foram definidas primeiramente as classes dos ATOs. Esta subatividade foi realizada através do editor de classes do Prosoft-Java, o qual é mostrado na figura 4.3.

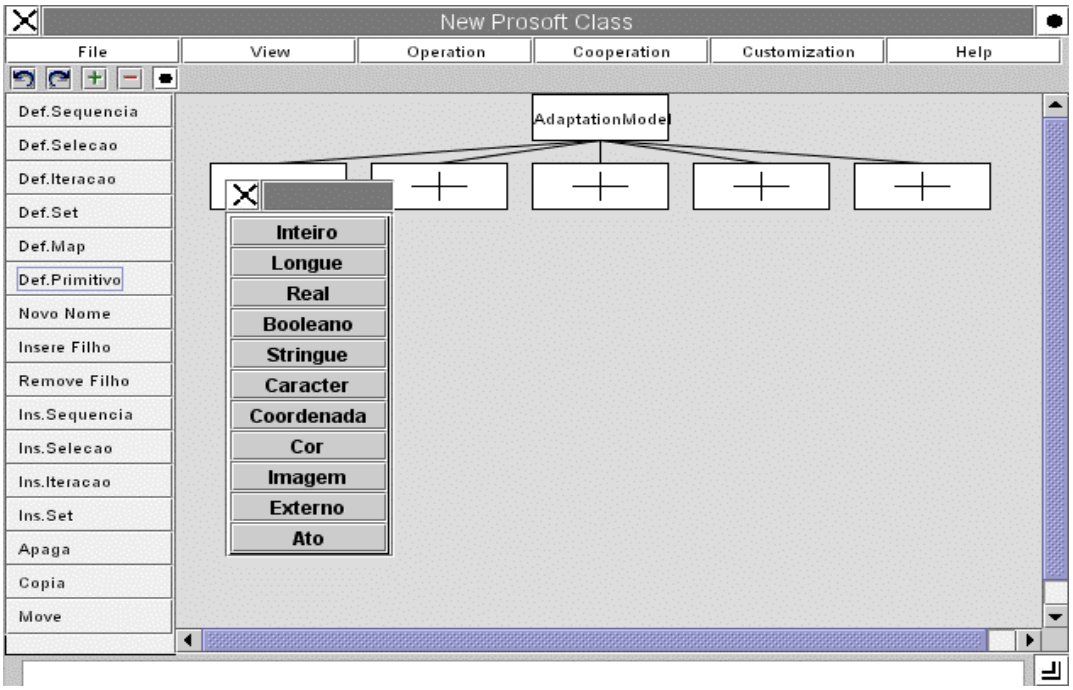


Figura 4.3: Editor de classes do Prosoft-Java

De posse das classes, foram definidas as interfaces das operações que poderiam ser aplicadas em cada uma delas. Desta atividade resultou também a lista de inclusões que cada ATO necessitaria. Embora as atividades 1, 2 e 3 sejam mostradas seqüencialmente,

na prática isso não aconteceu. A criação das interfaces, bem como dos axiomas, permitiu o refinamento das classes, realimentando o processo. Definidas as interfaces das operações, partiu-se para a semântica das mesmas, ou seja, construção dos axiomas. Para que os mesmos fossem genéricos, variáveis formais foram definidas e utilizadas. Por fim, tudo isso (classes, inclusões, interfaces, variáveis formais e axiomas) foi formatado na estrutura utilizada nesta dissertação, apresentada mais adiante, de modo a melhor esclarecer os TADs (Tipos Abstratos de Dados).

Elicitadas as regras informais para a garantia da coerência dos processos adaptados, pôde-se, através de um editor gráfico, construir as regras em GG. Esta atividade foi realizada utilizando-se um editor gráfico qualquer e se baseou no grafo tipo definido por Reis (2002), para seu modelo de *templates*.

A formalização das atividades do APSEE-Tail foi realizada utilizando-se a notação IDEF0, o que permitiu uma definição precisa das entradas, saídas, mecanismos, e controles das mesmas. À medida que o detalhamento das atividades era elucidado, sub-diagramas eram criados explodindo as macro-atividades. Conforme já mencionado no capitulo 3, as atividades do APSEE-Tail classificam-se em: de configuração e de apoio à adaptação.

Embora não seja o objetivo deste capítulo, é bom mencionar que, os ATOs algébricos, as regras em GG e as atividades IDEF0 foram utilizadas pelo programadores para prototipar o APSEE-Tail no ambiente Prosoft-Java. Este ambiente possui um gerador de código Java automático a partir das classes definidas no editor de classes. São gerados métodos básicos, cabendo ao programador definir os demais métodos e as interfaces dos ATOs em uma ferramenta Java qualquer, incluir os fontes no diretório do ambiente e recompilá-lo como um todo para integrar o novo ATO.

Por fim, cabem alguns comentários em relação à última atividade, Validação da Especificação. Rangel (2003) especificou e implementou no ambiente Prosoft-Java, uma ferramenta para prototipação de ATOs algébricos, o Protool. Ela permite a acoplagem de classes relacionadas, definição dos axiomas e geração do código OBJ. Usando o sistema de reescrita das linguagens OBJ, é possível realizar testes nos ATOs. Também existem ferramentas que validam regras em GG. Não é objetivo deste trabalho provar matematicamente a corretude das especificações, logo o mesmo é deixado como trabalho futuro.

4.3 Formalização dos Componentes

A especificação formal dos componentes do APSEE-Tail foi realizada usando-se o Prosoft-Algébrico, o que proporcionou uma derivação direta para implementação no ambiente Prosoft-Java, visto que há uma correspondência semântica entre os elementos usados na especificação e a implementação dos componentes de software descritos nesse paradigma. Além disso, o grupo de pesquisa no qual o trabalho foi desenvolvido possui uma vasta experiência no uso deste formalismo. Conforme já mencionado, o Prosoft-Algébrico foi utilizado para especificar os componentes do modelo (estruturas de dados e operações que podem ser aplicadas sobre tais estruturas), como também para especificar a semântica do mecanismo de adaptação.

4.3.1 Prosoft-Algébrico

O formalismo denominado Prosoft-Algébrico, descrito em (NUNES, 1992; 1994), permite a descrição de TADs através de um paradigma algébrico baseado em objetos. Segundo Nunes (1994), esse paradigma "adota uma abordagem *data-driven* para o desenvolvimento de software", isto é, estimula o desenvolvimento de software inicialmente através da composição dos tipos de dados necessários. Como método de especificação formal, apóia a reutilização e a modularização das especificações de TADs. A modularização torna possível quebrar o problema em partes menores e, portanto, mais fáceis de solucionar. A reutilização simplifica a especificação do TAD, tornando-a mais simples e mais rápida de ser construída. O Prosoft-Algébrico é, portanto, uma técnica orientada a propriedades que define um objeto matemático baseado nas relações entre as operações desse objeto.

Cada tipo de dados é instanciado a partir de um ATO. Cada ATO especifica somente um TAD. Qualquer termo desse tipo é chamado de objeto e, segundo Nunes (1994), não é similar com o conceito de objeto das linguagens de programação orientadas a objetos[28]. Segundo Daudt apud (REIS, 2002), "as instâncias de uma classe (objetos) são entidades passivas, que não têm capacidade de responder a estímulos (mensagens)". Portanto, os termos Prosoft armazenam dados, mas não podem manipulá-los: toda manipulação de dados é descrita através de funções definidas no escopo de um ATO.

A construção de um ou mais ATOs representa uma solução do problema (software). Assim, o lado esquerdo da figura 4.4 apresenta o esquema gráfico de um sistema arbitrário de software desenvolvido sob o paradigma do Prosoft, sendo composto por um número n de ATOs que comunicam-se através da ICS (Interface de Comunicação do Sistema), que será discutida mais adiante.

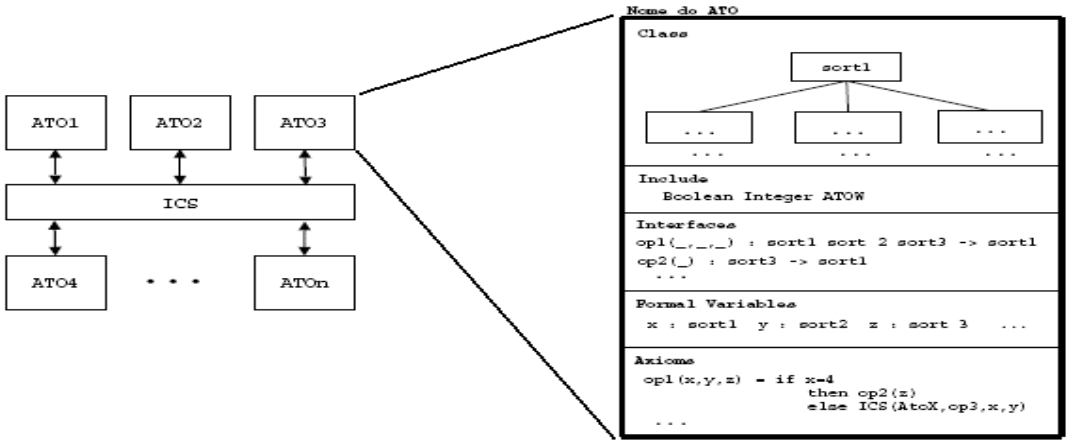


Figura 4.4: Composição de ATOs algébricos (RANGEL, 2003)

Ainda de acordo com a figura 4.4, 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

[28] Vale ressaltar que o conceito de classe empregado no Prosoft-Algébrico difere do conceito de classes em linguagens de programação orientadas a objetos. Uma classe Prosoft é uma estrutura de dados que corresponde ao que normalmente é chamado de atributos nas linguagens orientadas a objetos (REIS, 2002).

a funcionalidade (assinatura) das novas operações algébricas sobre o tipo de dado, as variáveis formais e a semântica das novas operações (axiomas).

Comparando-se a especificação algébrica convencional, como o método descrito por Watt (1991), e o Prosoft-Algébrico, pode-se dizer que a assinatura da especificação corresponde à instanciação e à interface do ATO. Os axiomas, por sua vez, correspondem às operações do ATO, aonde segundo Moraes (1997), "a funcionalidade de cada operação é definida na forma de equações". A vantagem fundamental do Prosoft-Algébrico é que, em função do mecanismo de inclusão automática de tipos compostos e primitivos em um ATO Prosoft, não é necessário explicitar a "importação" das especificações dos tipos externos como é feito tradicionalmente nos métodos algébricos. Portanto, a comunicação entre ATOs é feita explicitamente através da ICS, em um estilo muito similar à troca de mensagens do paradigma de objetos, aonde os componentes são encapsulados e descrevem interfaces bem definidas.

4.3.1.1 Descrição dos elementos de um ATO

Conforme visto na seção anterior, um ATO algébrico possui cinco componentes: classe, inclusões, interfaces, variáveis formais e axiomas. Cada um deles é descrito em detalhes a seguir.

4.3.1.1.1 Classe

No Prosoft-algébrico existem dois grupos de tipos de dado, primitivos (Integer, Real, 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 conjuntos de booleanos, etc.

Para facilitar a utilização do método Prosoft-algébrico, foi definida uma poderosa representação gráfica, inspirada nos diagramas de Jackson (1983), para definição das 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 mostrados na figura 4.5 através de exemplos. Para cada tipo composto e primitivo do Prosoft, existem operações[29] geradoras, modificadoras e observadoras.

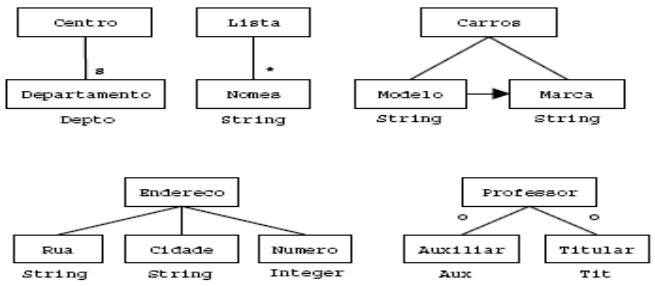


Figura 4.5: Tipos compostos do Prosoft-Algébrico (RANGEL, 2003)

[29] No anexo 1, podem ser encontradas as operações sobre cada tipo composto do Prosoft-Algébrico.

4.3.1.1.2 Cláusula *include*

Cada ATO algébrico manipula somente os tipos de dados que pertencem a sua classe. Em alguns casos é necessário também dispor de outros tipos de dados para definir novas operações neste ATO. Estes tipos de dados, que não pertencem à classe, são importados no ATO através da cláusula *Include* e podem ser primitivos ou também outros ATOs.

4.3.1.1.3 Operações algébricas

A instanciação de uma classe importa automaticamente todas as operações pré-definidas dos tipos compostos e primitivos que compõem a classe. Por exemplo, se a classe é uma lista de strings, todas as operações de criação e manipulação de listas e de strings são automaticamente incluídas na definição do ATO.

Para a definição de novas operações no ATO, utiliza-se a interface para declarar a assinatura[30] de cada operação. A assinatura é composta por uma string que representa o nome da operação, seguida de seu respectivo *rank*[31], como ilustrado no exemplo abaixo:

```
include-client : Clients Client -> Clients
```

Para cada nova operação definida num ATO, deve-se criar pelo menos um axioma para lhe dar semântica. Um axioma geralmente usa variáveis formais para generalizar sua definição. Abaixo são definidas as variáveis client e clients, com seus respectivos sorts.

```
client : Client
clients : Clients
```

A semântica da operação include-client é dada pelo axioma:

```
include-client(clients,client)=
if not (exist-client(clients,ICS(ATOClient,get-clientcode,client)))
then add(client,clients)
else clients
```

Nos métodos algébricos convencionais, variáveis são utilizadas nos axiomas. O Prosoft-Algébrico conta com o artifício "_", que facilita a construção de axiomas e conseqüentemente aumenta a legibilidade dos mesmos. Utiliza-se o símbolo "_" no lugar de variáveis que representam termos do lado esquerdo de um axioma que não são modificados no lado direito. O exemplo abaixo ilustra o uso deste recurso.

```
addterm(newterm,reg-Term(_,terms))
= reg-Term(_,cons(newterm,terms))
```

Por fim, cabe destacar algumas convenções que o autor adotou quando da criação dos axiomas, de modo a torná-los mais fáceis de ler. Métodos locais, quando utilizados dentro de um axioma, são sublinhados. Métodos auxiliares (aqueles que não produzem o resultado esperado quando não invocados como parte de um método local) aparecem em itálico no interior dos axiomas que os utilizam. E em alguns casos, é necessário

---

[30] No Prosoft, as operações são de primeira ordem.

[31] O rank de uma operação é composto pela aridade da operação seguido do sort valor (resultado) da mesma.

representar que uma variável formal, que não foi passada como parâmetro para o axioma, deve ser informada pelo usuário final. Nestes casos, convencionou-se colocar um ponto de interrogação após o nome da variável. Um exemplo do uso destas convenções é mostrado abaixo, onde se percebe que uma boa endentação também contribui para a legibilidade do axioma.

```
includeConditions(conds,exp,log-op)
= if ICS(ATOExpression,isEmpty,exp)
  then conds
  else if exist(conds,ICS(ATOExpression,getCharTypeId,exp))
       then conds
       else inc(conds,exp,log-op)
```

4.3.1.1.4 ICS

A Interface de Comunicação do Sistema tem função de integrar os diversos ATOs. Qualquer termo só pode ser criado ou modificado pelo ATO que contém sua classe. Assim, quando um ATO necessita processar termos que não pertencem à sua classe, ele deve requisitar operações, via ICS, dos ATOs que definem estes termos. A ICS é também uma operação algébrica, e sua sintaxe é:

ICS (<ATO>,<operação>,<seletor>,<argumentos*>)[32]

A pesquisa da operação é feita de cima para baixo dentro do ATO, se o nome do ATO coincidir e o nome da operação for uma operação pertencente ao ATO, os argumentos serão ligados aos parâmetros formais e então será executada a operação. Como uma operação pode possuir várias definições, um dos argumentos do tipo do *sort* definido pelo ATO - denominado seletor - é usado para encontrar a operação.

De acordo com Nunes, o conceito de modularização através da ICS evidencia que as operações algébricas do Prosoft são monádicas, ou seja, cada operação trata somente o tipo de dado definido na classe do seu ATO. Considere a criação de uma nova operação chamada `op`, cuja assinatura é

```
op : S1 S2 S3 ... Sn -> S
```

Onde Si e S são sorts. Esta operação op é interpretada, como no cálculo lambda, assim:

```
op : S1 -> (S2 S3 ... Sn -> S)
```

A operação op, pertencente ao ATO que define o sort S1, tem como domínio o sort S1 e como imagem operações de S2 S3 ... Sn -> S. Dependendo do valor de S1, será escolhido uma das operações de S2 S3 ... Sn -> S. Os valores de S1 podem ser interpretados como estados. Assim, o estado de S1 vai determinar qual operação será escolhida. Esse processo é recorrente. No final da computação, tem-se, como resultado, a aplicação de uma operação, criando ou alterando o estado de S.

---

[32] Onde, argumentos* representa os argumentos da operação ordenados em uma lista.

4.3.1.2 Estratégia para construção de ATOs algébricos

Na fase de análise do desenvolvimento de software, na maioria das vezes, o problema a ser solucionado é dividido em partes menores para facilitar sua resolução. O lado esquerdo da figura 4.6 ilustra os requisitos informais do software divididos em *n* problemas. Estes problemas menores podem ser analisados pela equipe de desenvolvimento que criará documentos de requisitos (informais e/ou semiformais) descrevendo o software. O emprego de uma notação formal na construção de software requer que, em algum instante, os documentos de requisitos sejam formalizados. A especificação formal permite que ambigüidades e inconsistências nos requisitos sejam reveladas, forçando o retorno à fase de análise dos requisitos.

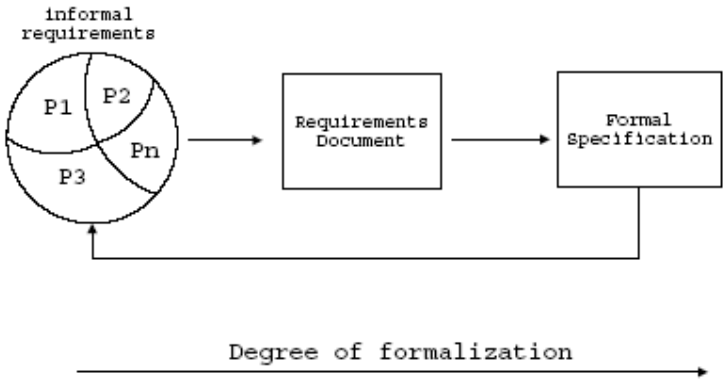


Figura 4.6: Etapas de formalização de requisitos (RANGEL, 2003)

No Prosoft, o desenvolvedor é incentivado a modularizar a formalização, ou seja, a especificação do software pode conter diversos ATOs, cada qual solucionando um problema específico. Tomando como exemplo o problema da figura 4.6, a especificação formal na notação Prosoft-algébrico conteria no mínimo *n* ATOs.

Para criar um ATO, o processo segue a estratégia *data-driven*, onde primeiramente são especificados os dados para depois serem criadas as operações sobre estes dados. Portanto, o desenvolvimento de um ATO algébrico demanda que o especificador crie basicamente:

1. A classe do ATO (TAD do ATO);
2. As operações que atuam sobre a classe.

Entre as etapas 1 e 2, existem outros passos intermediários para definição de importações (se necessário) do ATO e das variáveis formais que serão utilizadas nos axiomas. É importante ressaltar que o passo 2 consiste em definir a assinatura de cada operação criada para depois fornecer sua semântica através de axiomas.

A especificação algébrica convencional (WATT, 1991) é muito livre na definição semântica das operações. Se uma operação envolve sorts de especificações importadas, a definição da semântica desta operação pode conter muitos passos de computação, alguns sobre termos da especificação local e outros das especificações importadas. Este modo de especificar software tende a distrair o especificador do problema a ser solucionado, uma vez que é possível efetuar diretamente as computações sobre todos os sorts da especificação.

Com o recurso de modularização, provido pela operação ICS, é diferente. O especificador limita-se a definir operações de cada ATO somente com os tipos de dados da classe do ATO. Assim, quando uma operação chama uma operação de outro ATO, via ICS, o especificador não precisa se preocupar com os detalhes oriundos do tipo de dado do ATO chamado pela ICS. As chamadas ICS nos axiomas forçam o especificador a focar muito mais sua atenção no problema local que está sendo solucionado no ATO. Considere um exemplo que contêm os seguintes ATOs:

- *ATOData*: define as estruturas de dados do software;
- *ATOUseCases*: define diagramas use cases;
- *ATOAlgebraicProsoft*: define especificações na notação Prosoft-algébrico.

A partir dos ATOs acima, poderia-se criar outro ATO, chamado ATOCreateSpec, responsável por interpretar os requisitos funcionais (dados e seqüências) do software em construção e guiar o desenvolvedor no processo de criação da especificação formal na notação Prosoft. A definição dos axiomas da operação

```
generate-algeb-spec : Data Use-Cases -> AlgebraicProsoft
```

Conteria todo o processamento necessário para criar a especificação algébrica a partir das estruturas de dados e diagramas use cases, sem ter que se preocupar com os pormenores dos tipos Data, Use-Cases e AlgebraicProsoft já existentes. Logo, os ATOs podem ser considerados caixas-preta, onde sua funcionalidade é dada pelas operações que aparecem na interface do ATO. Note-se que ao criar o ATOCreateSpec, o especificador foca sua atenção principalmente no processo de geração de especificações formais.

4.3.2 Hierarquia das Classes Prosoft

A especificação dos tipos de dados necessários para prover suporte ao APSEE-Tail, resultou na criação de 18 classes e nos correspondentes ATOs algébricos que as manipulam. Segundo Reis (2002), a experiência e prática do grupo PROSOFT no desenvolvimento de software utilizando o formalismo Prosoft-Algébrico tem demonstrado que, a fim de promover o reuso de classes e a diminuição na complexidade dos axiomas, as classes não devem possuir uma profundidade (quantidade de níveis hierárquicos) alta. Partindo deste pressuposto, chegou-se à estrutura de classes, apresentada na figura 4.7[33]. A seguir, é apresentada uma rápida descrição destas classes.

- *AdaptationModel*: define os componentes do APSEE-Tail;
- *AbsProcessModel*: esta classe foi definida por Reis (2002), para representar seu modelo de *templates*. No APSEE-Tail, ela é utilizada para representar a estrutura dos modelos de processo do processo padrão e dos processos adaptados, já que ambos são processos abstratos;
- *Standard*: define o processo padrão da organização de software;
- *Rules*: define as regras que guiam a adaptação do processo padrão;

[33] Com a intenção de ilustrar a hierarquia entre as classes Prosoft do APSEE-Tail, utilizou-se um organograma. Assim, cada caixinha, ao invés de um cargo ou papel, irá representar uma classe.

- *RuleKind*: define o tipo de uma regra de adaptação, quais sejam positiva ou negativa;
- *Conditions*: define o conjunto de condições de uma regra de adaptação;
- *Expression*: define a expressão que representa uma condição;
- *ComparisonOp*: define os operadores relacionais que podem ser usados em uma expressão, quais sejam >, >=, <, <=, =, <>, between, not between, contains ou not contains;
- *Value*: define um valor ou um subconjunto de valores possíveis de um tipo de característica;
- *LogicOp*: define os operadores lógicos para relacionar duas condições, quais sejam And e Or;
- *AdaptedProcesses*: define os processos adaptados para projetos de software, a partir do padrão da organização de software;
- *CharacteristicTypes*: define os tipos de característica utilizados para caracterizar projetos de software;
- *CharKind*: define a espécie do tipo de característica, simples sem e com nivelamento ou múltipla;
- *Projects*: define os projetos de software a serem conduzidos na organização;
- *Characteristics*: define as características de um projeto de software;
- *Cases*: define os casos de adaptação;
- *SimilarCases*: conjunto de casos similares ao corrente;
- *Modifications*: define o conjunto de modificações realizadas no processo adaptado depois de sua criação;
- *ModKind*: define os tipos de modificações que podem ser realizadas no processo adaptado, quais sejam inclusão ou exclusão de atividades.

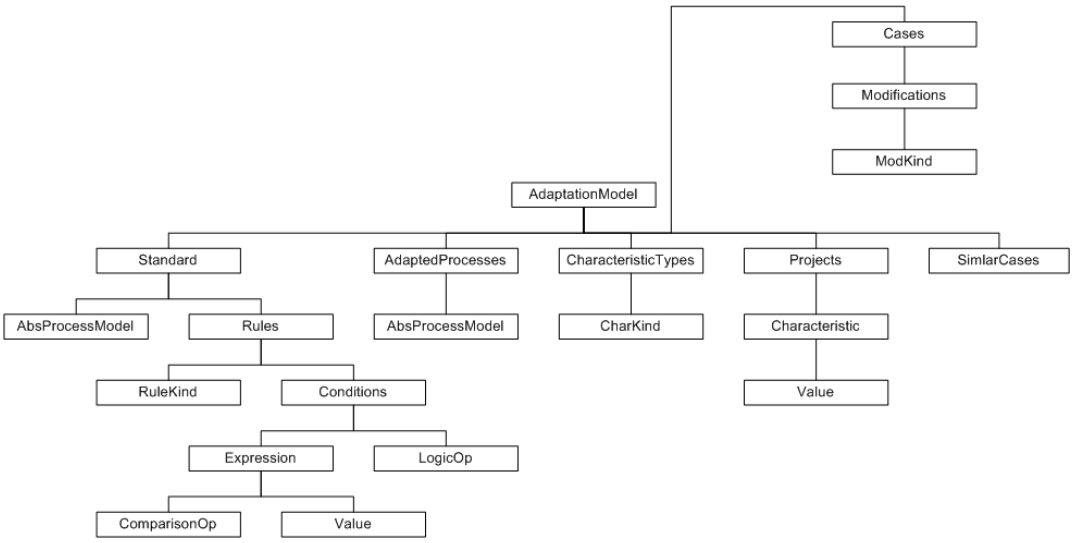


Figura 4.7: Hierarquia de classes do APSEE-Tail

4.4 Formalização das Regras para Coerência do Processo Adaptado

O fato de Gramáticas de Grafos serem formais e intuitivas ao mesmo tempo, influenciou na sua escolha para a definição das regras que determinam os estados de consistência dos componentes de um modelo de processo, permitindo garantir assim a coerência do processo como um todo.

4.4.1 Gramática de Grafos

Segundo Déharbe e Ribeiro apud (LIMA REIS, 2003), os métodos, técnicas e resultados na área de Gramáticas de Grafos já foram estudados e aplicados em uma grande variedade de campos da informática. Gramáticas de Grafos generalizam gramáticas de Chomsky usando grafos ao invés de strings e baseiam-se no processo de transformação que um grafo pode sofrer em função de um conjunto de regras previamente definidas. Um sistema é especificado em termos de estados, que são modelados por grafos, e mudanças de estados, modelados por regras ou derivações. A aplicação de uma regra a um grafo G é chamada passo de derivação, e isso só é possível se existe uma ocorrência (*match* em inglês) do lado esquerdo (L) da regra no grafo atual G. O lado direito (R) da regra define o grafo resultante da aplicação desta regra. A interpretação de uma regra r : L → R é feita da seguinte forma:

- Itens em L que não tem imagem em R são eliminados;
- Itens em L que são mapeados para R são preservados;
- Itens em R que não tem uma pré-imagem em L são criados.

Existem várias abordagens diferentes para Gramáticas de Grafos. Nesse trabalho será usada a abordagem algébrica através de mecanismos de "tipagem" nos grafos. Deste modo, será apresentado um grafo tipo representando os tipos de nodos e arcos do sistema. O grafo inicial e os grafos derivados das transformações devem ser compatíveis com o grafo tipo. Em seguida, serão apresentadas algumas regras de derivação do grafo inicial.

A aplicação de uma regra a um grafo N (passo de derivação) resulta em:

1. Adicionar a N tudo o que for criado pela regra;
2. Excluir do grafo gerado pelo passo 1 tudo o que deve ser excluído pela regra (itens que fazem parte do lado esquerdo e não do lado direito);
3. Excluir arcos pendentes. Caso vértices tenham sido excluídos no passo anterior e se existirem arcos conectados a eles, estes devem ser excluídos também.

Existem diversas aplicações para gramáticas de grafos apontadas por Ribeiro (2000), como por exemplo, sistemas concorrentes, linguagens visuais, reescrita de termos, programação em lógica, reconhecimento e geração de imagens, biologia, dentre outros.

4.4.2 Grafo Tipo

Conforme mencionado no início da seção 4.2, Gramática de Grafos foi utilizada como formalismo para especificar as regras que garantem a coerência do processo padrão durante sua adaptação para um projeto de software, pelos motivos já

mencionados. Uma vez que o APSEE-Tail é aplicado sobre processos que estão de acordo com o meta-modelo de *templates*, descrito em (REIS, 2002), optou-se por adotar a mesma abordagem algébrica daquele trabalho. O grafo tipo que descreve os elementos de um processo abstrato de software, bem como as conexões entre tais elementos, é o mesmo utilizado em (REIS, 2002), com pequenas modificações, sendo apresentado na figura 4.8. Os elementos, que não são necessários nas regras descritas neste trabalho, foram excluídos do grafo tipo.

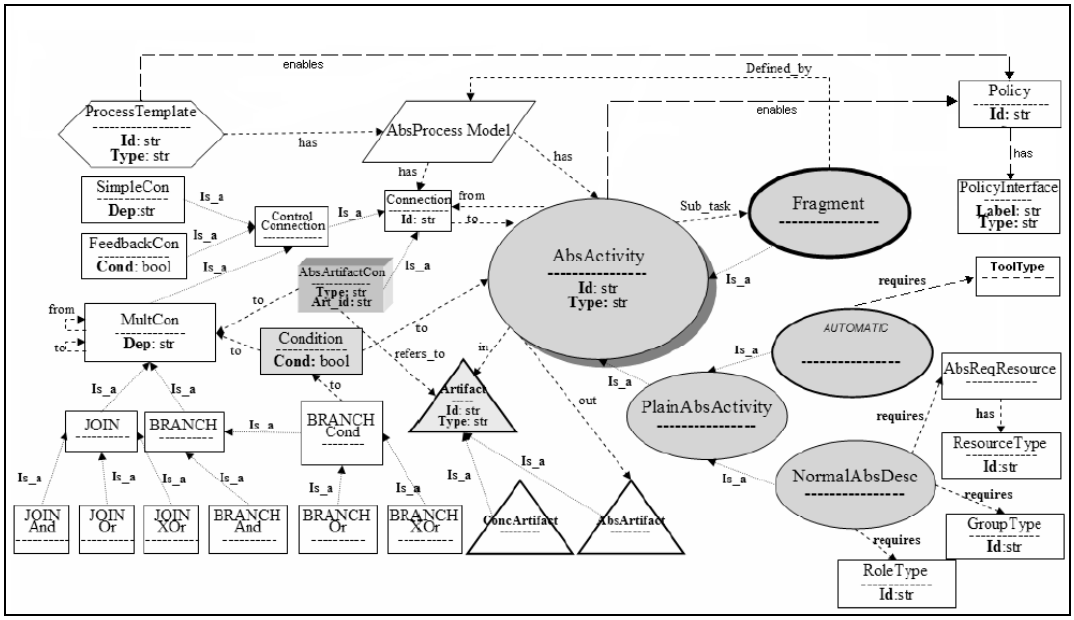


Figura 4.8: Grafo Tipo para processo abstratos – adaptado de (REIS,2002)

O grafo é composto por nodos e arcos, com significados específicos. Os **nodos** são rotulados, e possuem representação gráfica específica, estando relacionados às classes Prosoft apresentadas no anexo 2 deste trabalho. Alguns dos nodos do grafo tipo ainda descrevem - abaixo do rótulo do nodo - os atributos da classe Prosoft correspondente que são úteis para descrever as regras de transformação que compõem o sistema. Os **arcos** incluídos são divididos em dois tipos principais: os arcos com tracejado fino representam as associações de generalização-especialização do tipo **É-um** (*Is-a*), enquanto que as demais (com tracejado mais espaçado) descrevem associações diversas do modelo, devidamente evidenciadas pelos rótulos existentes. As associações do tipo é-um são orientadas no sentido da especialização para a generalização, enquanto que as demais associações são orientadas conforme as associações de tipos de dados apresentadas no anexo 2.

Cabe aqui salientar mais algumas observações sobre o grafo tipo, apresentado na figura 4.8:

- Da mesma forma que proposto por Lima Reis (2003), Reis (2002) incluíu nesse grafo tipo relacionamentos do tipo É-um (*Is_a*): esse tipo de arco representa o relacionamento de herança, com semântica similar ao construtor homônimo usado em linguagens de programação orientadas a objetos. Esse relacionamento *Is-a* simplifica bastante o grafo tipo e as regras de transformação correspondentes, uma vez que os atributos e relacionamentos são herdados para os subtipos de um nodo. Assim, por exemplo, quando o lado esquerdo de uma

regra contém referências ao tipo principal de atividade abstrata (i.e., o nodo rotulado com *AbsActivity*), então ele pode ser substituído por qualquer uma de suas especializações (*Fragment*, *PlainAbsActivity*, Automatic e *NormalAbsDesc*);

- Os arcos do grafo tipo denotam possibilidades de conexão entre os nodos, sendo que as restrições de cardinalidade e de dependência são verificadas e garantidas pela especificação algébrica do sistema.

4.4.3 Classificação das Regras em GG

Na subseção 4.4.1, foram apresentados, de forma genérica, o formato e a interpretação de regras em Gramática de Grafos. Cabe aqui ressaltar que, em algumas regras deste trabalho são utilizadas condições negativas (NAC - *Negative Application Condition*). A interpretação nesse caso é feita da seguinte forma: a regra é ativada se o lado esquerdo da regra ocorre no grafo e as condições negativas **não estão presentes**. As NACs são representadas por elementos cancelados com uma grande marca em forma de "X".

A figura 4.9 apresenta um exemplo, com o objetivo de ilustrar a composição dos elementos usados na descrição de uma regra. No exemplo, o nome da função envolvida (*includeNormalActivity*), assim como seus parâmetros, são incluídos acima do lado esquerdo da regra. O lado esquerdo da regra descreve a condição esperada para instanciar a função descrita: no caso, um objeto de *Template* deve possuir uma conexão *has* com um objeto de *AbsProcessModel*, o qual - pela NAC inserida - não pode estar conectado por *has* com um objeto *AbsActivity* qualquer que possua o identificador fornecido pelo usuário (*act_id*). O lado direito da regra apresenta o resultado obtido a partir da função: uma atividade *Normal* é inserida e conectada ao *AbsProcessModel* correspondente, sendo definida com o identificador *act_id* e os atributos *type* e *script* fornecidos.

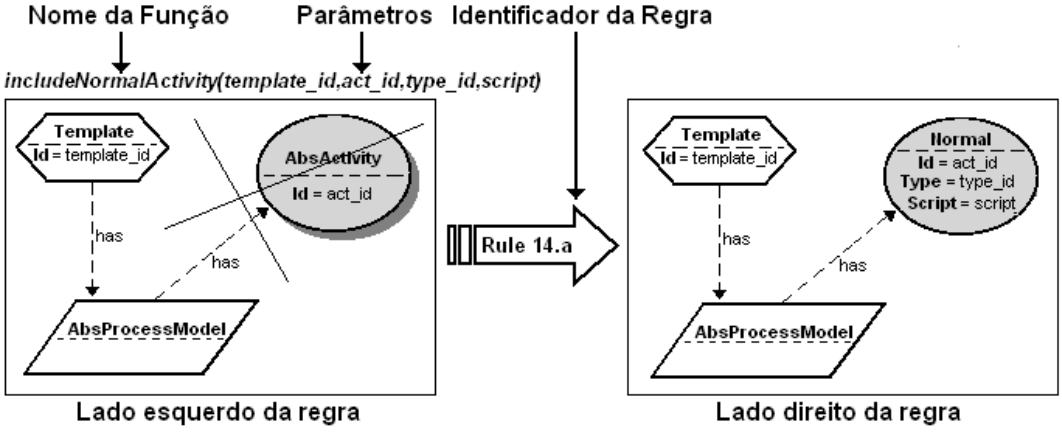


Figura 4.9: Exemplo de uma regra para inclusão de uma atividade normal

De acordo com a finalidade, as regras para garantia da coerência sintática de processos de software podem ser agrupadas como segue:

- *Regras para exclusão de atividades*: foram adaptadas do trabalho de Reis (2002), tratando da exclusão de atividades de um modelo de processo abstrato.

No APSEE-Tail, são aplicadas durante a adaptação do processo adaptado recuperado para o contexto do projeto corrente e na fase de ajustes manuais;

- *Regras para inclusão de atividades*: tratam da inclusão de atividades em um modelo de processo associado a um processo ou fragmento de processo. No modelo proposto, são aplicadas durante a adaptação do processo padrão para o projeto corrente e na fase de ajustes manuais;
- *Regras para redefinição dos componentes de uma atividade incluída*: com exceção da regra de definição de tipo de ferramenta para atividades automáticas, as demais foram retiradas do trabalho de Reis (2002). Tratam da redefinição dos elementos pertinentes a cada atividade incluída em um modelo de processo abstrato. No APSEE-Tail, são aplicadas durante a adaptação do processo padrão para o projeto corrente e na fase de ajustes manuais;
- *Regras para checagem da consistência de conexões*: são utilizadas para garantir a consistência das conexões de um modelo de processo, após a exclusão de atividades do mesmo. No modelo proposto, são aplicadas durante a adaptação do processo adaptado recuperado para o contexto do projeto corrente e na fase de ajustes manuais;
- *Regras para a inclusão de conexões*: tratam da inclusão de conexões em um modelo de processo, evitando duplicação das mesmas e criação de ciclos. As regras para inclusão de conexões de seqüência e de feedback foram retiradas do trabalho de Reis (2002). No modelo proposto, são aplicadas durante a adaptação do processo padrão para o projeto corrente e na fase de ajustes manuais;
- *Regras para redefinição das origens e destinos de conexões incluídas*: foram retiradas do trabalho de Reis (2002), tratando da redefinição das origens e destinos das conexões múltiplas e de artefato incluídas em um modelo de processo. No modelo proposto, são aplicadas durante a adaptação do processo padrão para o projeto corrente e na fase de ajustes manuais;
- *Regras para detecção de fluxo de controle*: utilizadas para detectar o fluxo de controle entre atividades de um processo. Foram originalmente desenvolvidas por Lima Reis (2003), sendo adaptadas para o tipo *AbsActivity* no trabalho de Reis (2002). No modelo APSEE, o teste de fluxo de controle é de extrema utilidade na detecção e prevenção de ciclos em modelos de processos abstratos (*templates*), instanciados ou executáveis. No modelo proposto, são aplicadas durante a adaptação do processo padrão para o projeto corrente e na fase de ajustes manuais.

4.5 Considerações

Neste capítulo, foram apresentados os formalismos utilizados na especificação do APSEE-Tail, bem como onde os mesmos foram utilizados e o porquê. Mostrou-se também, através do formalismo IDEF0, quais foram as etapas do processo de formalização do APSEE-Tail. Conforme apresentado, os componentes do modelo e a semântica do mecanismo de adaptação foram especificados utilizando-se o Prosoft-Algébrico, enquanto que Gramática de Grafos foi utilizada para especificar as regras para garantia da coerência dos processos adaptados.

A utilização de diversos formalismos para especificar os múltiplos aspectos do modelo de adaptação não foi uma tarefa trivial, especialmente ao se definir as fronteiras de utilização de cada um deles. Todavia, tal formalização permitiu um entendimento mais preciso dos componentes e comportamento do modelo proposto, bem como facilitou prototipação. No capítulo seguinte, é apresentado o protótipo do APSEE-Tail.

5 PROTOTIPAÇÃO DO APSEE-TAIL

A especificação formal - usando IDEF0, Prosoft-Algébrico e Gramática de Grafos - descrita no capítulo anterior serviu de base para a implementação de um protótipo no ambiente Prosoft-Java, cujo principal objetivo era o de fornecer a infra-estrutura computacional que permitisse a experimentação prática dos principais componentes propostos pelo modelo. A prototipação do APSEE-Tail foi realizada de forma cooperativa entre o autor e os senhores Heribert Schlebbe (pesquisador associado ao projeto de cooperação aprovado pela FAPERGS) e Tiago Cassol (então bolsista de IC do grupo PROSOFT).

Embora uma descrição e avaliação crítica do ambiente Prosoft-Java estejam fora do escopo desse texto, as seções a seguir apresentam uma visão geral da infra-estrutura de apoio disponível, enfatizando as características que influenciaram na implementação experimental do modelo APSEE-Tail. Portanto:

- A seção 5.1 descreve o ambiente Prosoft-Java;
- A seção 5.2 descreve o ambiente APSEE, PSEE escolhido para interfacear com o APSEE-Tail, uma vez que este possui uma versão implementada no próprio ambiente Prosoft-Java e há todo um conhecimento sobre o mesmo, por parte do grupo PROSOFT;
- A seção 5.3.1 descreve as interfaces disponíveis para configuração do APSEE-Tail, de acordo com a realidade da organização de software que pretende utilizá-lo;
- A seção 5.3.2 descreve as interfaces envolvidas no processo de adaptação de processos de software;
- Por fim, algumas considerações são feitas na seção 5.4.

5.1 O Ambiente Prosoft-Java

Prosoft-Java é a denominação da mais recente implementação do paradigma Prosoft[34], na forma de um ambiente homogêneo e integrado de desenvolvimento de software escrito na linguagem Java (SCHLEBBE, 1997). O desenvolvimento atual do Prosoft-Java é resultado do esforço cooperativo de estudantes e pesquisadores do

[34] A primeira versão monousuária do Prosoft foi desenvolvida em Solaris-Pascal, como descrito por Nunes (1992). Posteriormente, o Prosoft-Distribuído foi desenvolvido em Pascal e C, segundo descrito por Granville (1996) e Schlebbe (1995). Finalmente, segundo Schlebbe (1997), em 1997 foi selecionado Java como nova linguagem hospedeira para o ambiente.

PPGC-UFRGS e da *Fakultät Informatik* da *Universität Stuttgart* (Alemanha), sob orientação do Prof. Dr. Daltro José Nunes. O núcleo do Prosoft-Java consiste de um conjunto de classes e interfaces Java que cooperam entre si para fornecer a funcionalidade requerida pelo paradigma Prosoft, descrito em (NUNES, 1994). O principal objetivo do ambiente é estabelecer uma infra-estrutura que apóie o desenvolvimento de software de alta complexidade através da integração de ferramentas escritas em um paradigma próprio.

Atualmente, o Prosoft-Java é um ambiente portável[35], distribuído[36] e cooperativo[37] que integra ferramentas CASE para auxiliar diferentes fases do processo de software. É importante observar que há uma correspondência entre ATOs Algébricos e ATOs Java, conforme ilustrado pelo exemplo da figura 5.1. Como indicado pela figura, o conceito de classe do Prosoft-Algébrico é diretamente mapeado para implementação em Prosoft-Java, enquanto que métodos precisam ser implementados a partir de derivação das funções algébricas (axiomas) correspondentes. Na versão atual do ambiente, conforme mencionado no capítulo anterior, é possível especificar todo o ATO utilizando a ferramenta Protool. Entretanto, a especificação gerada serve apenas para fins de verificação de corretude através do sistema de reescrita de termos do ambiente OBJ, conforme descrito em (RANGEL, 2003).

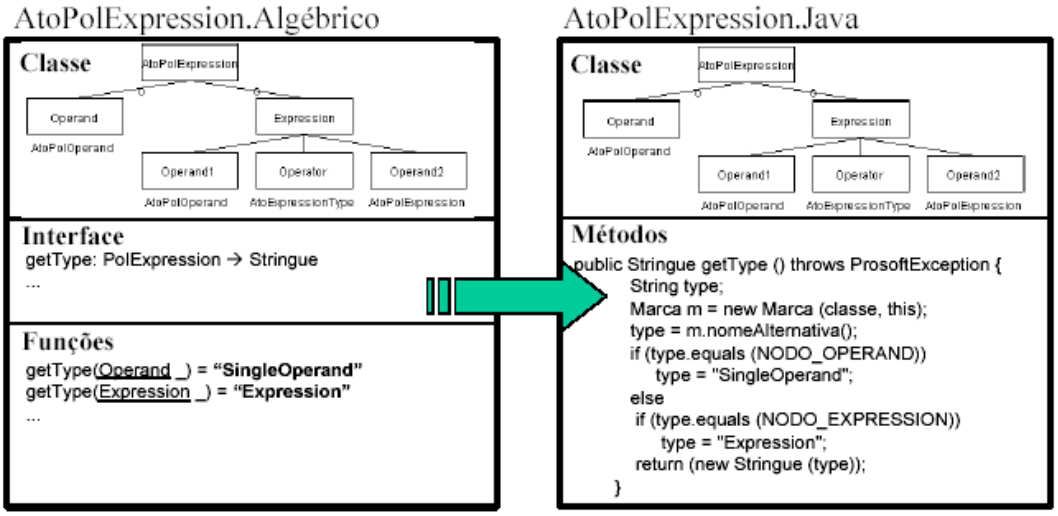


Figura 5.1: Derivação de ATOs Java a partir de ATOs Algébricos (REIS, 2002)

A ferramenta ATO-Classe está disponível para edição de classes Prosoft, conforme ilustrado no capítulo anterior, servindo para gerar automaticamente o código-fonte do ATO-Java correspondente, o qual inclui funções para criação e remoção de objetos e para obtenção (funções rotuladas com o prefixo *get*) e alteração (funções rotuladas com o prefixo *set*) dos valores armazenados nos nodos-folha do tipo definido.

---

[35] Portável no sentido em que o sistema é executável nas plataformas para as quais estão disponíveis a JVM (*Java Runtime Machine*).

[36] A distribuição entre ATOs é fornecida atualmente através do mecanismo *ad-hoc* de comunicação denominado ICS-Distribuído que está implementado sobre o protocolo Java-RMI.

[37] Funcionalidade para trabalho cooperativo está disponível a partir da extensão denominada Prosoft-Cooperativo, descrita por Reis (1998).

A definição de um ATO interativo em Java é dividida em dois arquivos. O primeiro é rotulado com o mesmo nome da classe fornecida, com a extensão *.java*. As funções de interação com o usuário são incluídas em um arquivo separado, cujo rótulo inclui o sufixo SI no nome do ATO, e inclui métodos que são relacionados com as opções de menus (denominada de "parte semântica" do ATO). Os esqueletos dos dois arquivos são gerados automaticamente pela operação "Gera Ato+Sem" disponível no ATO-Classe.

Na implementação atual do sistema Prosoft, as funções adicionais especificadas algebricamente devem ser traduzidas manualmente para métodos escritos de acordo com a sintaxe da linguagem Java, podendo-se utilizar os construtores disponíveis pelo pacote *prosoft.kernel.* Uma descrição completa acerca do desenvolvimento de ATOs-Java foi disponibilizada por Schlebbe (1997) (2005).

5.2 O Ambiente APSEE

A implementação atual do sistema APSEE é composta por mais de uma centena de ATOs-Java, como resultado do trabalho de diferentes membros do grupo de pesquisa PROSOFT. Os ATOs estão relacionados entre si como definido pela arquitetura apresentada na figura 5.2, na qual o gerenciador de processos (APSEE-Manager) interconecta os diferentes serviços para definição, visualização e execução de processos. O APSEE se vale dos serviços de apoio ao trabalho distribuído e cooperativo fornecidos pelo Prosoft-Java permitindo, por exemplo, que o gerenciador de processos execute em um servidor, enquanto que instalações remotas do Prosoft podem executar as agendas de tarefas.

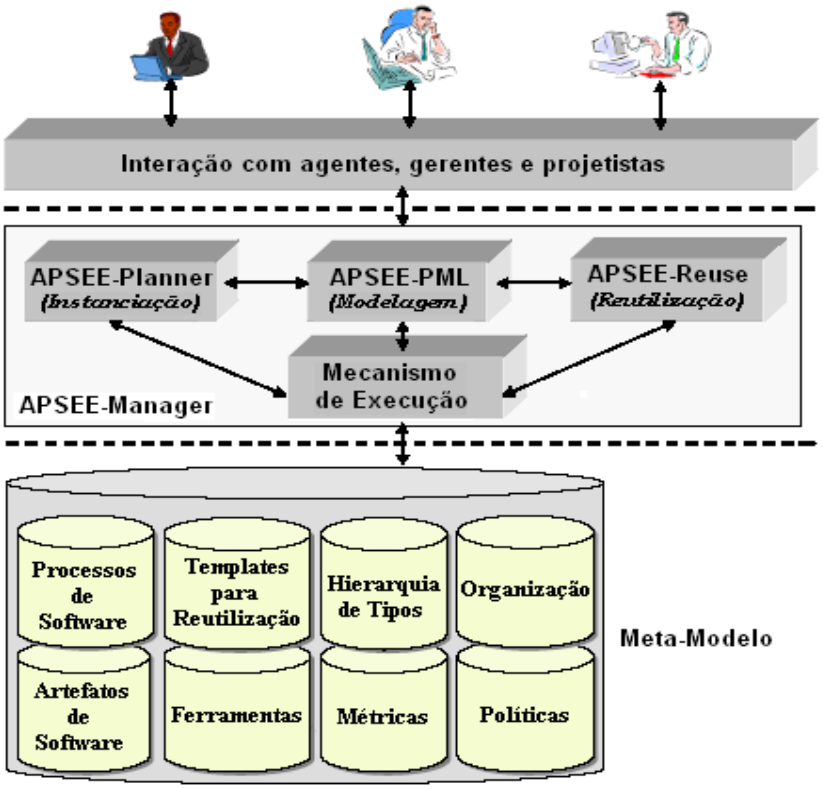


Figura 5.2: Visão geral dos componentes do ambiente APSEE (LIMA REIS, 2003)

Pela figura, nota-se que os principais componentes do sistema APSEE atual estão organizados em três camadas: interação com o usuário; mecanismos para gerência de processos; e repositório com os meta-tipos fornecidos. Uma rápida descrição a respeito destas camadas é fornecida abaixo.

5.2.1 Interação com o Usuário

A interação com o usuário é fornecida através de uma interface gráfica, sendo a mesma disponibilizada pelo ambiente Prosoft-Java, integrando serviços fornecidos pela linguagem de programação hospedeira do ambiente. A figura 5.3 apresenta um exemplo, com duas visões para a execução de um processo, por meio do console do gerente (ao fundo da figura) e da agenda do desenvolvedor (à frente).

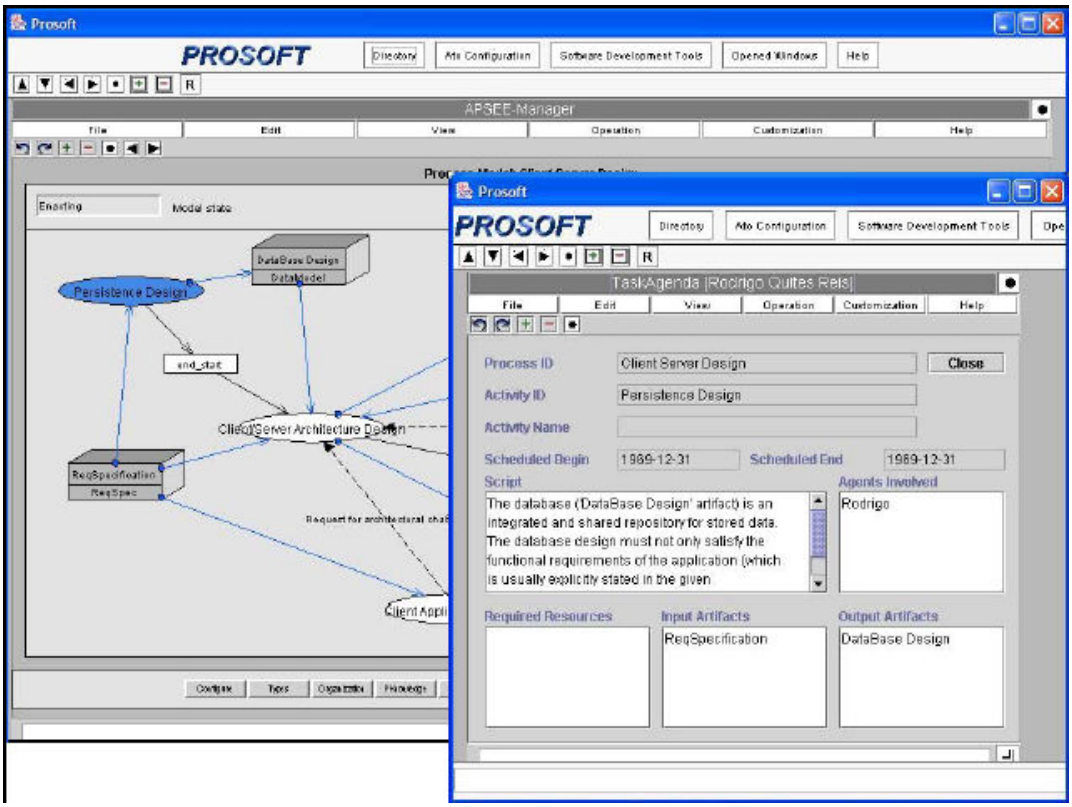


Figura 5.3: Execução distribuída de processos de software no APSEE (REIS, 2002)

5.2.2 Mecanismos para Gerência de Processos

A camada **mecanismos para gerência de processos** descreve um conjunto de serviços que constituem o componente denominado APSEE-Manager existente na implementação atual do sistema.

O mecanismo para execução de processos constitui a base que interpreta os modelos de processos (descritos com o editor APSEE-PML), e exerce um papel central na organização dos mecanismos existentes. Desse modo, o mecanismo de execução é responsável por fornecer dados sobre a dinâmica da execução de processos que são úteis

para o mecanismo de instanciação APSEE-Planner. Além disso, o mecanismo de execução fornece informação sobre processos abstratos, instanciados e executados para o componente responsável pela descrição de processos reutilizáveis.

O componente APSEE-Planner é responsável por fornecer assistência automática para instanciação de recursos e agentes durante a execução de processos de software através de políticas programáveis pelo usuário.

Finalmente, o editor da linguagem APSEE-PML auxilia a descrição de processos e *templates*, sendo também útil para habilitar políticas em modelos de processos de software.

5.2.3 O Meta-Modelo APSEE

O meta-modelo APSEE, camada mais inferior do diagrama da figura 5.2, apresenta alguns dos tipos de dados mais importantes da infra-estrutura proposta. Assim, estão dispostos elementos inter-relacionados, a saber:

- *Templates* e modelos de processos de software, instâncias descritas com editor APSEE-PML;
- As hierarquias de tipos do ambiente (APSEE-Types);
- A organização, representando as pessoas envolvidas, incluindo seus cargos, habilidades e grupos, e os recursos de apoio utilizados;
- Os artefatos de software, descrevendo os objetos de software utilizados e produzidos pelos processos;
- O item conhecimento sobre o processo, descrevendo métricas resultantes da execução dos processos definidos;
- As ferramentas utilizadas no desenvolvimento de software;
- As políticas, descrevendo elementos reutilizáveis usados na composição de processos e *templates*.

5.3 O Protótipo APSEE-Tail

Conforme mencionado no capítulo 3, as funcionalidades do APSEE-Tail dividem-se em duas categorias: funções de configuração, responsáveis por carregar o modelo com o conhecimento necessário à adaptação e que reflita a realidade da organização de software que pretende utilizá-lo; e funções de auxílio à adaptação, que, como o próprio nome sugere, são utilizadas durante o processo de adaptação. Tendo em vista esta classificação, as subseções a seguir estão organizadas como segue. A subseção 5.3.1 trata da integração do protótipo no ambiente Prosoft-Java. A subseção 5.3.2 apresenta as interfaces que provêem suporte às funções de configuração. Já a subseção 5.3.3 apresenta as interfaces que provêem suporte às funções de auxílio à adaptação.

5.3.1 Integração do APSEE-Tail ao Ambiente Prosoft-Java

A figura 5.4 mostra como deve ser chamado o APSEE-Tail no ambiente Prosoft-Java (Ato Configuration → AdaptationModel → New Adaptation Model). Na figura 5.5, é apresentada a tela principal do APSEE-Tail.

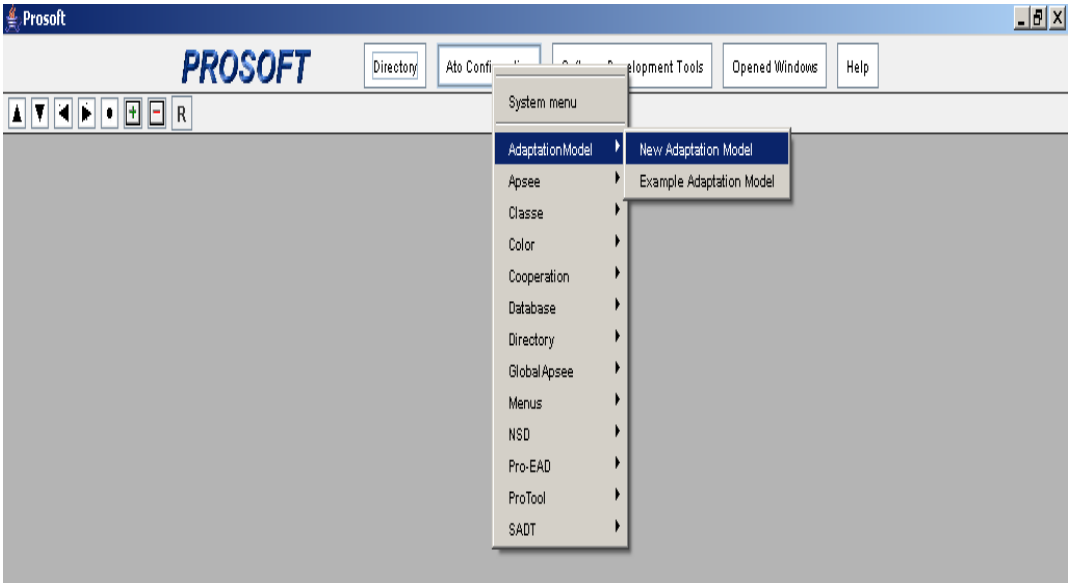


Figura 5.4: Acesso ao APSEE-Tail

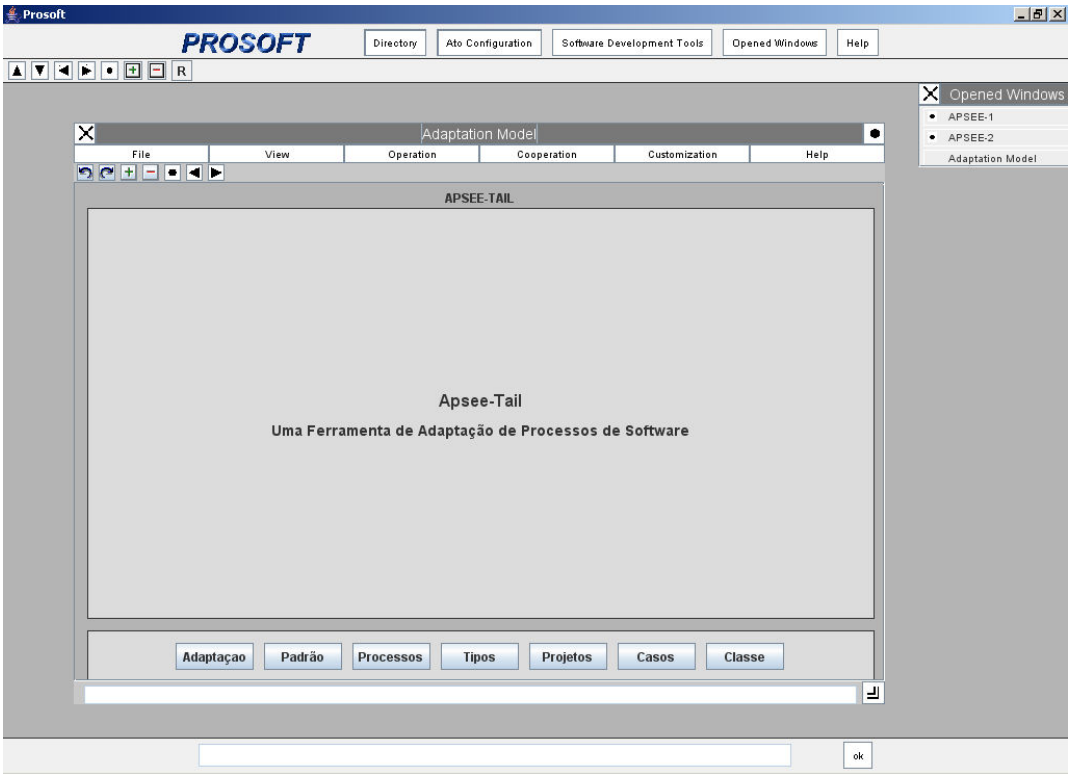


Figura 5.5: Tela principal do APSEE-Tail

5.3.2 Configurando o APSEE-Tail

Por ser uma ferramenta fortemente baseada em conhecimento, o APSEE-Tail precisa primeiramente ser configurado com este conhecimento, para só então ser utilizado eficazmente. É interessante frisar que parte deste conhecimento (o processo padrão, as regras de adaptação e os tipos de características) precisa ser informada à ferramenta, enquanto a outra parte (os casos de adaptação) é agregada automaticamente à medida que a mesma é utilizada. Sem o primeiro tipo de conhecimento, a ferramenta não tem utilidade. Daí a importância de a organização de software designar um engenheiro de processos ou uma equipe para realizar um estudo da sua realidade interna e maturidade em termos de processos. Este estudo tende a evoluir e elucidar a estrutura do processo padrão, as diretrizes que influenciam sua adaptação, bem como os tipos de características que realmente influenciam os projetos desta mesma organização. As interfaces que permitem que a ferramenta seja configurada são descritas a seguir.

5.3.2.1 O Editor do Processo Padrão

Inicialmente, é essencial que seja definido qual o processo padrão da organização de software que irá utilizar o APSEE-Tail. Esta definição pode e certamente será feita de forma incremental e evolutiva. Entretanto, cabe destacar que a modelagem deste processo não é responsabilidade do APSEE-Tail. No Prosoft-Java, a modelagem de processos abstratos é realizada através ambiente APSEE. A figura 5.6 apresenta o caminho para se chegar ao Editor de *Templates* do APSEE (Ato Configuration → APSEE → New APSEE para abrir uma instância do APSEE, Reuse → Process *Templates* para abrir o editor), enquanto as figuras 5.7 e 5.8 mostram, respectivamente, a tela de cadastro de um *template* e a tela de edição do seu modelo de processo. Nota-se que, na primeira tela, foi selecionado o *template "Programing Course"*, sendo apresentado o seu modelo na segunda.

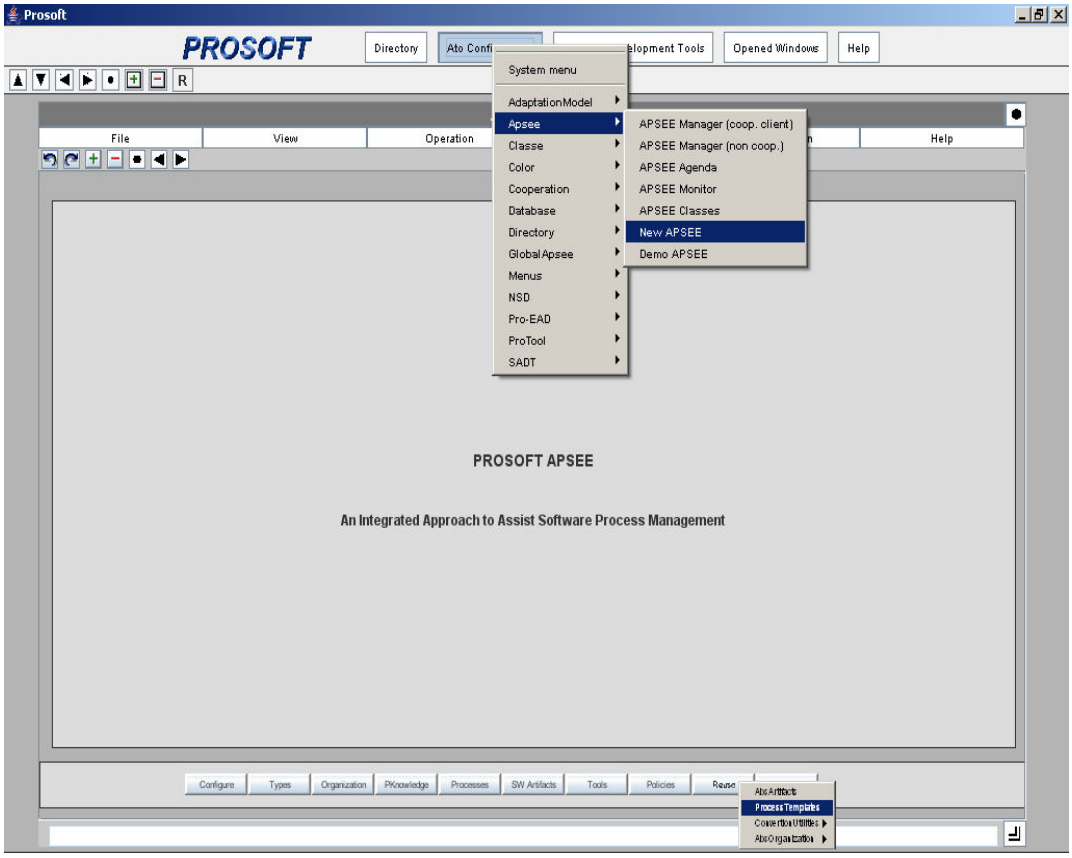


Figura 5.6: Acesso ao Editor de *Templates*

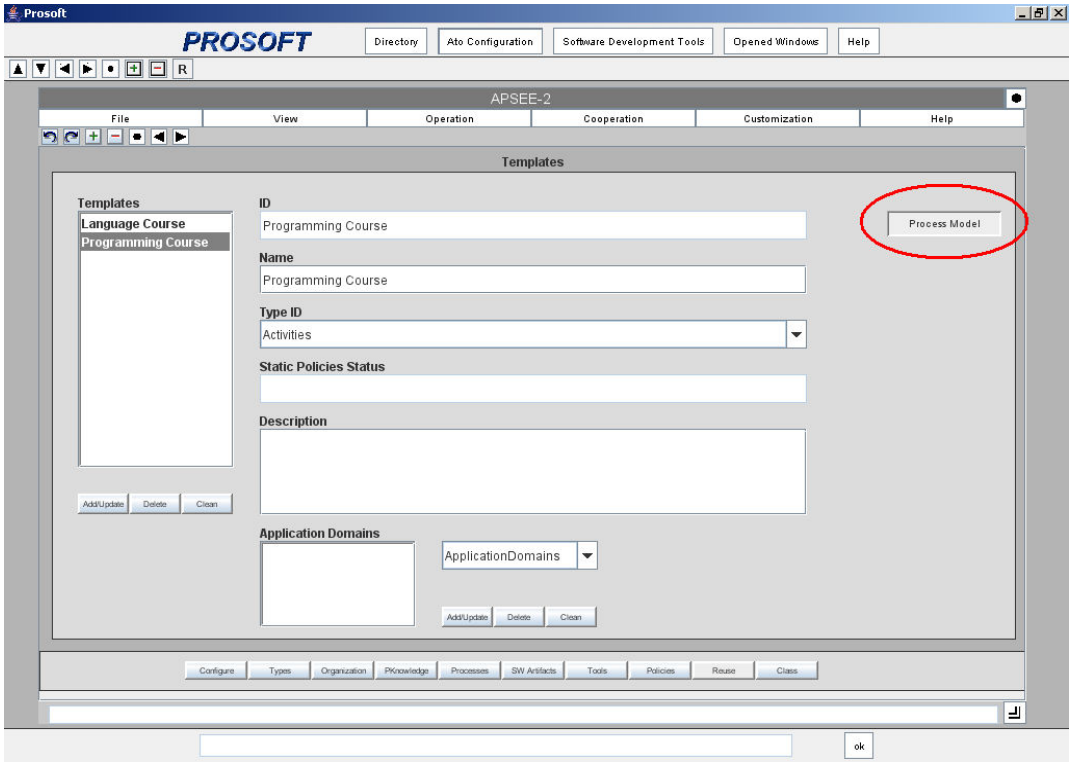


Figura 5.7: Cadastro de *templates*

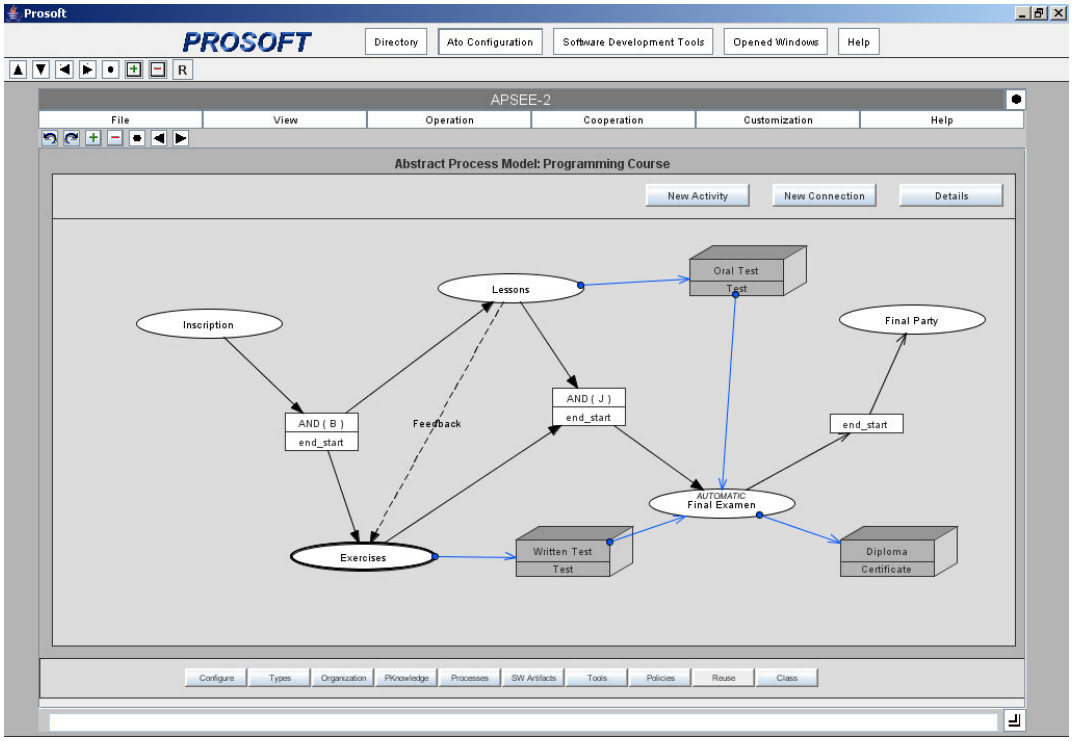


Figura 5.8: Editor gráfico de modelos de processo

Definido o processo padrão externamente, faz-se necessário importá-lo para o APSEE-Tail, por meio do Editor do Processo Padrão, o qual é apresentado na figura 5.9. Deve-se primeiramente selecionar a instância do ambiente APSEE, na qual o *template* modelado se encontra, no ComboBox "Instâncias de APSEE". No ComboBox "*Templates* de Processo" aparecerão todos os *templates* existentes na instância selecionada.

Depois de selecionado o *template*, duas ações são possíveis: define-se o *template* selecionado como um novo processo padrão, através do botão "Novo"; ou apenas atualiza-se o processo padrão com o modelo do *template* selecionado, através do botão "Atualizar". Ao se definir um novo processo padrão, o APSEE-Tail apaga todas as informações relativas ao antigo processo (suas regras de adaptação, os processos adaptados a partir do mesmo e os casos de adaptação). Por outro lado, atualizar o processo padrão significa que provavelmente o mesmo tenha sido apenas alterado externamente, mas ainda trata-se do mesmo processo (nesta situação, os casos de adaptação e processos adaptados são mantidos, enquanto as regras são apenas atualizadas para evitar que algumas apontem para atividades não mais existentes).

Através do botão "Exportar" é possível exportar o modelo do processo padrão para uma instância de APSEE qualquer e, assim, poder alterá-lo externamente. O botão "Regras" abre o formulário Editor de Regras de Adaptação, onde é realizado o cadastro das mesmas. Este formulário é apresentado mais adiante. No Memo "Processo Padrão" é apresentado, na forma de uma árvore hierárquica, o modelo do processo padrão. Ao clicar-se em uma de suas atividades, os dados da mesma aparecerão à direita do formulário, podendo ser limpos através do botão "Limpar". Vale ressaltar mais uma vez que o modelo do processo padrão e suas atividades podem ser visualizados e editados através do APSEE.

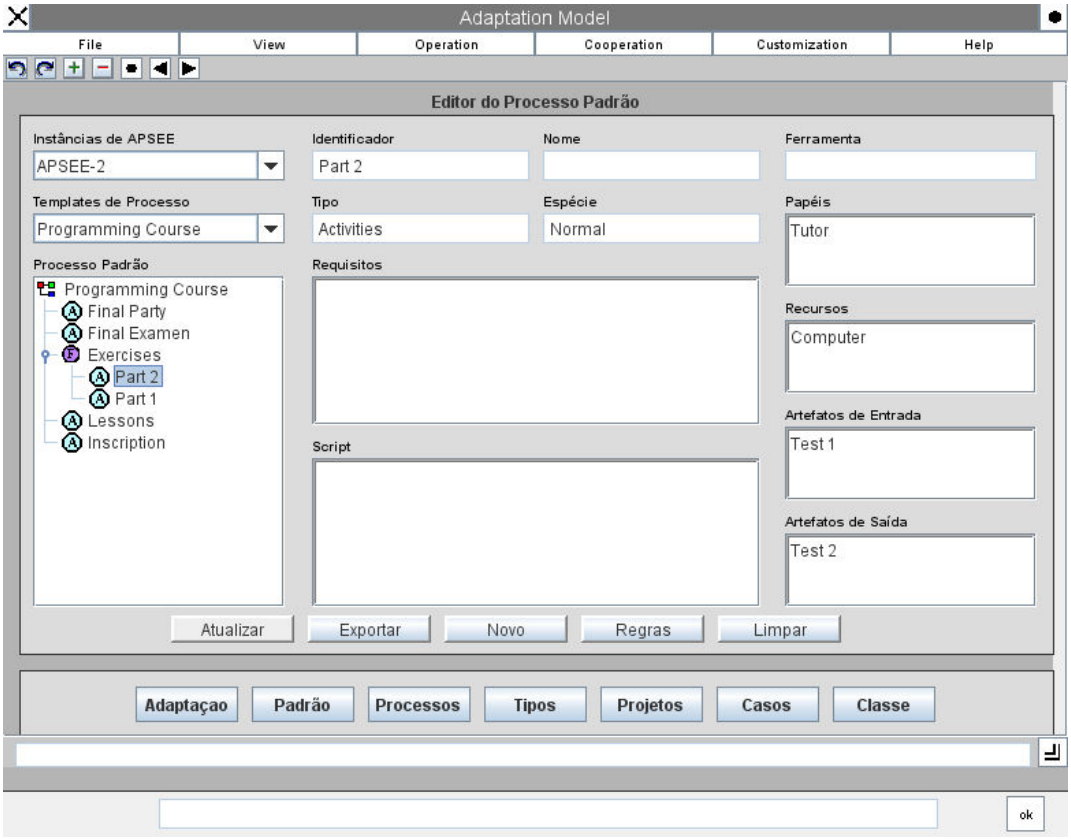


Figura 5.9: Editor do Processo Padrão

5.3.2.2 O Editor de Tipos de Características

Após a definição do processo padrão, é necessário que a organização de software realize um estudo acerca dos fatores, ou seja, das características que possuem influência sobre seus projetos. O cadastro destas características deve ser feito através do Editor de Tipos de Características, mostrado na figura 5.10.

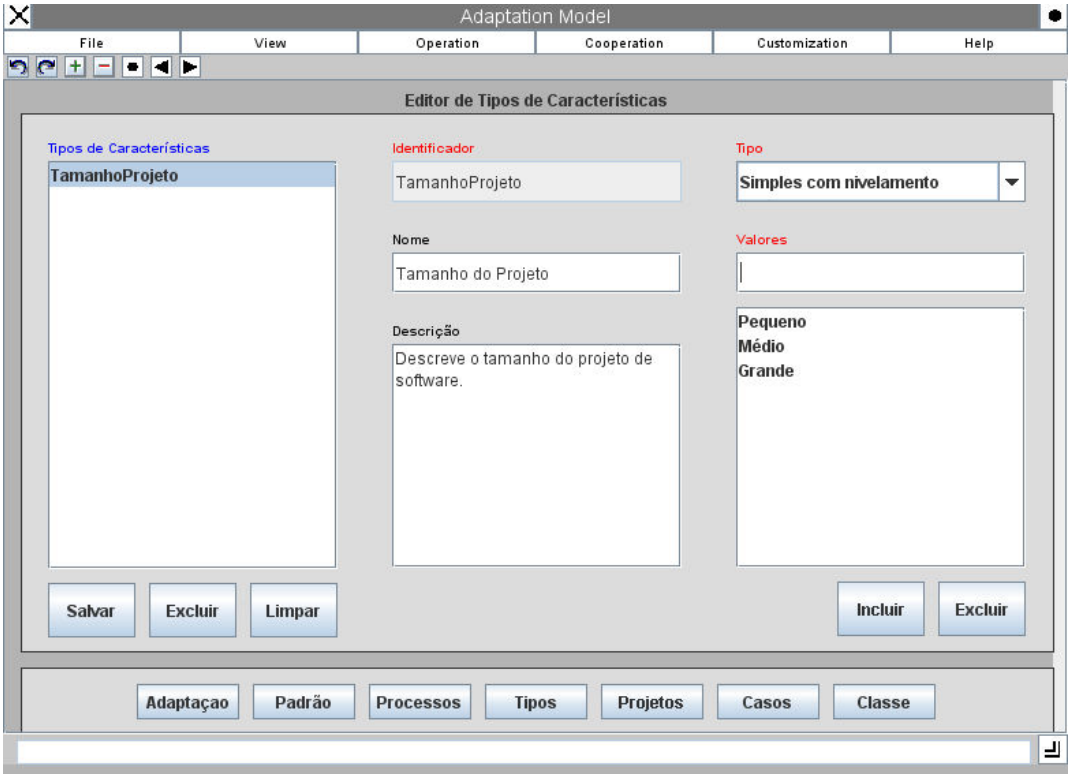


Figura 5.10: Editor de Tipos de Características

Para incluir uma nova característica (botão "Salvar"), devem ser informados um identificador, um nome, uma descrição e o tipo da característica (simples com e sem nivelamento e múltiplo). Após a inclusão de um tipo de característica, podem ser informados os possíveis valores da mesma. Para excluir uma característica ou alterar seus dados, a mesma não pode estar sendo utilizada em um projeto de software ou em uma regra de adaptação.

5.3.2.3 O Editor de Regras de Adaptação

O último passo da configuração do APSEE-Tail é o cadastro das regras de adaptação do processo padrão. Estas regras indicam como as características de um projeto influenciam a adaptação do processo padrão. Talvez este seja o mais difícil tipo de conhecimento a ser extraído da realidade de uma organização de software. Este cadastro deve ser realizado através do Editor de Regras de Adaptação, apresentado na figura 5.11.

No ComboBox "Atividade", encontram-se os identificadores de todas as atividades do processo padrão, à exceção daquelas que já possuem uma regra associada. Vale lembrar que uma atividade pode estar associada a uma e somente uma regra de adaptação. Se não o estiver, será considerada uma atividade obrigatória. Outra observação é que, sempre que uma regra for associada a um fragmento, todas as suas subtarefas serão excluídas deste ComboBox. Ao contrário, se uma de suas subtarefas for associada a uma regra, o fragmento é que é retirado. Além do identificador da atividade, é necessário escolher o tipo da regra, positiva ou negativa, no ComboBox "Tipo" e comentários podem ser feitos sobre a finalidade da regra no Memo "Comentários".

Após ter sido incluída a regra de adaptação, é hora de incluir suas condições, tal qual descrito na gramática do capítulo 3. Segundo esta gramática, uma condição possui o formato "Identificador da Característica + Operador Relacional + Valor + [Operador Lógico]". Sendo assim, para incluir um condição, deve-se selecionar o tipo de característica no ComboBox "Tipo de Característica". Serão então carregados seus possíveis valos no ListBox "Valores", bem como os operadores relacionais suportados no ComboBox "Operador Relacional". Seleciona-se então o operador e o(s) valor(es) desejado(s). Depois de incluída a condição, esta será mostrada no ListBox "Condições". A partir da segunda condição, o ComboBox "Operador Lógico" é habilitado.

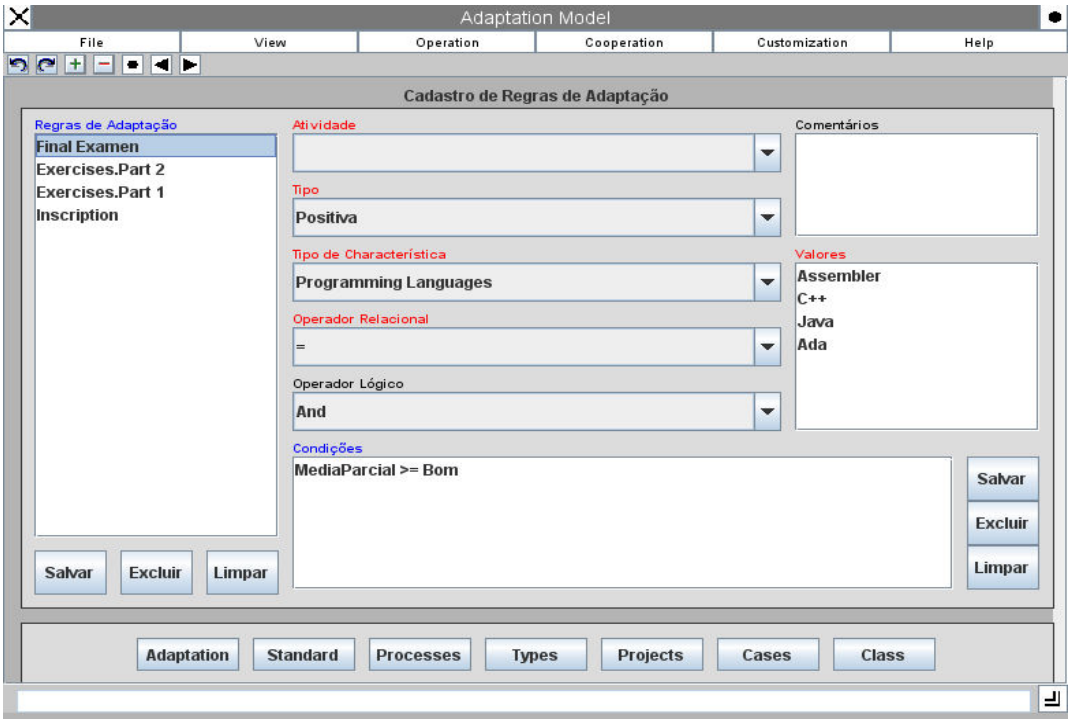


Figura 5.11: Editor de Regras de Adaptação

5.3.3 Adaptando o Processo Padrão para um Projeto de Software

Após sua configuração inicial, o APSEE-Tail está pronto para ser usado na adaptação do processo padrão da organização para a realidade de seus projetos. Conforme visto nos capítulos 3 e 4, este é um processo que exige os seguintes macro-passos:

- Definição e caracterização do projeto de software;
- Especificação dos parâmetros de adaptação;
- Realização da adaptação:
  - Seleção do caso de adaptação similar;
  - Adaptação do processo recuperado;
  - Adaptação do Processo Padrão;
  - Realização de ajustes manuais.

- Alteração do caso de adaptação;
- Atualização do processo adaptado.

5.3.3.1 Definindo e Caracterizando o Projeto de Software

A definição e caracterização do projeto de software, para o qual deseja-se obter um processo adequado, deve ser feita através do Editor de Projetos de Software, apresentado na figura 5.12.

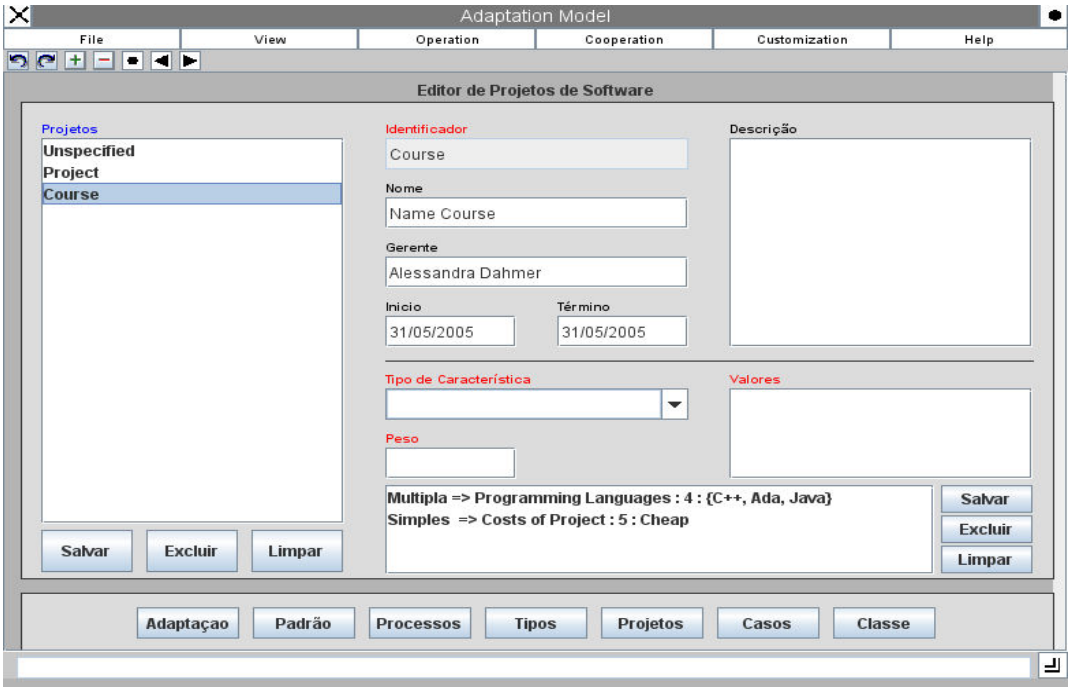


Figura 5.12: O Editor de Projetos de Software

Para definir um novo projeto de software, é necessário informar um identificador, nome, descrição, gerente e datas de início e término. Após ter sido criado o projeto, suas características podem ser informadas. Esta deve ser uma tarefa realizada em conjunto pelo gerente do projeto e pelo engenheiro de processos. Para cada característica relevante, seleciona-se a mesma no ComboBox "Tipo de Característica", seleciona-se dentre seus valores possíveis aquele(s) aplicáve(l)(is) ao projeto e defini-se sua importância (peso). Para alterar os dados de um projeto de software ou excluí-lo, é necessário que o mesmo não esteja associado a um caso de adaptação.

5.3.3.2 Especificando os Parâmetros da Adaptação

Após a definição e caracterização do projeto de software, é hora de especificar os parâmetros necessários para a realização da adaptação. Através do Adaptador de Processos, apresentado na figura 5.13, é necessário selecionar o projeto para o qual será realizada a adaptação, o grau de similaridade que se deseja usar para recuperação de casos de adaptação semelhantes e um identificador para o processo adaptado a ser criado. Para iniciar a adaptação deve-se clicar no botão "Adaptar".

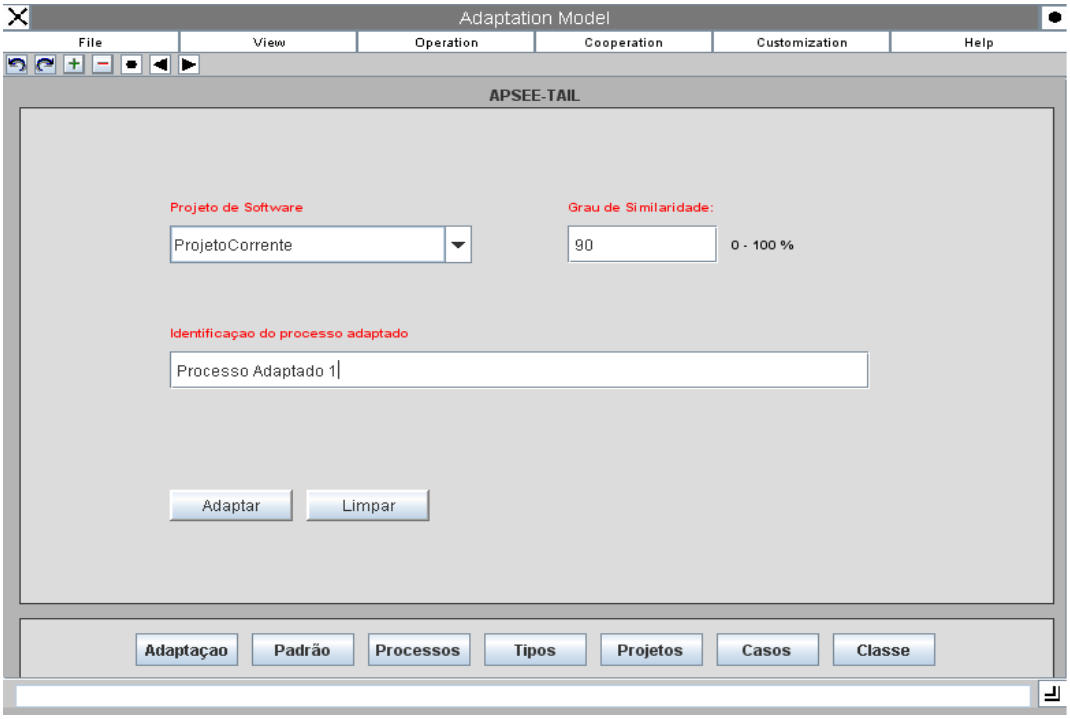


Figura 5.13: Adaptador de Processos

5.3.3.3 Realizando a Adaptação

Durante a adaptação do processo padrão para o projeto selecionado, a interação com o engenheiro de processos será necessária em alguns pontos e se dará de diferentes formas.

Inicialmente, se mais de um projeto similar tiver sido encontrado, o Editor de Casos de Adaptação será aberto mostrando os casos associados a tais projetos. Baseado nos comentários, avaliações e graus de similaridade de cada caso de adaptação, o engenheiro de processos deve selecionar um e clicar em "Prosseguir". Uma avaliação mais detalhada pode ser feita com base na visualização das características dos projetos ou dos modelos dos processos adaptados. Depois de selecionado o caso de adaptação, o processo adaptado associado ao mesmo será adaptado para realidade do projeto corrente. Pode haver a necessidade de questionamento do engenheiro de processos, através de caixas de diálogo, quanto à manutenção de certas atividades.

Após a adaptação do processo padrão para o projeto corrente e o seu merge com o processo adaptado até então, o engenheiro de processos pode realizar ajustes manuais no processo adaptado gerado, através do Editor de Atividades, apresentado na figura 5.14. Atividades podem ser excluídas do processo adaptado ou incluídas do processo padrão para o processo adaptado, através dos botões "Incluir/Excluir". Para cada uma destas ações deve ser fornecida uma justificativa. O botão "Restaurar" desfaz as modificações feitas até então no processo adaptado. O engenheiro de processos pode abandonar a adaptação ou finalizá-la, gerando então um caso de adaptação.

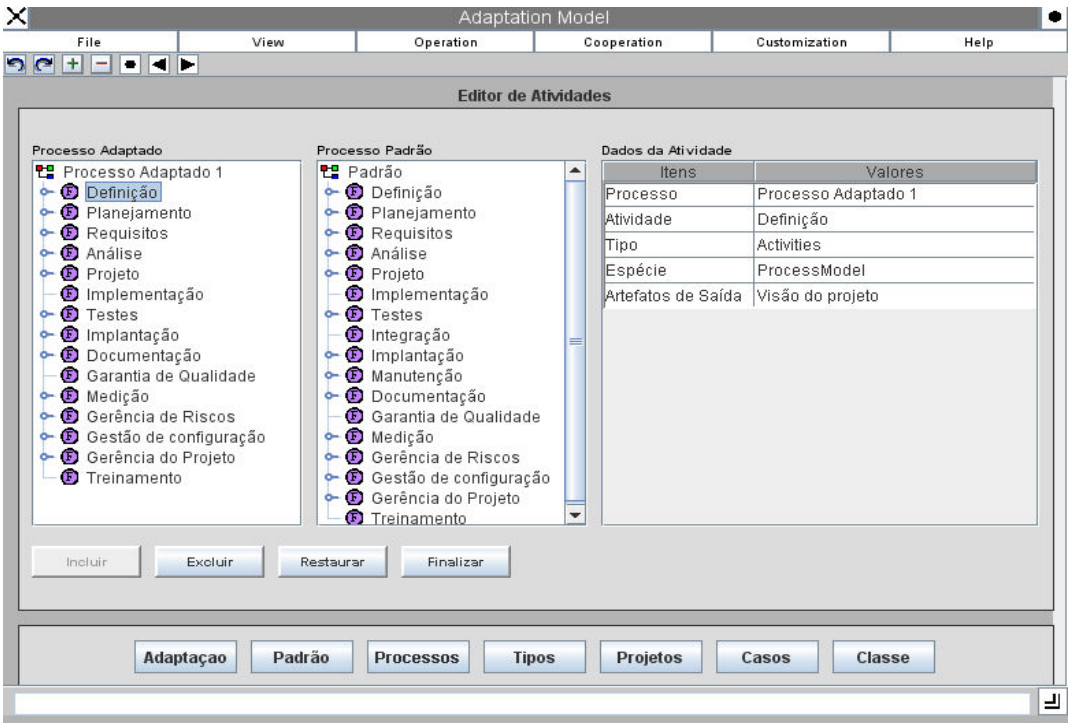


Figura 5.14: Editor de Atividades

5.3.3.4 Alterando o Caso de Adaptação

Finalizada a adaptação, o Editor de Casos, apresentado na figura 5.15, é aberto automaticamente com o caso de adaptação criado selecionado.

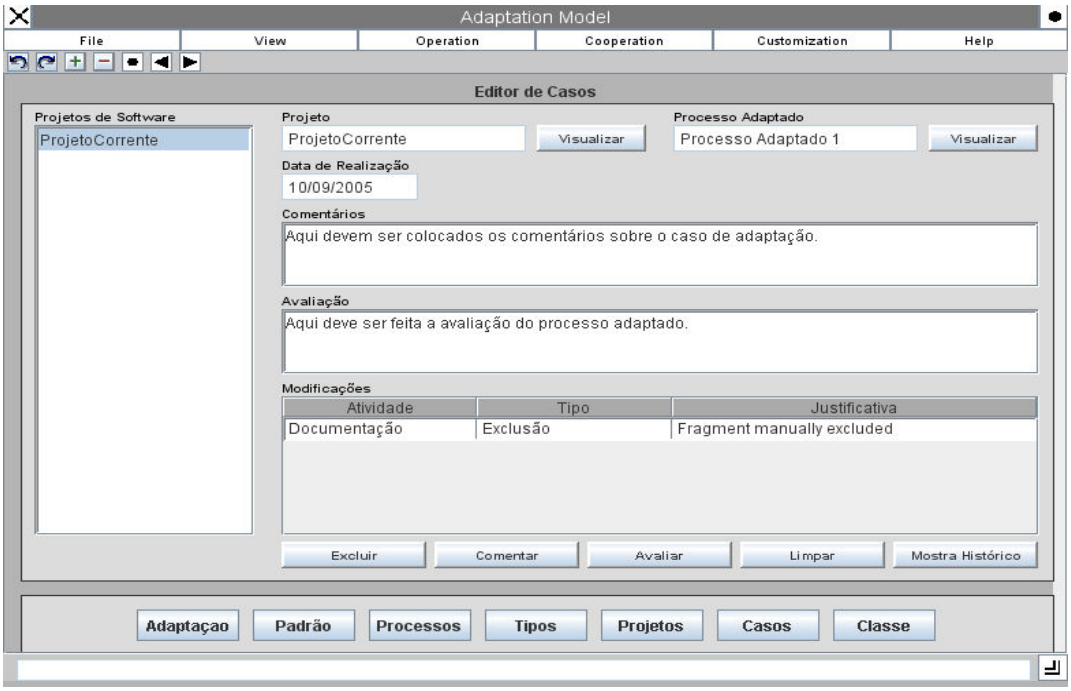


Figura 5.15: O Editor de Casos de Adaptação

O engenheiro de processos pode então deixar comentários sobre a adaptação realizada e, posteriormente, avaliar a aplicação do processo adaptado gerado no projeto de software real. Ele pode ainda, através do botão "Visualizar", abrir o Editor de Processos Adaptados e atualizar o processo adaptado em uma ferramenta externa. Por fim, clicando no botão "Mostrar Histórico", pode-se visualizar todos os passos realizados pela ferramenta durante a adaptação.

5.3.3.5 Atualizando o Processo Adaptado

Por fim, o processo adaptado gerado pode necessitar de ajustes, na visão do engenheiro de processos e do gerente do projeto, tais como inclusão, exclusão ou alteração das descrições das atividades, das conexões entre elas, etc. Tais mudanças devem ser providas em uma ferramenta de modelagem externa, no caso o APSEE. É função do APSEE-Tail permitir a exportação do processo adaptado e posterior importação, para garantia de consistência entre o que foi gerado pelo modelo e o que foi realmente utilizado no projeto de software real.

O Editor de Processos Adaptados, apresentado na figura 5.16, é responsável pela visualização e atualização dos processos adaptados. É possível exportar ou importar o processo adaptado, especificado no ComboBox "Processos", para/de a instância de APSEE, especificada no ComboBox "APSEE Instance". Ao importar um processo adaptado, o modelo do processo antigo é substituído pelo novo e o conjunto de modificações associadas pode ter de ser alterado.

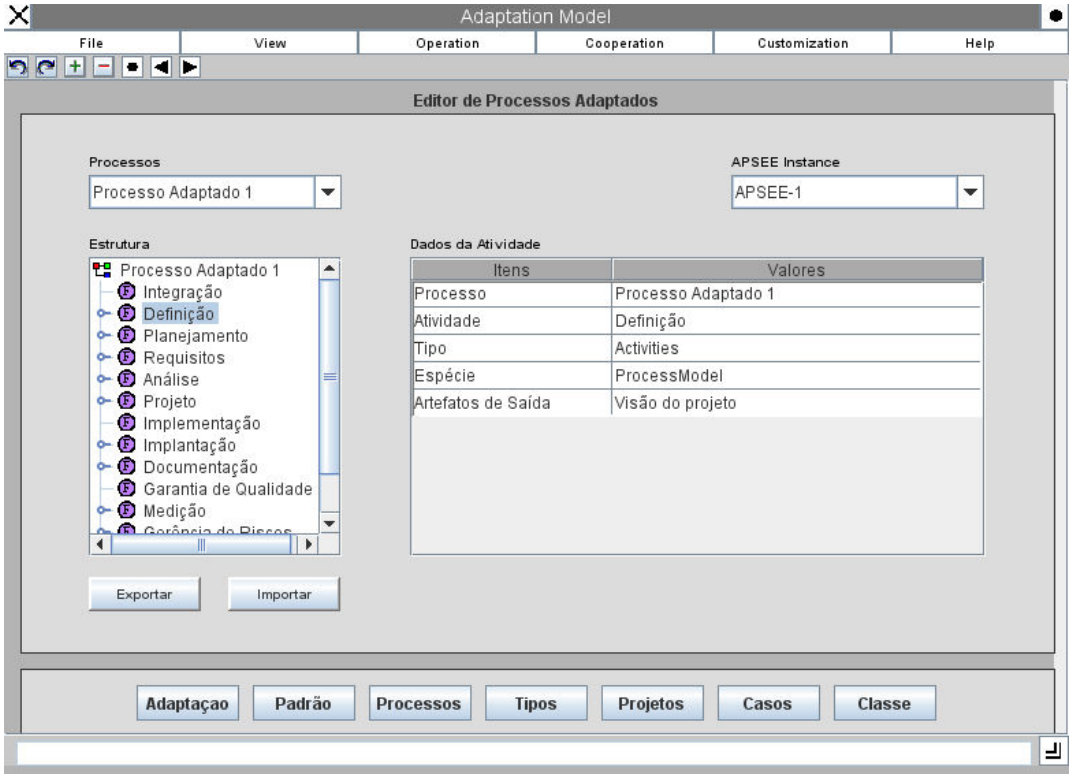


Figura 5.16: O Editor de Processos Adaptados

5.4 Considerações

Neste capítulo, foi apresentada a ferramenta APSEE-Tail, protótipo construído tendo por base a especificação formal apresentada no capítulo anterior, com o intuito de fornecer a infra-estrutura computacional necessária para a experimentação prática dos componentes do modelo de adaptação proposto.

Apresentou-se também, de forma superficial, o Prosoft-Java, ambiente no qual a ferramenta foi desenvolvida e acoplada. Tendo uma função bem definida dentro do meta-processo de software, a de adaptar processos abstratos, a ferramenta pode ser considerada auto-contida, assumindo-se certas premissas. Entretanto, é necessário que ela trabalhe em conjunto com um PSEE qualquer, de modo a completar o meta-processo do mesmo e aumentar seu nível de automação. O APSEE foi escolhido para realizar esta interface. No próximo capítulo, é apresentando um exemplo de uso desta ferramenta.

6 EXEMPLO DE APLICAÇÃO DO APSEE-TAIL

Com o intuito de apresentar a utilização do APSEE-Tail, através do protótipo construído, foi elaborado um cenário de exemplo. A aplicação deste cenário no protótipo e os resultados obtidos são descritos neste capítulo, cuja estrutura é como segue:

- A seção 6.1 define, passo a passo, a abordagem utilizada para elaborar o cenário de exemplo;
- A seção 6.2 descreve a aplicação do cenário no protótipo, apresentando os resultados obtidos;
- A Seção 6.3 apresenta as considerações acerca do capítulo.

6.1 Abordagem Utilizada na Elaboração do Cenário de Exemplo

A abordagem utilizada para a elaboração do cenário de exemplo compreendeu os seguintes passos:

1. *Definição do processo padrão*: o APM (*Adaptable Process Model*), definido por Pressman (2005), foi utilizado como ponto de partida para a elaboração do processo padrão;
2. *Definição dos tipos de característica*: o conjunto de tipos de característica utilizado para caracterizar os projetos exemplo foi extraído e adaptado daqueles encontrados em (AHN et al, 2003), (BERGER, 2003) e (COELHO, 2003). Os tipos de característica deste conjunto foram ainda agrupadas, para facilitar a atribuição dos pesos;
3. *Definição das regras de adaptação*: devido à ausência de um ambiente organizacional real para avaliação, algumas das diretrizes de adaptação foram geradas de forma empírica, enquanto outras foram extraídas da literatura pesquisada;
4. *Caracterização dos projetos*: foram definidos e caracterizados dois projetos, aqui referenciados como projetos A e B, propositadamente semelhantes, para que se pudesse utilizar o reuso de processos;
5. *Adaptação do processo padrão para o projeto A*: com base nas características do projeto A, foi realizada a adaptação do processo padrão através do APSEE-Tail. A adaptação foi realizada com base apenas nas regras de adaptação, já que não havia registros no repositório de casos de adaptação;

6. *Realização externa de melhorias no primeiro processo*: algumas modificações foram realizadas no processo adaptado para o projeto A, através da linguagem de modelagem do ambiente APSEE;
7. *Adaptação do processo padrão para o projeto B*: o processo padrão foi adaptado para o projeto B, porém como existiam registros no repositório de casos, o método de comparação de projetos foi aplicado, trazendo como resultado o processo adaptado para o projeto A. Este processo foi adaptado a situação corrente e, a seguir, alterado com as atividades aplicáveis do processo padrão;
8. *Análise dos resultados*: a análise dos resultados concentrou-se principalmente na checagem da coerência dos processos adaptados gerados e na usabilidade do protótipo.

6.2 Aplicação do Cenário de Exemplo

Eis uma rápida descrição do hipotético ambiente organizacional que se propôs a utilizar o APSEE-Tail: a EDSOFT (Empresa Desenvolvedora de Software) é uma organização de software, de porte médio, que atua tanto no desenvolvimento de software sob medida, quanto na produção de software de prateleira para diversos nichos de mercado. Esta empresa já possui boa parte de suas filiais com certificação CMM nível 2 e já possui um processo padrão definido e documentado. Justamente por estar tentado galgar o nível 3 deste mesmo modelo de maturidade, vislumbrou na utilização do APSEE-Tail uma oportunidade de atender um dos critérios deste nível (já mencionado no capítulo 1).

6.2.1 O Processo Padrão

O processo padrão da EDSOFT, apresentado nas figuras 6.1 e 6.2, foi desenvolvido com base no *Adaptable Process Model*[38] (PRESSMAN, 2005), em outros processos encontrados na literatura e na realidade e necessidade de seus projetos. Este processo se encontrava descrito em linguagem textual e disponibilizado através da *intranet* da empresa. Houve a necessidade de formalização do mesmo, através da APSEE-PML, no ambiente APSEE.

Em virtude de o fragmento principal ser consideravelmente grande, mostrá-lo com todas as informações necessárias, gerou uma visão muito poluída. Optou-se então por também mostrar o fragmento principal através das visões[39] funcional (figura 6.3, que mostra a relação entre as atividades e os seus artefatos) e comportamental (figura 6.4, que mostra a relação entre as atividades e suas dependências).

---

[38] Cabe aqui deixar claro que o modelo do processo padrão foi apenas inspirado no APM, o qual, por ser uma solução de mercado, é bem mais completo e complexo. Vale ressaltar ainda que tal processo possui uma metodologia própria de adaptação.

[39] Sousa (2003) agregou valor ao ambiente APSEE posibilitando suporte à múltiplas visões de processos de software. Esta ferramenta é chamada APSEE-Monitor e suporta a visualização de um processo de software, segundo cinco perspectivas: funcional, comportamental, hierárquica, organizacional e orientada a informações.

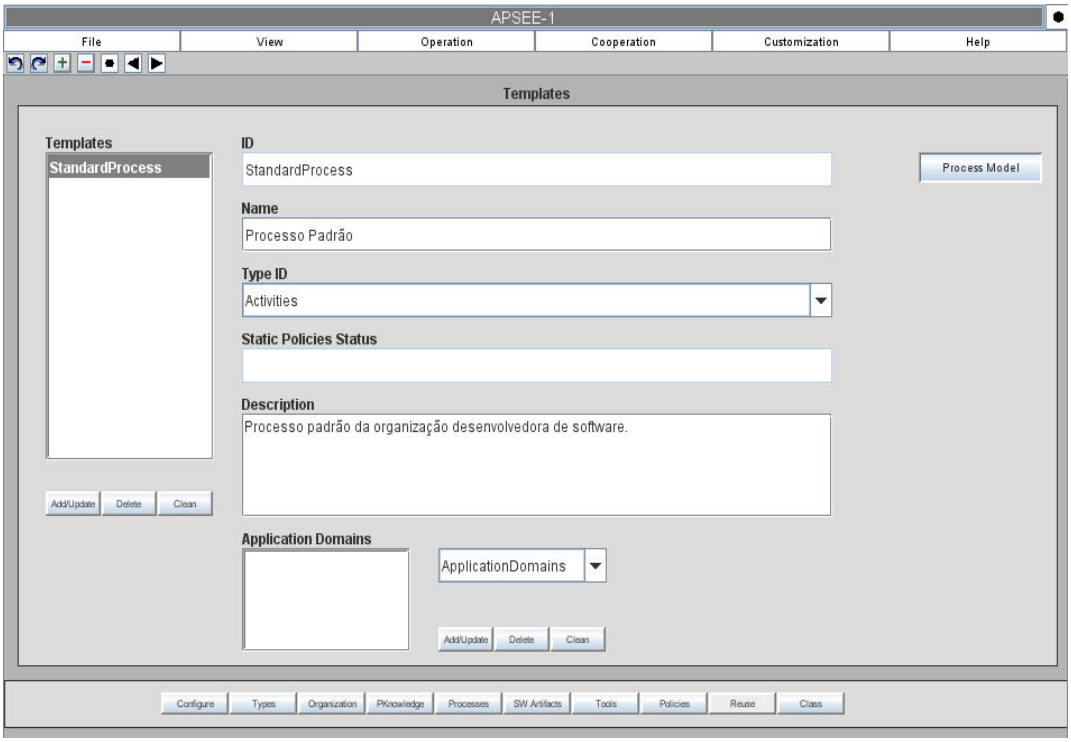


Figura 6.1: Processo padrão (informações de cadastro)

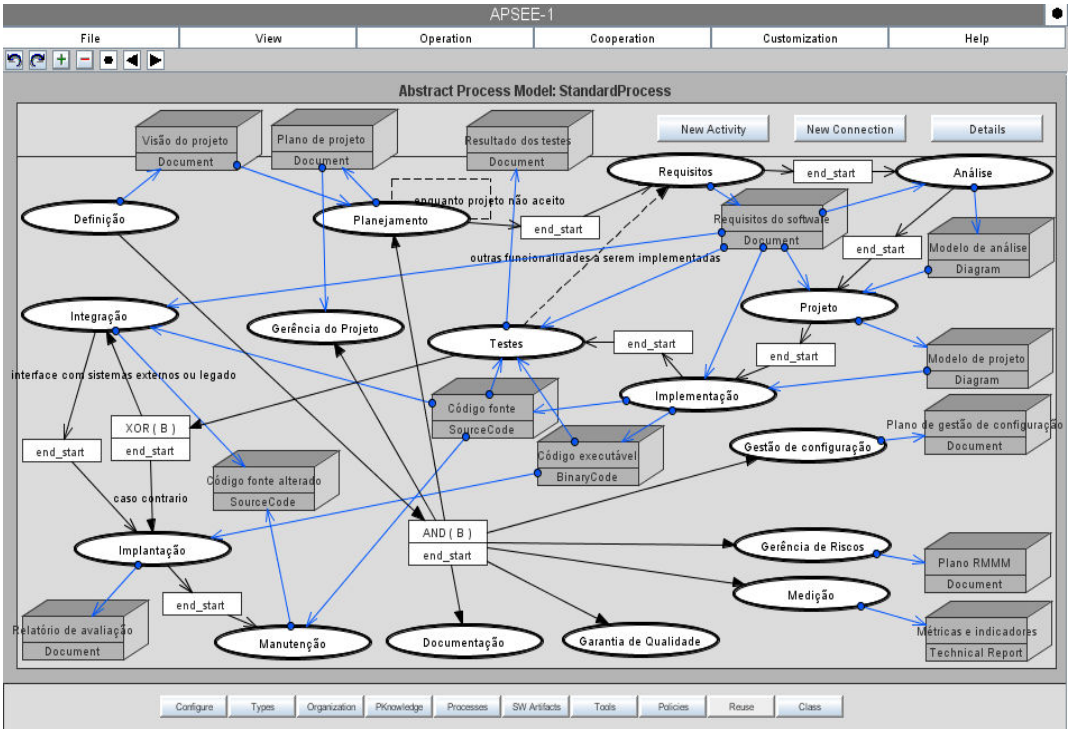


Figura 6.2: Processo padrão (fragmento principal)

Como pode ser visto, cada subatividade também se constitui como outro fragmento com seu respectivo modelo de processo. O importante aqui é frisar que estes fragmentos podem ser agrupados em dois tipos de atividades: atividades básicas e guarda-chuva.

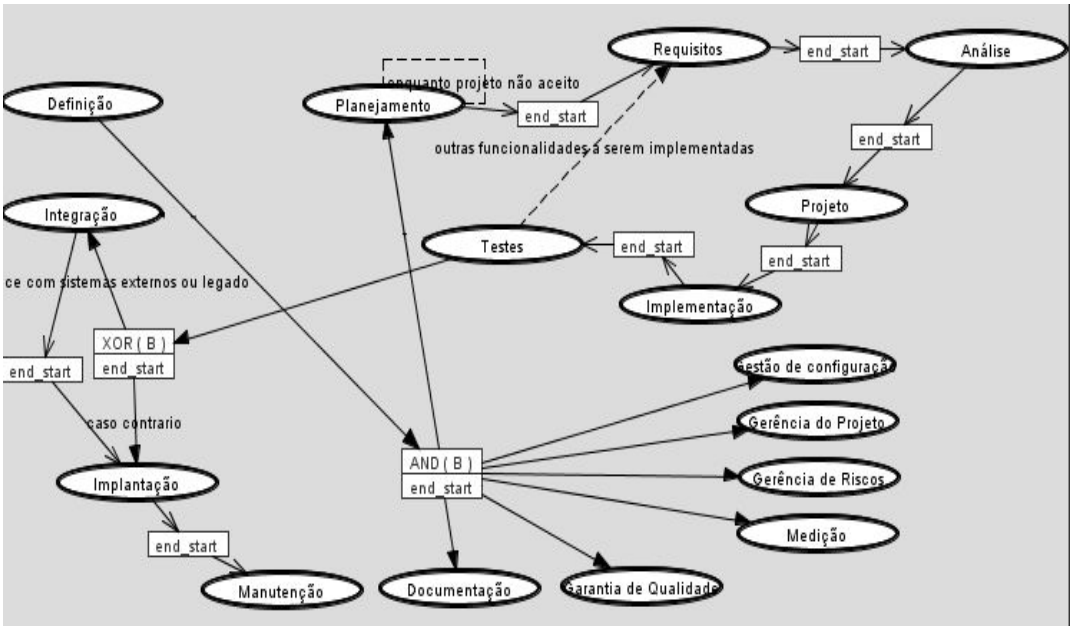


Figura 6.3: Visão funcional do processo padrão

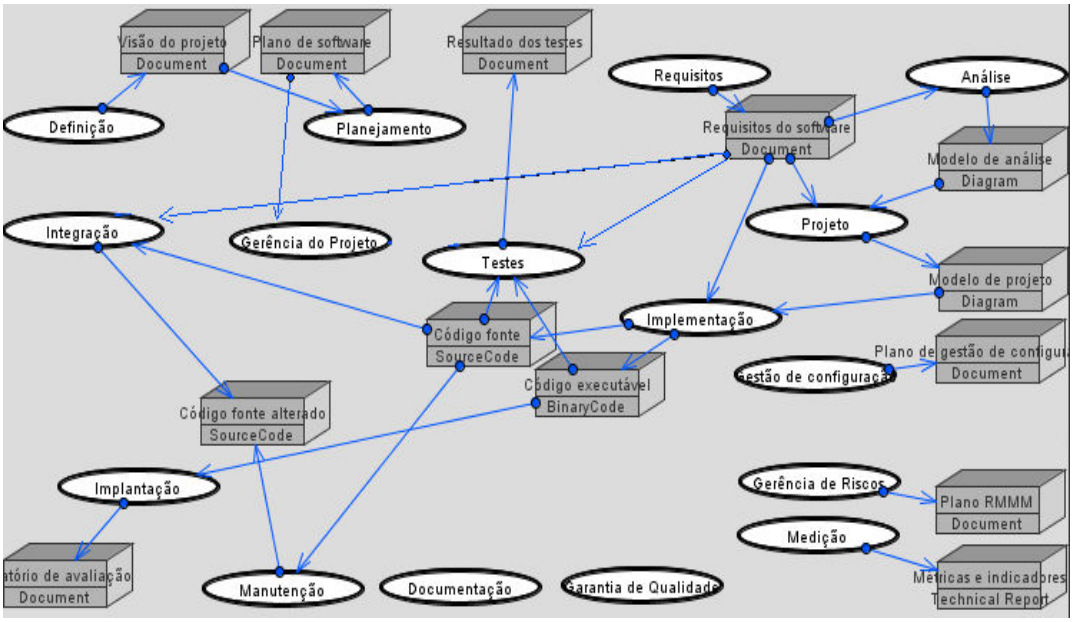


Figura 6.4: Visão Comportamental do processo padrão

6.2.1.1 Atividades básicas

conjunto horizontal de atividades do processo de software, ou seja, que participam diretamente da criação do produto de software final. São elas: definição (figura 6.5), planejamento (figura 6.6), requisitos (figura 6.7), análise (figura 6.8), projeto (figura 6.9), implementação, testes (figura 6.10), integração, implantação (figura 6.11) e manutenção (figura 6.12).

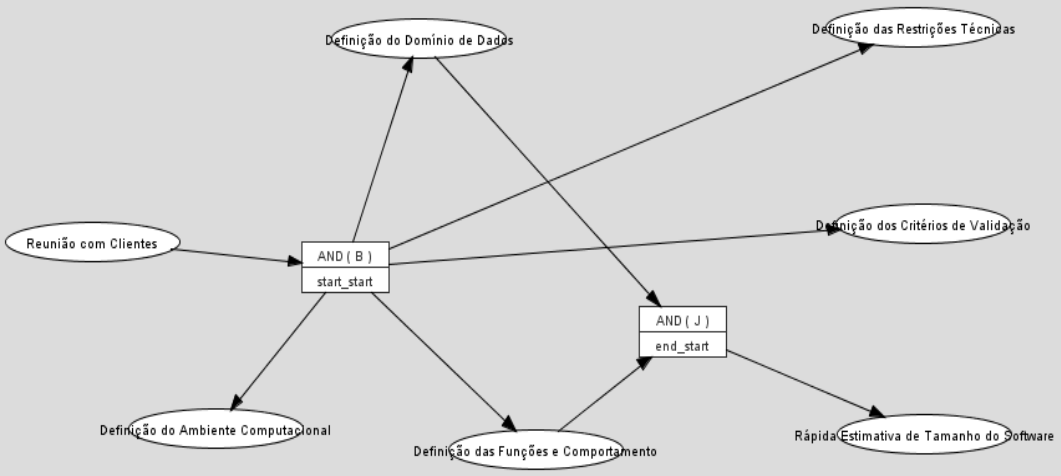


Figura 6.5: Processo padrão (fragmento Definição)

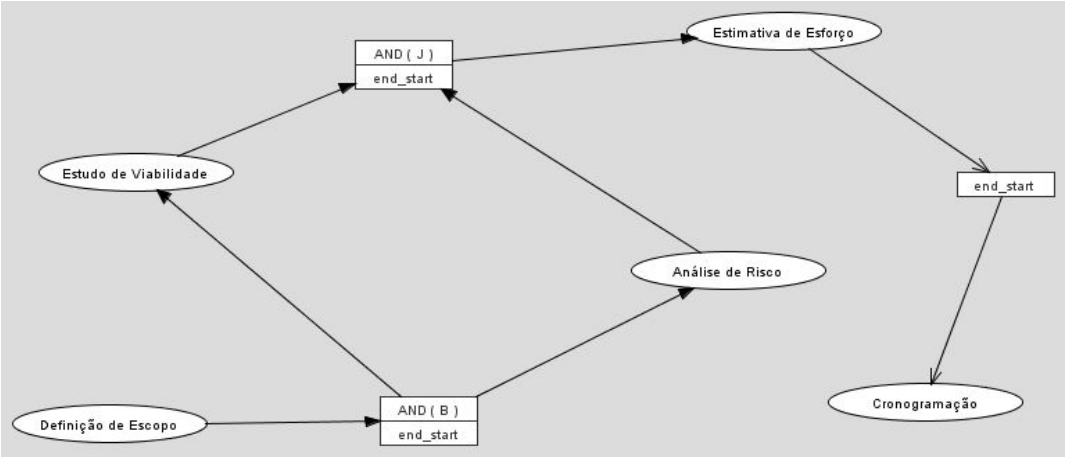


Figura 6.6: Processo padrão (fragmento Planejamento)

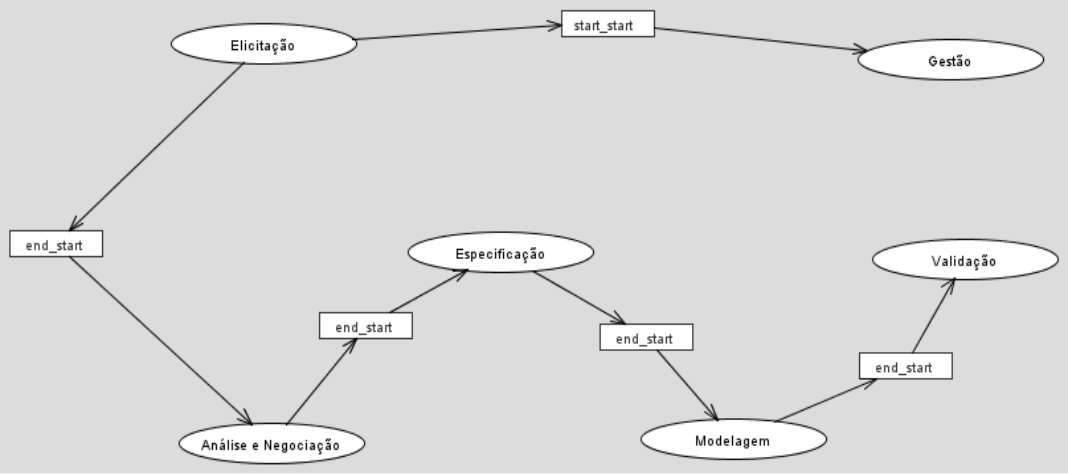


Figura 6.7: Processo padrão (fragmento Requisitos)

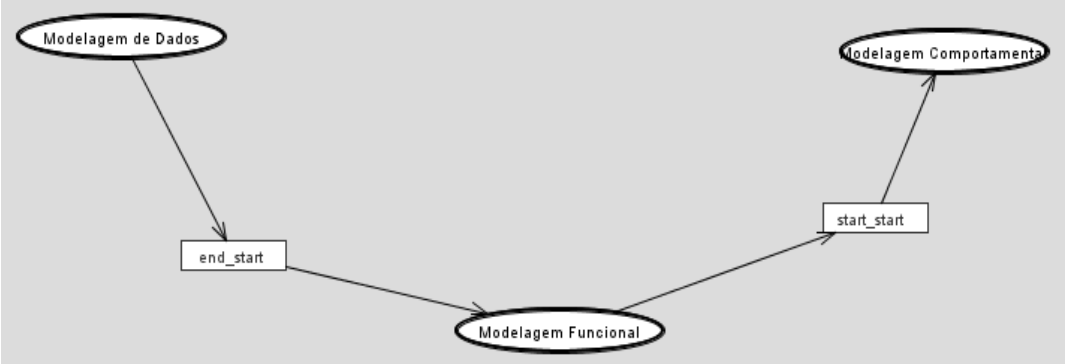


Figura 6.8: Processo padrão (fragmento Análise)

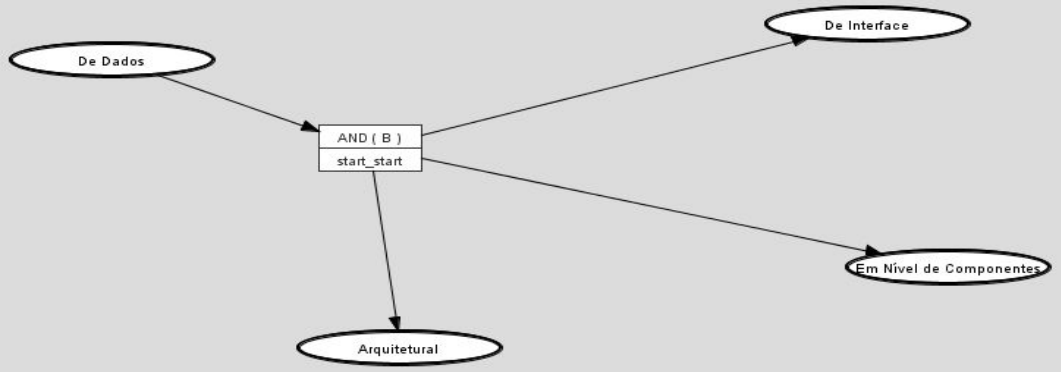


Figura 6.9: Processo padrão (fragmento Projeto)

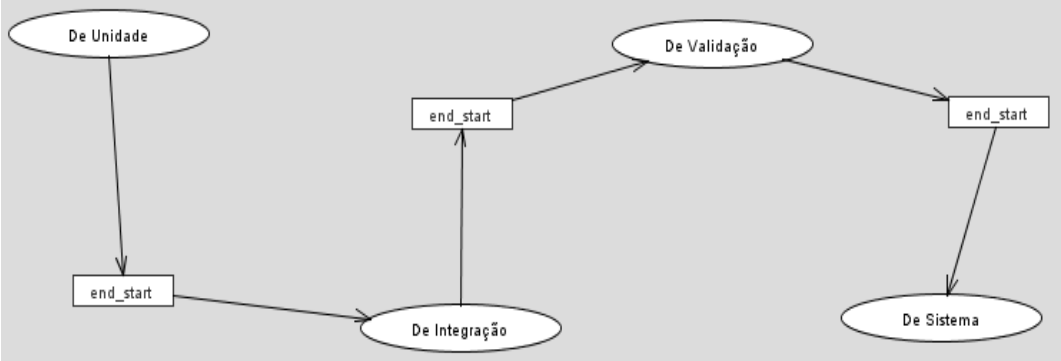


Figura 6.10: Processo padrão (fragmento Testes)

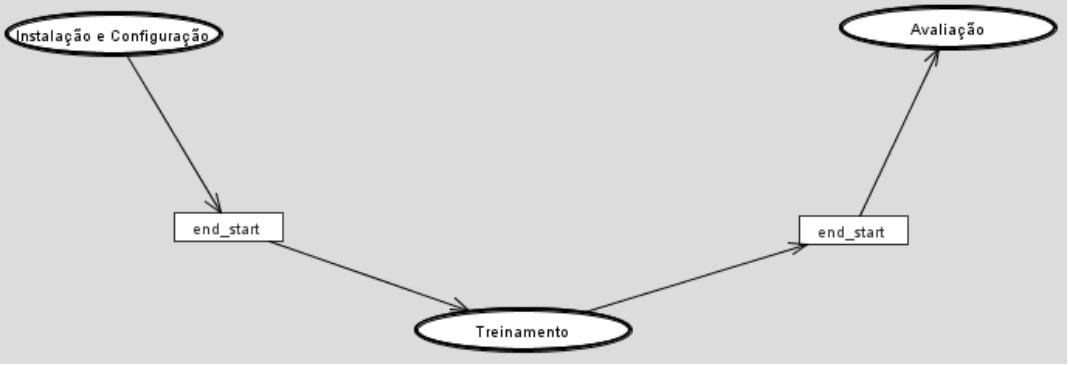


Figura 6.11: Processo padrão (fragmento Implantação)

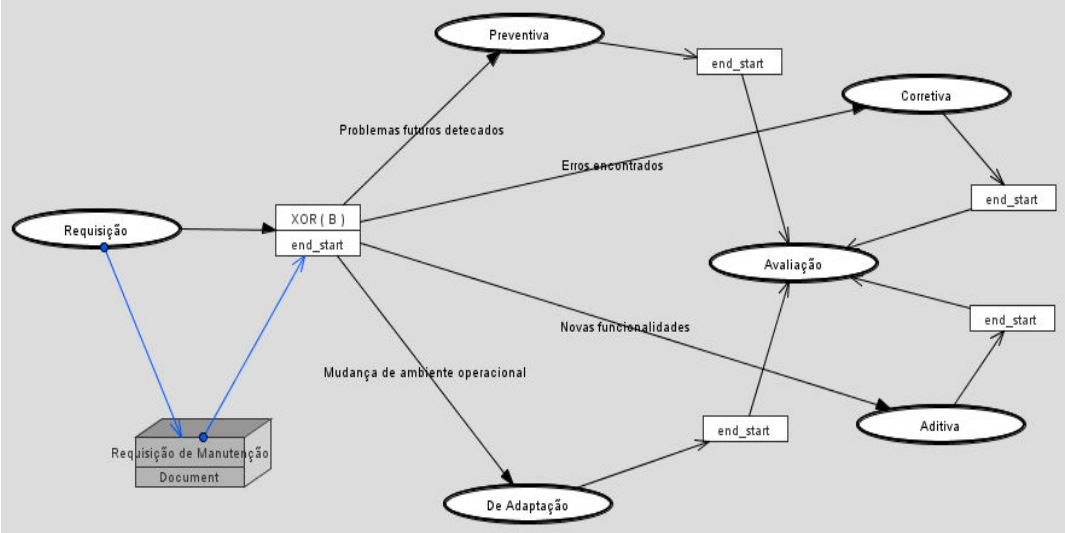


Figura 6.12: Processo padrão (fragmento Mantenção)

6.2.1.2 Atividades Guarda-Chuva

Conjunto vertical de atividades do processo de software, ou seja, aquelas aplicadas durante todo o projeto com o objetivo de garantir o gerenciamento dos diversos aspectos do mesmo. São elas: documentação (figura 6.13), medição (figura 6.14), gerência de configuração (figura 6.15), gerência de riscos (figura 6.16), gerência do projeto (figura 6.17) e garantia de qualidade (figura 6.18).

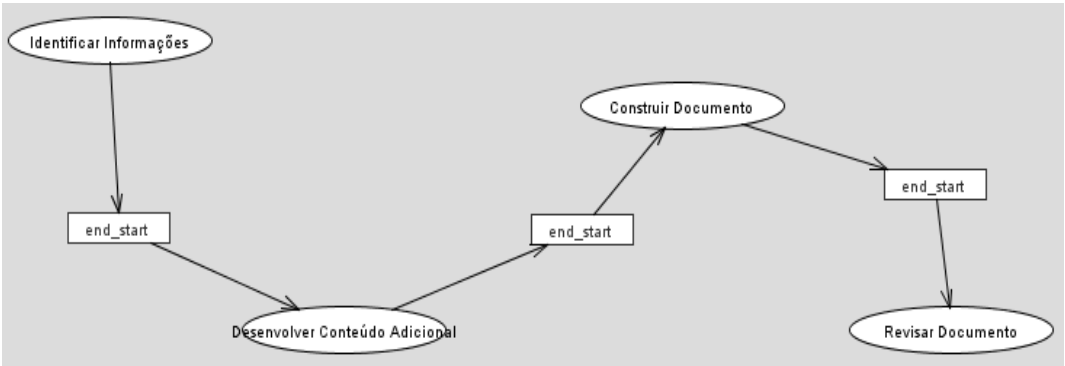


Figura 6.13: Processo padrão (fragmento Documentação)

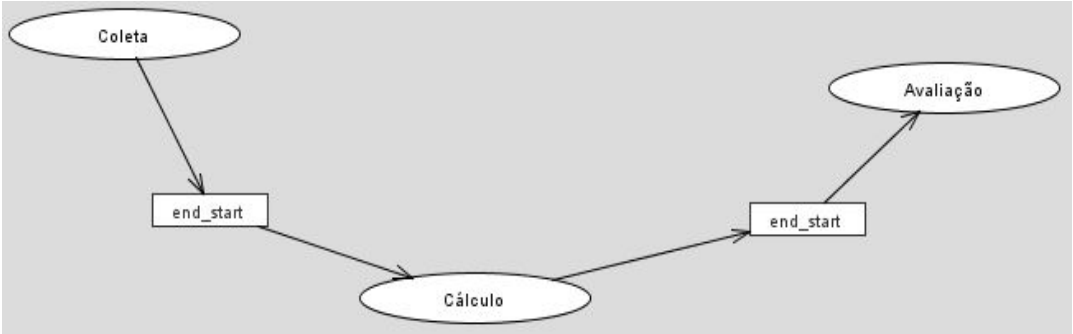


Figura 6.14: Processo padrão (fragmento Medição)

Figura 6.15: Processo padrão (fragmento Gestão de Configuração)

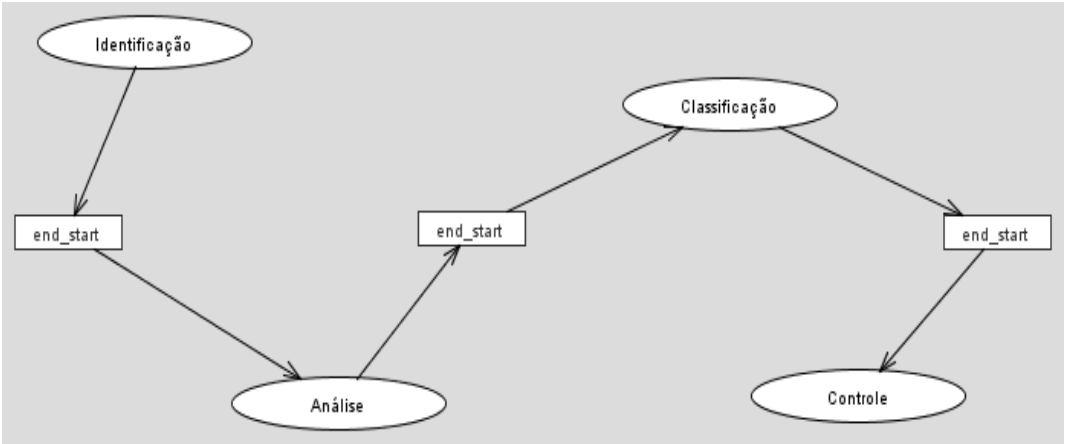


Figura 6.16: Processo padrão (fragmento Gestão de Riscos)

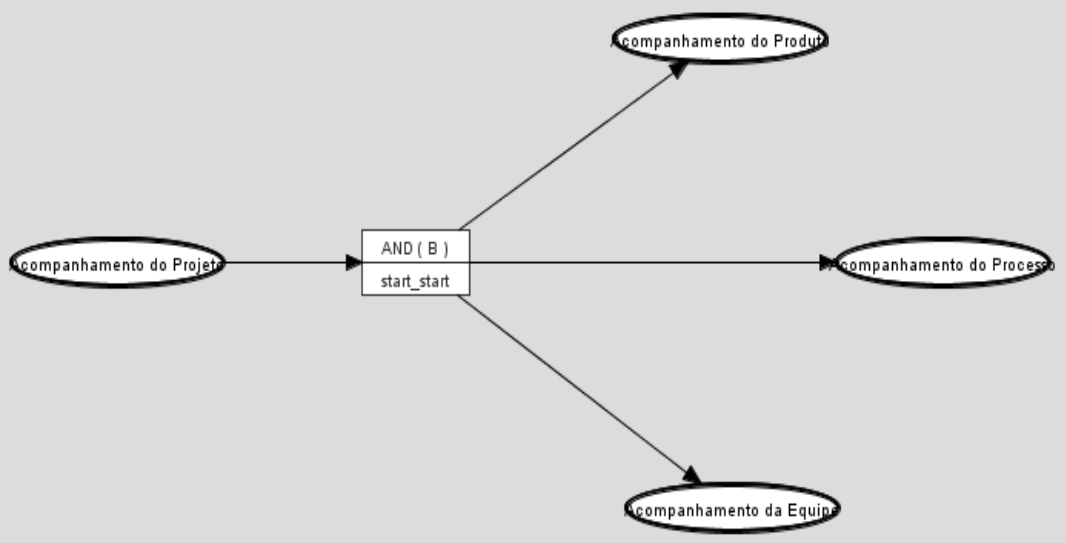


Figura 6.17: Processo padrão (fragmento Gerência do Projeto)

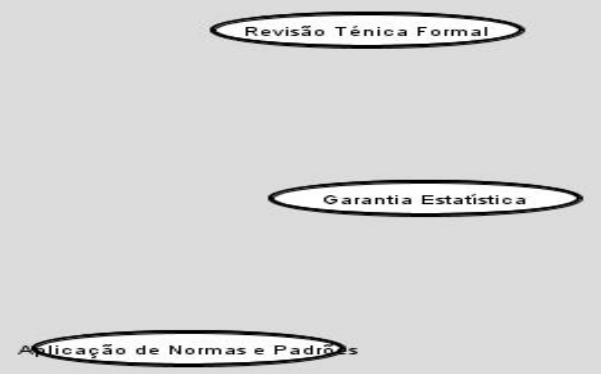


Figura 6.18: Processo padrão (fragmento Garantia de Qualidade)

6.2.2 Os Tipos de Característica

Pode-se dizer que, na EDSOFT, somente durante a fase de planejamento é que os projetos sofriam algum tipo de "caracterização", ainda que informalmente. Baseado na análise dos planos de projeto disponíveis e em trabalhos da literatura que abordam do assunto, (AHN et al, 2003), (BERGER, 2003) e (COELHO, 2003), o engenheiro de processos propôs à empresa um conjunto de vinte e quatro tipos de característica a ser utilizado na caracterização dos futuros projetos de software. Estes tipos de característica foram agrupados nas categorias: da organização, do projeto, do produto, do problema, dos usuários, da equipe, dos recursos, miscelânea. Esta classificação facilitaria a atribuição de pesos às características dentro dos projetos, embora fosse facultativa. Na figura 6.19 é mostrado o cadastro destes tipos de característica no APSEE-Tail, enquanto na tabela 6.1, são informados, para cada tipo de característica, sua classificação, espécie (1 para simples sem nivelamento, 2 para simples com nivelamento e 3 para múltipla), nome, descrição e possíveis valores.

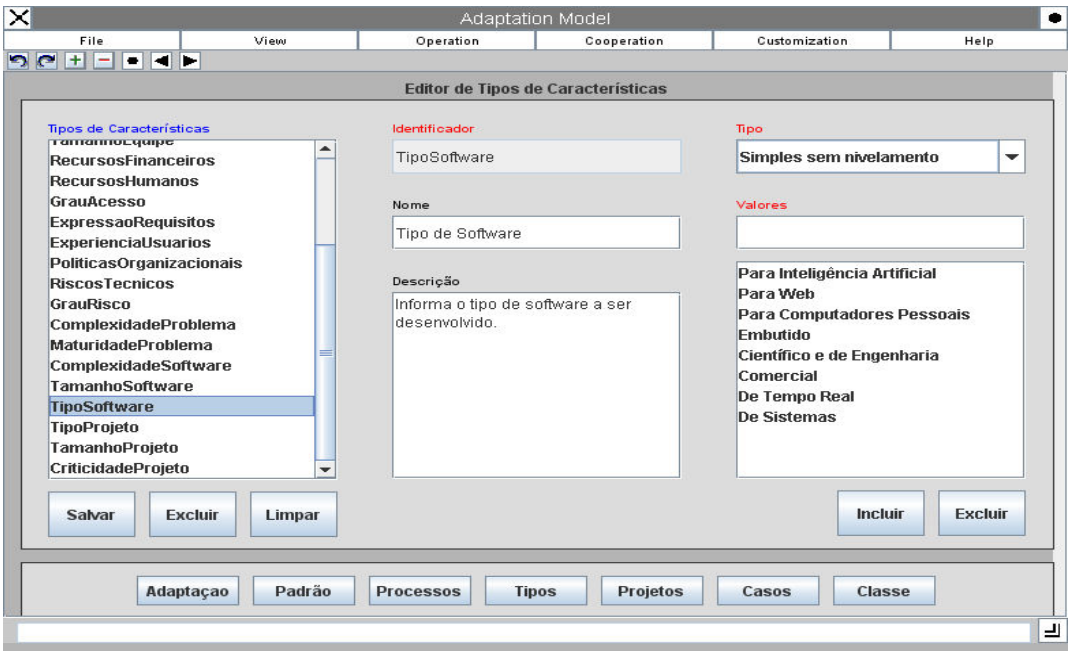


Figura 6.19: Tipos de característica cadastrados no APSEE-Tail

Tabela 6.1: Tipos de característica usados nos projetos da EDSOFT

Grupo	*Nome*	*Tipo*	*Descrição*	*Possíveis Valores*
Da Organização	Políticas Organizacionais	3	Políticas estabelecidas e seguidas pela organização	Conformidade com normas e padrões de qualidade, Terceirização, Gerência de riscos, Políticas de segurança
Do Projeto	Tipo do Projeto	1	Descreve a natureza do projeto	Configuração/Modificação de Pacote, Prototipação, Reuso, Reengenharia, Novo Desenvolvimento
	Tamanho do Projeto	2	Descreve o tamanho do projeto, em termos gerais	Pequeno, Médio, Grande
	Criticidade do Projeto	2	Define a criticidade de um projeto, em termos dos danos no caso de uma possível falha no sistema sendo desenvolvido	Perda de conforto; Prejuízos baixos, perdas facilmente recuperáveis; Prejuízos moderados, perdas recuperáveis, Prejuízos altos, perdas irrecuperáveis; Risco de vida
	Grau de Risco	2	Define o grau de risco do projeto	Baixo, Moderado, Alto
	Riscos Técnicos	3	Especifica os possíveis riscos técnicos do projeto	Tecnologia inovadora ou imatura, Interface com sistema externo ou legado, requisitos de tolerância a falhas, Requisitos de performance, Restrições de HW e SW, necessidade de repositório/ferramentas CASE
	Cronograma	2	Tipo de cronograma a ser seguido pelo projeto	Folgado, Adequado e Agressivo
Do Produto	Tipo de Software	1	Informa o tipo de software a ser desenvolvido	Para Inteligência Artificial, Para Web, Para Computadores Pessoais, Embutido, Científico e de Engenharia, Comercial, De Tempo Real, De Sistemas
	Tamanho do Software	2	Especifica o tamanho do software a ser construído	Pequeno, Médio, Grande
	Complexidade do Software	2	Define o grau de complexidade do software	Baixa, Moderada, Alta
Do Problema	Complexidade do Problema	2	Define o quão complexo de elucidar e entender é o problema	Baixa, Média e Alta
	Grau de Maturidade do Problema	2	Define o quão maduro está a área do problema	Baixa, Moderada e Alta

Dos Usuários	Grau de Acesso aos Usuários	2	Define o quão acessíveis são os usuários do sistema	Baixo, Médio e Alto
	Experiência dos Usuários no Domínio do Problema	2	Define o grau de experiência dos usuários no domínio do problema	Baixa, Média e Alta
	Facilidade em Expressar Requisitos	2	Facilidade dos usuários em expressar requisitos	Baixa, Média e Alta
Da Equipe	Tamanho da Equipe	2	Especifica o tamanho da equipe de desenvolvimento que participará do projeto	Muito Pequena (1-6 pessoas), Pequena (7-20 pessoas), Média (21-50 pessoas), Grande (51-100 pessoas), Muito Grande (+100 pessoas)
	Distribuição Geográfica da Equipe	2	Especifica a distribuição geográfica da equipe de desenvolvimento do projeto	Mesma sala; Mesmo prédio, salas diferentes; Mesma cidade, mesma empresa, prédios diferentes; Mesma cidade, empresas diferentes; cidades diferentes
	Experiência da Equipe no Processo	2	Especifica a experiência da equipe no processo de desenvolvimento do projeto	Baixa, Média, Alta
	Experiência da Equipe no Domínio	2	Especifica a experiência da equipe do projeto no domínio do problema	Baixa, Média, Alta
	Experiência da Equipe em Engenharia de Software	2	Especifica a experiência da equipe do projeto em engenharia de software	Baixa, Média, Alta
	Experiência da Gerência	2	Especifica a experiência da gerência do projeto	Baixa, Média, Alta
Dos Recursos	Recursos Financeiros	2	Especifica a disponibilidade de recursos financeiros para o projeto	Baixa, Adequada, Alta
	Recursos Humanos	2	Especifica a disponibilidade de recursos humanos	Baixa, Adequada, Alta
Miscelânea	Contrato de Manutenção	1	Especifica se o contratante requeriu contrato de manutenção	Sim, Não

6.2.3 As Diretrizes de Adaptação

Na EDSOFT, não havia critérios formais para a adaptação do processo padrão. Partindo do conjunto de características definido, o engenheiro de processos realizou um estudo histórico acerca da relação que havia entre as variantes, geradas informalmente a partir do processo padrão, e os projetos que as utilizaram. Tomou também por base, a literatura da área, em especial o trabalho de Coelho (2003). Entrevistou os gerentes de projeto da organização para tentar elicitar algumas regras que eram aplicadas

empiricamente por cada um. Ao final, uma reunião foi realizada com a equipe de melhoria contínua do processo padrão da organização, a fim de validar o conjunto inicial de regras. Essas regras foram então cadastradas no APSEE-Tail (conforme ilustrado na figura 6.20) e são apresentadas na tabela 6.2. Para cada regra, são apresentados: a atividade associada, o tipo (P para positiva e N para negativa), sua descrição e as condições associadas.

Algo interessante a ser ressaltado é que, de um modo geral, os fragmento normais sempre são obrigatórios para quase todos os projetos de uma organização, excetuando-se, talvez, algumas de suas subatividades. Ao contrário, a aplicação dos fragmentos guarda-chuva, quase sempre, é condicionada ao conjunto de tipos de característica. Também é bom destacar que a elicitação das diretrizes de adaptação é consideravelmente facilitada quando já existe um registro histórico dos projetos e seus processos.

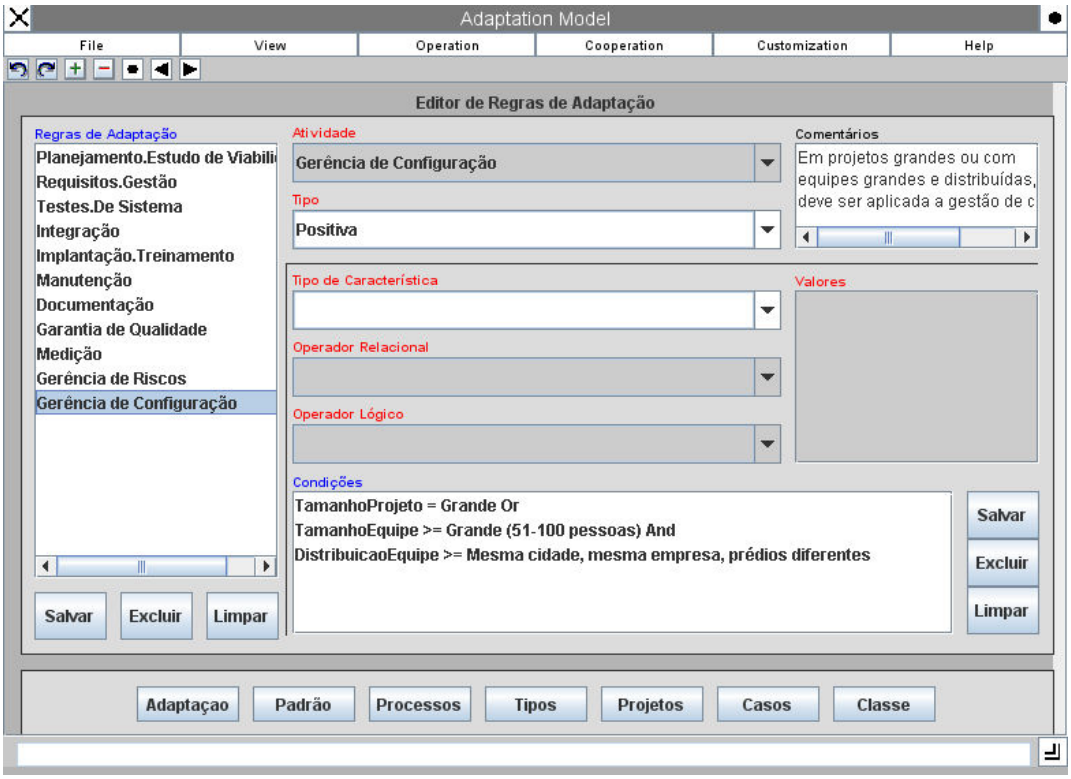


Figura 6.20: Regras cadastradas no APSEE-Tail

Tabela 6.2: Diretrizes de adaptação do processo padrão da EDSOFT

Atividade	*Tipo*	*Descrição*	*Condições*
Gerência de Configuração	P	Em projetos grandes ou com equipes grandes e distribuídas, deve ser aplicada a gestão de configuração	TamanhoProjeto = Grande Or TamanhoEquipe >= Grande (51-100 pessoas) And DistribuicaoEquipe >= Mesma cidade, mesma empresa, prédios diferentes
Gerência de Riscos	P	A gerência de riscos deve ser aplicada em projetos de médio porte em diante, com	TamanhoProjeto >= Médio Or CriticidadeProjeto >= Prejuízos moderados, perdas recuperáveis Or TamanhoEquipe >=

		equipes médias, distribuídas ou inexperientes	Média (21-50 pessoas) Or ExperienciaEquipeProcesso = Baixa Or DistribuicaoEquipe >= Mesma cidade, mesma empresa, prédios diferentes
Garantia de Qualidade	P	A garantia de qualidade deve ser aplicada em projetos grandes ou críticos, com equipes grandes, distribuídas ou inexperientes	TamanhoProjeto = Grande Or CriticidadeProjeto >= Prejuízos altos, perdas irrecuperáveis Or ExperienciaEquipeProcesso = Baixa Or TamanhoEquipe >= Grande (51-100 pessoas) Or DistribuicaoEquipe >= Mesma cidade, empresas diferentes Or PoliticasOrganizacionais contains {Terceirização, Conformidade com Normas e Padrões de Qualidade}
Medição	P	Em projetos grandes ou críticos, desde que haja mão de obra p/ tal, medições devem ser realizadas	RecursosHumanos >= Adequada And TamanhoProjeto = Grande Or CriticidadeProjeto >= Prejuízos altos, perdas irrecuperáveis
Documentação	P	Toda uma documentação transversal ao projeto deve ser produzida, quando este for maior que médio, com cronograma pelo menos adequado e houverem recursos humanos disponíveis	TamanhoProjeto >= Médio And Cronograma => Adequado And RecursosHumanos >= Adequada
Manutenção	P	A manutenção de um sistema deve ser fornecida, somente se um contrato tiver sido firmado	AditivosContrato contains {Manutenção}
Integração	P	Sempre que o software sendo implementado tiver interface com outros sistemas, a integração deve ser realizada	RiscosTecnicos contains {Interface com sistema externo ou legado}
Implantação (Treinamento)	P	O treinamento dos usuários deve ser fornecido, somente se um contrato tiver sido firmado.	AditivosContrato contains {Treinamento}
Planejamento (Estudo de Viabilidade)	P	O Estudo de viabilidade deve ser realizado somente em projeto de porte médio ou grande	TamanhoProjeto >= Médio
Requisitos (Gestão)	P	Sempre que o grau de acesso aos usuários do sistema for baixo ou a facilidade dos mesmos em expressar requisitos for baixa ou a experiência dos mesmos no domínio do problema for baixa, ou a complexidade do problema for alta, ou sua maturidade for baixa, deve ser aplicada uma rigorosa gestão de requisitos	GrauAcesso = Baixa Or ExpressaoRequisitos = Baixa Or ExperienciaUsuarios = Baixa Or ComplexidadeProblema = Alta Or MaturidadeProblema = Baixa

Testes (Sistema)	N	Desde que o cronograma do projeto não seja agressivo, devem ser realizados testes de sistema	Cronograma = Agressivo

6.2.4 O Projeto de Software A

O Projeto A trata da construção de uma nova aplicação Web, relativamente pequena e simples, que irá rodar na intranet da própria organização, para o gerenciamento do cadastro de seus clientes e contratos. Desta forma, a empresa detém o conhecimento claro e preciso do domínio do problema. O projeto é, portanto, de pequeno porte, com uma equipe bastante enxuta e concentrada fisicamente na mesma sala. O detalhe é que o projeto será implementado em Java, de ponta a ponta, com o SGBD MySQL e todos os seus relatórios serão gerados em XML. Haverá, portanto, a necessidade de treinamento da equipe. Na tabela 6.3, o projeto A é caracterizado através dos tipos de característica definidos anteriormente. Já na figura 6.21, seu cadastro no APSEE-Tail é apresentado.

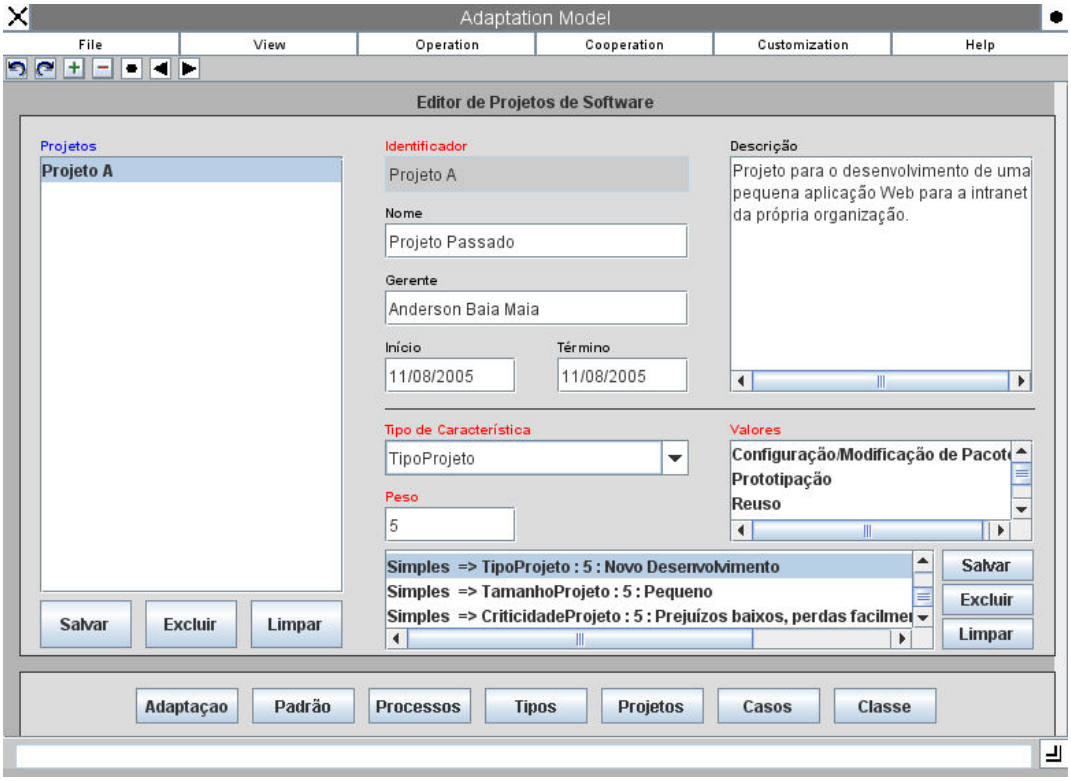


Figura 6.21: Cadastro do projeto A no APSEE-Tail

Tabela 6.3: Características do projeto A

Característica	*Valor*	*Peso*
Criticidade do Projeto	Prejuízos baixos, perdas facilmente recuperáveis	5
Tamanho do Projeto	Pequeno	5
Tipo do Projeto	Novo Desenvolvimento	5
Grau de Risco	Baixo	5

Cronograma do Projeto	Agressivo	5
Tipo de Software	Para Web	5
Tamanho do Software	Pequeno	5
Complexidade do Software	Baixa	5
Políticas Organizacionais	Conformidade com normas e padrões de qualidade	5
Maturidade do Problema	Alta	4
Complexidade do Problema	Baixa	4
Experiência dos usuários no domínio do problema	Alta	4
Facilidade dos usuários em expressar requisitos	Alta	4
Grau de acesso aos usuários	Alto	4
Tamanho da Equipe	Muito Pequena (1-6 pessoas)	3
Distribuição da Equipe	Mesma sala	3
Experiência da equipe no processo de software	Média	3
Experiência da equipe no domínio do problema	Baixa	3
Experiência da equipe em engenharia de software	Média	3
Experiência da gerência	Média	3
Recursos Humanos	Adequado	2
Recursos Financeiros	Adequado	2

6.2.5 Adaptação do Processo Padrão para o Projeto A

Conforme apresentado na figura 6.22, foram especificados, no Adaptador de Processos, o Projeto A, um grau de similaridade de 80% (que neste caso é indiferente, já que o repositório de casos de adaptação está vazio) e o nome do processo adaptado a ser gerado (Processo Adaptado 1).

Na figura 6.23, é apresentado o Editor de Atividades, que aparece após o botão "Adaptar", do Adaptador de Processos, ter sido pressionado. Nele, são apresentados o processo padrão e o processo adaptado gerado até então. O engenheiro de processos tem a possibilidade de visualizar os dados de uma atividade, incluir uma atividade do processo padrão para o adaptado ou excluí-la do processo adaptado. Percebe-se que, em virtude das diretrizes de adaptação do processo padrão e das características do projeto, as atividades "Integração", "Manutenção", "Documentação", "Garantia de Qualidade", "Medição", "Gerência de Riscos" e "Gerência de Configuração" não foram incluídas no processo adaptado. No fragmento "Planejamento", a subatividade "Estudo de Viabilidade" também não foi inserida. No fragmento "Requisitos", a subatividade "Gestão" não foi inserida. No fragmento "Testes", a subatividade "Teste de Sistema" não foi incluída. Por fim, no fragmento "Implantação", a subatividade "Treinamento" não foi incluída.

Alguns ajustes no processo adaptado foram realizados, através do Editor de Atividades: o fragmento “Garantia de Qualidade” e sua subatividade “Aplicação de Normas e Padrões” foram incluídos, pois a alta gerência exige um mínimo de qualidade nos sistemas produzidos para automatizar a sua gestão; no fragmento “Requisitos”, a subatividade “Análise e Negociação” foi retirada, pois devido ser um sistema importante para a organização, todos os seus requisitos terão de ser atendidos.

Após os ajustes manuais necessários no processo adaptado, a adaptação foi então finalizada (botão “Finalizar”, do Editor de Atividades). Como conseqüência, um caso de adaptação foi gerado e automaticamente aberto no Editor de Casos, conforme mostra a figura 6.24. Alguns comentários foram deixados pelo engenheiro de processos, porém a avaliação ainda não foi preenchida já que o processo não foi aplicado no projeto de software real. As modificações realizadas através do Editor de Atividades podem ser visualizadas na grade “Modificações”, bem como o projeto de software (no modo de leitura, através do botão “Visualizar Projeto”) e o processo adaptado (através do botão “Visualizar Processo Adaptado”). Outra possibilidade bem interessante é a visualização do *trace* da adaptação através do botão “Mostrar Histórico”, ou seja, dos passos realizados pelo mecanismo de adaptação até a geração do caso de adaptação, conforme mostrado na figura 6.25.

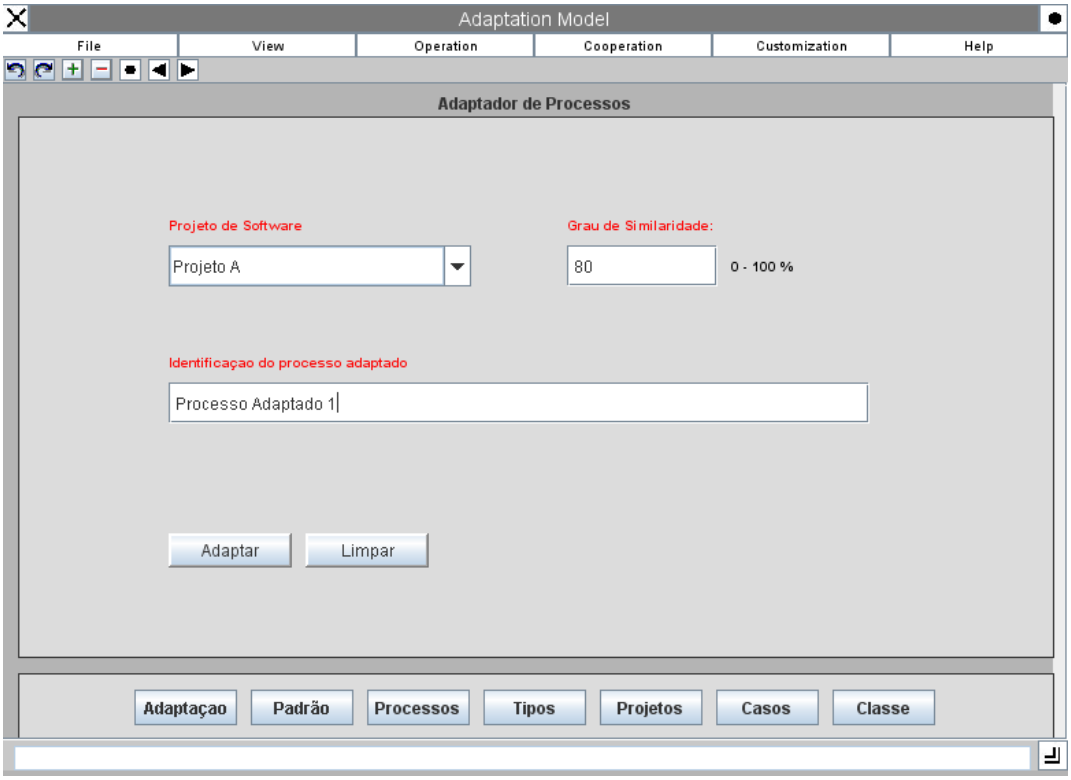


Figura 6.22: Especificação dos parâmetros gerais da adaptação

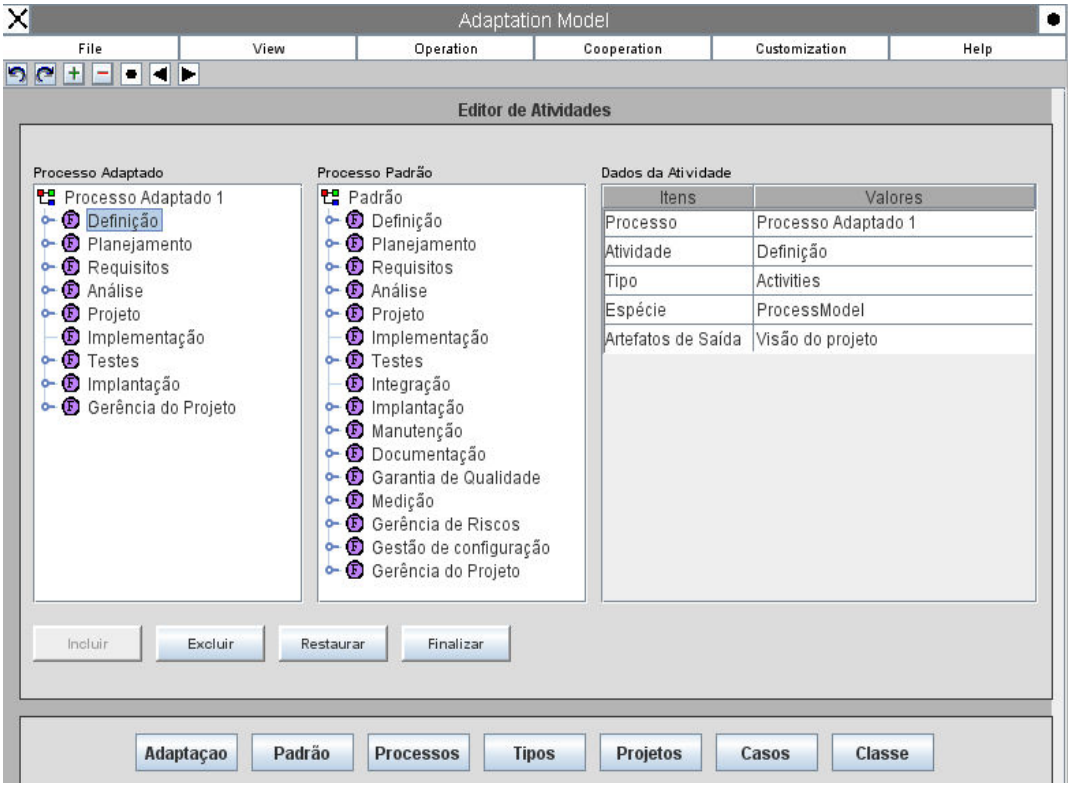


Figura 6.23: Realização de ajustes manuais no processo adaptado

Figura 6.24: Caso de adaptação gerado

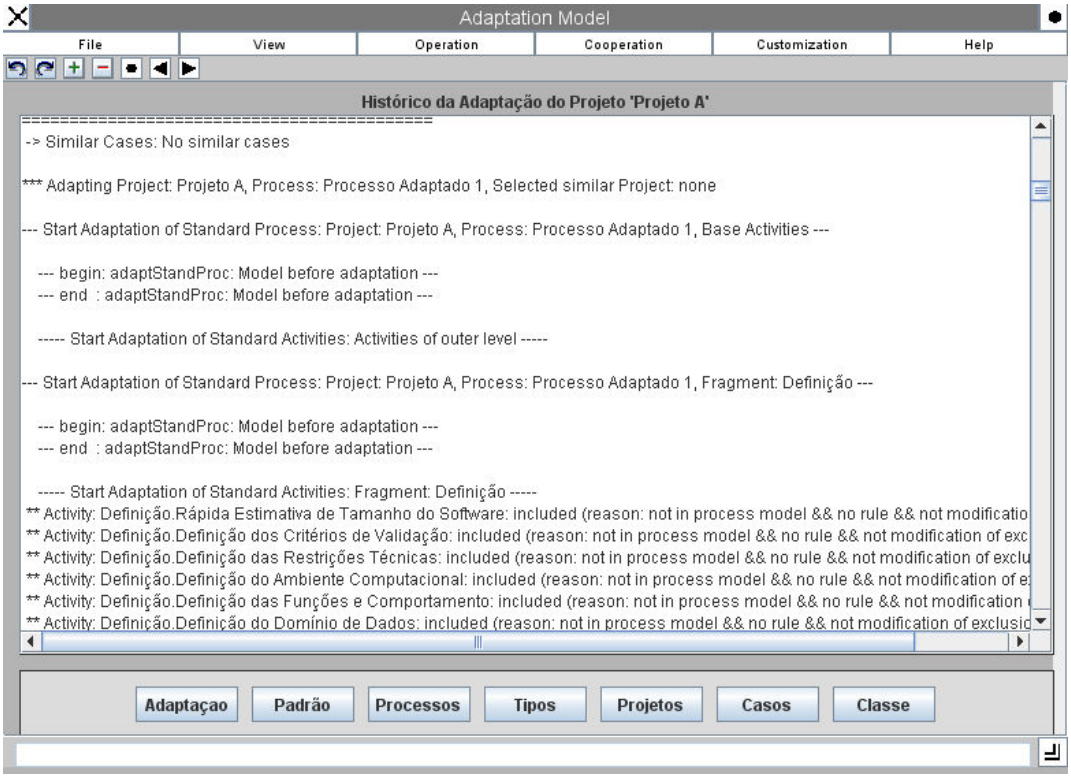


Figura 6.25: Visualização dos passos realizados pelo mecanismo de adaptação

6.2.6 Modificações Realizadas Externamente no Processo Adaptado

Conforme dito anteriormente, na descrição do projeto A, haveria a necessidade de treinamento da equipe do projeto, em virtude das tecnologias a serem utilizadas. Este treinamento, segundo o gerente do projeto, deveria ocorrer após o Planejamento e ser finalizado antes do início da fase de Análise, devendo ocorrer em paralelo com a Elicitação dos Requisitos. Desta maneira, identificou-se a necessidade de acrescentar ao processo adaptado uma atividade que não pertence ao processo padrão. Logo, isso não poderia ser realizado através do Editor de Atividades e sim utilizando a ferramenta de modelagem do ambiente APSEE.

Na figura 6.26, é apresentado o Editor de Processos Adaptados. Através dele, o engenheiro de processos selecionou o processo adaptado que havia sido gerado e a instância do ambiente APSEE para a qual seria exportado tal processo. Clicou no botão "Exportar" e confirmou a ação. Note que também através deste editor é possível visualizar os dados de uma atividade.

Na figura 6.27, é apresentado, através do editor de *templates* do ambiente APSEE, o modelo do processo adaptado antes da alteração externa. É interessante notar algumas modificações sintáticas que foram realizadas no modelo de processo para garantir sua coerência: todas as conexões simples que possuíam uma das atividades excluídas como origem ou destino, também foram excluídas; todas as conexões de artefato que perderam suas atividades produtoras também foram excluídas; o fragmento Gerência de Qualidade voltou a ser destino da conexão múltipla a qual pertencia; por fim, arcos sem vértice foram excluídos também.

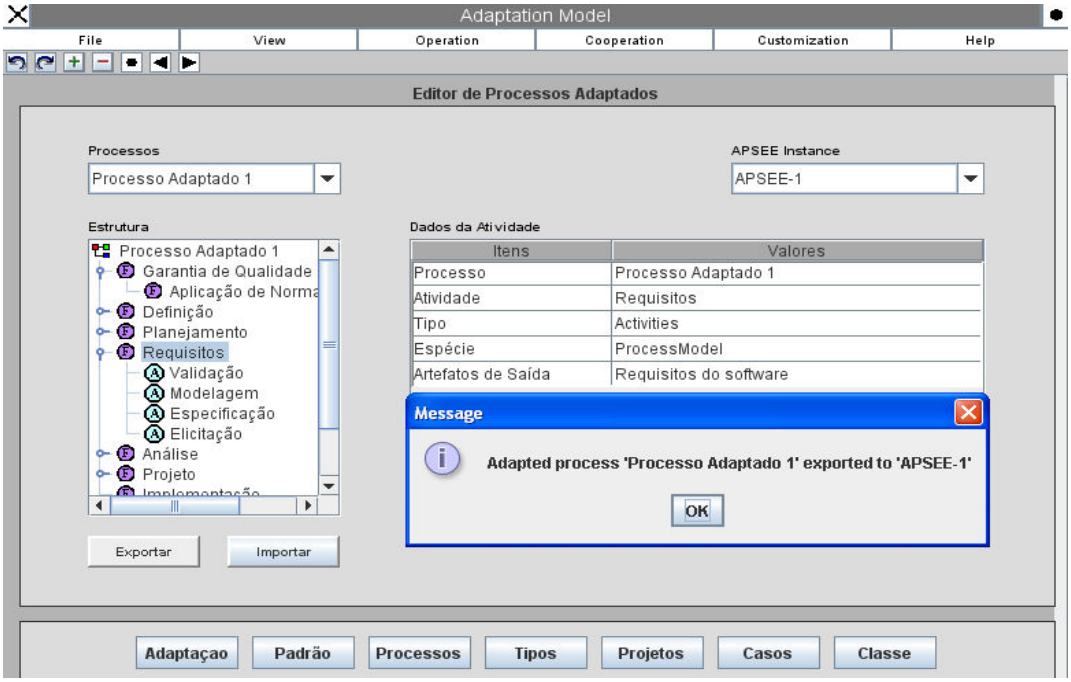


Figura 6.26: Exportação do processo adaptado para o ambiente APSEE

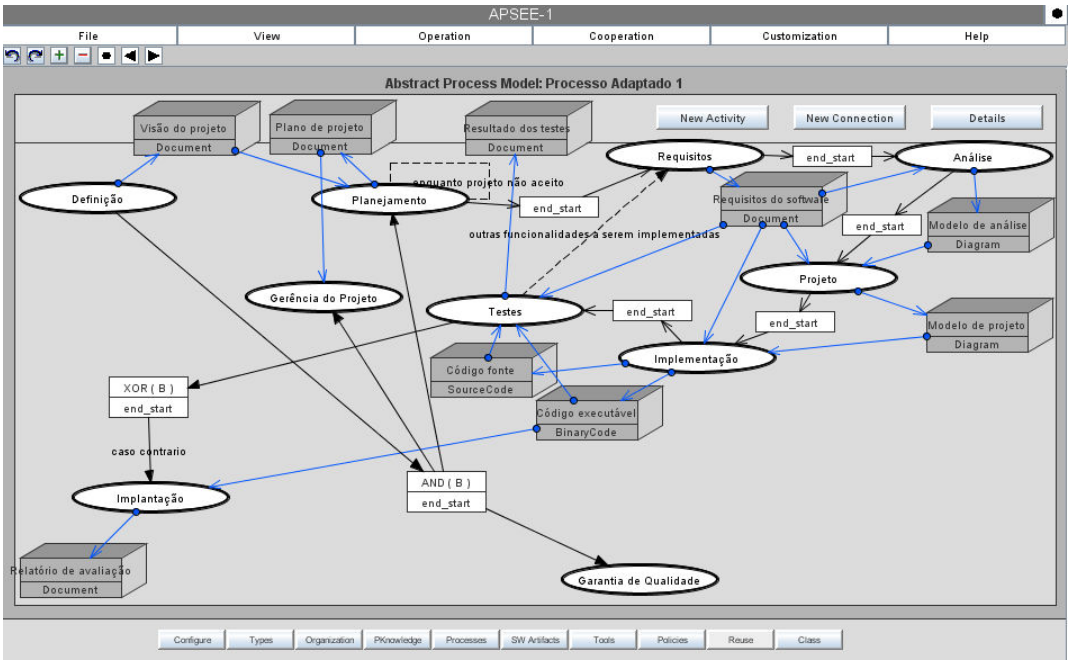


Figura 6.27: Modelo do processo adaptado antes da alteração externa

Já a figura 6.28 apresenta o modelo do processo adaptado após a alteração externa e reajuste de layout. Note que os fragmentos Treinamente e Requisitos tiveram de ser sincronizados através de duas conexões múltiplas. Após a alteração externa, faz-se necessário importar o processo adaptado atualizado, de modo a manter APSEE e APSEE-Tail sincronizados. Esta importação também é realizada pelo Editor de Processos Adaptados: seleciona-se o processo adaptado a ser importado, a instância do APSEE que contém a versão atualizada deste processo e clica-se no botão “Importar”. A

figura 6.29 apresenta o processo adaptado já atualizado externamente e importado para o APSEE-Tail. Note que, ao fragmento Treinamento foram inseridas as subatividades: contratar consultoria, especificar local e horário, realizar treinamento e avaliação.

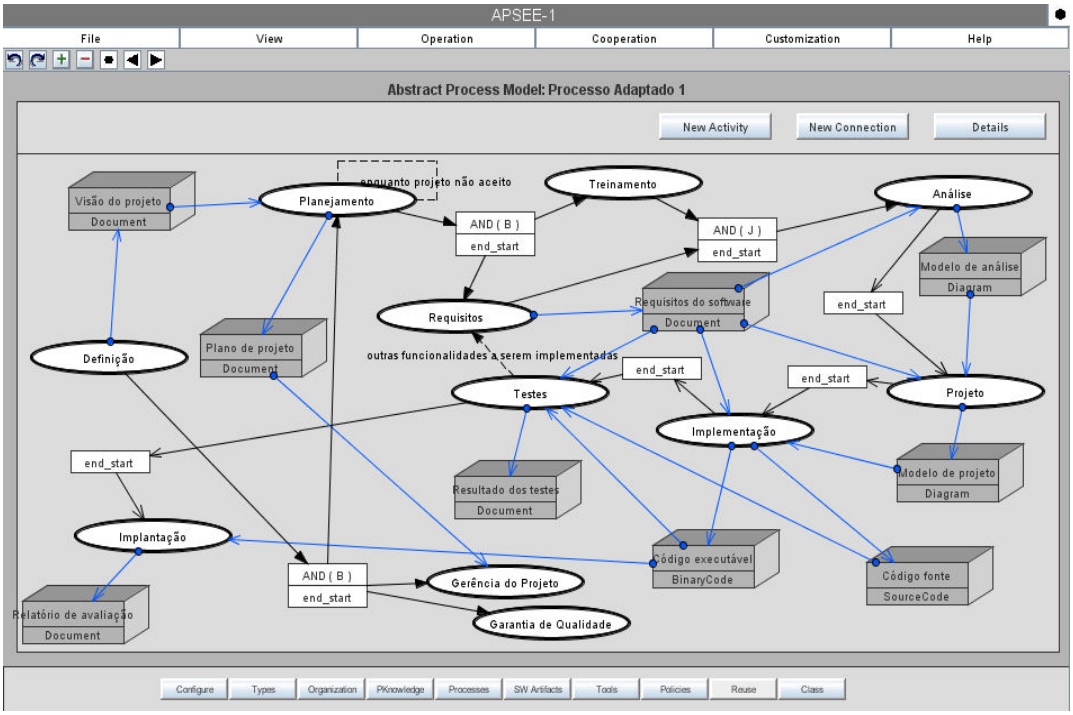


Figura 6.28: Modelo do processo adaptado alterado externamente

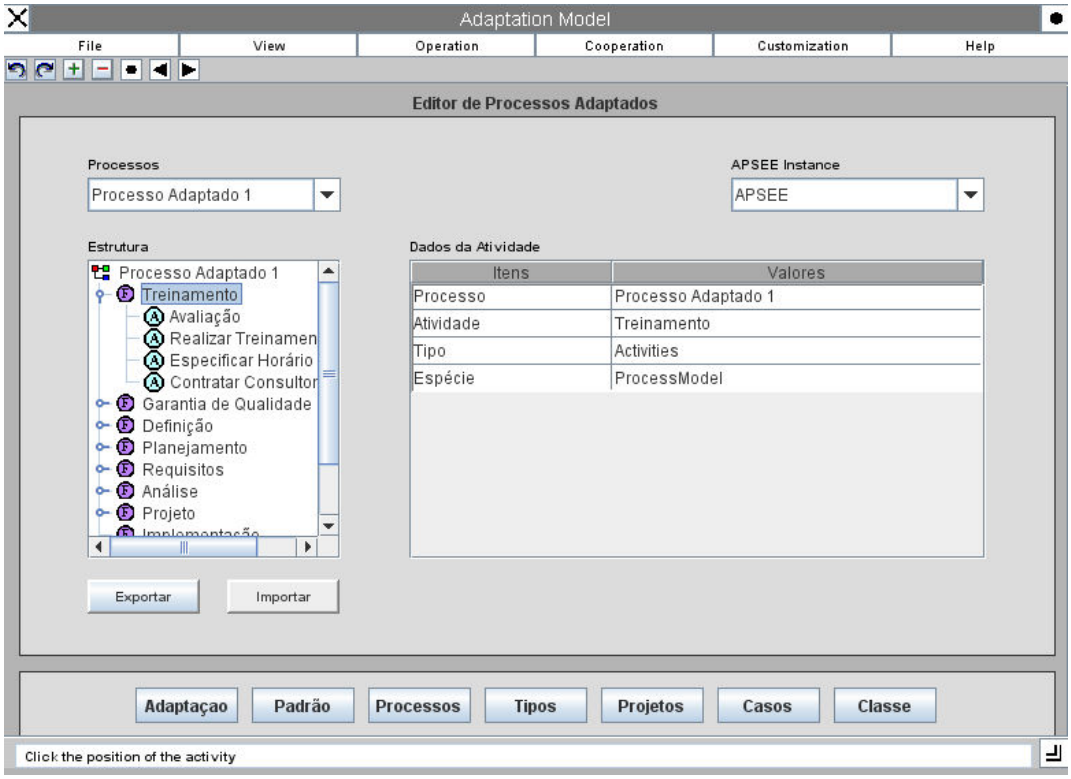


Figura 6.29: Importação do processo adaptado alterado no ambiente APSEE

6.2.7 O Projeto de Software B

O projeto B trata da reengenharia de uma aplicação cliente/servidor para a área financeira de uma empresa. Embora o domínio do problema não seja simples, o software a ser construído pode ser considerado pequeno, conseqüentemente o projeto também o é. Uma parte da equipe de desenvolvimento residirá nas instalações da empresa, a fim de observar o sistema atual em funcionamento e agilizar a prospecção de requisitos. O sistema fará interface com sistemas legados e terá requisitos de performance e tolerância à falhas. Na tabela 6.4, são apresentadas as características do projeto B, enquanto o seu cadastro no APSEE-Tail é apresentado na figura 6.30.

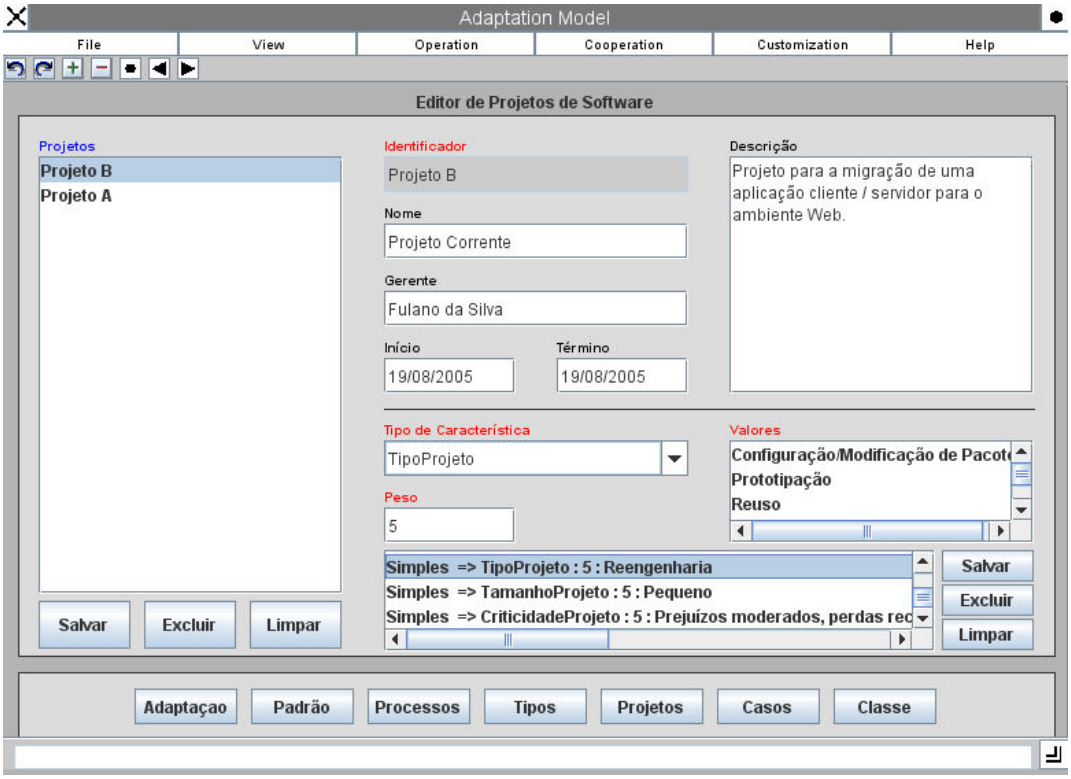


Figura 6.30: Cadastro do projeto B no APSEE-Tail

Tabela 6.4: Características do projeto B

Característica	*Valor*	*Peso*
Criticidade do Projeto	Prejuízos moderados, perdas recuperáveis	5
Tamanho do Projeto	Pequeno	5
Tipo do Projeto	Reengenharia	5
Grau de Risco	Moderado	5
Riscos Técnicos	{ Interface com sistema externo ou legado, Requisitos de tolerância à falhas, Requisitos de performance, Restrições de HW/SW }	5
Cronograma do Projeto	Agressivo	5
Tipo de Software	Para Web	5

Tamanho do Software	Pequeno	5
Complexidade do Software	Média	5
Políticas Organizacionais	{ Conformidade com normas e padrões de qualidade }	5
Maturidade do Problema	Alta	4
Complexidade do Problema	Média	4
Experiência dos usuários no domínio do problema	Alta	4
Facilidade dos usuários em expressar requisitos	Alta	4
Grau de acesso aos usuários	Alto	4
Tamanho da Equipe	Pequena (7-20 pessoas)	3
Distribuição da Equipe	Mesma cidade, mesma empresa, prédios diferentes	3
Experiência da equipe no processo de software	Média	3
Experiência da equipe no domínio do problema	Baixa	3
Experiência da equipe em engenharia de software	Alta	3
Experiência da gerência	Alta	3
Recursos Humanos	Adequado	2
Recursos Financeiros	Adequado	2
Aditivos no Contrato do Projeto	{ Manutenção, Treinamento }	1

6.2.8 Adaptação do Processo Padrão para o Projeto B

Desta vez, conforme ilustrado na figura 6.31, o engenheiro de processos especificou um grau de similaridade de 70%, de modo a aumentar a possibilidade de reuso de boas práticas em projetos anteriores. Ao se iniciar o processo de adaptação, foi detectado que o Projeto A é 76% semelhante ao projeto B (o cálculo da similaridade é apresentado na tabela 6.5), o que permitiu o reuso do processo recuperado. Note que, como o fragmento Garantia de Qualidade havia sido inserido no processo recuperado por decisão do próprio engenheiro e não por inferência do APSEE-Tail, o engenheiro é questionado se deseja mantê-lo no processo adaptado atual.

Tabela 6.5: Comparação entre os projetos A e B

Característica	*Peso no Projeto A*	*Similaridade de A e B*
Criticidade do Projeto	5	0,8
Tamanho do Projeto	5	1
Tipo do Projeto	5	0
Grau de Risco	5	0,67
Riscos Técnicos	5	0

Cronograma do Projeto	5	1
Tipo de Software	5	1
Tamanho do Software	5	1
Complexidade do Software	5	0,67
Políticas Organizacionais	5	1
Maturidade do Problema	4	1
Complexidade do Problema	4	0,67
Experiência dos usuários no domínio do problema	4	1
Facilidade dos usuários em expressar requisitos	4	1
Grau de acesso aos usuários	4	1
Tamanho da Equipe	3	0,8
Distribuição da Equipe	3	0,6
Experiência da equipe no processo de software	3	1
Experiência da equipe no domínio do problema	3	1
Experiência da equipe em engenharia de software	3	0,67
Experiência da gerência	3	0,67
Recursos Humanos	2	1
Recursos Financeiros	2	1
Aditivos no Contrato do Projeto	1	0

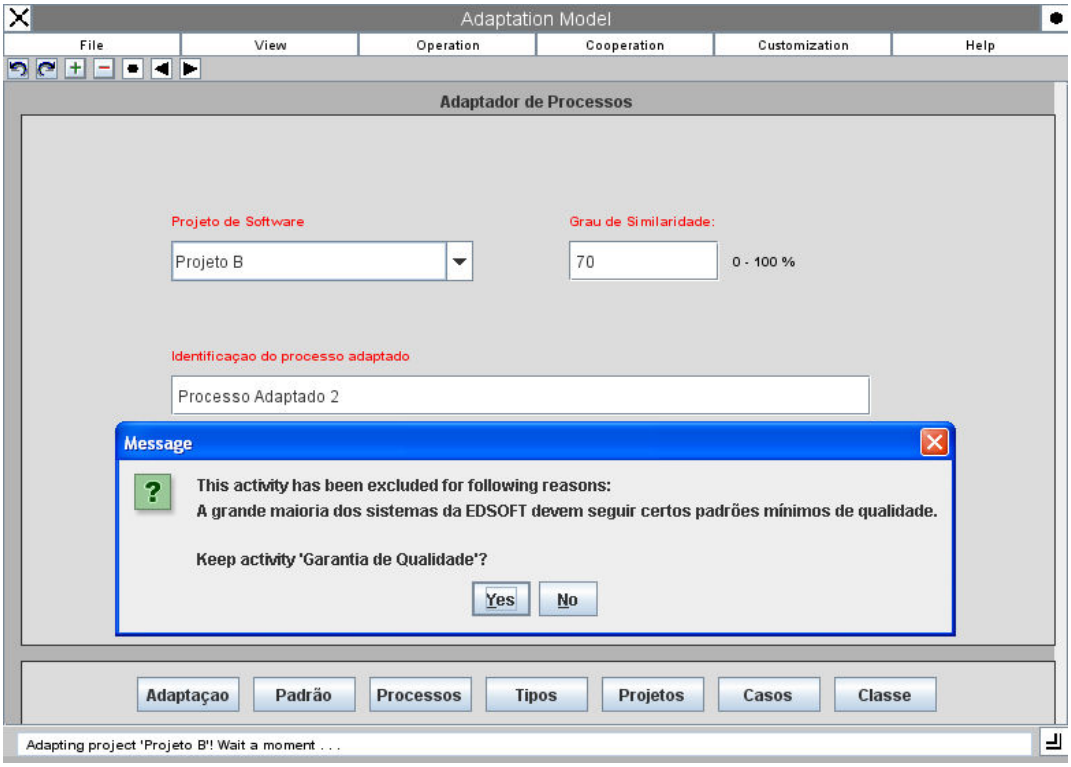


Figura 6.31: Especificação dos parâmetros gerais da adaptação para o Projeto B

Após o merge do processo recuperado com o processo padrão para formar a estrutura do processo adaptado, algumas mudanças ainda foram necessárias através do Editor de Atividades: o fragmento Treinamento já não fazia mais sentido, uma vez que boa parte dos membros da equipe do projeto B fez parte do projeto A e, portanto, já haviam sido treinados na tecnologia a ser utilizada.

Como pode ser visto na figura 6.32, os fragmentos Definição, Análise, Projeto, Implementação, Integração, Implantação, Manutenção, Gestão de Riscos e Gerência de Projeto foram mantidos sem nenhuma alteração. No fragmento Planejamento, a subatividade Estudo de viabilidade não foi incluída. No fragmento Requisitos, a subatividade de Gestão também não foi incluída. No fragmento Testes, a subatividade Teste de Sistema ficou de fora. Por fim, assim como aconteceu no Projeto A, apenas a subatividade Aplicação de Normas e Padrões foi mantida no fragmento Garantia de Qualidade. É interessante notar que pouquíssima alteração foi realizada através do Editor de Atividades, sendo que o processo adaptado não foi alterado externamente.

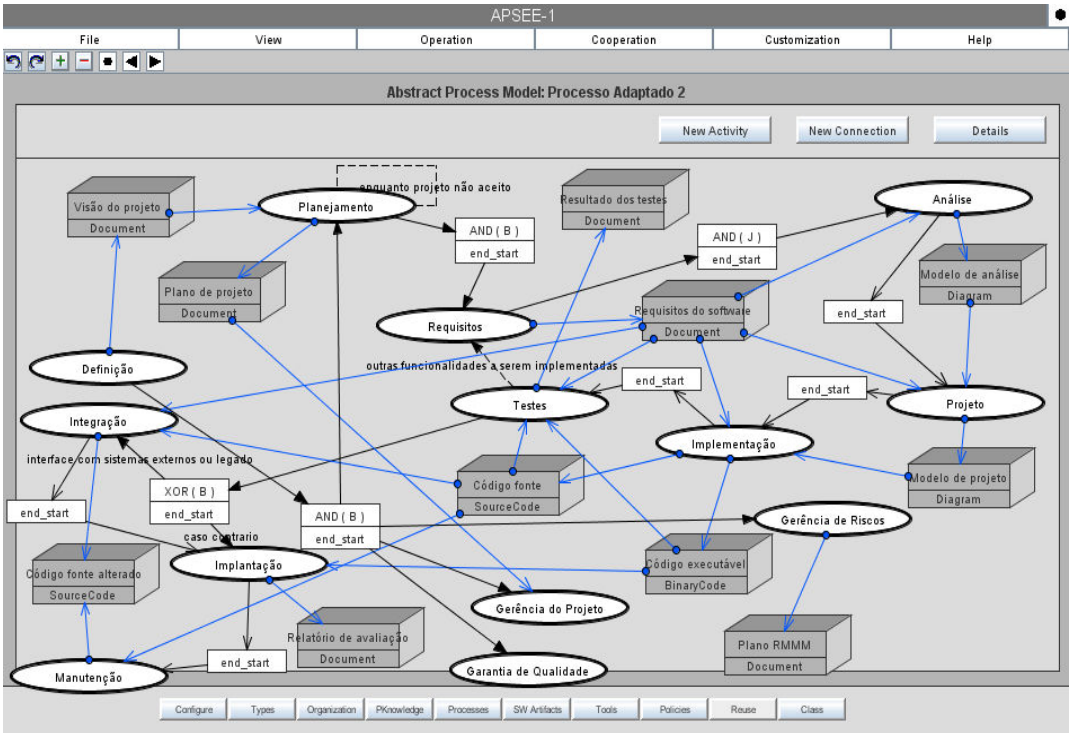


Figura 6.32: Processo adaptado gerado para o projeto B

6.3 Considerações

Neste capítulo, foi apresentado um exemplo de aplicação do APSEE-Tail, utilizando-se o protótipo. Apresentou-se a abordagem utilizada para a elaboração do cenário, bem como sua aplicação no protótipo e os resultados obtidos. Embora o APSEE-Tail tenha se comportado de acodo com o que foi especificado, somente a sua aplicação em projetos de software reais pode evidenciar sua aplicabilidade. Ainda assim, a elaboração do protótipo é um primeiro passo no caminho da experimentação prática. No capítulo seguinte, são encontradas as conclusões deste trabalho.

7 CONCLUSÃO

A pesquisa descrita neste texto resultou no desenvolvimento de um modelo de apoio à adaptação de processos de software organizacionais para o contexto de projetos individuais. Assim, embora o trabalho aqui apresentado não forneça uma solução completa para o problema investigado – e nem era a intenção do mesmo, em virtude da complexidade do tema – acredita-se que o modelo proposto possa agregar valor à área, através da base semântica gerada com a sua especificação formal e da construção de um protótipo que possui interface com um ambiente de execução, estando assim inserido no meta-processo de software como um todo.

Este capítulo contém as conclusões do trabalho e apresenta a estrutura abaixo:

- Na seção 7.1, o trabalho é comparado com as abordagens correlatas;
- Na seção 7.2, são apresentadas as contribuições do trabalho;
- Na seção 7.3, são apresentadas as limitações do trabalho;
- Na seção 7.4, são apresentados os trabalhos futuros;
- Por fim, na seção 7.5, são feitas as considerações finais da dissertação.

7.1 Trabalhos Relacionados

Existem inúmeros trabalhos que tratam sobre adaptação de processos de software, cada um com suas peculiaridades e contribuições. A seguir, serão brevemente descritos os trabalhos que mais influenciaram na definição do APSEE-Tail, enfatizando-se as semelhanças e diferenças.

Ahn et al (2003) apresentam um modelo de adaptação baseada nas características do projeto de software, na experiência adquirida em adaptações anteriores (armazenada na forma de casos, como no APSEE-Tail) 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 APSEE-Tail, também é orientada à atividades. Um projeto de software é caracterizado através de fatores de domínio (idêntico ao conceito de tipos de característica), entretanto, existe um conjunto fixo destes fatores e o conceito de nivelamento não é suportado (o que reduz a precisão 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 podem ser conflitantes, 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. Esta proposta está inserida no contexto do projeto QUEST, cujo objetivo é o desenvolvimento da arquitetura de uma ferramenta de SPI (*Software Process Improvement*), e encontrava-se, quando o autor tomou ciência de sua existência, parcialmente implementada.

Coelho (2003) 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 APSEE-Tail, 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), 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. Não foi implementado um protótipo para suportá-la, no entanto a mesma foi "avaliada" (foram gerados processos adaptados para dois projetos em andamento e obtidos *feedbacks* dos gerentes ao compararem tais processos com os que estavam realmente sendo utilizados) em casos de uso reais.

7.2 Contribuições

Antes de serem listadas aquelas que o autor acreditam sejam as contribuições do trabalho, faz-se necessário avaliar se o APSEE-Tail atende os requisitos listados na seção 1.1 da introdução. Para cada requisito, são informados o status de atendimento (atendido, parcialmente atendido ou não atendido) e a justiticativa para tal.

- *Requisito 1*: exigência do processo reutilizável candidato à adaptação e a descrição rigorosa do contexto de utilização;
    - Atendido: no nível conceitual, ao especificar os tipos de dados para representação do processo reutilizável (classe *Standard*) e do contexto de utilização (classe *Projects*) e os correspondentes ATOs, incluindo-os nos argumentos da operação de adaptação (método *adapt*), o APSEE-Tail formalizou suas representações, ao mesmo tempo em que os transformou em parâmetros necessários para o seu funcionamento. No nível de implementação, tal requisito é atendido pelo Editor do Processo Padrão (através de sua interface com o ambiente APSEE), pelo Editor de Projetos de Software e pelo Adaptador de Processos.
- *Requisito 2*: fornecer, como resultado da adaptação, um processo adaptado sintaticamente coerente, pronto para ser instanciado e executado, ou aperfeiçoado ainda mais pelo engenheiro de processos;
    - Parcialmente Atendido: foram definidas e formalizadas as regras a serem aplicadas nos diversos pontos da adaptação onde o processo adaptado pode

sofrer alterações, de modo a mantê-lo sintaticamente coerente e evitar a formação de ciclos. Estas regras foram definidas com base no meta-modelo de processos reutilizáveis adotado (classe *AbsProcessModel*) e, juntamente com a premissa de que qualquer processo importado pelo APSEE-Tail deva estar sintaticamente coerente, permite, numa primeira análise, afirmar que o requisito em questão foi atendido. Entretanto, apenas com a experimentação prática será possível avaliar se alguma situação não foi tratada.

- *Requisito 3*: documentar as alterações realizadas no processo de software original, de forma adequada, tal que uma análise das variações deste processo seja possível;
    - Parcialmente Atendido: foi criada uma estrutura de dados (classe *Cases*) para armazenar um conjunto de informações teoricamente úteis para serem reutilizadas em adaptações futuras, bem como permitir análises que possam indicar boas práticas dentro da organização-usuária. Some-se a isso o registro das variantes geradas em todas as adaptações (classe *AdaptedProcesses*). Novamente vale ressaltar que apenas a experimentação prática poderá verificar se tais informações são realmente suficientes e úteis, ou se são insuficientes e estão mal organizadas.
- *Requisito 4*: tornar a tarefa de adaptar processos mais simples e menos custosa.
    - Parcialmente Atendido: apesar da atenção destinada à fundamentação teórica da proposta, evidenciada na formalização do APSEE-Tail, a sua definição foi pautada desde o início pelos aspectos de utilidade e simplicidade para o usuário. A idéia de armazenar o conhecimento útil a esta atividade, que antes estava apenas na cabeça de alguns indivíduos da organização de software, e permitir o raciocínio sobre ele, bem como tirar a responsabilidade do engenheiro de processos em garantir a coerência sintática do processo a ser gerado, tenta ir de encontro ao requisito em questão. O próprio suporte automatizado, fornecido através do protótipo, tenta alcançar este objetivo, e é provável que alcance, se comparado à situação onde a adaptação é feita de forma manual. Não cabe ao autor realizar qualquer juízo de valor sobre os possíveis ganhos do APSEE-Tail. Aqui, mais do que nunca, a experimentação prática é fundamental na resposta a este requisito.

Dado o exposto, são listadas então aquelas que o autor acredita sejam as contribuições do trabalho desenvolvido nesta dissertação de mestrado:

- Definição de um modelo para apoiar o engenheiro de processos na tarefa de adaptar o processo padrão de uma organização de software para as características de cada um de seus projetos, baseado no armazenamento do conhecimento necessário e na capacidade de raciocínio sobre o mesmo. O APSEE-Tail tem, como objetivo final, o estabelecimento de uma "família" de processos adaptados, todos derivados do processo padrão, em que cada "membro" da "família" é um processo adaptado, testado e aprovado para uma circunstância específica. Um artigo, contendo a visão geral do modelo, foi publicado no *Congresso Brasileiro de Computação 2004* (MAIA, 2004a);
- A definição de uma arquitetura para caracterização de projetos de software, que permite: a definição de quais tipos de característica influenciam os projetos da organização de software (neste sentido, a definição das três espécies de um tipo

de característica representa um avanço no sentido de retratar de forma mais adequada as características de um projeto); a instanciação dos tipos de característica para refletir as características de cada projeto; a determinação do grau de semelhança entre projetos quanto às suas características (adaptou-se o algoritmo da vizinhança, levando-se em conta as espécies de tipos de característica existentes);

- A definição de um componente para especificação dos critérios de adaptação do processo padrão. Devido poderem ser associadas a uma e somente uma atividade e não poderem aparecer simultaneamente em um fragmento e suas subatividades, eliminou-se qualquer possibilidade de conflito de regras;
- Os componentes do APSEE-Tail constituem, de fato, um repositório de conhecimento acerca da realidade da organização de software. A medida em que o uso do APSEE-Tail aumenta, mais rico em conhecimento se torna este repositório, passando a ser tão ou mais importante para a organização do que o próprio mecanismo de adaptação, podendo ser utilizado para outras finalidades;
- Os diferentes componentes envolvidos na definição do APSEE-Tail foram especificados algebricamente, constituindo uma base semântica de alto nível de abstração, que deu origem a um protótipo implementado no ADS Prosoft e que possui interface com o ambiente APSEE. Um artigo, descrevendo os componentes do modelo, foi publicado no *Congresso Argentino de Ciência da Computação 2004, Workshop de Engenharia de Software* (MAIA, 2004b);
- A interface com o APSEE agregou valor ao ambiente, visto que o mesmo não possuía suporte automatizado à adaptação de processos;
- Um conjunto de regras foi definido para garantir que, dado um processo padrão sintaticamente coerente, o APSEE-Tail gere processos adaptados também coerentes sintaticamente, retirando assim mais uma responsabilidade das mãos do engenheiro de processos. Estas regras foram especificadas algebricamente através de Gramática de Grafos e, juntamente com a parte especificada em Prosoft-Algébrico, formalizam o comportamento do mecanismo de adaptação do APSEE-Tail;
- A implementação do protótipo acabou por demandar algumas melhorias no ambiente Prosoft: o ATOTexto foi alterado para ficar mais eficiente; o visualizador de processos concretos do APSEE, o APSEE-Monitor, foi estendido para suportar processos abstratos;
- A definição de uma metodologia a ser seguida para realização da adaptação de processos, a qual é aplicada pelo APSEE-Tail e foi especificada utilizando-se a notação IDEF0.

7.3 Limitações

Existem alguns pontos relativos ao APSEE-Tail que não foram tratados nesta dissertação de mestrado, dentre outros motivos, por não terem sido considerados críticos para o alcance dos objetivos traçados. Dependendo do ponto de vista, tais pontos podem ser encarados como melhorias a serem feitas no APSEE-Tail ou limitações do mesmo. O autor deixa esta avaliação para o leitor, limitando a listar tais pontos:

- *Ausência de uma estratégia para avaliação e melhoria do processo adaptado durante sua execução*. Seria interessante permitir que o gerente de projetos realizasse avaliações parciais da execução do processo adaptado gerado, bem como uma avaliação final. Essa informação seria valiosa para o engenheiro de processos quando do reuso de um caso de adaptação, além de possibilitar ao APSEE-Tail fornecer sugestões de melhoria do processo em utilização. De preferência, este mecanismo de avaliação deveria poder ser costumizado pelo engenheiro de processos à realidade da organização;
- *Impossibilidade de melhoria ou extensão do processo padrão dentro APSEE-Tail*. Neste sentido, o APSEE-Tail poderia suportar a ação de padronizar uma atividade do processo adaptado, definida externamente ou não, desde que detectado os seus benefícios para projetos posteriores;
- *Realização de extensões na gramática que define a parte condicional de uma regra de adaptação* para, por exemplo, suportar o uso de parênteses, novos operadores lógicos ou relacionais, testes feitos sobre a importância das características nos projetos, entre outras;
- *Extensão do modelo para suporte a múltiplos paradigmas de adaptação, a princípio excludentes*. Assim, por exemplo, poderiam ser suportadas a adaptação orientada à atividades ou artefatos;
- Criação de grupos de tipos de características, além da permissão da atribuição de pesos ao grupo, ao tipo de característica e às suas instâncias. Para o projeto, o peso de uma característica seria aquele associado à instância, ou aquele associado ao tipo, caso o peso da instância não estivesse especificado, ou o do grupo, caso os do tipo e da instância não estivessem especificados;
- *Suporte a todos os três tipos de adaptação*, definidos por Reis (2002). O tipo de adaptação poderia estar associado à adaptação em si (tornando-se mais uma de suas entradas, juntamente com o projeto de software, grau de similaridade e identificador do processo adaptado) ou ao processo padrão (podendo ser redefinido apenas com a definição de um novo processo padrão).

7.4 Trabalhos Futuros

Algumas questões ainda estão em aberto, definindo pontos que devem ser explorados em trabalhos que sigam a mesma linha. Deste modo, abaixo são apresentados os elementos da pesquisa para serem investigados no futuro próximo:

- A realização desse trabalho, por si só, não garante um aumento da qualidade, produtividade ou melhoria dos custos para a modelagem de processos ou para o software resultante. Assim, vislumbra-se como uma importante atividade a ser realizada a condução de uma avaliação empírica que determine, através de estudos de casos aplicados na indústria, os benefícios reais obtidos quando o APSSE-Tail é utilizado, comparando com situações similares quando nenhum apoio automatizado está disponível. Isto também permitiria a identificação de possibilidades de melhoria;
- Definição de políticas ou regras para adaptação dinâmica. Tais regras deveriam permitir testes sobre as propriedades de quaisquer componentes do processo

concreto ou de seu modelo, ou ainda sobre as já utilizadas características do projeto, para definir então pela permanência ou não, ou pela mudança de alguma propriedade de determinado componente. Um exemplo: se, em um projeto crítico, um conjunto de atividades-chave está atrasado, então incluir e executar a atividades Gestão de Subcontratação e Renegociação de Cronograma e Custos;

- A adaptabilidade é um conceito abstrato que, para o modelo proposto, pode ser avaliado segundo várias perspectivas. Este tópico discorre essencialmente acerca da adaptabilidade do APSEE-Tail para diferentes PSEEs ou PMLs. À primeira vista, isto é possível uma vez que os componentes e o funcionamento do APSEE-Tail foram especificados formalmente, portanto sua descrição é precisa e em alto nível de abstração, o que possivelmente pode guiar diferentes implementações. Em se mantendo a implementação atual, uma alternativa seria criar um arquivo de mapeamento, em um formato portável como o XML (*eXtensible Markup Language*), de modo a permitir relacionar os elementos do meta-modelo de processos reutilizáveis do APSEE-Tail com os do adotado pelo ambiente externo. Em geral, acredita-se ser possível adaptar o APSEE-Tail para ambientes e linguagens que adotem um paradigma orientado à atividades. Todavia, investigação adicional sobre esse tópico ainda é necessária;
- O APSEE-Tail foi especificado formalmente com o objetivo de fornecer uma descrição precisa e em alto nível de abstração e, a partir da mesma, construir um protótipo que permitisse a sua experimentação prática. Apesar disso, não foi objetivo deste trabalho realizar verificações formais nas especificações, sendo este também um tópico relevante para trabalhos futuros nesta área. O autor consegue vislumbrar, em uma primeira análise, um ponto do modelo que poderiam ser verificado e beneficiado com isso: se as regras em Gramáticas de Grafos realmente garantem a coerência sintática dos processos adaptados.

7.5 Considerações Finais

Esse trabalho representa um passo na evolução da área de Processos de Software no sentido de proporcionar uma abordagem automatizada para apoiar a adaptação de processos. O autor acredita que o aprofundamento da pesquisa nesse assunto pode levar a um aumento significativo na qualidade dos processos adotados e, por conseqüência, nos produtos resultantes de organizações de desenvolvimento de software.

Além disso, o desenvolvimento do modelo proposto permitiu um melhor entendimento da área de Processos de Software e provocou uma reflexão sobre os benefícios e as dificuldades da adaptação de processos. Desse melhor entendimento e dessa reflexão surgiu a necessidade de melhorá-lo, apontando para a realização de trabalhos futuros que o tornarão mais completo e aplicável, aproximando-o cada vez mais de um produto de uso prático para a indústria de software.

REFERÊNCIAS

ABEL, M.; CASTILHO, J. M. V. **Um Estudo sobre Raciocínio Baseado em Casos**. 1996. Trabalho Individual (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

ALVES, R. C. de O. **Uma Proposta de apoio para decisões de grupo no ambiente PROSOFT.** 2002. 173 f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

AHN, Y. W. et al. Knowledge and Case-Based Reasoning for Customization of Software Processes - A Hybrid Approach. **International Journal of Software Engineering and Knowledge Engineering**, [S.l.], v.13, n.3, 2003.

BASILI, V.; ROMBACH, D. Tailoring the Software Process to Project Goals and Environments. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGEERING, 1987. **Proceedings…** [S.l. : s.n.], 1987. p.345-357.

BERGER, P. M. **Instanciação de Processos de Software em Ambientes Configurados na Estação TABA**. 2003. 128f. Dissertação (Mestrado em Engenharia de Sistemas e Computação) – COPPE, UFRJ, Rio de Janeiro.

BUDLONG, F. C. et al. **Process Tailoring for Software Project Plans**. 1996. Disponível em: <http://www.stsc.hill.af.mil/resources/tech_docs/process_plan/>. Acesso em: out. 2003.

COELHO, C. C. **MAPS**: Um Modelo de Adaptação de Processos de Software. 2003. 162f. Dissertação (Mestrado em Ciência da Computação) - Centro de Informática, UFPE, Pernambuco.

DAHMER, A. **Um Ambiente para Desenvolvimento e Gerência de Cursos em Educação a Distância** (tese a ser defendida). 2005. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

DERNIAME, J.; KABA, B.; WASTELL, D. (Ed.) **Software Process**: Principles, Methodology and Technology. Berlin: Springer-Verlag, 1999. (Lecture Notes in Computer Science, v. 1500).

DOWSON, M.; NEJMEH, B.; RIDDLE, W. Fundamental Software Process Concepts. In: EUROPEAN WORKSHOP ON SOFTWARE PROCESS MODELLING, 1., 1991, Milan, Italy. **Proceedings...** [S.l.]: AICA Press, 1991.

EMAN, K.; DROUIN, J.N.; MELO, W. (Ed.) **SPICE**: The theory and practice of software process improvement and capability determination. Los Alamitos: IEEE Computer Society Press, 1998.

FEILER, P.; HUMPHREY, W. Software Process Development and Enactment: Concepts and Definitions. In: INTERNATIONAL CONFERENCE ON THE SOFTWARE PROCESS, ICSP, 2., 1993. **Proceedings…** Berlin, Germany: IEEE Computer Society Press, 1993.

FIPS (*Federal Information Processing Standarts*). **Integration definition for function modeling – IDEF0**. Draft Federal Information Processing Standards Publication.ed. Gaithersburg, MD, USA: Federal Information Processing Standards Publications (FIPS PUBS), 1993.

FREITAS, A. V. P. **APSEE-Global**: um Modelo de Gerência de Processos Distribuídos (a ser defendida). 2005. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

FUGGETTA, A. Software Process: A Roadmap. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, ICSE, 22., 2000, Limerick, Ireland. **Proceedings...** New York: ACM Press, 2000.

GRANVILLE, L.Z.; SCHLEBBE, H. **Distributed PROSOFT**: management of tools and memory. Stuttgart: [s.n.], 1996. 26 p. Technical Report.

HENNINGER, S. An Environment for Reusing Software Processes. INTERNATIONAL CONFERENCE ON SOFTWARE REUSE, 5., 1998. Disponível em: <http://pooh.unl.edu/~scotth/publications.html>. Acesso em: out. 2003.p.103-112.

HUMPHREY, W. S. **Managing the Software Process**. New York: Addison-Wesley, 1989.

ISO/IEC TR 15504, 1998, Parts 1-9: Information Technology - Software Process Assessment.

JACKSON, M. **System Development.** New York: Prenctice-Hall International, 1993.

LIMA REIS, C. A. **Um Gerenciador de Processos de Software para o Ambiente PROSOFT**. 1998. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

LIMA REIS, C. A. **Uma Abordagem Flexível para Execução de Processos de Software Evolutivos**. 2003. 267f. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

LONCHAMP, J. A Structured Conceptual and Terminological Framework for Software Process Engineering. In: INTERNATIONAL CONFERENCE ON SOFTWARE PROCESS, ICSP, 2., Berlin, Germany. **Proceedings…** Los Alamitos: IEEE Computer Society Press, 1993. p.41-53.

MAIA, A. B.; FREITAS, A. V. P. de; NUNES, D. J. Um Modelo para Auxiliar a Adaptação de Processos de Software. In: CONGRESSO BRASILEIRO DE COMPUTAÇÃO, 4., 2004, Itajaí. **Anais. . .** Itajaí: Univali, 2004. p.155–160.

MAIA, A. B.; FREITAS, A. V. P. de; NUNES, D. J.; STEINMACHER, I. Componentes de um Modelo para Adaptação de Processos de Software. In: CONGRESO ARGENTINO DE CIENCIAS DE LA COMPUTACIóN, 10., 2004, Buenos Aires. **Anais. . .** [S.l.: s.n.], 2004.

MACHADO, L. F. et al. Def-Pro: Apoio automatizado para Definição de Processos de Software. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, 14., 2000, João Pessoa, PB. **Anais...** João Pesoa: CETEF-PB, 2000. p.359-362.

MORAES, S. M. W. **Um Ambiente Expert para Apoio ao Desenvolvimento de Software**. 1997. 139 f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

MÜNCH, J.; SCHMITZ, M.; VERLAGE, M. Tailoring groβer Prozeβmodelle auf der Basis von MVP-L. In: WORKSHOP DER FACHGRUPPE: VORGEHENSMODELLE - EINFÜHRUNG, BETRIEBLICHER EINSATZ, WERKZEUG-UNTERSTÜTZUNG UND MIGRATION, 4., 1997. **Proceedings…** [S.l: s.n.], 1997.

NUNES, D. J. Estratégia Data-driven no Desenvolvimento de Software. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, SBES, 6., 1992, Gramado. **Anais...** [S.l.]: Sociedade Brasileira de Computação, 1992. p. 81-95.

NUNES, D. J. **PROSOFT**: Um Ambiente de Desenvolvimento de Software Baseado no Método Algébrico. Porto Alegre: Instituto de Informática, Universidade Federal do Rio Grande do Sul, 1994. Relatório de Pesquisa Interno. Disponível em: <http://www.inf.ufrgs.br/~prosoft>. Acesso em: set. 2004.

OSTERWEIL, L. Software Processes are Software Too. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 9., Monterey, CA. **Proceedings...** Los Alamitos: IEEE Computer Society Press, 1987.

PAULK, M. C. et al. (Ed.). **The Capability Maturity Model**: Guidelines for Improving the Software Process. Carnegie Mellon University, Software Engineering Institute, Addison-Wesley Longman Inc, 1997.

PERRY, D. E. Practical Issues in Process Reuse. In: INTERNATIONAL SOFTWARE PROCESS WORKSHOP, ISPW, 10., France. **Proceedings...** Los Alamitos: IEEE Computer Society Press, 1996.

PIMENTA, A. **Especificação formal de uma ferramenta de reutilização de especificações de requisitos**. 1998. 120 f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

PRESSMAN, R. **Engenharia de Software**. 5.ed. Rio de Janeiro: McGraw-Hill, 2002.

RANGEL, G. S. **ProTool**: uma ferramenta de prototipação de software para o ambiente PROSOFT. 2003. 223 f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

REIS, R. Q. **Uma Avaliação dos Paradigmas de Linguagens de Processo de Software**. 1999. Trabalho Individual II (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

REIS, R. Q. **Reutilização de Processos de Software**. 2000. Exame de Qualificação (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

REIS, R. Q. **APSEE-Reuse**: um Meta-Modelo para Apoiar a Reutilização de Processos de Software. 2002. 215f. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

RIBEIRO, L. Métodos Formais de Especificação: Gramáticas de Grafos. In: ESCOLA DE INFORMÁTICA DA SBC – SUL, 2000, Ijuí. **Livro texto**. Santa Maria: UFSM, 2000. p.1-34.

ROSS, D. Applications and Extensions of SADT. **IEEE Computer**, New York, USA, v.18, n.4, p.25–35, March 1985.

RUPPRECHT, C. et al. Capture and Dissemination of Experience About The Construction of Engineering Processes. In: CONFERENCE ON ADVACED INFORMATION SYSTEMS ENGINEERING, CAISE, 12., 2000, Stockholm, Sweden **Proceedings...** [S. l. : s.n.], 2000.

SCHLEBBE, H. **Distributed PROSOFT**: report on a working stay at the institute of computer science of the state university of Rio Grande do Sul (UFRGS) at Porto Alegre, Brazil from May 1 to June 15, 1994. Stuttgart: Universität Stuttgart. Fakultät Informatik, 1994.

SCHLEBBE, H. **Java PROSOFT Manual.** Disponível em: <http://www.informatik.uni-stuttgart.de/ifi/bs/schlebbe/prosoft_doc/guide> Acesso em: set. 2005.

SCHLEBBE, H.; SCHIMPF, S. **Reenginering of PROSOFT in Java.** Technical Report. Stuttgart: Universität Stuttgart. Fakultät Informatik, 1997.

SILVA, F. A. das D. **Um modelo de simulação de processos de software baseado em conhecimento para o ambiente PROSOFT**. 2001. 123 f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

SOUSA, A. L. R. de. **APSEE-Monitor**: um mecanismo de apoio à visualização de modelos de processos de software. 2003. 112 f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

SCHMID, K.; WIDEN, T. Customizing the PuLSETM Product Line Approach to the Demands of an Organization. In: EUROPEAN WORKSHOP ON SOFTWARE PROCESS TECHNOLOGY, EWSPT, 7., 2000, Kaprun, Austria. **Proceedings…** [S. l.: s.n.], 2000. p.221-238.

TRAVASSOS, G. H. **O Modelo de Integração de Ferramentas da Estação TABA.** Tese (Doutorado em Ciência da Computação) – Programa de Engenharias de Sistemas, COPPE/UFRJ, Rio de Janeiro.

WATT, D. **Programming Language Syntax and Semantics**. New York: Prentice-Hall, 1991.

WELZEL, D.; HAUSEN, H.; SCHMIDT, W. Tailoring and Conformance Testing of Software Processes: The ProcePT Approach. In: IEEE SOFTWARE ENGINEERING STANDARDS SYMPOSIUM, 2., 1995, Montreal, Quebec, Canada. **Proceedings…** [S.l. : s.n], 1995. p.41-49.

XU, P. et al. A Tool for Capture and Use of Process Knowledge in Process Tailoring. In: HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES, HICSS, 36., 2002. **Proceedings...** Los Alamitos: IEEE Computer Society Press, 2002.

ANEXO A OPERAÇÕES DOS CONSTRUTORES DE TIPOS DO PROSOFT-ALGÉBRICO

Este anexo apresenta as operações algébricas dos construtores de tipos do Prosoft-Algébrico. Na tabela abaixo, para as operações de cada construtor, são apresentados seu nome, funcionalidade (*rank*) e um exemplo de utilização. O símbolo "_" nas operações indica o *place-holder* (posição ou lugar) onde cada argumento está localizado dentro da operação sob a sintaxe mixfix. Quando não aparecer este símbolo, então a operação tem sintaxe pré-fixada, caso não seja um construtor.

Tabela A.1: Lista das operações dos construtores de tipos do Prosoft-Algébrico (RANGEL, 2003)

Tipo	*Operação*	*Funcionalidade*	*Exemplo de Uso*
Conjunto	Emptyset	→ Set	{}
	Add	Element Set → Set	add(x1,{x2,x3}) = {x1,x2,x3}
	_U_	Set Set → Set	{x1,x2} U {x3} = {x1,x2,x3}
	belongsto	Element Set → Boolean	x1 belongsto {x1,x2,x3} = true
	Cardinality	Set → Integer	cardinality({x1,x2,x3}) = true
	complement	Set Set → Set	complement({x1,x2,x3},{x2}) = {x1,x3}
	contain	Set Set → Boolean	{x1,x2,x3} contain {x1,x3} = true
	Delete	Element Set → Set	delete(x2,{x1,x2,x3}) = {x1,x3}
	Equal	Set Set → Boolean	equal({x1,x2},{x3}) = false
	intersection	Set Set → Set	{x1,x2,x3} intersection {x1,x3} = {x1,x3}
	Isempty	Set → Boolean	isempty({x1}) = false
Lista	Emptylist	→ List	<>
	Cons	Element List → List	cons(x1,<>) = <x1>
	Concat	List List → List	concat(<x1,x2>,<x1>) = <x1,x2,x1>
	Elements	List → Set	elements(<x1,x2,x3>) = {x1,x2,x3}
	Head	List → Element	head(<x1,x2,x3>) = x1
	Last	List → Element	last(<x1,x2,x3>) = x3
	Length	List → Integer	length(<x1,x2,x3>) = 3

	Projection	List Integer → Element	projection(<x1,x2,x3>,2) = x2
	Replace	List Integer Element → List	replace(<x1,x2,x3>,2,x5) = <x1,x5,x3>
	Tail	List → List	tail(<x1,x2,x3>) = <x2,x3>
	Isin	Element List → Boolean	isin(x2,<x1,x2,x3>) = true
	Delete	Element List → List	delete(x2,<x1,x2,x3>) = <x1,x3>
	Invert	List → List	invert(<x1,x2,x3>) = <x3,x2,x1>
	Isempty	List → Boolean	isempty(<>) = true
Mapeamento	Emptymap	→ Map	[]
	Modify	Domain Range Map → Map	modify(x1,y1,[]) = [x1 → y1]
	composition	Map Map → Map	composition([x1 → y1],[x2 → y2]) = [x1 → y2]
	Domain	Map → Set	domain([x1 → y1,x2 → y2]) = {x1,x2}
	Imageof	Domain Map → Range	imageof(x1,[x1 → y1,x2 → y2]) = y1
	Merge	Map Map → Map	merge([x1 → y1],[x2 → y2]) = [x1 → y1,x2 → y2]
	Override	Map Map → Map	override([x1 → y1,x2 → y2],[x1→ y3]) = [x1 → y3,x2 → y2]
	Range	Map → Set	range([x1 → y1,x2 → y2]) = {y1,y2}
	Restrictto	Map Set → Map	restrictto([x1 → y1,x2 → y2],{x2}) = [x2 → y2]
	restrictwith	Map Set → Map	restrictwith([x1 → y1,x2 → y2],{x2}) = [x1 → y1]
	Isempty	Map → Boolean	isempty([]) = true
Registro	Reg	D1 D2 D3 Dn → Register	reg-Type(2,true)
	select-Dn	Register → Dn	select-Integer(reg-Type(2,true)) = 2
União Disjunta	DesjoinUnion-Dn	Dn → DesjoinUnion	Type-Number(2)
	get-Dn	DesjoinUnion → Dn	get-Number(Type-Number(2)) = 2
	is-Dn	DesjoinUnion → Boolean	is-Number(Type-Number(2)) = true

ANEXO B A CLASSE *ABSPROCESSMODEL*

Neste anexo, são apresentadas as principais classes, definidas por Reis (2003) para a especificação do modelo de reutilização de processos de software APSEE-Reuse, relacionadas ao tipo *AbsProcessModel*, o qual define modelos de processo abstratos. Esta classe foi utilizada na especificação das estruturas do processo padrão e dos processos adaptados. Uma descrição detalhada das classes abaixo foge ao escopo deste trabalho, podendo ser encontrada em (REIS, 2003).

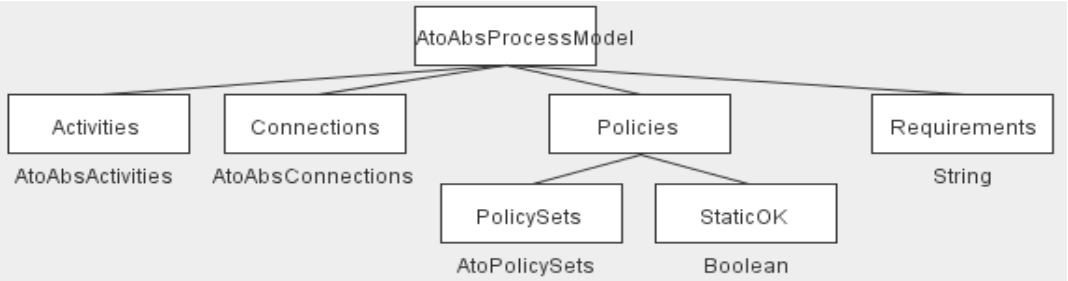


Figura B.1: A classe *AbsProcessModel*

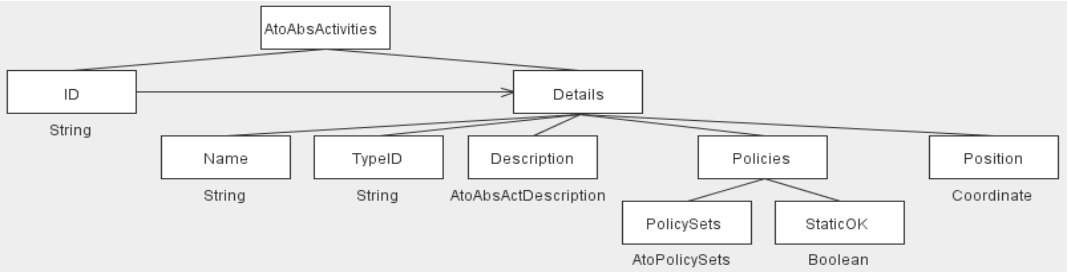


Figura B.2: A classe *AbsActivities*

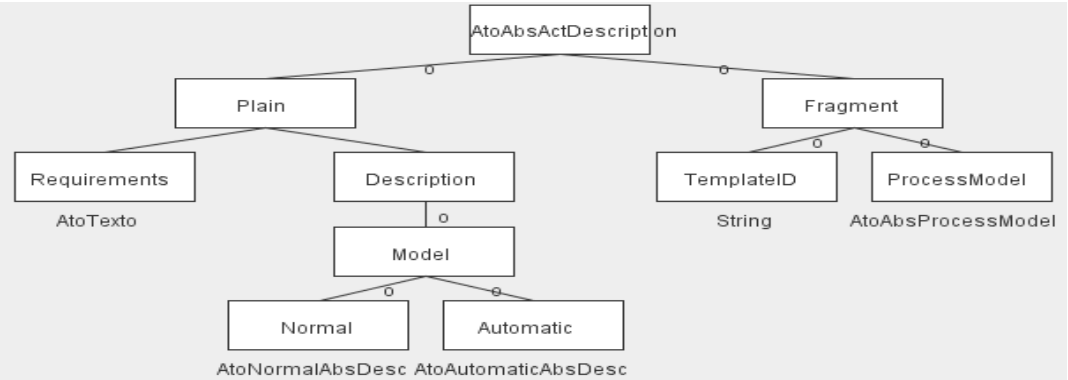


Figura B.3: A classe *AbsActDescription*

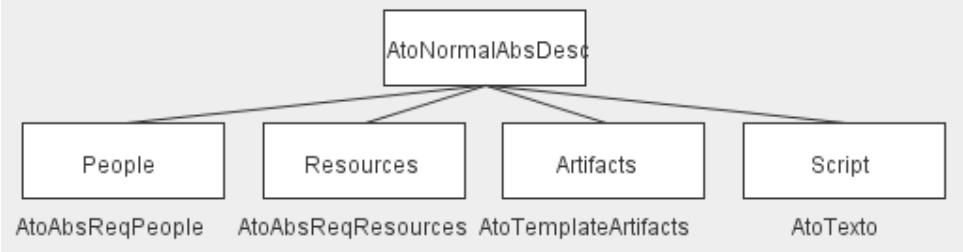


Figura B.4: A classe *NormalAbsDesc*

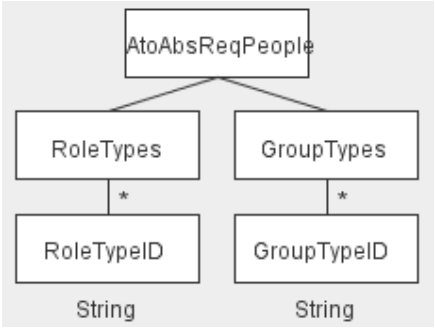


Figura B.5: A classe *AbsReqPeople*

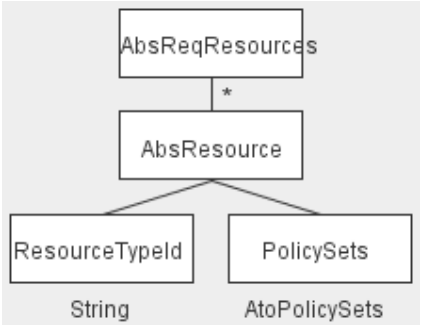


Figura B.6: A classe *AbsReqResources*

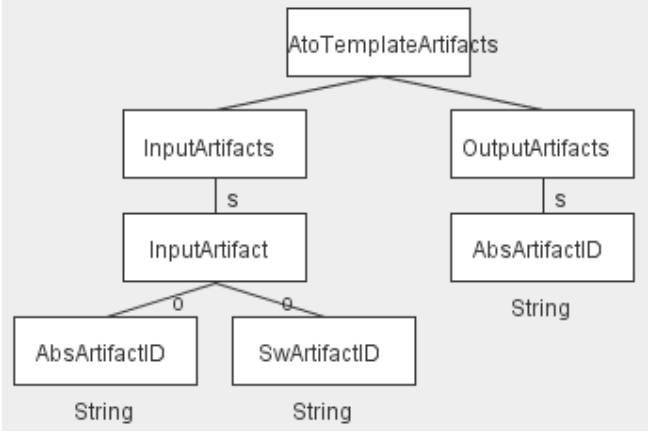


Figura B.7: A classe *TemplateArtifacts*

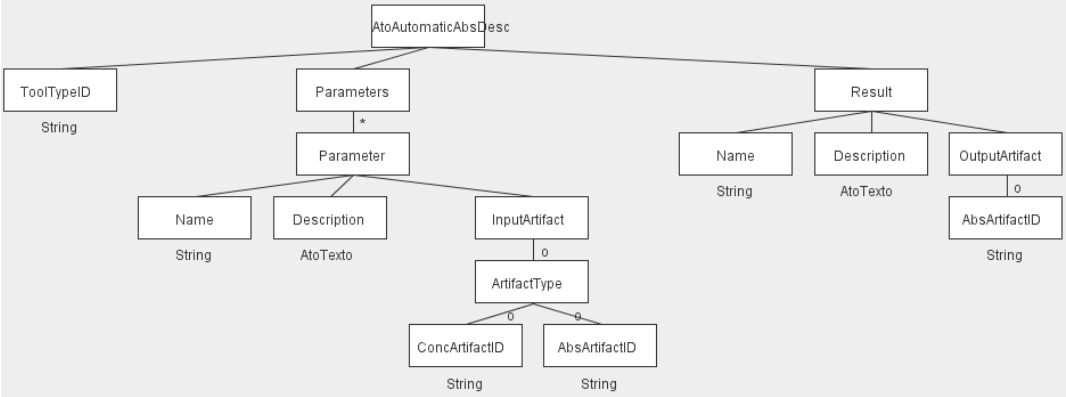


Figura B.8: A classe *AutomaticAbsDesc*

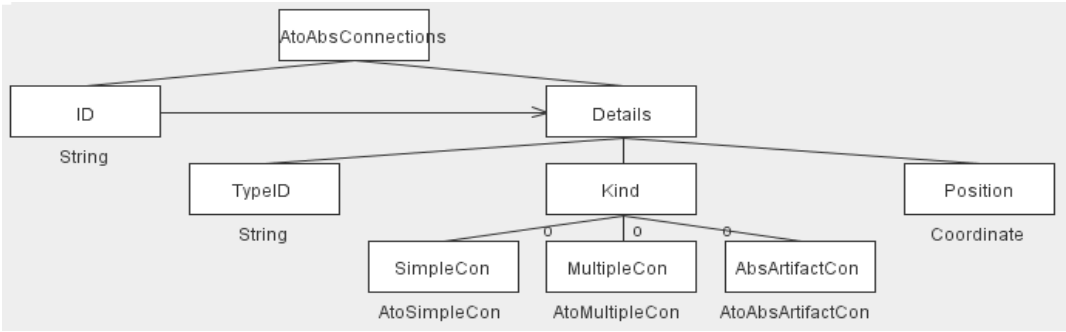


Figura B.9: A classe *AbsConnections*

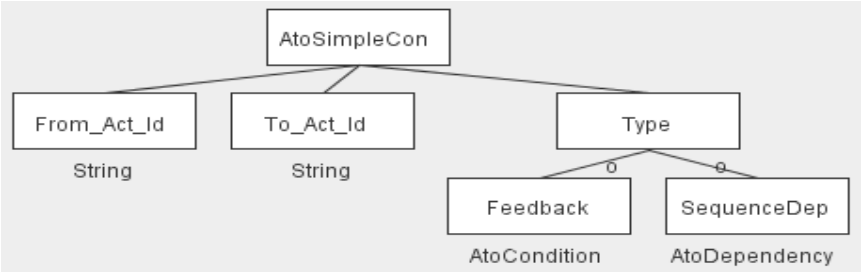


Figura B.10: A classe *SimpleCon*

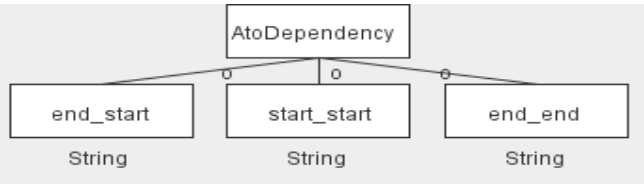


Figura B.11: A classe *Dependency*

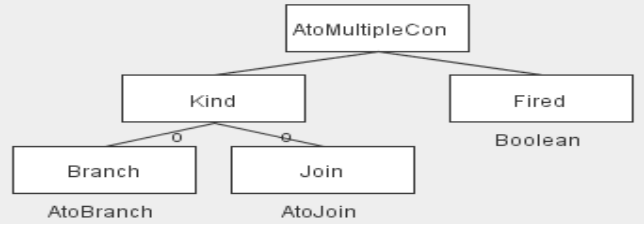


Figura B.12: A classe *MultipleCon*

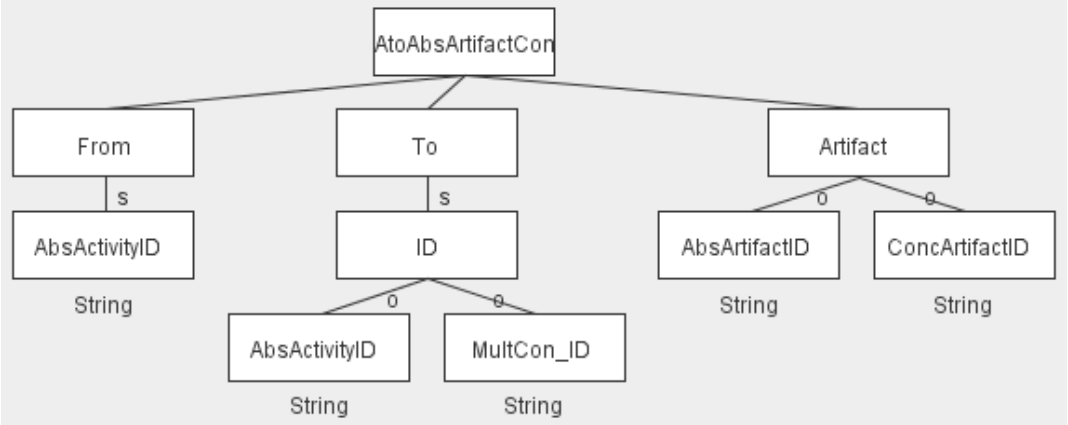


Figura B.13: A classe *AbsArtifactCon*

APÊNDICE A ATOS ALGÉBRICOS

Nas seções a seguir, são apresentados os ATOs algébricos que representam os principais componentes do APSEE-Tail. Para cada ATO, são apresentados: uma descrição informal de suas operações; a classe que define os sorts por ele tratado; as inclusões que foram necessárias na sua especificação; as interfaces das suas operações; as variáveis formais utilizadas; por fim, os axiomas.

A.1 ATOAdaptationModel

Abaixo é apresentada uma descrição de alto nível para as principais operações deste ATO:

- *adapt*: adapta o processo padrão para um projeto de software. Por representar a semântica do mecanismo de adaptação, será apresentado em detalhes no próximo apêndice;
- *getStandard*: obtém o modelo do processo padrão para poder exportá-lo;
- *newStandard*: define um novo processo padrão;
- *updateStandard*: atualiza o modelo do processo padrão;
- *includeRule*: associa uma regra a uma atividade do processo padrão;
- *excludeRule*: exclui uma regra de adaptação;
- *alterRule*: altera os dados de uma regra de adaptação;
- *includeCondition*: inclui uma condição em uma regra de adaptação;
- *isWellFormedExpression*: verifica se uma expressão é bem formada;
- *isApplicableCompOpToCharKind*: verifica se o operador relacional é aplicável ao tipo de característica;
- *isApplicableValueToCompOp*: verifica se o operador relacional é aplicável ao valor em questão;
- *excludeCondition*: exclui uma condição de uma regra de adaptação;
- *alterCondition*: altera os dados de uma condição;
- *getAdaptedProcess*: obtém o modelo do processo adaptado para poder exportá-lo;
- *createAdaptedProcess*: cria um novo processo adaptado;

- *updateAdaptedProcess*: atualiza um processo adaptado existente;
- *updateModifications*: atualiza as modificações referentes ao processo adaptado atualizado;
- *includeActivity*: inclui uma atividade no processo adaptado;
- *excludeActivity*: exclui uma atividade do processo adaptado;
- *includeCharacteristicType*: inclui o tipo de característica;
- *excludeCharacteristicType*: exclui o tipo de característica;
- *alterCharacteristicType*: altera o tipo de característica;
- *includeValue*: inclui um valor no tipo de característica;
- *excludeValue*: exclui um valor do tipo de característica;
- *includeProject*: inclui um projeto de software;
- *excludeProject*: exclui um projeto de software;
- *alterProject*: altera os dados de um projeto de software;
- *includeCharacteristic*: inclui uma característica em um projeto de software;
- *excludeCharacteristic*: exclui uma característica de um projeto de software;
- *alterCharacteristic*: altera os dados de uma característica do projeto de software;
- *excludeCase*: exclui o caso de adaptação;
- *commentCase*: comenta o caso de adaptação;
- *avaliateCase*: avalia o caso de adaptação;
- *showTrace*: mostra o histórico da adaptação.

CLASS

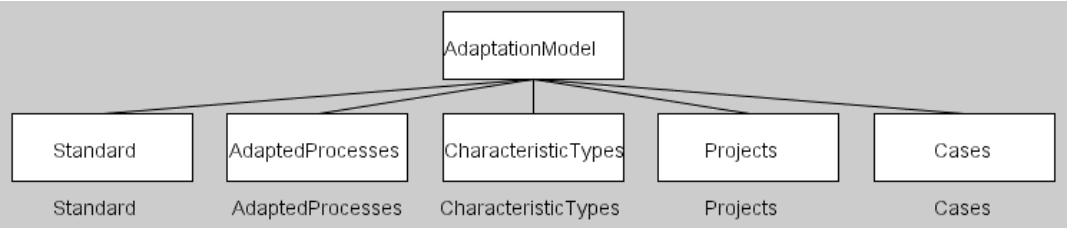


Figura A.1: Classe que define o ATOAdaptationModel

INCLUDE

INTERFACES

```
adapt : AdaptationModel String String Double → AdaptationModel

getStandard : AdaptationModel → AbsProcessModel

newStandard : AdaptationModel AbsProcessModel → AdaptationModel

updateStandard : AdaptationModel AbsProcessModel → AdaptationModel

includeRule : AdaptationModel String Text RuleKind → AdaptationModel

excludeRule : AdaptationModel String → AdaptationModel

alterRule : AdaptationModel String Text RuleKind → AdaptationModel

includeCondition : AdaptationModel String Expression LogicOp →
                   AdaptationModel

isWellFormedExpression : Expression CharacteristicType → Boolean

isApplicableCompOpToCharKind : ComparisonOp CharKind → Boolean

isApplicableValueToCompOp : Value ComparisonOp → Boolean

excludeCondition : AdaptationModel String String → AdaptationModel

alterCondition : AdaptationModel String Expression LogicOp →
                 AdaptationModel

getAdaptedProcess : AdaptationModel String → AbsProcessModel

createAdaptedProcess : AdaptationModel String AbsProcessModel →
                       AdaptationModel

updateAdaptedProcess : AdaptationModel String AbsProcessModel →
                       AdaptationModel

updateModifications : Modifications SetOfString AbsActivities
                      AbsActivities AbsActivities Rules Projects
                      String CharacteristicTypes → Modifications
```

```
updateModification : Modifications String ModKind String
                     ActivityDescription ActivityDescription
                     AbsActivities Rules Projects String
                     CharacteristicTypes → Modifications

includeActivity : AdaptationModel String String → AdaptationModel

excludeActivity : AdaptationModel String String → AdaptationModel

includeCharacteristicType : AdaptationModel String String Text
                            CharKind → AdaptationModel

excludeCharacteristicType : AdaptationModel String → AdaptationModel

alterCharacteristicType : AdaptationModel String String Text
                          CharKind → AdaptationModel

includeValue : AdaptationModel String String → AdaptationModel

excludeValue : AdaptationModel String String → AdaptationModel

includeProject : AdaptationModel String String Text String Date Date
                 → AdaptationModel

excludeProject : AdaptationModel String → AdaptationModel

alterProject : AdaptationModel String String Text String Date Date →
               AdaptationModel

includeCharacteristic : AdaptationModel String String Value Integer
                        → AdaptationModel

excludeCharacteristic : AdaptationModel String String →
                        AdaptationModel

alterCharacteristic : AdaptationModel String String Value Integer →
                      AdaptationModel
```

```
excludeCase : AdaptationModel String → AdaptationModel

commentCase : AdaptationModel String Text → AdaptationModel

avaliateCase : AdaptationModel String Text → AdaptationModel

showTrace: AdaptationModel String → Text
```

FORMAL VARIABLES

```
proj-id proc-id name manager char-id value act-id : String
stand : Standard
proc-mod : AbsProcessModel
projs : Projects
desc comm reason aval : Text
start-dt end-dt : Date
cases : Cases
char-types : CharacteristicTypes
value : Value
weight : Integer
sim-degree : Double
char-kind : CharKind
rule-kind : RuleKind
exp : Expression
log-op : LogicOp
procs : AdaptedProcesses
```

AXIOMS

```
adapt(reg-AdaptationModel(stand,procs,char-types,projs,cases),proj-
id,proc-id,sim-degree)
= ?

getStandard(reg-AdaptationModel(stand,_,_,_,_))
= ICS(ATOStandard,getProcessModel,stand)

newStandard(reg-AdaptationModel(stand,_,_,projs,cases),proc-mod)
= if ICS(ATOAbsProcessModel,isEmpty,proc-mod)
  then reg-AdaptationModel(stand,_,_,projs,cases)
```

```
  else reg-AdaptationModel(
         ICS(ATOStandard,new,<proc-mod>),
         emptymap,
         _,
         ICS(ATOProjects,exclude,
           projs,<ICS(ATOCases,getEvaluated,cases)>
         ),
         emptymap
       )

updateStandard(reg-AdaptationModel(stand,_,_,_,_),proc-mod)
= if ICS(ATOAbsProcessModel,isEmpty,proc-mod)
  then reg-AdaptationModel(stand,_,_,_,_)
  else reg-AdaptationModel(
         ICS(ATOStandard,update,stand,<proc-mod>),
         _,_,_,_
       )

includeRule(reg-AdaptationModel(stand,_,_,_,_),act-id,comm,rule-kind)
= reg-AdaptationModel(
    ICS(ATOStandard,includeRule,
      stand,<act-id,comm,rule-kind>),_,_,_,_
    )

excludeRule(reg-AdaptationModel(stand,_,_,_,_),act-id)
= reg-AdaptationModel(
    ICS(ATOStandard,excludeRule,stand,<act-id>),
    _,_,_,_
  )

alterRule(reg-AdaptationModel(stand,_,_,_,_),act-id,comm,rule-kind)
= reg-AdaptationModel(
    ICS(ATOStandard,alterRule,stand,<act-id,comm,rule-kind>),_,_,_,_
  )

includeCondition(reg-AdaptationModel(stand,_,char-types,_,_),act-
id,exp,log-op)
= if ICS(ATOExpression,isEmpty,exp)
  then reg-AdaptationModel(stand,_,char-types,_,_)
  else if not isWellFormedExpression(exp,char-types)
```

```
       then reg-AdaptationModel(stand,_,char-types,_,_)
       else reg-AdaptationModel(
              ICS(ATOStandard,includeCondition,
                stand,<act-id,exp,log-op>
              ),
              _,char-types,_,_
            )

isWellFormedExpression(reg-Expression(char-id,comp-op,value),char-
types)
= ICS(ATOCharacteristicTypes,exist,char-types,<char-id>) and
  ICS(ATOCharKind,getValues,
      ICS(ATOCharacteristicTypes,getKind,char-types,<char-id>))
  contain
  ICS(ATOValue,getAsSet,value) and
  isApplicableCompOpToCharKind(
    comp-op,
    ICS(ATOCharacteristicTypes,getKind,char-types,<char-id>)
  ) and
  isApplicableValueToCompOp(value,comp-op)

isApplicableCompOpToCharKind(comp-op,char-kind)
= if ICS(ATOCharKind,isMultiple,char-kind)
  then ICS(ATOComparisonOp,isEqual,comp-op)) xor
       ICS(ATOComparisonOp,isDifferent,comp-op) xor
       ICS(ATOComparisonOp,isContains,comp-op) xor
       ICS(ATOComparisonOp,isNotContains,comp-op)
  else if ICS(ATOCharKind,isSingleWithLevel,char-kind)
       then ICS(ATOComparisonOp,isEqual,comp-op) xor
            ICS(ATOComparisonOp,isDifferent,comp-op) xor
            ICS(ATOComparisonOp,isGreater,comp-op) xor
            ICS(ATOComparisonOp,isGreaterEqual,comp-op) xor
            ICS(ATOComparisonOp,isSmaller,comp-op) xor
            ICS(ATOComparisonOp,isSmallerEqual,comp-op) xor
            ICS(ATOComparisonOp,isBetween,comp-op) xor
            ICS(ATOComparisonOp,isNotBetween,comp-op)
       else ICS(ATOComparisonOp,isEqual,comp-op) xor
            ICS(ATOComparisonOp,isDifferent,comp-op)

isApplicableValueToCompOp(value,comp-op)
```

```
= if ICS(ATOValue,isItem,value)
  then ICS(ATOComparisonOp,isEqual,comp-op) xor
       ICS(ATOComparisonOp,isDifferent,comp-op) xor
       ICS(ATOComparisonOp,isGreater,comp-op) xor
       ICS(ATOComparisonOp,isGreaterEqual,comp-op) xor
       ICS(ATOComparisonOp,isSmaller,comp-op) xor
       ICS(ATOComparisonOp,isSmallerEqual,comp-op)
  else ICS(ATOComparisonOp,isEqual,comp-op) xor
       ICS(ATOComparisonOp,isDifferent,comp-op) xor
       ICS(ATOComparisonOp,isBetween,comp-op) xor
       ICS(ATOComparisonOp,isNotBetween,comp-op)
       ICS(ATOComparisonOp,isContains,comp-op) xor
       ICS(ATOComparisonOp,isNotContains,comp-op)


excludeCondititon(reg-AdaptationModel(stand,_,_,_,_),act-id,char-id)
= reg-AdaptationModel(
    ICS(ATOStandard,excludeCondition,stand,<act-id,char-id>),_,_,_,_
  )


alterCondition(reg-AdaptationModel(stand,_,char-types,_,_),act-
id,exp,log-op)
= if ICS(ATOExpression,isEmpty,exp)
  then reg-AdaptationModel(stand,_,char-types,_,_)
  else if not isWellFormedExpression(exp,char-types)
       then reg-AdaptationModel(stand,_,char-types,_,_)
       else reg-AdaptationModel(
              ICS(ATOStandard,alterCondition,
                stand,<act-id,exp,log-op>
              ),
              _,char-types,_,_
            )


getAdaptedProcess(reg-AdaptationModel(_,procs,_,_,_),proc-id)
= ICS(ATOAdaptedProcesses,getProcessModel,procs,<proc-id>)


createAdaptedProcess(reg-AdaptationModel(_,procs,_,_,_),proc-id,proc-
mod)
= reg-AdaptationModel(
    _,ICS(ATOAdaptedProcesses,include,procs,<proc-id,proc-mod>),_,_,_
  )
```

```
updateAdaptedProcess(reg-AdaptationModel(stand,procs,char-
types,projs,cases),proj-id,proc-mod)
= if not ICS(ATOAdaptedProcesses,exist,procs,
             <ICS(ATOCases,getProcessId,cases,<proj-id>)>)
  then reg-AdaptationModel(stand,procs,char-types,projs,cases)
  else if ICS(ATOCases,wasApplied,cases,<proj-id>)
       then reg-AdaptationModel(stand,procs,char-types,projs,cases)
       else if ICS(ATOAbsProcessModel,isEmpty,proc-mod)
            then reg-AdaptationModel(
                   stand,procs,char-types,projs,cases
                 )
            else reg-AdaptationModel(
                   stand,
                   ICS(ATOAdaptedProcesses,alter,procs,<
                     ICS(ATOCases,getProcessId,cases,<proj-id>),
                     proc-mod
                   >),
                   char-types,
                   projs,
                   ICS(ATOCases,setModifications,cases,<
                     proj-id,
                     updateModifications(
                       ICS(ATOCases,getModifications,cases,<proj-id>),
                       ICS(ATOAbsActivitivies,getActivitiesIds,
                         ICS(ATOAbsProcessModel,getActivities,proc-id)
                       )
                       U
                       ICS(ATOAbsActivitivies,getActivitiesIds,
                         ICS(ATOAbsProcessModel,getActivities,
                           ICS(ATOAdaptedProcesses,getProcessModel,
                             procs,
                             <ICS(ATOCases,getProcessId,
                                cases,<proj-id>
                             )
                           >)
                         )
                       ),
                       ICS(ATOAbsProcessModel,getActivities,
                         proc-mod
```

```
                        ),
                        ICS(ATOAbsProcessModel,getActivities,
                          ICS(ATOAdaptedProcesses,getProcessModel,
                            procs,
                            <ICS(ATOCases,getProcessId,
                               cases,<proj-id>
                             )
                           >)
                        ),
                        ICS(ATOAbsProcessModel,getActivities,
                          ICS(ATOStandard,getProcessModel,stand)
                        ),
                        ICS(ATOStandard,getRules,stand),
                        projs,
                        proj-id,
                        char-types
                      )
                    >)
                  )

updateModifications(mods,emptyset,_,_,_,_,_,_,_)
= mods

updateModifications(mods,add(act-id,acts),acts-new,acts-
old,stand,rules,projs,proj-id,char-types)
= if ICS(ATOAbsActivities,exist,acts-new,<act-id>) and
     ICS(ATOAbsActivities,exist,acts-old,<act-id>)
  then //→ Manutenção da Atividade
       updateModifications(
         updateModification(
           mods,
           act-id,
           ,
           reason?,
           ICS(ATOAbsActivities,getDescription,acts-new,<act-id>),
           ICS(ATOAbsActivities,getDescription,acts-old,<act-id>),
           stand,
           rules,
           projs,
           proj-id,
```

```
          char-types
        ),
        acts,
        acts-new,
        acts-old,
        stand,
        rules,
        projs,
        proj-id,
        char-types
      )
 else //→ Inclusão da Atividade
      if ICS(ATOAbsActivities,exist,acts-new,<act-id>)
      then updateModifications(
             updateModification(
               mods,
               act-id,
               ModKind-Inclusion(“”),
               reason?,
               ICS(ATOAbsActivities,getDescription,
                 acts-new,<act-id>
               ),
               ,
               stand,
               rules,
               projs,
               proj-id,
               char-types
             ),
             acts,
             acts-new,
             acts-old,
             stand,
             rules,
             projs,
             proj-id,
             char-types
           )
      else //→ Exclusão da Atividade
```

```
            updateModifications(
              updateModification(
                mods,
                act-id,
                ModKind-Exclusion(“”),
                reason?,
                ,
                ICS(ATOAbsActivities,getDescription,
                  acts-old,<act-id>
                ),
                stand,
                rules,
                projs,
                proj-id,
                char-types
              ),
              acts,
              acts-new,
              acts-old,
              stand,
              rules,
              projs,
              proj-id,
              char-types
            )

updateModification(mods,act-id,mod-kind,reason,desc-new,desc-
old,stand,rules,projs,proj-id,char-types)
= if not ICS(ATOAbsActivities,exist,stand,<act-id>)
  then ICS(ATOModifications,include,
         ICS(ATOModifications,exclude,mods,<act-id>),
         <act-id,mod-kind,reason>
       )
  else if not ICS(ATORules,exist,rules,<act-id>)
       then if ICS(ATOModKind,isExclusion,mod-kind)
            then ICS(ATOModifications,include,
                   ICS(ATOModifications,exclude,mods,<act-id>),
                   <act-id,mod-kind,reason>
                 )
            else if not ICS(ATOActivityDescription,
```

```
          isFragment,desc-new
        )
then ICS(ATOModifications,exclude,mods,<act-id>)
else if ICS(ATOModKind,isInclusion,mod-kind)
     then updateModifications(
            ICS(ATOModifications,exclude,
              mods,<act-id>
            ),
            ICS(ATOAbsActivities,getActivitiesIds,
              ICS(ATOAbsProcessModel,getActivities,
              ICS(ATOActivityDescription,getFragment,
                desc-new
              )
              )
            ),
            ICS(ATOAbsProcessModel,getActivities,
              ICS(ATOActivityDescription,
                getFragment,
                desc-new
              )
            ),
            emptymap,
            ICS(ATOAbsProcessModel,getActivities,
              ICS(ATOActivityDescription,
                getFragment,
                ICS(ATOAbsActivities,
                  getActivityDescription,
                  stand,
                  <act-id>
                )
              )
            ),
            rules,
            projs,
            proj-id,
            char-types
          )
     else updateModifications(
            ICS(ATOModifications,
              exclude,
```

```
  mods,
  <act-id>
),
ICS(ATOAbsActivities,getActivitiesIds,
  ICS(ATOAbsProcessModel,getActivities,
    ICS(ATOActivityDescription,
      getFragment,
      desc-new
    )
  )
)
U
ICS(ATOAbsActivities,getActivitiesIds,
  ICS(ATOAbsProcessModel,getActivities,
    ICS(ATOActivityDescription,
      getFragment,
      desc-old
    )                              )
),
ICS(ATOAbsProcessModel,getActivities,
  ICS(ATOActivityDescription,
    getFragment,
    desc-new
  )
),
ICS(ATOAbsProcessModel,getActivities,
  ICS(ATOActivityDescription,
    getFragment,
    desc-old
  )
),
ICS(ATOAbsProcessModel,getActivities,
  ICS(ATOActivityDescription,getFragment,
    ICS(ATOAbsActivities,
      getActivityDescription,
      stand,
      <act-id>
    )
  )
),
```

```
                              rules,
                              projs,
                              proj-id,
                              char-types
                            )
                       fi
                  fi
             fi
        else if evaluateRule(rules,act-id,projs,proj-id,char-types) ==
                true
             then if ICS(ATOModKind,isExclusion,mod-kind)
                  then ICS(ATOModifications,include,
                         ICS(ATOModifications,exclude,mods,<act-id>),
                         <act-id,mod-kind,reason>
                       )
                  else ICS(ATOModifications,exclude,mods,<act-id>)
                  fi
             else if ICS(ATOModKind,isExclusion,mod-kind)
                  then ICS(ATOModifications,exclude,mods,<act-id>)
                  else ICS(ATOModifications,include,
                         ICS(ATOModifications,exclude,mods,<act-id>),
                         <act-id,mod-kind,reason>
                       )
                  fi
             fi
        fi
  fi

includeActivity(reg-AdaptationModel(stand,procs,_,_,_),proc-id,act-id)
= reg-AdaptationModel(
    stand,
    ICS(ATOAdaptedProcesses,
      includeActivity,
      procs,
      <proc-id,getStandard(stand),act-id>
    ),
    _,_,_
  )

excludeActivity(_,procs,_,_,_),proc-id,act-id)
```

```
= reg-AdaptationModel(
    _,
    ICS(ATOAdaptedProcesses,excludeActivity,procs,<proc-id,act-id>)
    ,_,_,_
  )


includeCharacteristicType(reg-AdaptationModel(_,_,char-
types,_,_),char-id,name,desc,char-kind)
= reg-AdaptationModel(
    _,_,
    ICS(ATOCharacteristicTypes,include,char-types,<
      char-id,name,desc,char-kind
    >),
    _,_
  )


excludeCharacteristicType(reg-AdaptationModel(stand,_,char-
types,projs,_),char-id)
= if ICS(ATOStandard,isCharacteristicBeenUsed,stand,<char-id>)
  then reg-AdaptationModel(stand,_,char-types,projs,_)
  else if ICS(ATOProjects,isCharacteristicBeenUsed,projs,<char-id>)
       then reg-AdaptationModel(stand,_,char-types,projs,_)
       else reg-AdaptationModel(
              stand,
              _,
              ICS(ATOCharacteristicTypes,exclude,
                char-types,<char-id>
              ),
              projs,
              _
            )


alterCharacteristicType(reg-AdaptationModel(stand,_,char-
types,projs,_),char-id,name,desc,char-kind)
= if ICS(ATOStandard,isCharacteristicBeenUsed,stand,<char-id>)
  then reg-AdaptationModel(stand,_,char-types,projs,_)
  else if ICS(ATOProjects,isCharacteristicBeenUsed,projs,<char-id>)
       then reg-AdaptationModel(stand,_,char-types,projs,_)
       else reg-AdaptationModel(
              stand,
              _,
```

```
              ICS(ATOCharacteristicTypes,alter,char-types,<
                char-id,name,desc,char-kind
              >),
              projs,
              _
            )

includeValue(reg-AdaptationModel(stand,_,char-types,projs,_),char-
id,value)
= if ICS(ATOStandard,isCharacteristicBeenUsed,stand,<char-id>)
  then reg-AdaptationModel(stand,_,char-types,projs,_)
  else if ICS(ATOProjects,isCharacteristicBeenUsed,projs,<char-id>)
       then reg-AdaptationModel(stand,_,char-types,projs,_)
       else reg-AdaptationModel(
              stand,
              _,
              ICS(ATOCharacteristicTypes,includeValue,char-types,<
                char-id,value
              >),
              projs,
              _
            )

excludeValue(reg-AdaptationModel(stand,_,char-types,projs,_),char-
id,value)
= if ICS(ATOStandard,isCharacteristicBeenUsed,stand,<char-id>)
  then reg-AdaptationModel(stand,_,char-types,projs,_)
  else if ICS(ATOProjects,isCharacteristicBeenUsed,projs,<char-id>)
       then reg-AdaptationModel(stand,_,char-types,projs,_)
       else reg-AdaptationModel(
              stand,
              _,
              ICS(ATOCharacteristicTypes,excludeValue,char-types,<
                char-id,value
              >),
              projs,
              _
            )

includeProject(reg-AdaptationModel(_,_,_,projs,_),proj-
id,name,desc,manager,start-dt,end-dt)
```

```
= reg-AdaptationModel(
    _’_’_’
    ICS(ATOProjects,include,
      projs,<proj-id,name,desc,manager,start-dt,end-dt>
    ),
    _
  )

excludeProject(reg-AdaptationModel(_,_,_,projs,cases),proj-id)
= if ICS(ATOCases,exist,cases,<proj-id>)
  then reg-AdaptationModel(_,_,_,projs,cases)
  else reg-AdaptationModel(
         _’_’_’
         ICS(ATOProjects,exclude,projs,<proj-id>),
         cases
       )

alterProject(reg-AdaptationModel(_,_,_,projs,cases),proj-id,name,desc,
manager,start-dt,end-dt)
= if ICS(ATOCases,exist,cases,<proj-id>)
  then reg-AdaptationModel(_,_,_,projs,cases)
  else reg-AdaptationModel(
         _’_’_’
         ICS(ATOProjects,alter,projs,<
           proj-id,name,desc,manager,start-dt,end-dt
         >),
         cases
       )

includeCharacteristic(reg-AdaptationModel(_,_,char-
types,projs,cases),proj-id,char-id,value,weight)
= if ICS(ATOCases,exist,cases,<proj-id>)
  then reg-AdaptationModel(_,_,char-types,projs,cases)
  else if not ICS(ATOCharacteristicTypes,exist,char-types,<char-id>)
       then reg-AdaptationModel(_,_,char-types,projs,cases)
       else reg-AdaptationModel(
              _’_’
              char-types,
              ICS(ATOProjects,includeCharacteristic,projs,<
                proj-id,char-id,value,weight
```

```
            >),
            cases
          )
excludeCharacteristic(reg-AdaptationModel(_,_,_,projs,cases),proj-
id,char-id)
= if ICS(ATOCases,exist,cases,<proj-id>)
  then reg-AdaptationModel(_,_,_,projs,cases)
  else reg-AdaptationModel(
         _,_,_,
         ICS(ATOProjects,excludeCharacteristic,
           projs,<proj-id,char-id>
         ),
         cases
       )

alterCharacteristic(reg-AdaptationModel(_,_,_,projs,cases),proj-
id,char-id,value,weight)
= if ICS(ATOCases,exist,cases,<proj-id>)
  then reg-AdaptationModel(_,_,_,projs,cases)
  else reg-AdaptationModel(
         _,_,_,
         ICS(ATOProjects,alterCharacteristic,projs,<
           proj-id,char-id,value,weight
         >),
         cases
       )

excludeCase(reg-AdaptationModel(_,procs,_,projs,cases),proj-id)
= if not ICS(ATOCases,exist,cases,<proj-id>)
  then reg-AdaptationModel(_,procs,_,projs,cases)
  else if not ICS(ATOCases,wasApplied,cases,<proj-id>)
       then reg-AdaptationModel(
              _,
              ICS(ATOAdaptedProcesses,exclude,procs,<
                ICS(ATOCases,getProcessId,cases,<proj-id>)
              >),
              _,
              projs,
              ICS(ATOCases,exclude,cases,<proj-id>)
            )
```

```
       else reg-AdaptationModel(
              _,
              ICS(ATOAdaptedProcesses,exclude,procs,<
                ICS(ATOCases,getProcessId,cases,<proj-id>)
              >),
              _,
              ICS(ATOProjects,exclude,projs,<proj-id>),
              ICS(ATOCases,exclude,cases,<proj-id>)
            )

commentCase(reg-AdaptationModel(_,_,_,_,cases),proj-id,comm)
= reg-AdaptationModel(
    _,_,_,_,
    ICS(ATOCases,setComments,cases,<proj-id,comm>)
  )

avaliateCase(reg-AdaptationModel(_,_,_,_,cases),proj-id,aval)
= reg-AdaptationModel(
    _,_,_,_,
    ICS(ATOCases,setAvaliation,cases,<proj-id,aval>)
  )

showTrace(reg-AdaptationModel(_,_,_,_,cases),proj-id)
= ICS(ATOCases,getTrace,cases,<proj-id>)
```

A.2 ATOStandard

Abaixo é apresentada uma descrição de alto nível para as principais operações deste ATO:

- *getProcessModel*: retorna o modelo do processo padrão;
- *getRules*: retorna o conjunto de regras de adaptação do processo padrão;
- *new*: define um novo processo padrão;
- *update*: atualiza o processo padrão;
- *includeRule*: associa uma regra de adaptação a uma atividade do processo padrão;
- *excludeRule*: exclui uma regra de adaptação do processo padrão;
- *alterRule*: altera os dados de uma regra de adaptação do processo padrão;
- *includeCondition*: inclui uma condição em uma regra de adaptação do processo padrão;

- *excludeCondition*: exclui uma condição de uma regra de adaptação do processo padrão;
- *alterCondition*: altera uma condição de uma regra de adaptação do processo padrão;
- *isCharacteristicBeenUsed*: verificar se alguma regra de adaptação utiliza o tipo de característica em questão.

CLASS

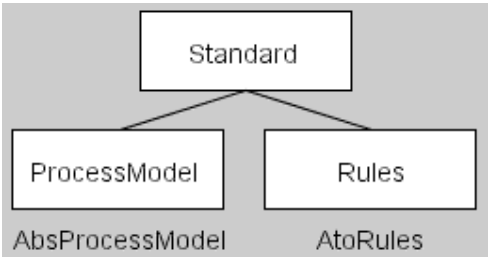


Figura A.2: Classe que define o ATOStandard

INCLUDE

INTERFACES

```
getProcessModel : Standard → AbsProcessModel

getRules : Standard → Rules

new : AbsProcessModel → Standard

update: Standard AbsProcessModel → Standard

includeRule : Standard String Text RuleKind → Standard

excludeRule : Standard String → Standard

alterRule : Standard String Text RuleKind → Standard
```

includeCondition : Standard String Expression LogicOp → Standard

excludeCondition : Standard String String → Standard

alterCondition : Standard String Expression LogicOp → Standard

isCharacteristicBeenUsed : Standard String → Boolean

FORMAL VARIABLES

```
stand : Standard
proc-mod: AbsProcessModel
act-id char-id : String
comm : Text
rule-kind : RuleKind
exp : Expression
log-op : LogicOp
```

AXIOMS

```
getProcessModel(stand)
= select-ProcessModel(stand)

getRules(stand)
= select-Rules(stand)

new(proc-mod)
= reg-Standard(proc-mod,emptymap)

update(stand,proc-mod)
= reg-Standard(proc-mod,
               ICS(ATORules,
                   update,
                   getRules(stand),
                  <ICS(ATOAbsProcessModel,getActivities,proc-mod)>))
```

```
includeRule(stand,act-id,comm,rule-kind)
= if not
ICS(ATOAbsProcessModel,hasActivity,getProcessModel(stand),<act-id>)
  then stand
  else reg-Standard(getProcessModel(stand),
                    ICS(ATORules,
                        include,
                        getRules(stand),
                        <act-id,comm,rule-kind>))

excludeRule(stand,act-id)
= reg-Standard(getProcessModel(stand),
               ICS(ATORules,exclude,getRules(stand),<act-id>))

alterRule(stand,act-id,comm,rule-kind)
= reg-Standard(
    getProcessModel(stand),
    ICS(ATORules,alter,getRules(stand),<act-id,comm,rule-kind>)
  )

includeCondition(stand,act-id,exp,log-op)
= reg-Standard(getProcessModel(stand),
               ICS(ATORules,
                   includeCondition,
                   getRules(stand),
                   <act-id,exp,log-op>))

excludeCondition(stand,act-id,char-id)
= reg-Standard(getProcessModel(stand),
               ICS(ATORules,
                   excludeCondition,
                   getRules(stand),
                   <act-id,char-id>))

alterCondition(stand,act-id,exp,log-op)
= reg-Standard(getProcessModel(stand),
               ICS(ATORules,
                   alterCondition,
                   getRules(stand),
                   <act-id,exp,log-op>))
```

```
isCharacteristicBeenUsed(stand,char-id)
= ICS(ATORules,isCharacteristicBeenUsed,getRules(stand),<char-id>)
```

A.3 ATORules

Abaixo é apresentada uma descrição de alto nível para as principais operações deste ATO:

- *exist*: verifica se uma atividade possui uma regra de adaptação associada;
- *getComments*: retorna os comentários da regra de adaptação;
- *getKind*: retorna o tipo da regra de adaptação;
- *getConditions*: retorna o conjunto de condições da regra de adaptação;
- *include*: inclui uma regra de adaptação;
- *exclude*: exclui uma regra de adaptação;
- *alter*: altera os dados de uma regra de adaptação;
- *update*: atualiza o conjunto de regras de adaptação;
- *includeCondition*: inclui uma condição em uma regra de adaptação;
- *excludeCondition*: exclui uma condição de uma regra de adaptação;
- *alterCondition*: altera uma condição de uma regra de adaptação;
- *isCharacteristicBeenUsed*: verifica se a característica está sendo utilizada em alguma regra.

CLASS

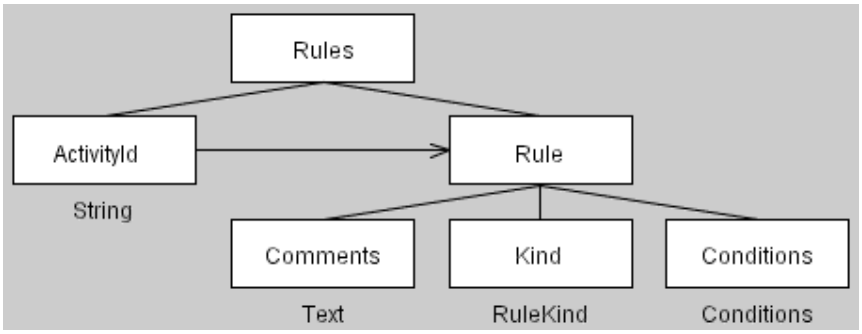


Figura A.3: Classe que define o ATORules

INCLUDE

Boolean

INTERFACES

exist : Rules String → Boolean

getComments : Rules String → Text

getKind : Rules String → RuleKind

getConditions : Rules String → Conditions

include : Rules String Text RuleKind → Rules

exclude : Rules String → Rules

alter : Rules String Text RuleKind → Rules

update : Rules SetOfString → Rules

includeCondition : Rules String Expression LogicOp → Rules

excludeCondition : Rules String String → Rules

alterCondition : Rules String Expression LogicOp → Rules

isCharacteristicBeenUsed : Rules String → Boolean

FORMAL VARIABLES

rules : Rules
act-id char-id : String
acts : SetOfString
comm : Text
rule-kind : RuleKind

```
conds : Conditions
exp : Expression
log-op : LogicOp
```

AXIOMS

```
exist(rules,act-id)
= act-id belongsto domain(rules)

getComments(rules,act-id)
= if exist(rules,act-id)
  then select-Comments(imageof(act-id,rules))

getKind(rules,act-id)
= if exist(rules,act-id)
  then select-Kind(imageof(act-id,rules))

getConditions(rules,act-id)
= if exist(rules,act-id)
  then select-Conditions(imageof(act-id,rules))

include(rules,act-id,comm,rule-kind)
= if exist(rules,act-id)
  then rules
  else if not ICS(ATORuleKind,isSelected,rule-kind)
       then rules
       else modify(act-id,
                   reg-Rule(
                     comm,
                     rule-kind,
                     ICS(ATOConditions,emptyConditions)),
                   rules)

exclude(rules,act-id)
= if not exist(rules,act-id)
  then rules
  else restrictwith(rules,add(act-id,emptyset))
```

```
alter(rules,act-id,comm,rule-kind)
= if not exist(rules,act-id)
  then rules
  else if not ICS(ATORuleKind,isSelected,rule-kind)
       then rules
       else override(rules,
                     modify(act-id,
                            reg-Rule(
                              comm,
                              rule-kind,
                              getConditions(rules,act-id)),
                            emptymap))

update(rules,acts)
= restrictto(rules,acts)

includeCondition(rules,act-id,exp,log-op)
= if not exist(rules,act-id)
  then rules
  else override(rules,
                modify(act-id,
                       reg-Rule(getComments(rules,act-id),
                                getKind(rules,act-id),
                                ICS(ATOConditions,
                                    include,
                                    getConditions(rules,act-id),
                                    <exp,log-op>)),
                       emptymap))

excludeCondition(rules,act-id,char-id)
= if not exist(rules,act-id)
  then rules
  else override(rules,
                modify(act-id,
                       reg-Rule(getComments(rules,act-id),
                                getKind(rules,act-id),
                                ICS(ATOConditions,
                                    exclude,
                                    getConditions(rules,act-id),
                                    <char-id>)),
```

```
                        emptymap))

alterCondition(rules,act-id,exp,log-op)
= if not exist(rules,act-id)
    then rules
    else override(rules,
                  modify(act-id,
                         reg-Rule(getComments(rules,act-id),
                                  getKind(rules,act-id),
                                  ICS(ATOConditions,
                                      alter,
                                      getConditions(rules,act-id),
                                      <exp,log-op>)),
                         emptymap))

isCharacteristicBeenUsed(emptymap,_)
= false

isCharacteristicBeenUsed(modify(_,reg-Rule(_,_,conds),rules),char-id)
= if ICS(ATOConditions,isCharacteristicBeenUsed,conds,<char-id>)
  then true
  else isCharacteristicBeenUsed(rules,char-id)
```

A.4 ATORuleKind

Abaixo é apresentada uma descrição de alto nível para as principais operações deste ATO:

- *isPositive*: verifica se o tipo de regra é positivo;
- isNegative: verifica se o tipo de regra é negativo;
- *isSelected*: verifica se o tipo de regra está selecionado.

CLASS

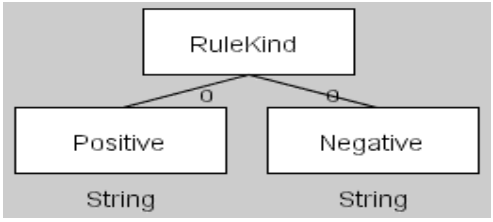


Figura A.4: Classe que define o ATORuleKind

INCLUDE

```
Boolean
```

INTERFACES

```
isPositive : RuleKind → Boolean

isNegative : RuleKind → Boolean

isSelected : RuleKind → Boolean
```

FORMAL VARIABLES

```
rule-kind : RuleKind
```

AXIOMS

```
isPositive(rule-kind)
= is-Positive(rule-kind)

isNegative(rule-kind)
= is-Negative(rule-kind)

isSelected(rule-kind)
= isPositive(rule-kind) xor isNegative(rule-kind)
```

A.5 ATOConditions

Abaixo é apresentada uma descrição de alto nível para as principais operações deste ATO:

- *getExpression*: retorna a primeira condição;
- *hasOptionalConds*: verifica se o conjunto de condições restantes existe;
- *getLogicOp*: retorna o primeiro operador lógico;
- *getConditions*: retorna o conjunto de condições restantes;
- *isEmpty*: verifica se o conjunto de condições é vazio;
- *exist*: verifica se determinada condição existe;
- *include*: inclui uma condição;
- *exclude*: exclui uma condição;
- *alter*: altera uma condição.

CLASS

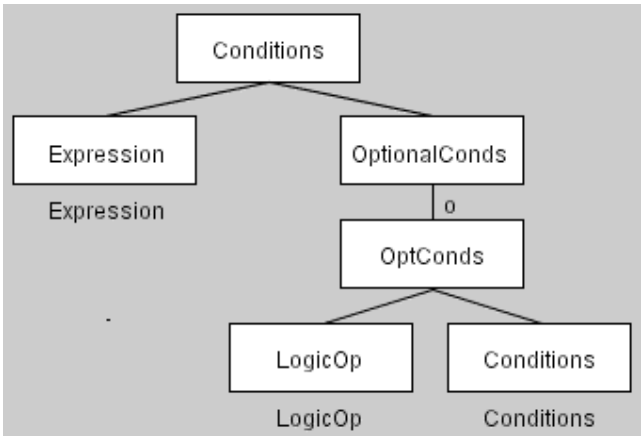


Figura A.5: Classe que define o ATOConditions

INCLUDE

```
Boolean
```

INTERFACES

getExpression : Conditions → Expression

hasOptionalConds : Conditions → Boolean

getLogicOp : Conditions → LogicOp

getConditions : Conditions → Conditions

isEmpty : Conditions → Boolean

exist : Conditions String → Boolean

include : Conditions Expression LogicOp → Conditions

inc : Conditions Expression LogicOp → Conditions

exclude : Conditions String → Conditions

exc : Conditions String → Conditions

alter : Conditions Expression LogicOp → Conditions

alt : Conditions Expression LogicOp → Conditions

FORMAL VARIABLES

conds : Conditions
exp : Expression
log-op : LogicOp
char-id : String

AXIOMS

```
getExpression(conds)
= select-Expression(conds)

hasOptionalConds(conds)
= is-OptConds(select-OptionalConds(conds))

getLogicOp(conds)
= if hasOptionalConds(conds)
  then select-LogicOp(get-OptConds(select-OptionalConds(conds)))

getConditions(conds)
= if hasOptionalConds(conds)
  then select-Conditions(get-OptConds(select-OptionalConds(conds)))

isEmpty(conds)
= ICS(ATOExpression,isEmpty,getExpression(conds))

exist(conds,char-id)
= if isEmpty(conds)
  then false
  else if ICS(ATOExpression,getCharTypeId,getExpression(conds))
          == char-id
       then true
       else if not hasOptionalConds(conds)
            then false
            else exist(getConditions(conds),char-id)

include(conds,exp,log-op)
= if ICS(ATOExpression,isEmpty,exp)
  then conds
  else if exist(conds,ICS(ATOExpression,getCharTypeId,exp))
       then conds
       else inc(conds,exp,log-op)

inc(conds,exp,log-op)
= if isEmpty(conds)
  then reg-Conditions(exp,)
  else if not ICS(ATOLogicOp,isSelected,log-op)
       then conds
       else if not hasOptionalConds(conds)
```

```
            then reg-Conditions(
                   getExpression(conds),
                   OptionalConds-OptConds(
                     reg-OptConds(
                       log-op,
                       reg-Conditions(exp,)
                     )
                   )
                 )
            else reg-Conditions(
                   getExpression(conds),
                   OptionalConds-OptConds(
                     reg-OptConds(
                       getLogicOp(conds),
                       inc(getConditions(conds),exp,log-op)
                     )
                   )
                 )

exclude(conds,char-id)
= if not exist(conds,char-id)
  then conds
  else exc(conds,char-id)

exc(conds,char-id)
= if ICS(ATOExpression,getCharTypeId,getExpression(conds)) == char-id
  then if not hasOptionalConds(conds)
       then reg-Conditions(ICS(ATOExpresion,create,<"",,>),)
       else getConditions(conds)
       fi
  else if ICS(ATOExpression,getCharTypeId,
            getExpression(getConditions(conds))
          ) == char-id
       then reg-Conditions(
              getExpression(conds),
              select-OptionalConds(getConditions(conds)))
       else reg-Conditions(
              getExpression(conds),
              OptionalConds-OptConds(
                reg-OptConds(
```

```
                getLogicOp(conds),
                exc(getConditions(conds),char-id)
              )
            )
          )
      fi
  fi

alter(conds,exp,log-op)
= if ICS(ATOExpression,isEmpty,exp)
  then conds
  else if not exist(conds,ICS(ATOExpression,getCharTypeId,exp))
       then conds
       else alt(conds,exp,log-op)

alt(conds,exp,log-op)
= if ICS(ATOExpression,getCharTypeId,getExpression(conds)) ==
     ICS(ATOExpression,getCharTypeId,exp)
  then reg-Conditions(exp,select-OptionalConds(conds))
  else if not ICS(ATOLogicOp,isSelected,log-op)
       then conds
       else if ICS(ATOExpression,getCharTypeId,
                   getExpression(getConditions(conds))
                ) == ICS(ATOExpression,getCharTypeId,exp)
            then reg-Conditions(
                   getExpression(conds),
                   OptionalConds-OptConds(
                     reg-OptConds(
                       log-op,
                       reg-Conditions(
                         exp,
                         select-OptionalConds(getConditions(conds))
                       )
                     )
                   )
                 )
            else reg-Conditions(
                   getExpression(conds),
                   OptionalConds-OptConds(
                     reg-OptConds(
```

```
                    getLogicOp(conds),
                    alt(getConditions(conds),exp,log-op)
                  )
                )
              )
          fi
      fi
```

A.6 ATOExpression

Abaixo é apresentada uma descrição de alto nível para as principais operações deste ATO:

- *getCharTypeId*: retorna o tipo de característica;
- *getComparisonOp*: retorna o operador relacional;
- *getValue*: retorna o valor;
- *isEmpty*: verifica se expressão é vazia;
- *create*: cria uma expressão.

CLASS

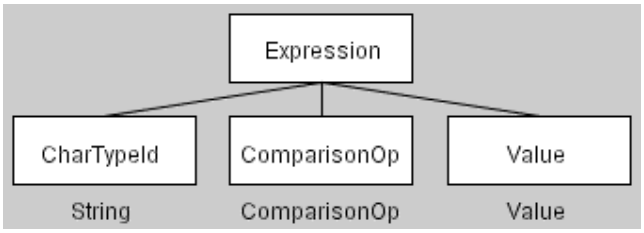


Figura A.6: Classe que define o ATOExpression

INCLUDE

```
Boolean
```

INTERFACES

```
getCharTypeId : Expression → String
```

```
getComparisonOp : Expression → ComparisonOp

getValue : Expression → Value

isEmpty : Expression → Boolean

create : String ComparisonOp Value → Expression
```

FORMAL VARIABLES

```
char-id : String
comp-op : ComparisonOp
value : Value
exp : Expression
```

AXIOMS

```
getCharTypeId(exp)
= select-CharTypeId(exp)

getComparisonOp(exp)
= select-ComparisonOp(exp)

getValue(exp)
= select-Value(exp)

isEmpty(exp)
= length(repl(" ","",getCharTypeId(exp))) == 0 or
  not ICS(ATOComparisonOp,isSelected,getComparisonOp(exp)) or
  not ICS(ATOValue,isEmpty,getValue(exp))

create(char-id,comp-op,value)
= reg-Expression(char-id,comp-op,value)
```

A.7 ATOComparisonOp

Abaixo é apresentada uma descrição de alto nível para as principais operações deste ATO:

- *isEqual*: verifica se o operador "=" está selecionado;
- *isDifferent*: verifica se o operador "<>" está selecionado;
- *isGreater*: verifica se o operador ">" está selecionado;
- *isGreaterEqual*: verifica se o operador ">=" está selecionado;
- *isSmaller*: verifica se o operador "<" está selecionado;
- *isSmallerEqual*: verifica se o operador "<=" está selecionado;
- *isBetween*: verifica se o operador "between" está selecionado;
- *isNotBetween*: verifica se o operador "not between" está selecionado;
- *isContains*: verifica se o operador "contains" está selecionado;
- *isNotContains*: verifica se o operador "not contains" está selecionado;
- *isSelected*: verifica algum operador está selecionado.

CLASS

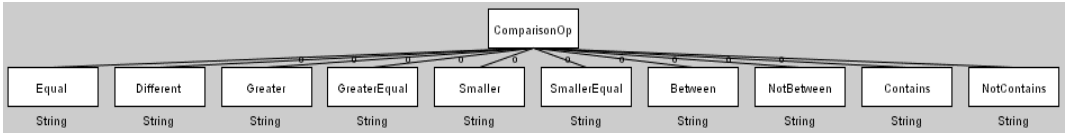


Figura A.7: Classe que define o ATOComparisonOp

INCLUDE

```
Boolean
```

INTERFACES

```
isEqual : ComparisonOp → Boolean

isDifferent : ComparisonOp → Boolean
```

isGreater : ComparisonOp → Boolean

isGreaterEqual : ComparisonOp → Boolean

isSmaller : ComparisonOp → Boolean

isSmallerEqual : ComparisonOp → Boolean

isBetween : ComparisonOp → Boolean

isNotBetween : ComparisonOp → Boolean

isContains : ComparisonOp → Boolean

isNotContains : ComparisonOp → Boolean

isSelected: ComparisonOp → Boolean

FORMAL VARIABLES

comp-op : ComparisonOp

AXIOMS

isEqual(comp-op)
= is-Equal(comp-op)

isDifferent(comp-op)
= is-Different(comp-op)

isGreater(comp-op)
= is-Greater(comp-op)

isGreaterEqual(comp-op)
= is-GreaterEqual(comp-op)

```
isSmaller(comp-op)
= is-Smaller(comp-op)

isSmallerEqual(comp-op)
= is-SmallerEqual(comp-op)

isBetween(comp-op)
= is-Between(comp-op)

isNotBetween(comp-op)
= is-NotBetween(comp-op)

isContains(comp-op)
= is-Contains(comp-op)

isNotContains(comp-op)
= is-NotContains(comp-op)

isSelected(comp-op)
= isEqual(comp-op) xor
  isDifferent(comp-op) xor
  isGreater(comp-op) xor
  isGreaterEqual(comp-op) xor
  isSmaller(comp-op) xor
  isSmallerEqual(comp-op) xor
  isBetween(comp-op) xor
  isNotBetween(comp-op) xor
  isContains(comp-op) xor
  isNotContains(comp-op)
```

A.8 ATOLogicOp

Abaixo é apresentada uma descrição de alto nível para as principais operações deste ATO:

- *isAnd*: verifica se o operador “And” está selecionado;
- *isOr*: verifica se o operador “Or” está selecionado;
- *isSelected*: verifica se algum operador lógico está selecionado.

CLASS

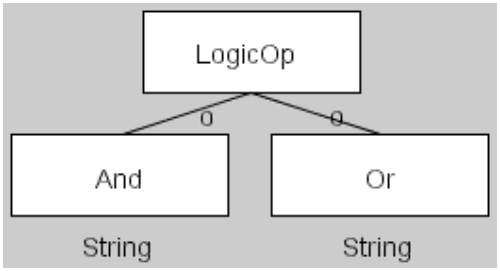


Figura A.8: Classe que define o ATOLogicOp

INCLUDE

```
Boolean
```

INTERFACES

```
isAnd : LogicOp → Boolean

isOr : LogicOp → Boolean

isSelected : LogicOp → Boolean
```

FORMAL VARIABLES

```
log-op : LogicOp
```

AXIOMS

```
isAnd(log-op)
= is-And(log-op)

isOr(log-op)
```

```
= is-Or(log-op)

isSelected(log-op)
= isAnd(log-op) xor isOr(log-op)
```

A.9 ATOAdaptedProcesses

Abaixo é apresentada uma descrição de alto nível para as principais operações deste ATO:

- *exist*: verifica se o processo adaptado existe;
- *getProcessModel*: retorna o modelo do processo adaptado;
- *include*: cria o processo adaptado;
- *exclude*: exclui o processo adaptado;
- *alter*: modifica um processo adaptado já existente;
- *includeActivity*: inclui uma atividade no processo adaptado;
- *excludeActivity*: exclui uma atividade do processo adaptado.

CLASS

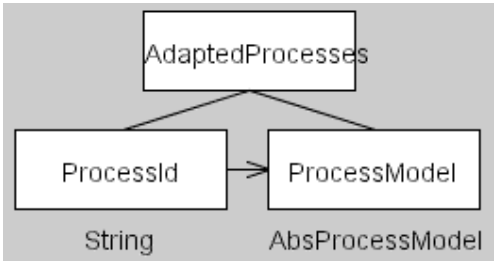


Figura A.9: Classe que define o ATOAdaptecProcesses

INCLUDE

INTERFACES

```
exist : AdaptedProcesses String → Boolean

getProcessModel : AdaptedProcesses String → AbsProcessModel
```

```
include : AdaptedProcesses String AbsProcessModel → AdaptedProcesses

exclude : AdaptedProcesses String → AdaptedProcesses

alter : AdaptedProcesses String AbsProcessModel → AdaptedProcesses

includeActivity : AdaptedProcesses String AbsProcessModel String →
                  AdaptedProcesses

excludeActivity : AdaptedProcesses String String → AdaptedProcesses
```

FORMAL VARIABLES

```
procs : Processes
proc-id act-id : String
proc-mod : ProcessModel
```

AXIOMS

```
exist(procs,proc-id)
= proc-id belongsto domain(procs)

getProcessModel(procs,proc-id)
= if exist(procs,proc-id)
  then select-ProcessModel(imageof(proc-id,procs))

include(procs,proc-id,proc-mod)
= if exist(procs,proc-id)
  then procs
  else modify(proc-id,reg-ProcessModel(proc-mod),procs)

exclude(procs,proc-id)
= if not exist(procs,proc-id)
  then procs
  else restrictwith(procs,add(proc-id,emptyset))
```

```
alter(procs,proc-id,proc-mod)
= if not exist(procs,proc-id)
  then procs
  else override(
         procs,modify(proc-id,reg-ProcessModel(proc-mod),emptymap)
       )

includeActivity(procs,proc-id,proc-mod,act-id)
= if not exist(procs,proc-id)
  then procs
  else alter(
         procs,
         proc-id,
         ICS(ATOAbsProcessModel,includeActivity,
           getProcessModel(procs,proc-id),
           <proc-mod,act-id>
         )
       )

excludeActivity(procs,proc-id,act-id)
= if not exist(procs,proc-id)
  then procs
  else alter(
         procs,
         proc-id,
         ICS(ATOAbsProcessModel,excludeActivity,
           getProcessModel(procs,proc-id),
           <act-id>
         )
       )
```

A.10 ATOCharacteristicTypes

Abaixo é apresentada uma descrição de alto nível para as principais operações deste ATO:

- *exist*: verifica se o tipo de característica existe;
- *getName*: retorna o nome do tipo de característica;
- *getDescription*: retorna a descrição do tipo de característica;

- *getKind*: retorna a espécie do tipo de característica;
- *include*: inclui o tipo de característica;
- *exclude*: exclui o tipo de característica;
- *alter*: altera as informações do tipo de característica;
- *includeValue*: inclui um valor possível ao tipo de característica;
- *excludeValue*: exclui um valor possível do tipo de característica.

CLASS

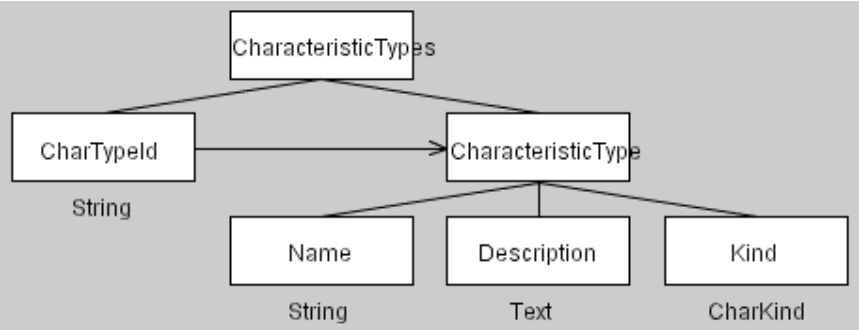


Figura A.10: Classe que define o ATOCharacteristicTypes

INCLUDE

```
Boolean
```

INTERFACES

```
getName : CharacteristicTypes String → String

getDescription : CharacteristicTypes String → Text

getKind : CharacteristicTypes String → CharKind

exist : CharacteristicTypes String → Boolean

include : CharacteristicTypes String String Text CharKind →
```

```
          CharacteristicTypes

exclude : CharacteristicTypes String → CharacteristicTypes

alter : CharacteristicTypes String String Text CharKind →
        CharacteristicTypes

includeValue : CharacteristicTypes String String →
               CharacteristicTypes

excludeValue : CharacteristicTypes String String →
               CharacteristicTypes
```

FORMAL VARIABLES

```
char-types : CharacteristicTypes
char-id name value : String
desc : Text
kind : CharKind
```

AXIOMS

```
exist(char-types,char-id)
= char-id belongsto domain(char-types)

getName(char-types,char-id)
= if exist(char-types,char-id)
  then select-Name(imageof(char-id,char-types))

getDescription(char-types,char-id)
= if exist(char-types,char-id)
  then select-Description(imageof(char-id,char-types))

getKind(char-types,char-id)
= if exist(char-types,char-id)
  then select-Kind(imageof(char-id,char-types))
```

```
include(char-types,char-id,name,desc,kind)
= if exist(char-types,char-id)
  then char-types
  else if not ICS(ATOCharKind,isSelected,kind)
       then char-types
       else modify(
             char-id,
             reg-CharacteristicType(name,desc,kind),
             char-types
           )

exclude(char-types,char-id)
= if not exist(char-types,char-id)
  then char-types
  else restrictwith(char-types,add(char-id,emptyset))

alter(char-types,char-id,name,desc,kind)
= if not exist(char-types,char-id)
  then char-types
  else if not ICS(ATOCharKind,isSelected,kind)
       then char-types
       else override(
              char-types,
              modify(
                char-id,
                reg-CharacteristicType(name,desc,kind),
                emptymap
              )
            )

includeValue(char-types,char-id,value)
= if not exist(char-types,char-id)
  then char-types
  else override(
         char-types,
         modify(
           char-id,
           reg-CharacteristicType(
             getName(char-types,char-id),
```

```
            getDescription(char-types,char-id),
            ICS(ATOCharKind,includeValue,
              getKind(char-types,char-id),
              <value>
            )
          ),
          emptymap
        )
      )

excludeValue(char-types,char-id,value)
= if not exist(char-types,char-id)
  then char-types
  else override(
         char-types,
         modify(
           char-id,
           reg-CharacteristicType(
             getName(char-types,char-id),
             getDescription(char-types,char-id),
             ICS(ATOCharKind,excludeValue,
               getKind(char-types,char-id),
               <value>
             )
           ),
           emptymap
         )
       )
```

A.11 ATOCharKind

Abaixo é apresentada uma descrição de alto nível para as principais operações deste ATO:

- *isMultiple*: verifica se a espécie do tipo de característica é múltipla;
- *isSingleWithoutLevel*: verifica se a espécie do tipo de característica é simples sem nivelamento;
- *isSingleWithLevel*: verifica se a espécie do tipo de característica é simples com nivelamento;
- *isSelected*: verifica se a espécie do tipo de característica está selecionada;

- *getValues*: retorna os valores possíveis do tipo de caraterística;
- *isEmpty*: verifica se a espécie do tipo de característica é vazia;
- *getLevelOf*: retorna o nível associado a um valor de uma característica simples com nivelamento;
- *getMaxLevel*: retorna o nível máximo de uma característica simples com nivelamento;
- *includeValue*: adiciona um valor possível ao tipo de característica;
- *excludeValue*: exclui um valor possível do tipo de característica.

CLASS

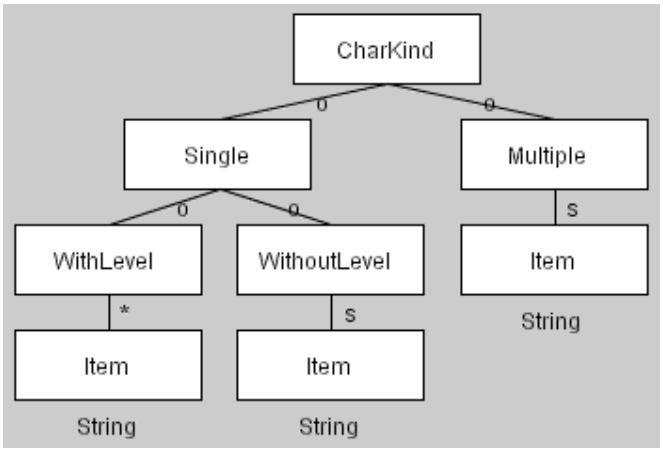


Figura A.11: Classe que define o ATOCharKind

INCLUDE

```
Boolean Integer
```

INTERFACES

```
isMultiple : CharKind → Boolean
```

```
isSingleWithoutLevel : CharKind → Boolean
```

```
isSingleWithLevel : CharKind → Boolean

isSelected : CharKind → Boolean

getValues : CharKind → SetOfString

isEmpty : CharKind → Boolean

getLevelOf : CharKind String → Integer

getMaxLevel : CharKind → Integer

includeValue : CharKind String → CharKind

excludeValue : CharKind String → CharKind.
```

FORMAL VARIABLES

```
char-kind : CharKind
item : String
```

AXIOMS

```
isMultiple(char-kind)
= is-Multiple(char-kind)

isSingleWithoutLevel(char-kind)
= if not is-Single(char-kind)
  then false
  else if not is-WithoutLevel(get-Single(char-kind))
       then false
       else true

isSingleWithLevel(char-kind)
= if not is-Single(char-kind)
  then false
```

```
  else if not is-WithLevel(get-Single(char-kind))
       then false
       else true

isSelected(char-kind)
= isMultiple(char-kind) xor
  isSingleWithoutLevel(char-kind) xor
  isSingleWithLevel(char-kind)

getValues(char-kind)
= if not isSelected(char-kind)
  then emptyset
  else if isMultiple(char-kind)
       then get-Multiple(char-kind)
       else if isSingleWithoutLevel(char-kind)
            then get-WithoutLevel(get-Single(char-kind))
            else elements(get-WithLevel(get-Single(char-kind)))

isEmpty(char-kind)
= if not isSelected(char-kind)
  then true
  else isempty(getValues(char-kind))

getLevelOf(char-kind,item)
= positionof(get-WithLevel(get-Single(char-kind)),item)

getMaxLevel(char-kind)
= length(get-WithLevel(get-Single(char-kind)))

includeValue(char-kind,item)
= if length(repl(" ","",item)) > 0
     then if isMultiple(char-kind)
          then CharKind-Multiple(add(item,getValues(char-kind)))
          else if isSingleWithoutLevel
               then CharKind-Single(
                      Single-WithoutLevel(
                        add(item,getValues(char-kind))
                      )
                    )
               else if isin(item,get-WithLevel(get-Single(char-kind)))
```

```
                    then char-kind
                    else CharKind-Single(
                           Single-WithLevel(
                             cons(
                               item,
                               get-WithLevel(get-Single(char-kind))
                             )
                           )
                         )
                       fi
                  fi
             fi
       else char-kind
       fi

excludeValue(char-kind,item)
= if isMultiple(char-kind)
  then CharKind-Multiple(delete(item,getValues(char-kind)))
  else if isSingleWithoutLevel
       then CharKind-Single(
              Single-WithoutLevel(
                delete(item,getValues(char-kind))
              )
            )
       else CharKind-Single(
              Single-WithLevel(
                delete(item,get-WithLevel(get-Single(char-kind)))
              )
            )
```

A.12 ATOProjects

Abaixo é apresentada uma descrição de alto nível para as principais operações deste ATO:

- *exist*: verifica se o projeto existe;
- *getName*: retorna o nome do projeto;
- *getDescription*: retorna a descrição do projeto;
- *getManager*: retorna o gerente do projeto;
- *getStart*: retorna a data de início do projeto;

- *getExpectedEnd*: retorna a data de término esperada do projeto;
- *getCharacteristics*: retorna as características do projeto;
- *include*: inclui o projeto;
- *exclude*: exclui o projeto;
- *exclude*: exclui um conjunto de projetos;
- *alter*: altera o projeto;
- *includeCharacteristic*: inclui a característica no projeto;
- *excludeCharacteristic*: exclui a característica do projeto;
- *alterCharacteristic*: altera a característica do projeto;
- *isCharacteristicBeenUsed*: verifica se característica está sendo utilizada em algum projeto.

CLASS

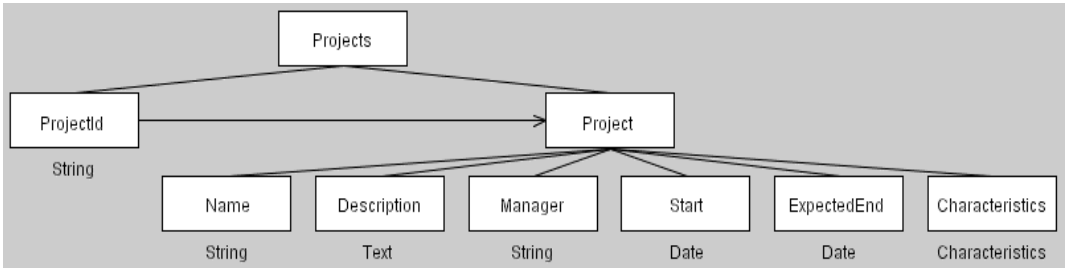


Figura A.12: Classe que define o ATOProjects

INCLUDE

```
Boolean
```

INTERFACES

```
exist : Projects String → Boolean

getName : Projects String → String

getDescription : Projects String → Text
```

```
getManager : Projects String → String

getStart : Projects String → Date

getExpectedEnd : Projects String → Date

getCharacteristics : Projects String → Characteristics

include : Projects String String Text String Date Date → Projects

exclude : Projects String → Projects

exclude : Projects SetOfString → Projects

alter : Projects String String Text String Date Date → Projects

includeCharacteristic : Projects String String Value Integer →
                        Projects

excludeCharacteristic : Projects String String → Projects

alterCharacteristic : Projects String String Value Integer →
                      Projects

isCharacteristicBeenUsed : Projects String → Boolean
```

FORMAL VARIABLES

```
projs : Projects
proj-id char-id name manager : String
proj-ids : SetOfString
desc : Text
start-dt end-dt : Date
value : Value
weight : Integer
chars : Characteristics
```

AXIOMS

```
exist(projs,proj-id)
= proj-id belongsto domain(projs)

getName(projs,proj-id)
= if exist(projs,proj-id)
  then select-Name(imageof(proj-id,projs))

getDescription(projs,proj-id)
= if exist(projs,proj-id)
  then select-Description(imageof(proj-id,projs))

getManager(projs,proj-id)
= if exist(projs,proj-id)
  then select-Manager(imageof(proj-id,projs))

getStart(projs,proj-id)
= if exist(projs,proj-id)
  then select-Start(imageof(proj-id,projs))

getExpectedEnd(projs,proj-id)
= if exist(projs,proj-id)
  then select-ExpectedEnd(imageof(proj-id,projs))

getCharacteristic(projs,proj-id)
= if exist(projs,proj-id)
  then select-Characteristics(imageof(proj-id,projs))

include(projs,proj-id,name,desc,manager,start-dt,end-dt)
= if exist(projs,proj-id)
  then projs
  else modify(
         proj-id,
         reg-Project(name,desc,manager,start-dt,end-dt,emptymap),
         projs
       )
```

```
exclude(projs,proj-id)
= if not exist(projs,proj-id)
  then projs
  else restrictwith(projs,add(proj-id,emptyset))

exclude(_,emptyset)
= _

exclude(projs,add(proj-id,proj-ids))
= exclude(exclude(projs,proj-id),proj-ids)

alter(projs,proj-id,name,desc,manager,start-dt,end-dt)
= if not exist(projs,proj-id)
  then projs
  else override(
         projs,
         modify(
           proj-id,
           reg-Project(
             name,desc,manager,start-dt,end-dt,
             getCharacteristics(projs,proj-id)
           ),
           emptymap
         )
       )

includeCharacteristic(projs,proj-id,char-id,value,weight)
= if not exist(projs,proj-id)
  then projs
  else override(
         projs,
         modify(
           proj-id,
           reg-Project(
             getName(projs,proj-id),
             getDescription(projs,proj-id),
             getManager(projs,proj-id),
             getStart(projs,proj-id),
             getExpectedEnd(projs,proj-id),
```

```
            ICS(ATOCharacteristics,include,
              getCharacteristics(projs,proj-id),
              <char-id,value,weight>
            )
          ),
          emptymap
        )
      )

excludeCharacteristic(projs,proj-id,char-id)
= if not exist(projs,proj-id)
  then projs
  else override(
         projs,
         modify(
           proj-id,
           reg-Project(
             getName(projs,proj-id),
             getDescription(projs,proj-id),
             getManager(projs,proj-id),
             getStart(projs,proj-id),
             getExpectedEnd(projs,proj-id),
             ICS(ATOCharacteristics,exclude,
               getCharacteristics(projs,proj-id),
               <char-id>
             )
           ),
           emptymap
         )
       )

alterCharacteristic(projs,proj-id,char-id,value,weight)
= if not exist(projs,proj-id)
  then projs
  else override(
         projs,
         modify(
           proj-id,
           reg-Project(
             getName(projs,proj-id),
```

```
            getDescription(projs,proj-id),
            getManager(projs,proj-id),
            getStart(projs,proj-id),
            getExpectedEnd(projs,proj-id),
            ICS(ATOCharacteristics,alter,
              getCharacteristics(projs,proj-id),
              <char-id,value,weight>
            )
          ),
          emptymap
        )
      )

isCharacteristicBeenUsed(emptymap,_)
= false

isCharacteristicBeenUsed(modify(proj-id,reg-Project(_,_,_,_,_,chars),
projs),char-id)
= if ICS(ATOCharacteristics,exist,chars,<char-id>)
  then true
  else isCharacteristicBeenUsed(projs,char-id)
```

A.13 ATOCharacteristics

Abaixo é apresentada uma descrição de alto nível para as principais operações deste ATO:

- *exist*: verifica se a característica existe;
- *getCharacteristicsIds*: retorna os identificadores das características existentes;
- *getValue*: retorna o valor da característica;
- *getWeight*: retorna o peso da característica;
- *include*: inclui a característica;
- *exclude*: exclui a característica;
- *alter*: altera a característica.

CLASS

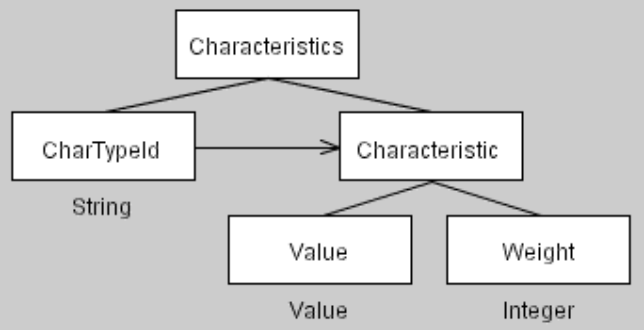


Figura A.13: Classe que define o ATOCharacteristics

INCLUDE

Boolean

INTERFACES

exist : Characteristics String → Boolean

getCharacteristicsIds : Characteristics → SetOfString

getValue : Characteristics String → Value

getWeight : Characteristics String → Integer

include : Characteristics String Value Integer → Characteristics

exclude : Characteristics String → Characteristics

alter : Characteristics String Value Integer → Characteristics

FORMAL VARIABLES

chars : Characteristics
char-id : String
value : Value

```
weight : Integer
```

AXIOMS

```
exist(chars,char-id)
= char-id belongsto domain(chars)

getCharacteristicsIds(chars)
= domain(chars)

getValue(chars,char-id)
= if exist(chars,char-id)
  then select-Value(imageof(char-id,chars))

getWeight(chars,char-id)
= if exist(chars,char-id)
  then select-Weight(imageof(char-id,chars))

include(chars,char-id,value,weight)
= if exist(chars,char-id)
  then chars
  else if not ICS(ATOValue,isEmpty,value)
       then chars
       else if weight <= 0
            then chars
            else modify(
                   char-id,reg-Characteristic(value,weight),chars
                 )

exclude(chars,char-id)
= if not exist(chars,char-id)
  then chars
  else restrictwith(chars,add(char-id,emptyset))

alter(chars,char-id,value,weight)
= if not exist(chars,char-id)
  then chars
  else if not ICS(ATOValue,isEmpty,value)
```

```
then chars
else if weight <= 0
     then chars
     else override(
            chars,
            modify(
              char-id,
              reg-Characteristic(value,weight),
              emptymap
            )
          )
```

A.14 ATOValue

Abaixo é apresentada uma descrição de alto nível para as principais operações deste ATO:

- *isItem*: verifica se o valor é um item;
- *isSetOf*: verifica se o valor é um conjunto de itens;
- *isSelected*: verifica se o valor está selecionado;
- *getItem*: retorna o item do valor;
- *getSetOf*: retorna o conjunto de itens do valor;
- *getAsSet*: retorna o valor como um conjunto de itens;
- *isEmpty*: verifica se o valor é vazio;
- *create*: cria um valor como um item;
- *create*: cria um valor como um conjunto de itens.

CLASS

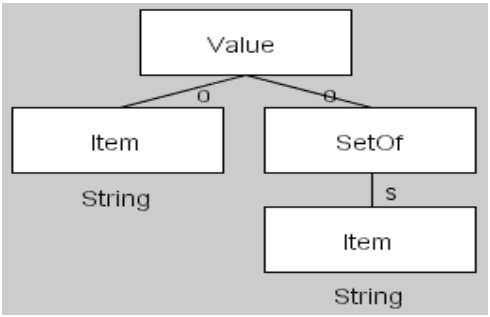


Figura A.14: Classe que define o ATOValue

INCLUDE

Boolean

INTERFACES

isItem : Value → Boolean

isSetOf : Value → Boolean

isSelected : Value → Boolean

getItem : Value → String

getSetOf : Value → SetOfString

getAsSet : Value → SetOfString

isEmpty : Value → Boolean

create : String → Value

create : SetOfString → Value

FORMAL VARIABLES

value : Value
item : String
itens : SetOfString

AXIOMS

```
isItem(value)
= is-Item(value)

isSetOf(value)
= is-SetOf(value)

isSelected(value)
= isItem(value) xor isSetOf(value)

getItem(Value-Item(item))
= item

getSetOf(Value-SetOf(itens))
= itens

getAsSet(value)
= if isSelected(value)
  then if isItem(value)
       then add(getItem(value),emptyset)
       else getSetOf(value)
       fi
  else emptyset
  fi

isEmpty(value)
= if isSelected(value)
  then if isItem(value)
       then length(repl(" ","",getItem(value))) == 0
       else isempty(getSetOf(value))
       fi
  else true
  fi

create(item)
= Value-Item(item)

create(itens)
= Value-SetOf(itens)
```

A.15 ATOCases

Abaixo é apresentada uma descrição de alto nível para as principais operações deste ATO:

- *exist*: verifica se o caso existe;
- *getCasesIds*: retorna os identificadores dos casos existentes;
- *getProcessId*: retorna o processo adaptado;
- *getModifications*: retorna as modificações;
- *getComments*: retorna os comentários;
- *wasApplied*: verifica se o caso foi avaliado;
- *getAvaliation*: retorna a avaliação;
- *getRealization*: retorna a data de realização;
- *getTrace*: retorna o caminho da adaptação
- *getAvaliated*: retorna os casos avaliados;
- *include*: inclui o caso;
- *exclude*: exclui o caso;
- *exclude*: exclui o conjunto de casos;
- *setModifications*: define o conjunto de modificações;
- *setComments*: comenta o caso;
- *setAvaliation*: avalia o caso.

CLASS

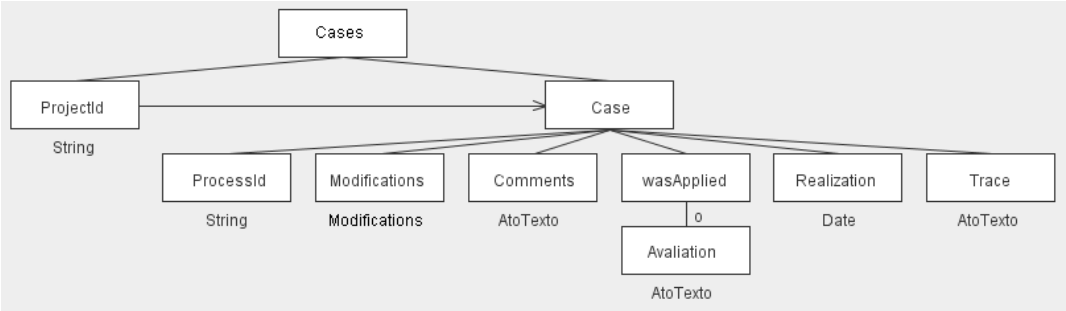


Figura A.15: Classe que define o ATOCases

INCLUDE

Boolean

INTERFACES

exist : Cases String → Boolean

getCasesIds : Cases → SetOfString

getProcessId : Cases String → String

getModifications : Cases String → Modifications

getComments : Cases String → Text

wasApplied : Cases String → Boolean

getAvaliation : Cases String → Text

getRealization : Cases String → Date

getTrace : Cases String → Text

getAvaliated: Cases → SetOfString

include : Cases String String Date → Cases

exclude : Cases String → Cases

exclude : Cases SetOfString → Cases

setModifications : Cases String Modifications → Cases

setComments : Cases String Text → Cases

setAvaliation : Cases String Text → Cases

FORMAL VARIABLES

```
cases : Cases
proj-id proc-id : String
projs : SetOfString
sysdate : Date
comm aval: Text
mods : Modifications
```

AXIOMS

```
exist(cases,proj-id)
= proj-id belongsto domain(cases)

getCasesIds(cases)
= domain(cases)

getProcessId(cases,proj-id)
= if exist(cases,proj-id)
  then select-ProcessId(imageof(proj-id,cases))

getModifications(cases,proj-id)
= if exist(cases,proj-id)
  then select-Modifications(imageof(proj-id,cases))

getComments(cases,proj-id)
= if exist(cases,proj-id)
  then select-Comments(imageof(proj-id,cases))

wasApplied(cases,proj-id)
= is-Avaliation(select-wasApplied(imageof(proj-id,cases)))

getAvaliation(cases,proj-id)
= if exist(cases,proj-id) and wasApplied(cases,proj-id)
  then get-Avaliation(select-Avaliation(imageof(proj-id,cases)))

getRealization(cases,proj-id)
```

```
= if exist(cases,proj-id)
  then select-Realization(imageof(proj-id,cases))

getTrace(cases,proj-id)
= if exist(cases,proj-id)
  then select-Trace(imageof(proj-id,cases))

getAvaliated(emptymap)
= emptyset

getAvaliated(modify(proj-id,reg-Cases(_,_,_,aval,_),cases))
= if is-Avaliation(aval)
  then add(proj-id,getAvaliated(cases))
  else getAvaliated(cases)

include(cases,proj-id,proc-id,sysdate)
= if exist(cases,proj-id)
  then cases
  else modify(
         proj-id,
         reg-Case(proc-id,emptymap,emptylist,,sysdate),
         cases
       )

exclude(cases,proj-id)
= if not exist(cases,proj-id)
  then cases
  else restrictwith(cases,add(proj-id,emptyset))

exclude(_,emptyset)
= _

exclude(cases,add(proj-id,projs))
= exclude(exclude(cases,proj-id),projs)

setModifications(cases,proj-id,mods)
= if not exist(cases,proj-id)
  then cases
  else if wasApplied(cases,proj-id)
       then cases
```

```
       else override(
              cases,
              modify(
                proj-id,
                reg-Case(
                  getProcessId(cases,proj-id),
                  mods,
                  getComments(cases,proj-id),,
                  getRealization(cases,proj-id)
                ),
                emptymap
              )
            )

setComments(cases,proj-id,comm)
= if not exist(cases,proj-id)
  then cases
  else if wasApplied(cases,proj-id)
       then cases
       else override(
              cases,
              modify(
                proj-id,
                reg-Case(
                  getProcessId(cases,proj-id),
                  getModifications(cases,proj-id),
                  comm,,
                  getRealization(cases,proj-id)
                ),
                emptymap
              )
            )

setAvaliation(cases,proj-id,aval)
= if not exist(cases,proj-id)
  then cases
  else if wasApplied(cases,proj-id)
       then cases
       else override(
              cases,
```

```
modify(
  proj-id,
  reg-Case(
    getProcessId(cases,proj-id),
    getModifications(cases,proj-id),
    getComments(cases,proj-id),
    wasApplied-Avaliation(aval),
    getRealization(cases,proj-id)
  ),
  emptymap
 )
)
```

A.16 ATOModifications

Abaixo é apresentada uma descrição de alto nível para as principais operações deste ATO:

- *exist*: verifica se a modicação existe;
- *getKind*: retorna o tipo da modificação;
- *getReason*: retorna o motivo da modificação;
- *include*: inclui a modificação;
- *exclude*: exclui a modificação.

CLASS

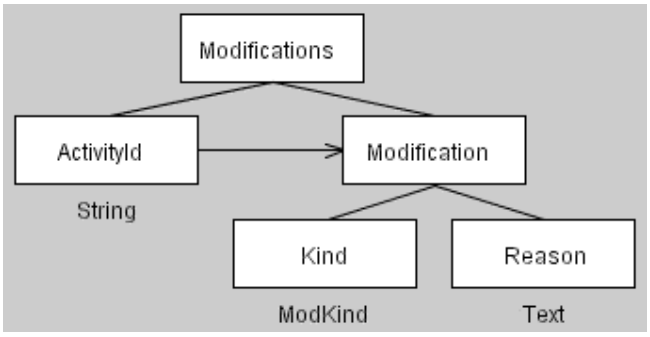


Figura A.16: Classe que define o ATOModifications

INCLUDE

Boolean

INTERFACES

exist : Modifications String → Boolean

getKind : Modifications String → ModKind

getReason : Modifications String → Text

include : Modifications String ModKind Text → Modifications

exclude : Modifications String → Modifications

FORMAL VARIABLES

mods : Modifications
act-id : String
kind : ModKind
reason : Text

AXIOMS

exist(mods,act-id)
= act-id belongsto domain(mods)

getKind(mods,act-id)
= if exist(mods,act-id)
  then select-Kind(imageof(act-id,mods))

getReason(mods,act-id)
= if exist(mods,act-id)
  then select-Reason(imageof(act-id,mods))

```
include(mods,act-id,kind,reason)
= if exist(mods,act-id)
  then mods
  else if not ICS(ATOModKind,isSelected,kind)
       then mods
       else if isempty(reason)
            then mods
            else modify(act-id,reg-Modification(kind,reason),mods)

exclude(mods,act-id)
= if not exist(mods,act-id)
  then mods
  else restrictwith(mods,add(act-id,emptyset))
```

A.17 ATOModKind

Abaixo é apresentada uma descrição de alto nível para as principais operações deste ATO:

- *isInclusion*: verifica se a modificação é uma inclusão;
- *isExclusion*: verifica se a modificação é uma exclusão;
- *isSelected*: verifica se a modificação está selecionada.

CLASS

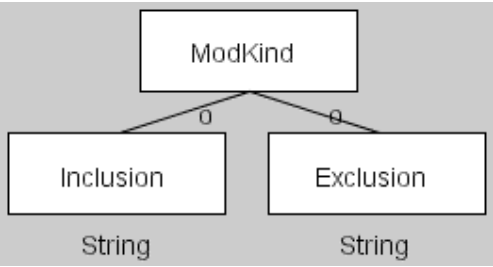


Figura A.17: Classe que define o ATOModKind

INCLUDE

```
Boolean
```

INTERFACES

```
isInclusion : ModKind → Boolean

isExclusion : ModKind → Boolean

isSelected : ModKind → Boolean
```

FORMAL VARIABLES

```
mod-kind : ModKind
```

AXIOMS

```
isInclusion(mod-kind)
= is-Inclusion(mod-kind)

isExclusion(mod-kind)
= is-Exclusion(mod-kind)

isSelected(mod-kind)
= isInclusion(mod-kind) xor isExclusion(mod-kind)
```

A.18 ATOSimilarCases

Abaixo é apresentada uma descrição de alto nível para as principais operações deste ATO:

- *exist*: verifica se o caso é similar;
- *getSimilarity*: retorna o grau de similaridade;
- *include*: inclui caso similar;
- *exclude*: exclui caso similar;
- *alter*: altera o grau de similaridade.

CLASS

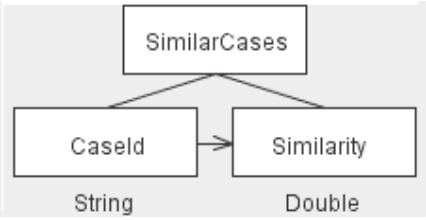


Figura A.18: Classe que define o ATOSimilarCases

INCLUDE

```
Boolean
```

INTERFACES

```
exist : SimilarCases String → Boolean

getSimilarity : SimilarCases String → Double

include : SimilarCases String Double → SimilarCases

alter : SimilarCases String Double → SimilarCases

exclude : SimilarCases String → SimilarCases
```

FORMAL VARIABLES

```
cases : SimilarCases
proj-id: String
sim-degree : Double
```

AXIOMS

```
exist(cases,proj-id)
= proj-id belongsto domain(cases)

getSimilarity(cases,proj-id)
= if exist(cases,proj-id)
  then imageof(proj-id,cases)

include(cases,proj-id,sim-degree)
= if exist(cases,proj-id)
  then cases
  else modify(proj-id,sim-degree),cases)

alter(cases,proj-id,sim-degree)
= if not exist(cases,proj-id)
  then cases
  else override(cases,modify(proj-id,sim-degree,emptymap))

exclude(cases,proj-id)
= if not exist(cases,proj-id)
  then cases
  else restrictwith(cases,add(proj-id,emptyset))
```

APÊNDICE B SEMÂNTICA ALGÉBRICA DO MECANISMO DE ADAPTAÇÃO

Neste apêndice, é apresentada a semântica do mecanismo de adaptação, formalizada através do Prosoft-Algébrico[40]. Conforme visto no capítulo 3, o processo de adaptação é formado pelos seguintes passos:

- Reusar casos de adaptação:
  - Comparar projetos de software;
  - Selecionar caso de adaptação;
  - Adaptar processo adaptado recuperado.
- Adaptar processo padrão;
- Ajustar processo adaptado;
- Criar caso de adaptação.

B.1 Adaptar Processo de Software

Função global de adaptação, tendo sido especificada através do método *adapt*, cuja assinatura, passos e semântica são apresentados abaixo.

```
/*****************************************************************/
/*     Nome: adapt                                               */
/* Objetivo: adaptar o processo padrão para um projeto de software  */
/*   Passos: - verificar se processo padrão é vazio              */
/*           - verificar se projeto existe                       */
/*           - verificar se projeto está associado a um caso     */
/*           - verificar se o processo adaptado existe           */
/*           - verificar os limites do grau de similaridade      */
```

[40] Durante a adaptação, o modelo do processo adaptado irá sofrer mudanças (inclusão ou exclusão de atividades) e sua coerência precisará ser garantida. Tanto as operações de inclusão/exclusão de atividades quanto as regras para coerência foram especificadas em GG, e são refeenciadas na semântica algébrica do mecanismo de adaptação da seguinte forma: no caso das operações, apenas a chamada dos métodos é referenciada; no caso das regras, a aplicação das mesma em um modelo de processo abstrato mod' será evidenciada com o uso de duas apóstrofes, passando então a ficar mod''.

```
/*           - recuperar casos de adaptação + semelhantes           */
/*           - selecionar caso de adaptação                         */
/*           - adaptar processo adaptado recuperado                 */
/*           - adaptar processo padrão                              */
/*           - ajustar processo adaptado                            */
/*           - gerar caso de adaptação                              */
/*******************************************************************/


adapt : AdaptationModel String String Double → AdaptationModel


adapt(reg-AdaptationModel(stand,procs,char-types,projs,cases),proj-
id,proc-id,sim-degree)
=
if
ICS(ATOAbsProcessModel,isEmpty,ICS(ATOStandard,getProcessModel,stand))
    then reg-AdaptationModel(stand,procs,char-types,projs,cases)
    else if not ICS(ATOProjects,exist,projs,<proj-id>)
    then reg-AdaptationModel(stand,procs,char-types,projs,cases)
    else if ICS(ATOCases,exist,cases,<proj-id>)
    then reg-AdaptationModel(stand,procs,char-types,projs,cases)
    else if ICS(ATOAdaptedProcesses,exist,procs,<proc-id>)
    then reg-AdaptationModel(stand,procs,char-types,projs,cases)
    else if sim-degree < 0 or sim-degree > 1
    then reg-AdaptationModel(stand,procs,char-types,projs,cases)
    else reg-AdaptationModel(
           stand,
           ICS(ATOAdaptedProcesses,include,procs,<
             proc-id,
             adaptStandProc(
               ICS(ATOStandard,getProcessModel,stand),
               ICS(ATOStandard,getRules,stand),
               selectSimilarCase(
                 ICS(ATOCases,exclude,cases,<complement(
                   ICS(ATOCases,getCasesIds,cases),
                   ICS(ATOSimilarCases,getCasesIds,
                     returnSimilarCases(
                       cases,projs,proj-id,sim-degree,char-types
                     )
                   )
                 )>),
```

```
                proj-past?,
                procs,
                ICS(ATOStandard,getProcessModel,stand),
                ICS(ATOStandard,getRules,stand),
                projs,
                proj-id,
                char-types
              ),
              ICS(ATOCases,getModifications,cases,<proj-past?>),
              projs,
              proj-id,
              char-types
            )
          >),
          char-types,
          projs,
          ICS(ATOCases,setModifications,
            ICS(ATOCases,include,cases,<proj-id,proc-id,sysdate>),
            <proj-id,mods?>
          )
        )
```

B.2 Retornar Projetos Similares

Função responsável por comparar o projeto de software corrente com cada um dos projetos passados e retornar o mais similares. Foi especificada através do método *returnSimilarCases*, o qual por sua vez invoca o método *calculateSimilarity* para calcular o grau de similaridade entre dois projetos de software. Os passos, assinatura e semântica destes dois métodos são apresentados abaixo.

```
/*****************************************************************/
/*     Nome: returnSimilarCases                                  */
/* Objetivo: retornar casos de adaptação mais similares ao projeto  */
/*           corrente                                            */
/*****************************************************************/


returnSimilarCases : Cases Projects String Double CharacteristicTypes
                   → SimilarCases


returnSimilarCases(emptymap,_,_,_,_)
= emptymap
```

```
returnSimilarCases(modify(proj-past,reg-Case(_,_,_,_,_),cases),projs,
proj-cur,sim-degree,char-types)
= if calculateSimilarity(projs,proj-cur,proj-past,char-types)
     >= sim-degree
  then ICS(ATOSimilarCases,include,
         returnSimilarCases(
           cases,projs,proj-cur,sim-degree,char-types
         ),
         <proj-past,
          calculateSimilarity(
            projs,proj-cur,proj-past,char-types
          )
         >
       )
  else returnSimilarCases(cases,projs,proj-cur,sim-degree,char-types)

/*********************************************************************/
/*     Nome: calculateSimilarity                                     */
/* Objetivo: calcula a similaridade entre dois projetos de software */
/*********************************************************************/

calculateSimilarity : Projects String String CharacteristicTypes
                      → Double

similaritySomatory : SetOfString Characteristics Characteristics
                     CharacteristicTypes → Double

similarityFunction : String Characteristics Characteristics
                     CharacteristicTypes → Double

weightSomatory : SetOfString Characteristics → Integer

calculateSimilarity(projs,proj1,proj2,char-types)
= similaritySomatory(
    ICS(ATOCharacteristics,getCharacteristicsIds,
        ICS(ATOProjects,getCharacteristics,projs,<proj1>))
    U
    ICS(ATOCharacteristics,getCharacteristicsIds,
        ICS(ATOProjects,getCharacteristics,projs,<proj2>)),
```

```
    ICS(ATOProjects,getCharacteristics,projs,<proj1>),
    ICS(ATOProjects,getCharacteristics,projs,<proj2>),
    char-types
  )
  /
  weightSomatory(
    ICS(ATOCharacteristics,getCharacteristicsIds,
        ICS(ATOProjects,getCharacteristics,projs,<proj1>))
    U
    ICS(ATOCharacteristics,getCharacteristicsIds,
        ICS(ATOProjects,getCharacteristics,projs,<proj2>)),
    ICS(ATOProjects,getCharacteristics,projs,<proj1>),
    ICS(ATOProjects,getCharacteristics,projs,<proj2>)
  )

weightSomatory(emptyset,_)
= 0

weightSomatory(add(char-id,char-ids),chars)
= if not ICS(ATOCharacteristics,exist,chars,<char-id>)
  then 1 + weightSomatory(char-ids,chars)
  else ICS(ATOCharacteristics,getWeight,chars,<char-id>)
       +
       weightSomatory(char-ids,chars)

similaritySomatory(emptyset,_,_,_)
= 0

similaritySomatory(add(char-id,chars),chars1,chars2,char-types)
= if not ICS(ATOCharacteristics,exist,chars1,<char-id>)
  then similaritySomatory(chars,chars1,chars2,char-types)
  else similarityFunction(char-id,chars1,chars2,char-types)
       *
       iCS(ATOCharacteristics,getWeight,chars1,<char-id>)
       +
       similaritySomatory(chars,chars1,chars2,char-types)

similarityFunction(char-id,chars1,chars2,char-types)
= if not ICS(ATOCharacteristics,exist,chars1,<char-id>) or
     not ICS(ATOCharacteristics,exist,chars2,<char-id>)
```

```
then 0
else if ICS(ATOCharKind,isSingleWithoutLevel,
          ICS(ATOCharacteristicTypes,getKind,char-types,<char-id>))
     then if ICS(ATOValue,getItem,
                ICS(ATOCharacteristics,getValue,chars1,<char-id>))
             =
             ICS(ATOValue,getItem,
               ICS(ATOCharacteristics,getValue,chars2,<char-id>))
          then 1
          else 0
          fi
     else if ICS(ATOCharKind,isSingleWithLevel,
          ICS(ATOCharacteristicTypes,getKind,char-types,<char-id>))
          Then 1 - (
                 mod(
                   ICS(ATOCharKind,getLevelOf,
                     ICS(ATOCharacteristicTypes,getKind,
                       char-types,
                       <char-id>
                     ),
                     <ICS(ATOValue,getItem,
                       ICS(ATOCharacteristics,getValue,
                         chars1,
                         <char-id>
                       )
                     )>
                   )
                   -
                   ICS(ATOCharKind,getLevelOf,
                     ICS(ATOCharacteristicTypes,getKind,
                       char-types,
                       <char-id>
                     ),
                     <ICS(ATOValue,getItem,
                        ICS(ATOCharacteristics,getValue,
                          chars2,<char-id>
                        )
                     )>
                   )
                 )
```

```
        /
        ICS(ATOCharKind,getMaxLevel,
          ICS(ATOCharacteristicTypes,getKind,
            char-types,
            <char-id>
          )
        )
      )
else cardinality(
        ICS(ATOValue,getSetOf,
          ICS(ATOCharacteristics,getValue,
            chars1,
            <char-id>
          )
        )
        Intersection
        ICS(ATOValue,getSetOf,
          ICS(ATOCharacteristics,getValue,
            chars2,
            <char-id>
          )
        )
      )
      /
      cardinality(
        ICS(ATOValue,getSetOf,
          ICS(ATOCharacteristics,getValue,
            chars1,
            <char-id>
          )
        )
        U
        ICS(ATOValue,getSetOf,
          ICS(ATOCharacteristics,getValue,
            chars2,
            <char-id>
          )
        )
      )
fi
```

```
      fi
  fi
```

B.3 Selecionar Caso Similar

Embora a ação de selecionar um caso de adaptação dentre aqueles cujos projetos associados são similares ao projeto corrente seja semi-automática, ou seja, pode haver a necessidade de interação com o engenheiro de processos através do Editor de Casos de Adaptação, houve a necessidade de especificá-la algebricamente, através do recurso da "?" após variáveis. O método *selectSimilarCase* formaliza esta ação e seus passos, assinatura e semântica são apresentados abaixo.

```
/*****************************************************************/
/*     Nome: selectSimilarCase                                   */
/* Objetivo: selecionar caso de adaptação mais similar ao projeto   */
/*           corrente                                            */
/*   Passos: - selecionar caso similar                           */
/*           - adaptar caso similar                              */
/*****************************************************************/

selectSimilarCase : Cases String AdaptedProcesses AbsProcessModel
                    Rules Projects String CharacteristicTypes
                    → AbsProcessModel

//→ Nenhum caso similar
selectSimilarCase(emptymap,_,_,_,_,_,_,_)
= reg-AbsProcessModel(
    emptymap,
    emptymap,
    reg-Policies(
      reg-PolicySets(
        emptymap,emptymap,reg-Instantiation(emptymap,emptymap)),
      false
    ),
    ""
  )

//→ Um caso similar
selectSimilarCase(modify(proj-id,reg-Case(proc-id,mods,_,_,_),
emptymap),_,procs,stand,rules,projs,proj-cur,char-types)
= adaptSimilarCase(
```

```
    ICS(ATOAdaptedProcesses,getProcessModel,procs,<proc-id>),
    mods,
    stand,
    rules,
    projs,
    proj-cur,
    char-types
  )

//→ N casos similares
selectSimilarCase(cases,proj-past,procs,stand,rules,projs,proj-cur,
char-types)
= adaptSimilarCase(
    ICS(ATOAdaptedProcesses,getProcessModel,procs,<
      ICS(ATOCases,getProcessId,cases,<proj-past>)
    >),
    ICS(ATOCases,getModifications,cases,<proj-past>),
    stand,
    rules,
    projs,
    proj-cur,
    char-types
  )
```

B.4 Adaptar Processo Selecionado

O método *adaptSimilarCase* é responsável pela adaptação do processo adaptado, associado ao projeto de software mais similar ao corrente, para o contexto do projeto corrente. Seus passos, assinatura e semântica são apresentados abaixo.

```
/*****************************************************************/
/*     Nome: adaptSimilarCase                                    */
/* Objetivo: adaptar o processo de software associado ao caso    */
/*           recuperado                                          */
/*****************************************************************/

adaptSimilarCase : AbsProcessModel Modifications AbsProcessModel Rules
                   Projects String CharacteristicTypes
                   → AbsProcessModel
```

```
adaptActivities : AbsActivities Modifications AbsActivities Rules
                  Projects String CharacteristicTypes → AbsActivities

adaptSimilarCase(reg-AbsProcessModel(acts,cons',_,_),mods,stand,rules,
projs,proj-id,char-types)
= reg-AbsProcessModel(
    adaptActivities(
      acts,
      mods,
      ICS(ATOAbsProcessModel,getActivities,stand),
      rules,
      projs,
      proj-id,
      char-types
    ),
    cons'',
    _'_
  )

adaptActivities(emptymap,_,_,_,_,_,_)
= emptymap

adaptActivities(modify(act-id,reg-Details(_,_,desc,_,_),acts),
mods,stand,rules,projs,proj-id,char-types)
= if not ICS(ATOAbsActivities,exist,stand,<act-id>)
  then if not ICS(ATOModifications,exist,mods,<act-id>)
       then modify(
              act-id,
              reg-Details(_,_,desc,_,_),
              adaptActivities(
                acts,mods,stand,rules,projs,proj-id,char-types
              )
            )
       else if manter-atividade? == true
            then modify(
                   act-id,
                   reg-Details(_,_,desc,_,_),
                   adaptActivities(
                     acts,mods,stand,rules,projs,proj-id,char-types
```

```
                    )
                )
            else adaptActivities(
                    acts,mods,stand,rules,projs,proj-id,char-types
                )
            fi
        fi
    else if not ICS(ATORules,exist,rules,<act-id>)
        then if not ICS(ATOModifications,exist,mods,<act-id>)
            then if not ICS(ATOActivityDescription,isFragment,desc)
                then modify(
                        act-id,
                        reg-Details(_,_,desc,_,_),
                        adaptActivities(
                          acts,mods,stand,rules,projs,proj-id,
                          char-types
                        )
                      )
                else modify(
                        act-id,
                        reg-Detail(
                          _,_,
                          adaptSimilarCase(
                            ICS(ATOActivityDescription,getFragment,
                              desc
                            ),
                            mods,
                            ICS(ATOActivityDescription,getFragment,
                              ICS(ATOAbsActivities,getDescription,
                                stand,
                                <act-id>
                              )
                            ),
                            rules,
                            projs,
                            proj-id,
                            char-types
                          ),
                          _,_
                        ),
```

```
                adaptActivities(
                  acts,mods,stand,rules,projs,proj-id,
                  char-types
                )
              )
        fi
  else if not manter-atividade? == true
        then adaptActivities(
               acts,mods,stand,rules,projs,proj-id,char-types
              )
        else if not ICS(ATOActivityDescription,isFragment,
                        desc)
              then modify(
                     act-id,
                     reg-Details(_,_,desc,_,_),
                     adaptActivities(
                       acts,mods,stand,rules,projs,
                       proj-id,char-types
                     )
                   )
              else modify(
                     act-id,
                     reg-Detail(
                       _,_,
                       adaptSimilarCase(
                         ICS(
                           ATOActivityDescription,
                           getFragment,
                           desc
                         ),
                         mods,
                         ICS(
                           ATOActivityDescription,
                           getFragment,
                           ICS(
                             ATOAbsActivities,
                             getDescription,
                             stand,
                             <act-id>
                           )
```

```
                        ),
                        rules,
                        projs,
                        proj-id,
                        char-types
                      ),
                      _,_
                    ),
                    adaptActivities(
                      acts,mods,stand,rules,projs,
                      proj-id,char-types
                    )
                  )
              fi
          fi
     fi
else if evaluateRule(rules,act-id,projs,proj-id,char-types)
        == true
     then modify(
            act-id,
            reg-Details(_,_,desc,_,_),
            adaptActivities(
              acts,mods,stand,rules,projs,proj-id,char-types
            )
          )
     else if not ICS(ATOModifications,exist,mods,<act-id>)
          then adaptActivities(
                 acts,mods,stand,rules,projs,proj-id,char-types
               )
          else if not manter-atividade? == true
               then adaptActivities(
                      acts,mods,stand,rules,projs,proj-id,
                      char-types
                    )
               else modify(
                      act-id,
                      reg-Details(_,_,desc,_,_),
                      adaptActivities(
                        acts,mods,stand,rules,projs,
                        proj-id,char-types
```

```
                                )
                            )
                        fi
                    fi
                fi
            fi
        fi
```

B5 Adaptar Processo Padrão

Por fim, o método *adaptStandProc* é responsável pela adaptação do processo padrão para as características do projeto corrente, como também o seu merge com o processo adaptado recuperado. Este método, bem como o método *adaptSimilarCase*, invoca o método *evaluateRule*, responsável pela avaliação de uma regra de adaptação no contexto do projeto corrente. Os passos, assinatura e semântica destes dois métodos são apresentados abaixo.

```
/*****************************************************************/
/*     Nome: adaptStandProc                                      */
/* Objetivo: adaptar o processo padrão para um projeto de software */
/*****************************************************************/

adaptStandProc : AbsProcessModel Rules AbsProcessModel Modifications
                 Projects String CharacteristicTypes → AbsProcessModel

adaptActivities : AbsActivities Rules AbsActivities Modifications
                  Projects String CharacteristicTypes → AbsActivities

adaptStandProc(reg-AbsProcessModel(acts1,cons1,pols1,reqs1),rules,
reg-AbsProcessModel(acts2,cons2',pols2',reqs2'),mods,projs,proj-id,
char-types)
= reg-AbsProcessModel(
    adaptActivities(acts1,rules,acts2,mods,projs,proj-id,char-types),
    cons2'',
    pols2'',
    reqs2'
  )

adaptActivities(emptymap,_,acts,_,_,_,_)
= acts
```

```
adaptActivities(modify(act-id,reg-Details(_,_,desc,_,_),acts1),rules,
acts2,mods,projs,proj-id,char-types)
= if ICS(ATOAbsActivities,exist,acts2,<act-id>)
  then adaptActivities(
         acts1,rules,acts2,mods,projs,proj-id,char-types
       )
  else if not ICS(ATORules,exist,rules,<act-id>)
       then if not ICS(ATOModifications,exist,mods,<act-id>)
            then if not ICS(ATOActivityDescription,isFragment,desc)
                 then adaptActivities(
                        acts1,
                        rules,
                        ICS(ATOAbsActivities,include,acts2,<
                          act-id,reg-Detail(_,_,desc,_,_)
                        >),
                        mods,
                        projs,
                        proj-id,
                        char-types
                      )
                 else adaptActivities(
                        acts1,
                        rules,
                        ICS(ATOAbsActivities,include,acts2,<
                          act-id,
                          reg-Detail(
                            _'_'
                            Description-Fragment(
                              Fragment-ProcessModel(
                                adaptStandProc(
                                  ICS(
                                    ATOActivityDescrition,
                                    getFragment,
                                    desc
                                  ),
                                  rules,
                                  reg-AbsProcessModel(
                                    emptymap,
                                    ICS(
                                      ATOAbsProcessModel,
```

```
                getConnections,
                ICS(
                  ATOActivityDescrition,
                  getFragment,
                  desc
                )
              ),
              ICS(
                ATOAbsProcessModel,
                getPolicies,
                ICS(
                  ATOActivityDescrition,
                  getFragment,
                  desc
                )
              ),
              ICS(
                ATOAbsProcessModel,
                getRequirements,
                ICS(
                  ATOActivityDescrition,
                  getFragment,
                  desc
                )
              ),
            ),
            mods,
            projs,
            proj-id,
            char-types
          )
        )
      ),
      _'_
    )
>),
mods,
projs,
proj-id,
char-types
```

```
          )
     fi
else if not incluir-atividade? == true
     then adaptActivities(
          acts1,rules,acts2,mods,projs,proj-id,char-types
          )
     else if not
             ICS(ATOActivityDescription,isFragment,desc)
          then adaptActivities(
                 acts1,
                 rules,
                 ICS(ATOAbsActivities,include,acts2,<
                   act-id,reg-Detail(_,_,desc,_,_)
                 >),
                 mods,
                 projs,
                 proj-id,
                 char-types
               )
          else adaptActivities(
                 acts1,
                 rules,
                 ICS(
                   ATOAbsActivities,
                   include,
                   acts2,<
                     act-id,
                     reg-Detail(
                       _,_,
                       Description-Fragment(
                         Fragment-ProcessModel(
                           adaptStandProc(
                             ICS(
                               ATOActivityDescrition,
                               getFragment,
                               desc
                             ),
                             rules,
                             reg-AbsProcessModel(
                               emptymap,
```

```
                    ICS(
                      ATOAbsProcessModel,
                      getConnections,
                      ICS(
                        ATOActivityDescrition,
                        getFragment,
                        desc
                      )
                    ),
                    ICS(
                      ATOAbsProcessModel,
                      getPolicies,
                      ICS(
                        ATOActivityDescrition,
                        getFragment,
                        desc
                      )
                    ),
                    ICS(
                      ATOAbsProcessModel,
                      getRequirements,
                      ICS(
                        ATOActivityDescrition,
                        getFragment,
                        desc
                      )
                    ),
                  ),
                  mods,
                  projs,
                  proj-id,
                  char-types
                )
              )
            ),
            _'_
          )
        >
      ),
      mods,
```

```
                    projs,
                    proj-id,
                    char-types
                  )
            fi
       fi
  fi
else if evaluateRule(rules,act-id,projs,proj-id,char-types)
     == true
  then if not ICS(ATOModifications,exist,mods,<act-id>)
       then adaptActivities(
              acts1,
              rules,
              ICS(ATOAbsActivities,include,acts2,<
                act-id,reg-Detail(_,_,desc,_,_)
              >),
              mods,
              projs,
              proj-id,
              char-types
            )
       else if incluir-atividade? == true
            then adaptActivities(
                   acts1,
                   rules,
                   ICS(ATOAbsActivities,include,acts2,<
                     act-id,reg-Detail(_,_,desc,_,_)
                   >),
                   mods,
                   projs,
                   proj-id,
                   char-types
                 )
            else adaptActivities(
                   acts1,rules,acts2,mods,projs,proj-id,
                   char-types
                 )
            fi
       fi
  else adaptActivities(
```

```
                  acts1,rules,acts2,mods,projs,proj-id,char-types
                )
            fi
       fi
  fi

/*****************************************************************/
/*     Nome: evaluateRule                                        */
/* Objetivo: avaliar uma regra de adaptação em relação ao projeto */
/*           corrente                                            */
/*   Passos: estão descritos em pseudo-código                    */
/*****************************************************************/

evaluateRule : Rules String Projects String CharacteristicTypes
               → Boolean

evaluateConditions : Conditions Projects String CharacteristicTypes
                     → Boolean

evaluateExpression : Expression Projects Sting CharacteristicTypes
                     → Boolean

applyComparisonOp : Value ComparisonOp Value CharKind → Boolean

evaluateRule(rules,act-id,projs,proj-id,char-type)
= if not ICS(ATORules,exist,rules,<act-id>)
  then true
  else if evaluteConditions(
            ICS(ATORules,getConditions,rules,<act-id>),
            projs,proj-id,char-types
          )== true
       then ICS(ATORuleKind,isPositive,
              ICS(ATORules,getKind,rules,<act-id>))
       else not ICS(ATORuleKind,isPositive,
                  ICS(ATORules,getKind,rules,<act-id>))

evaluateConditions(conds,projs,proj-id,char-types)
= if not ICS(ATOConditions,hasOptionalConds,conds)
  then evaluateExpression(
```

```
        ICS(ATOConditions,getExpression,conds),
        projs,
        proj-id,
        char-types
      )
  else if ICS(ATOLogicOp,isAnd,ICS(ATOConditions,getLogicOp,conds))
      then evaluateExpression(
             ICS(ATOConditions,getExpression,conds),
             projs,
             proj-id,
             char-types
           )
           and
           evaluateConditions(
             ICS(ATOConditions,getConditions,conds),
             projs,
             proj-id,
             char-types
           )
      else evaluateExpression(
             ICS(ATOConditions,getExpression,conds),
             projs,
             proj-id,
             char-types
           )
           or
           evaluateConditions(
             ICS(ATOConditions,getConditions,conds),
             projs,
             proj-id,
             char-types
           )

evaluateExpression(exp,projs,proj-id,char-types)
= if not ICS(ATOCharacteristics,exist,
          ICS(ATOProjects,getCharacteristics,projs,<proj-id>),
            <ICS(ATOExpression,getCharTypeId,exp)>)
  then false
  else applyComparisonOp(
         ICS(ATOCharacteristics,getValue,
```

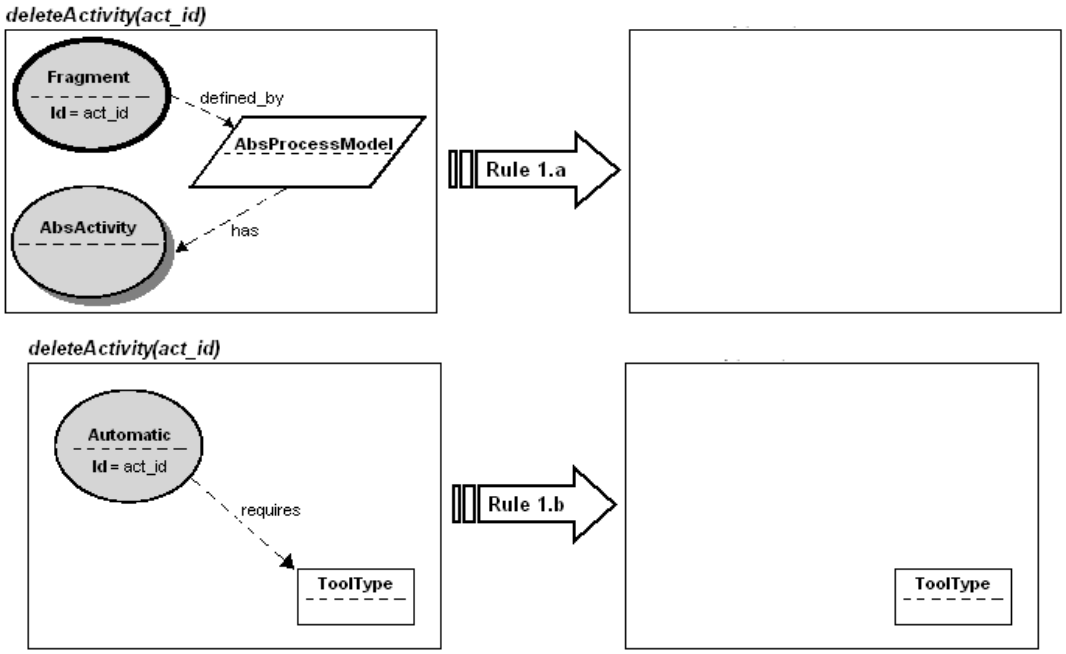
```
          ICS(ATOProjects,getCharacteristics,projs,<proj-id>),
            <ICS(ATOExpression,getCharTypeId,exp)>),
        ICS(ATOExpression,getComparisonOp,exp),
        ICS(ATOExpression,getValue,exp),
        ICS(ATOCharacteristicTypes,getKind,char-types,
          ICS(ATOExpression,getCharTypeId,exp))
      )

applyComparisonOp(opd1,comp-op,opd2,char-kind)
= if ICS(ATOComparisonOp,isEqual,comp-op)
  then if ICS(ATOValue,isItem,opd1)
       then ICS(ATOValue,getItem,opd1) == ICS(ATOValue,getItem,opd2)
       else equal(
              ICS(ATOValue,getSetOf,opd1),
              ICS(ATOValue,getSetOf,opd2)
            )
       fi
  else if ICS(ATOComparisonOp,isDifferent,comp-op)
  then if ICS(ATOValue,isItem,opd1)
       then ICS(ATOValue,getItem,opd1) =/= ICS(ATOValue,getItem,opd2)
       else not equal(
                  ICS(ATOValue,getSetOf,opd1),
                  ICS(ATOValue,getSetOf,opd2)
                )
       fi
  else if ICS(ATOComparisonOp,isGreater,comp-op)
  then ICS(ATOCharKind,getLevelOf,char-kind,
       ICS(ATOValue,getItem,opd1)) >
       ICS(ATOCharKind,getLevelOf,char-kind,
       ICS(ATOValue,getItem,opd2))
  else if ICS(ATOComparisonOp,isGreaterEqual,comp-op)
  then ICS(ATOCharKind,getLevelOf,char-kind,
       ICS(ATOValue,getItem,opd1)) >=
       ICS(ATOCharKind,getLevelOf,char-kind,
       ICS(ATOValue,getItem,opd2))
  else if ICS(ATOComparisonOp,isSmaller,comp-op)
  then ICS(ATOCharKind,getLevelOf,char-kind,
       ICS(ATOValue,getItem,opd1)) <
       ICS(ATOCharKind,getLevelOf,char-kind,
       ICS(ATOValue,getItem,opd2))
```

```
else if ICS(ATOComparisonOp,isSmallerEqual,comp-op)
then ICS(ATOCharKind,getLevelOf,char-kind,
     ICS(ATOValue,getItem,opd1)) <=
     ICS(ATOCharKind,getLevelOf,char-kind,
     ICS(ATOValue,getItem,opd2))
else if ICS(ATOComparisonOp,isBetween,comp-op)
then ICS(ATOCharKind,getLevelOf,char-kind,
     ICS(ATOValue,getItem,opd1)) >=
     ICS(ATOCharKind,getLevelOf,char-kind,
     ICS(ATOValue,getItemOfSet,opd2,1))
     and
     ICS(ATOCharKind,getLevelOf,char-kind,
     ICS(ATOValue,getItem,opd1)) <=
     ICS(ATOCharKind,getLevelOf,char-kind,
     ICS(ATOValue,getItemOfSet,opd2,2))
else if ICS(ATOComparisonOp,isNotBetween,comp-op)
then ICS(ATOCharKind,getLevelOf,char-kind,
     ICS(ATOValue,getItem,opd1)) <
     ICS(ATOCharKind,getLevelOf,char-kind,
     ICS(ATOValue,getItemOfSet,opd2,1))
     and
     ICS(ATOCharKind,getLevelOf,char-kind,
     ICS(ATOValue,getItem,opd1)) >
     ICS(ATOCharKind,getLevelOf,char-kind,
     ICS(ATOValue,getItemOfSet,opd2,2))
else if ICS(ATOComparisonOp,isContains,comp-op)
then ICS(ATOValue,getSetOf,opd1) contain ICS(ATOValue,getSetOf,opd2)
else not ICS(ATOValue,getSetOf,opd1) contain I
     ICS(ATOValue,getSetOf,opd2)
fi
```

APÊNDICE C REGRAS PARA GARANTIA DA COERÊNCIA SINTÁTICA DOS PROCESSOS ADAPTADOS

C.1 Regras para Exclusão de Atividades

A remoção de uma atividade de um modelo de processo abstrato é definida pela função *deleteActivity*, a qual é especificada pelas regras apresentadas na figura C.1. São três os casos tratados: a regra 1.a trata da remoção de fragmentos (deve-se remover também os modelos de processos definidos pelos fragmentos juntamente com as subatividades); a regra 1.b trata da remoção de atividades automáticas (deve-se manter o tipo de ferramenta utilizado pela atividade); A regra 1.c (REIS, 2003) trata da remoção de atividades normais (removendo também o item *AbsRegResource* e mantendo os tipos de grupo e cargo que estiverem associados).



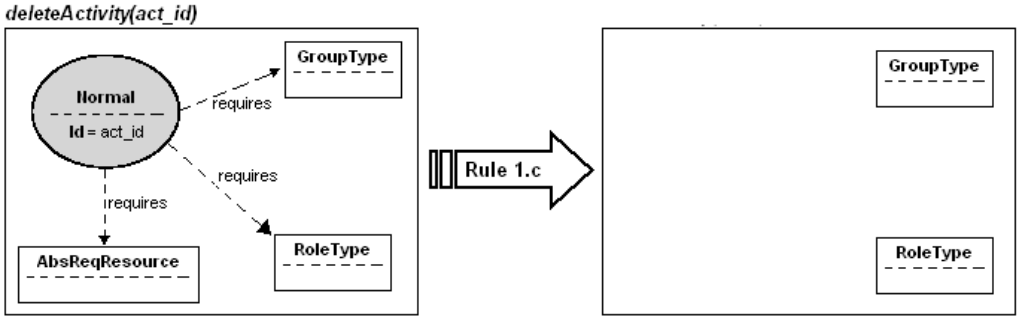


Figura C.1: Regras para remoção de atividades

C.2 Regras para Inclusão de Atividades

A inclusão de uma atividade no modelo de processo de um template ou de um fragmento é tratada por três funções distintas:

- *IncludeNormalActivity*: trata da inclusão de atividades normais, sendo especificada pelas regras 14.a e 14.b. Além do identificador da atividade a ser incluída, deve-se informar também o seu tipo e script.

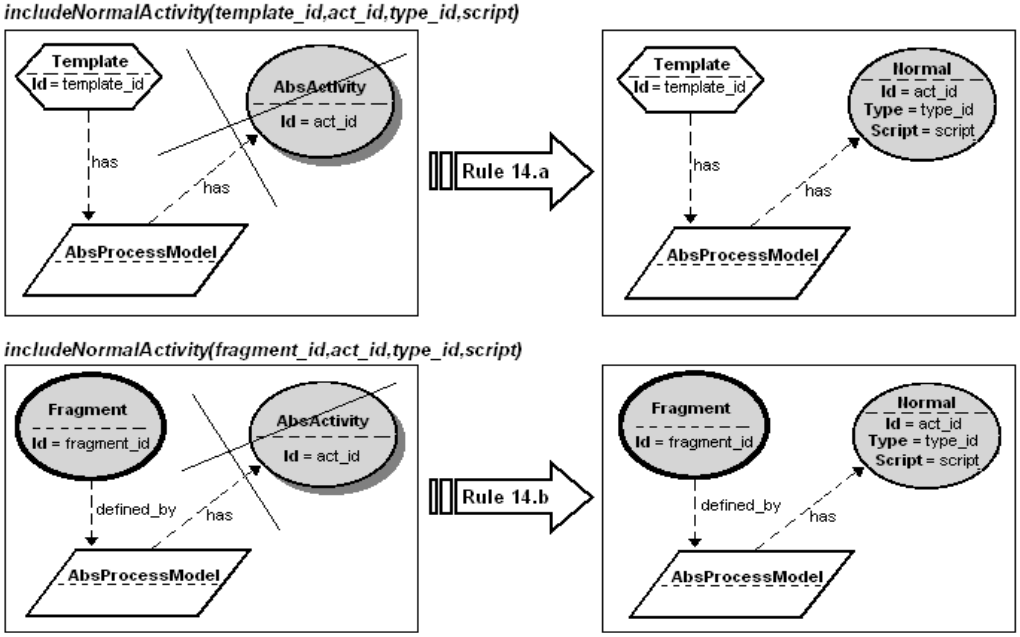


Figura C.2: Regras para inclusão de uma atividade normal

- *IncludeAutomaticActivity*: trata da inclusão de atividades automáticas, sendo especificada pelas regras 14.c e 14.d. Além do identificador da atividade a ser incluída, deve-se informar também o seu tipo.

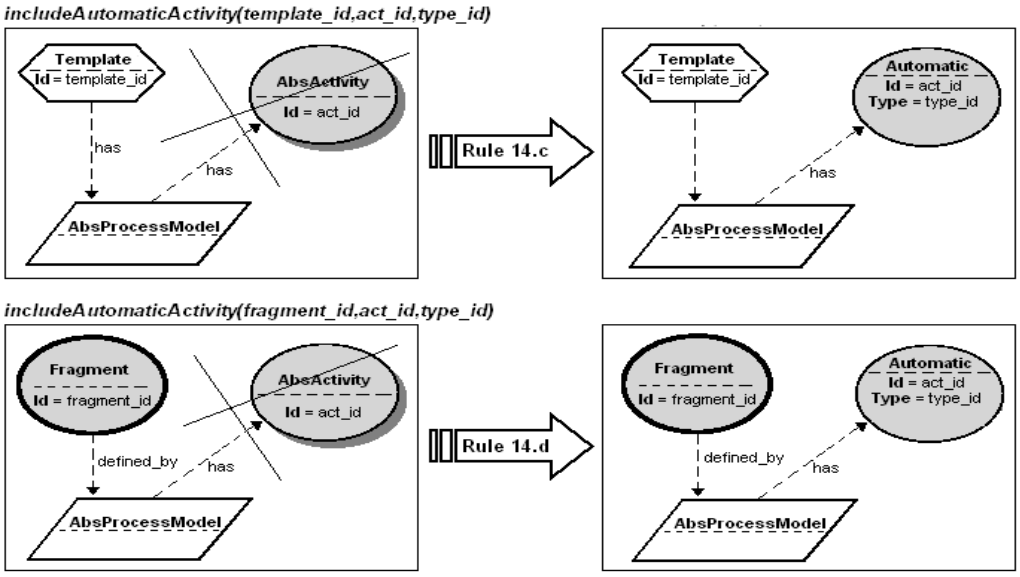


Figura C.3: Regras para inclusão de uma atividade automática

- *IncludeFragment*: trata da inclusão de fragmentos, sendo especificada pelas regras 14.e e 14.f. Além do identificador da atividade a ser incluída, deve-se informar também o seu tipo.

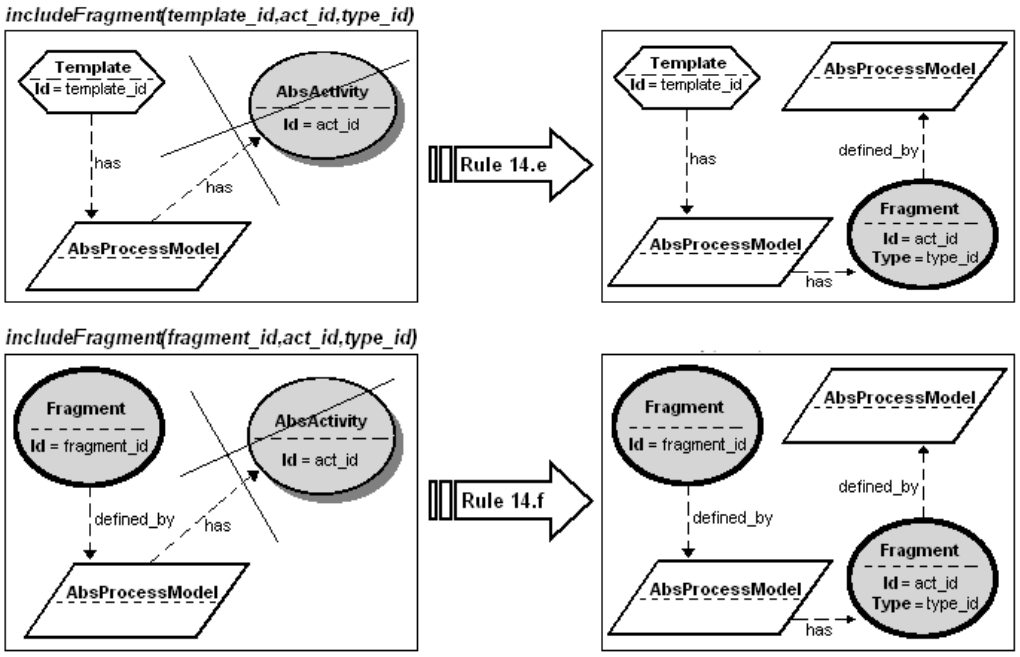


Figura C.4: Regras para inclusão de um fragmento

C.3 Regras para Redefinição dos Componentes de uma Atividade Incluída

Após a inclusão de uma atividade, pertencente ao processo padrão, em um processo adaptado, é necessário redefinir os elementos aos quais ela se relacionava, todavia no

novo modelo de processo. As funções a seguir são responsáveis pela redefinição dos elementos pertencentes a uma atividade incluída:

- *defineInputArtifact*: define os artefatos consumidos por uma atividade abstrata.

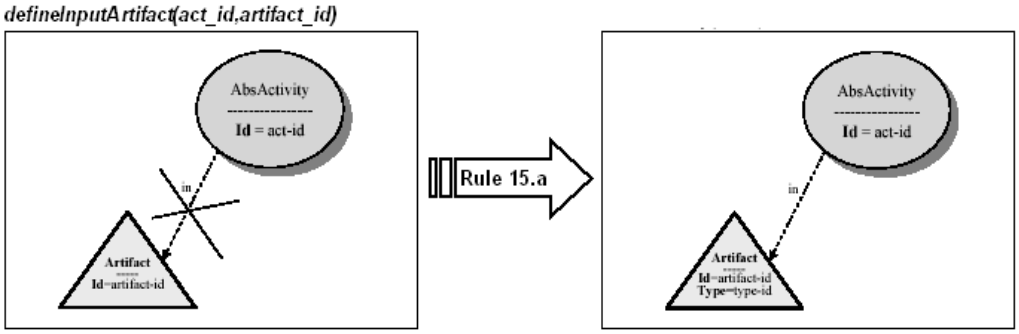


Figura C.5: Regra para definição dos artefatos de entrada de uma atividade (REIS,2002)

- *defineOutputArtifact*: define os artefatos produzidos por uma atividade abstrata.

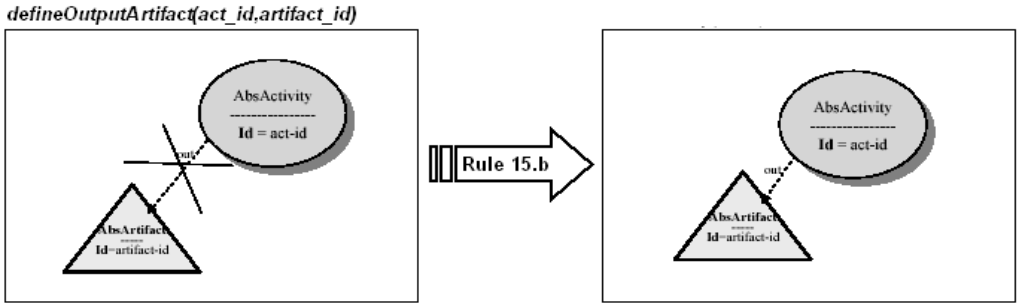


Figura C.6: Regra para definição dos artefatos de saída de uma atividade (REIS,2002)

- *addRequiredResource*: define os recursos abstratos exigidos por uma atividade normal.

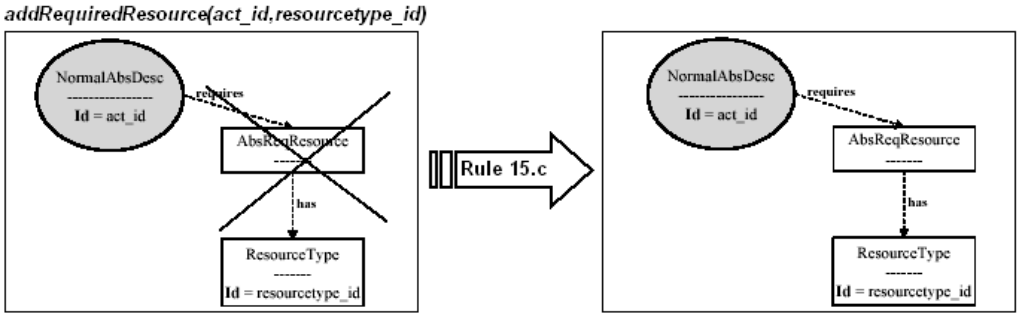


Figura C.7: Regra para definição dos recursos exigidos por uma atividade (REIS,2002)

- *addRequiredRoleType*: define os papéis que desempenharão a atividade normal.

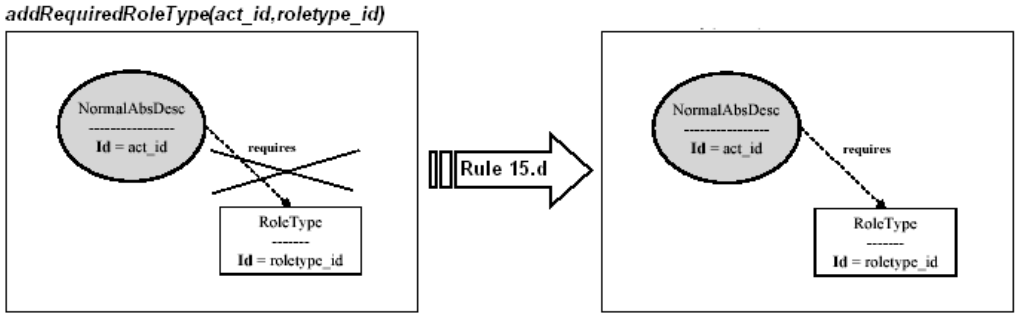


Figura C.8: Regra para definição dos papéis que realizam a atividade (REIS,2002)

- *addRequiredGroupType*: define os grupos que desempenharão a atividade normal.

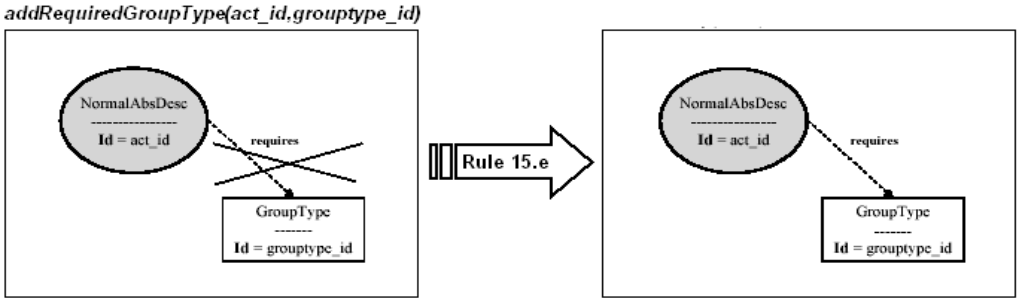


Figura C.9: Regra para definição dos grupos que realizam a atividade (REIS,2002)

- *enablePolicyInActivity*: habilita uma política em uma atividade abstrata.

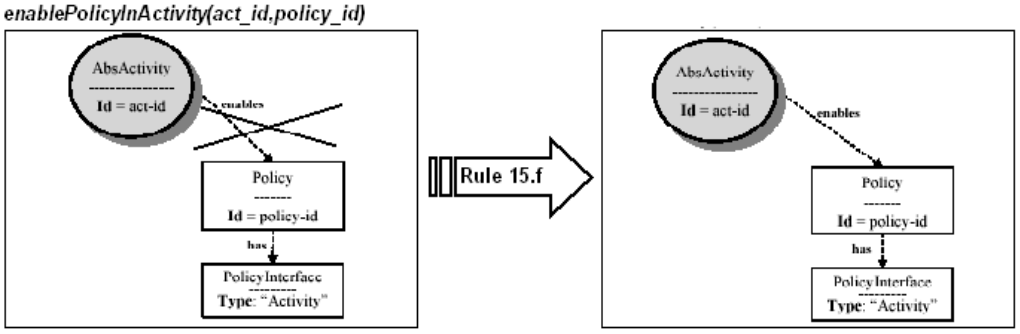


Figura C.10: Regra para habilitação de uma política em uma atividade (REIS,2002)

- *addRequiredToolType*: define o tipo da ferramenta utilizada em uma atividade automática.

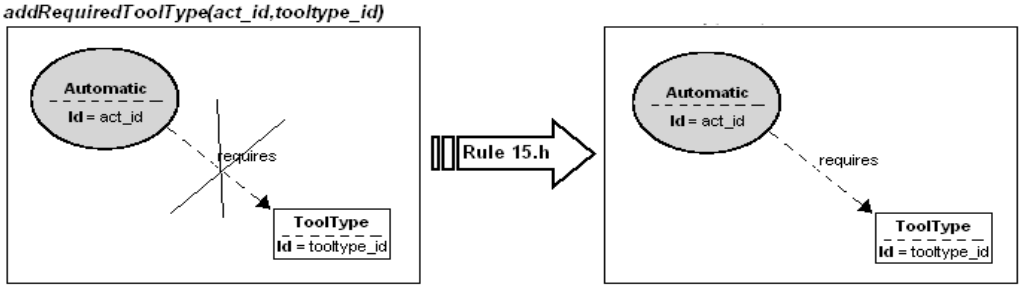


Figura C.11: Regra para definição da ferramenta utilizada na atividade

C.4 Regras para Checagem da Consistência de Conexões

Sempre que uma atividade é excluída de um modelo de processo, algumas das conexões, que possuíam esta atividade como origem ou destino, podem ficar inconsistentes, devendo ser normalizadas. As funções a seguir não possuem parâmetros, tratam da consistência de conexões e são invocadas automaticamente sempre que a condição esperada é atendida após a exclusão de uma atividade:

- *checkConsistencyOfSimpleCons*: trata da consistência de conexões simples de dependência, as quais devem possuir sempre uma origem e um destino. É definida pelas regras 2.a e 2.b.

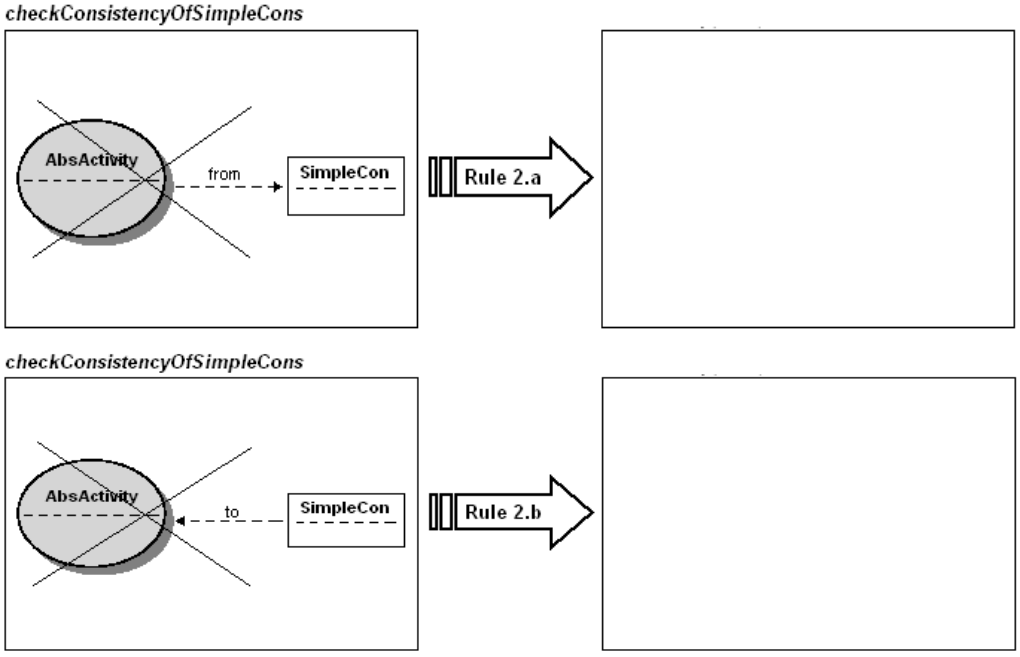
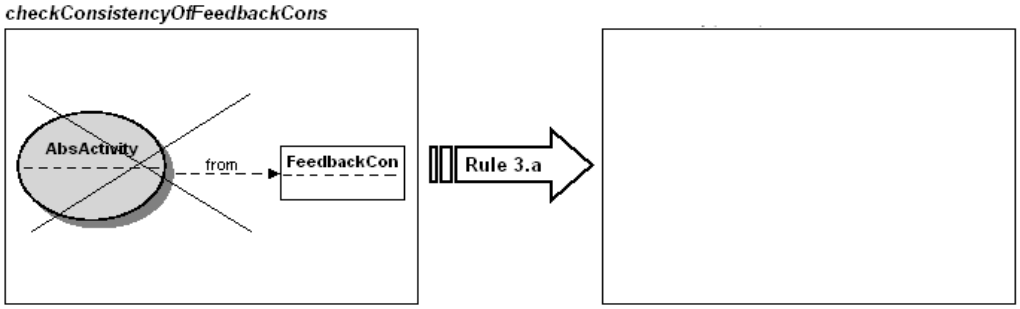


Figura C.12: Regras para checagem da consistência de conexões simples

- *checkConsistencyOfFeedbackCons*: trata da consistência de conexões simples de feedback, as quais devem possuir sempre uma origem e um destino. É definida pelas regras 3.a e 3.b.



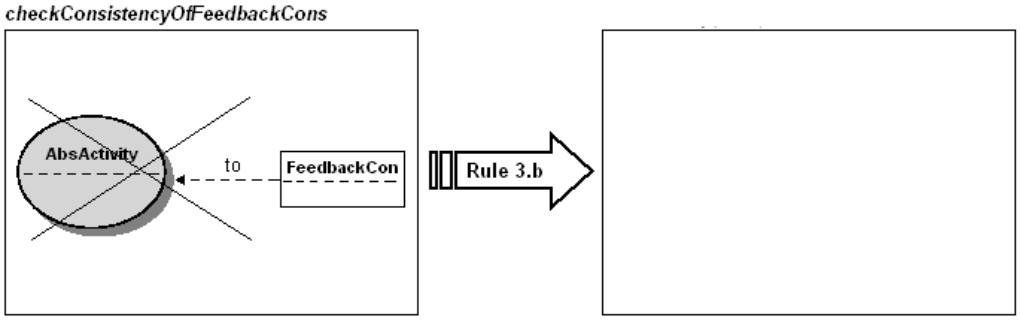
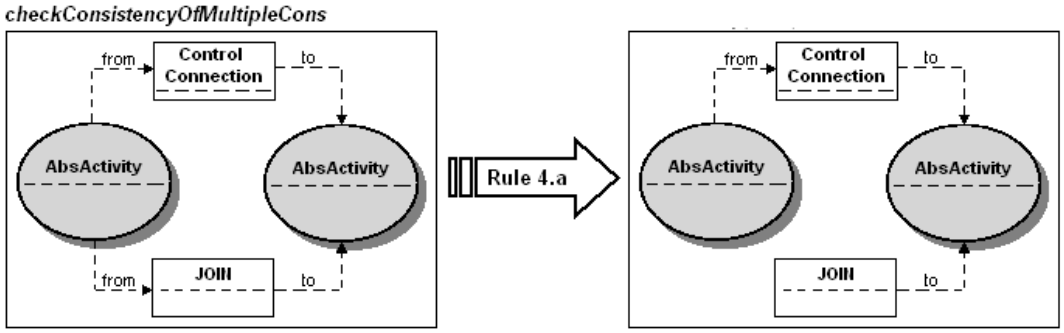
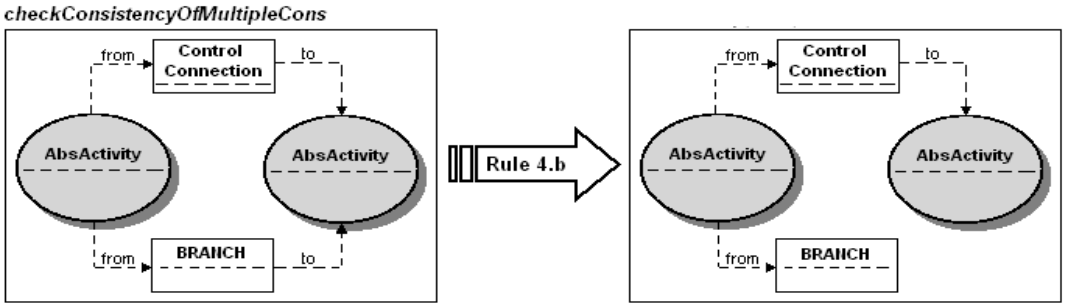
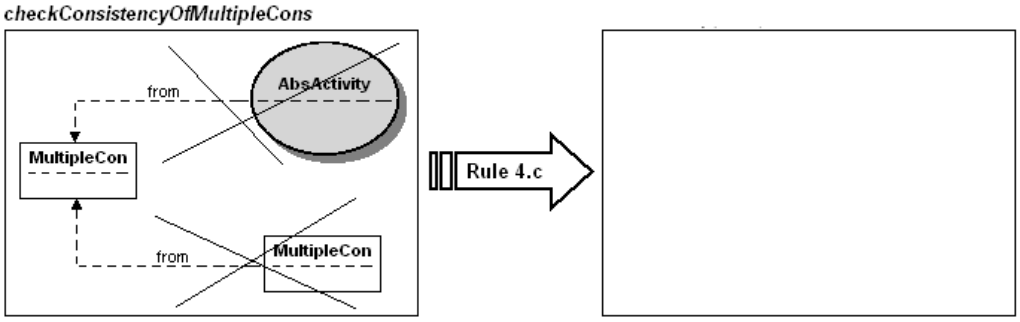


Figura C.13: Regras para checagem da consistência de conexões de *feedback*

- *checkConsistencyOfMultipleCons*: trata da consistência de conexões múltiplas, as quais devem possuir sempre pelo menos uma origem e um destino. É definida pelas regras 4.*.







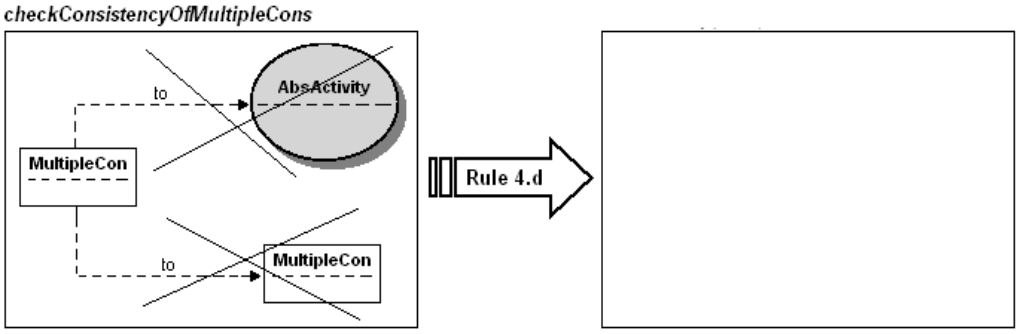


Figura C.14:Regras para consistência de conexões múltiplas

- *checkConsistencyOfArtifactCons*: trata da consistência de conexões de artefatos, as quais, se estiverem ligadas a um artefato concreto devem possuir pelo menos uma atividade ou conexão múltipla como destino e, se estiverem ligadas a um artefato abstrato devem apresentar pelo menos uma atividade como origem. È definida pelas regras 5.a e 5.b.

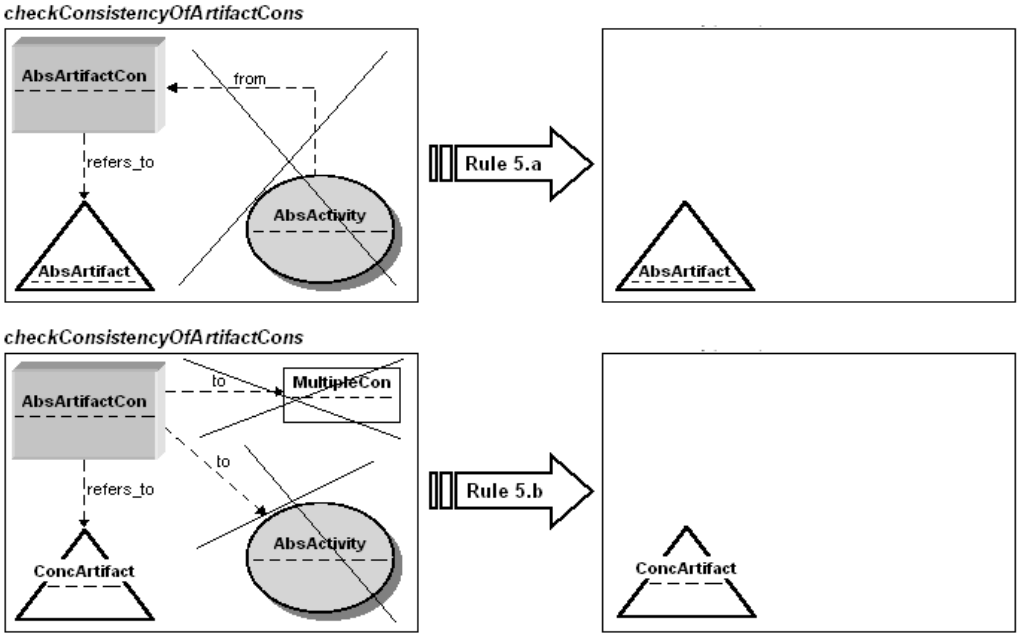


Figura C.15: Regras para checagem da consistência de conexões concretas de artefato

- *checkActiviesIO*: embora pareça meio fora do contexto, esta função é complementar à anterior, só que se aplica às atividades abstratas.

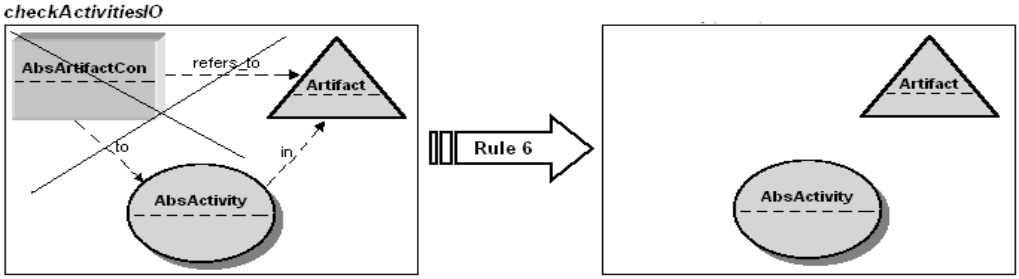


Figura C.16: Regra para checagem dos artefatos de entrada/saída de uma atividade

C.5 Regras para Inclusão de Conexões

Sempre que uma atividade é incluída de um modelo de processo para outro, as conexões, que envolvem a mesma e que permanecerão consistentes no novo modelo de processo, também devem ser incluídas juntamente. As funções a seguir são responsáveis por este procedimento, incluindo as conexões sem que haja duplicação ou formação de ciclos no novo modelo de processo:

- *addSimpleCon*: responsável pela inclusão de conexões simples, recebendo como parâmetros os identificadores das atividades de origem e destino, além do tipo de dependência. É definida pela regra 7.

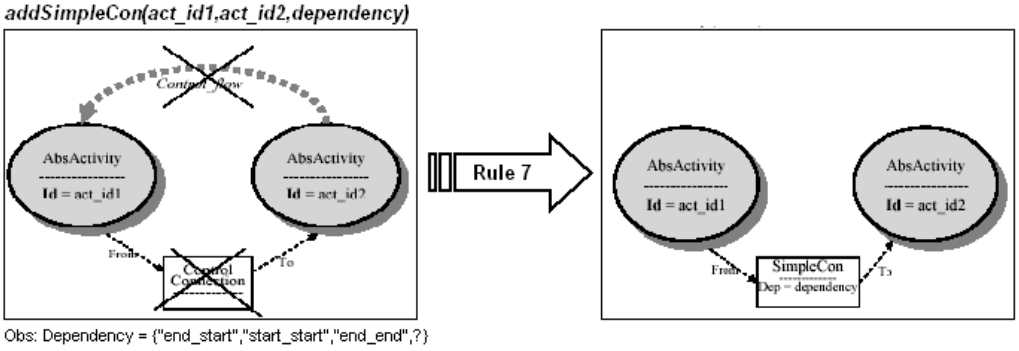


Figura C.17: Regra para inclusão de conexões simples (REIS,2002)

- *addFeedbackCon*: responsável pela inclusão de conexões de feedback, recebendo como parâmetros os identificadores das atividades de origem e destino, além da condição de retorno. É definida pela regra 8.

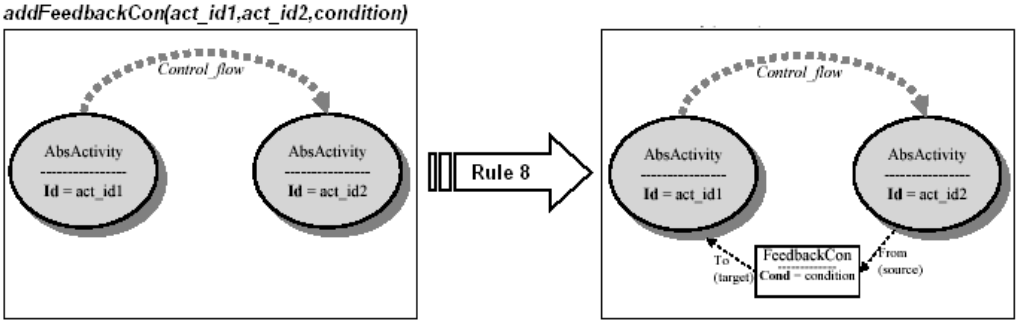


Figura C.18: Regra para inclusão de conexões de *feedback* (REIS, 2002)

- O conjunto de funções a seguir é responsável pela inclusão de conexões múltiplas, recebendo como parâmetros o identificador da conexão e o tipo de dependência.

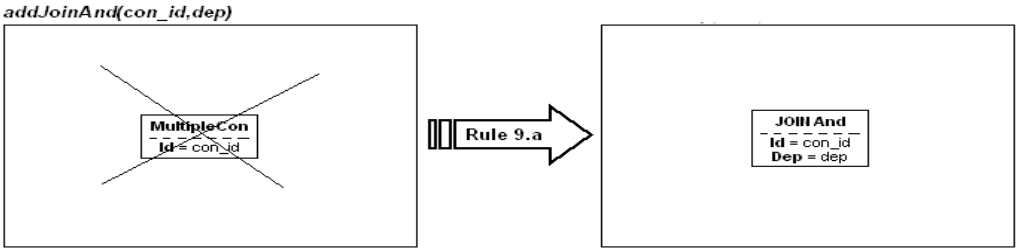


Figura C.19: Regra para inclusão de conexões Join And

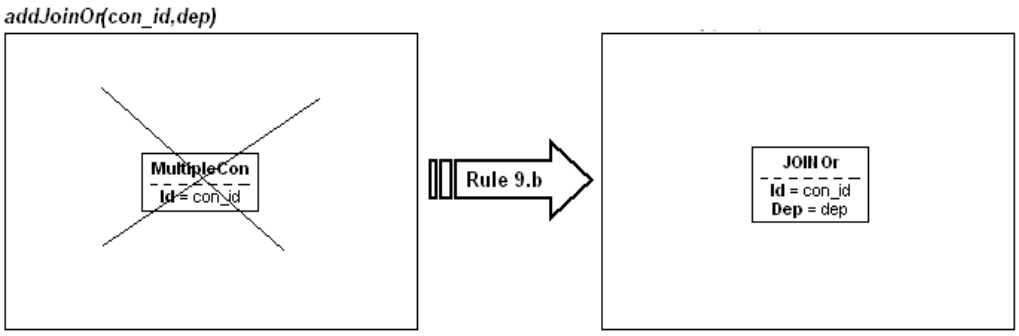


Figura C.20: Regra para inclusão de conexões Join Or

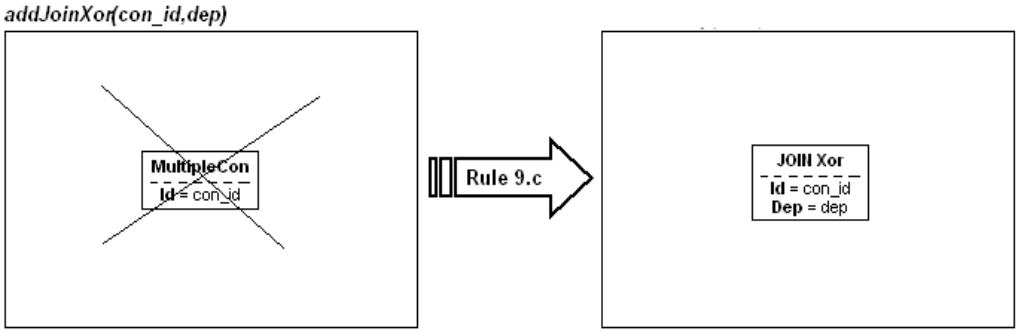


Figura C.21: Regra para inclusão de conexões Join Xor

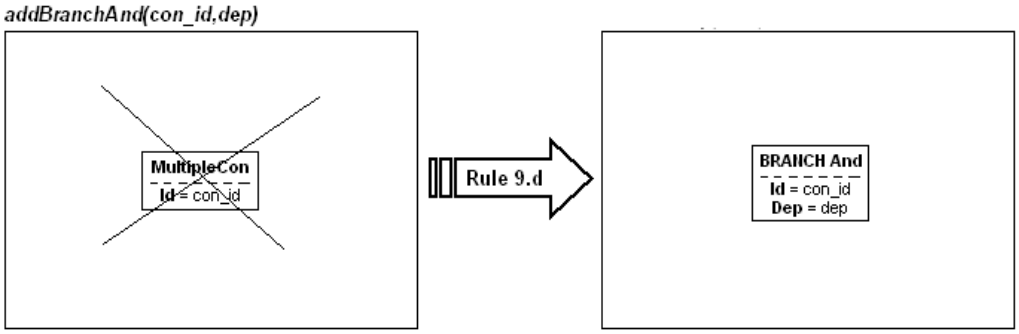


Figura C.22: Regra para inclusão de conexões Branch And

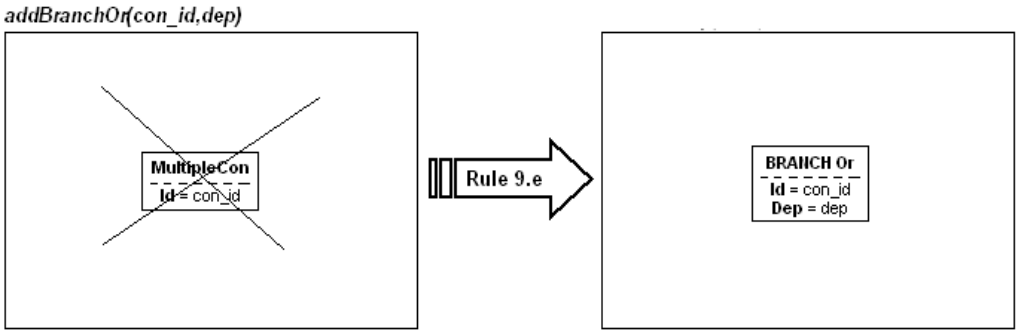


Figura C.23: Regra para inclusão de conexões Branch Or

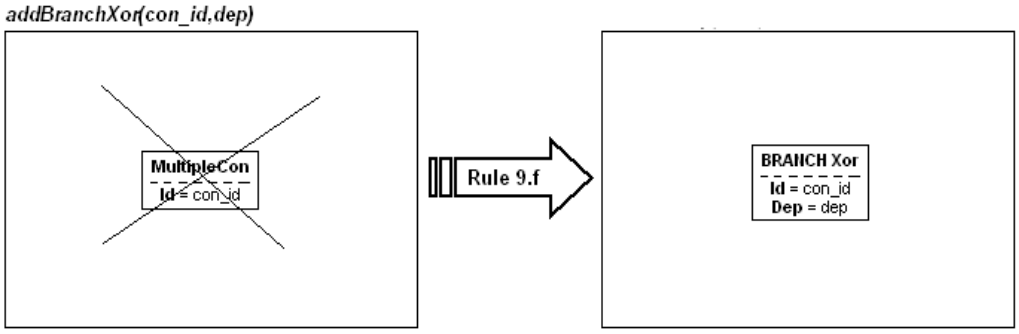


Figura C.24: Regra para inclusão de conexões Branch Xor

- *addArtifactCon*: responsável pela inclusão de conexões de artefato, recebendo como parâmetros o identificador e o tipo do artefato que será instanciado com a conexão. É definida pela regra 11.

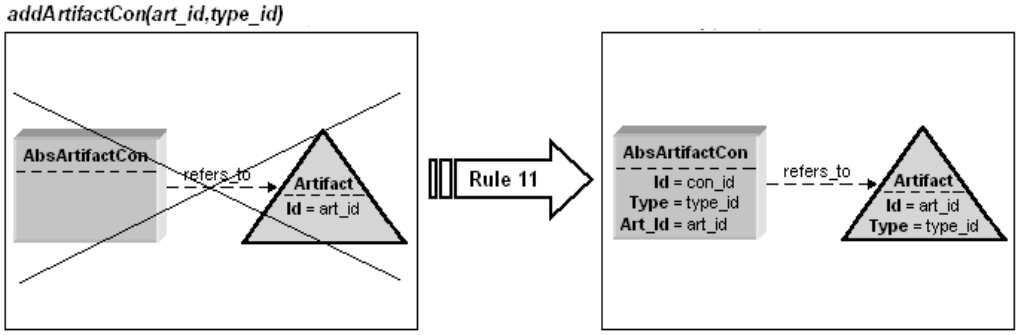
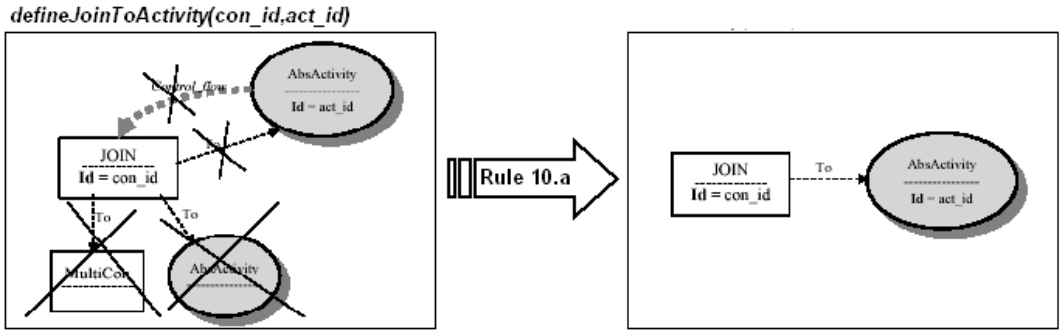


Figura C.25: Regra para inclusão de conexões de artefato

C.6 Regras para Redefinição das Origens e Destinos de Conexões Incluídas

Sempre que uma conexão múltipla ou de artefato é copiada de um modelo de processo para outro, se faz necessário redefinir suas origens e destinos ainda válidos no novo modelo de processo. O conjunto de funções definidas pelas regras 10.* tratam das conexões múltiplas, enquanto aquelas definidas pelas regras 12.* tratam de conexões de artefato.



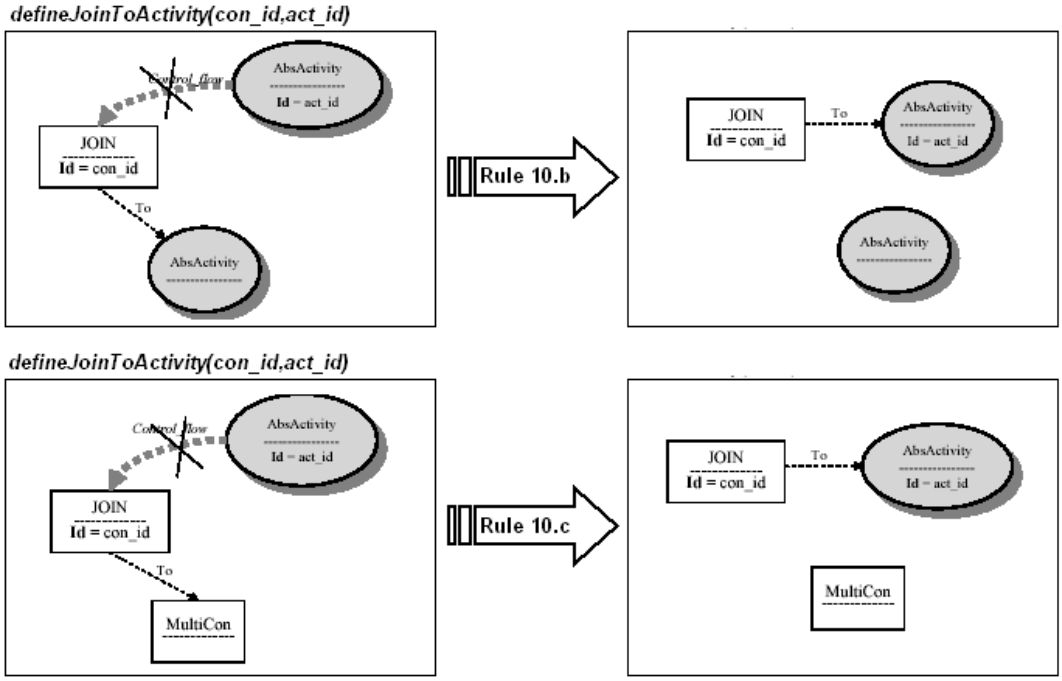


Figura C.26: Regras para definir uma atividade abstrata como destino da conexão Join (REIS,2002)

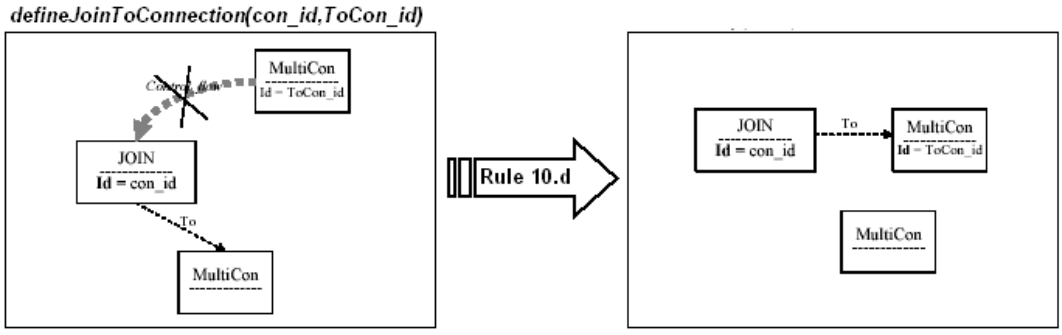
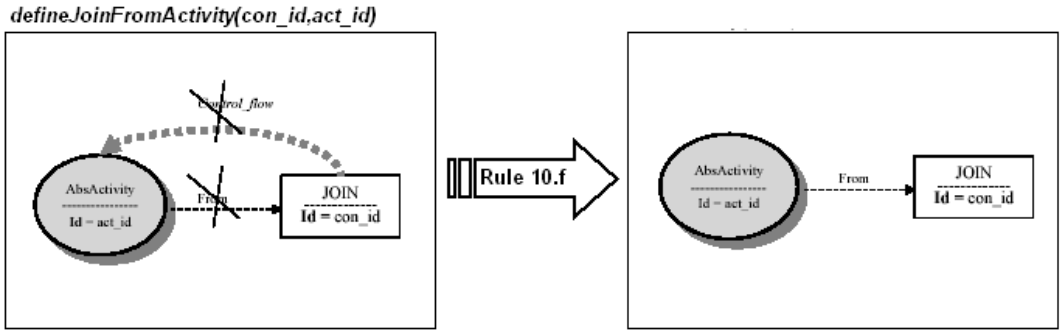


Figura C.27: Regra para definir uma conexão múltipla como destino da conexão Join (REIS,2002)



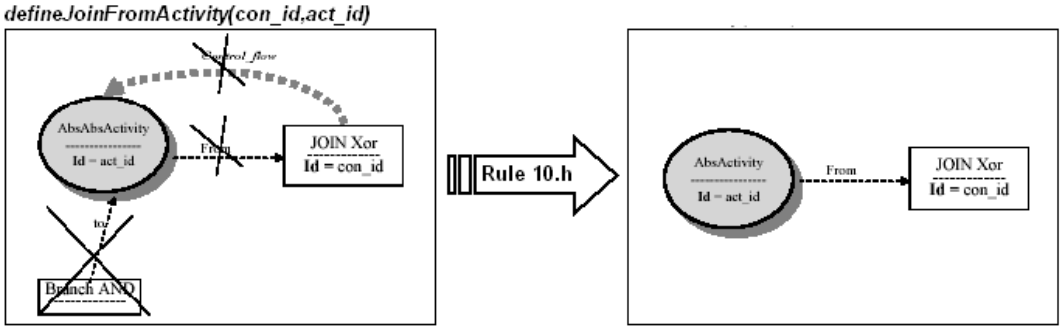


Figura C.28: Regras para definir uma atividade abstrata como origem da conexão Join (REIS,2002)

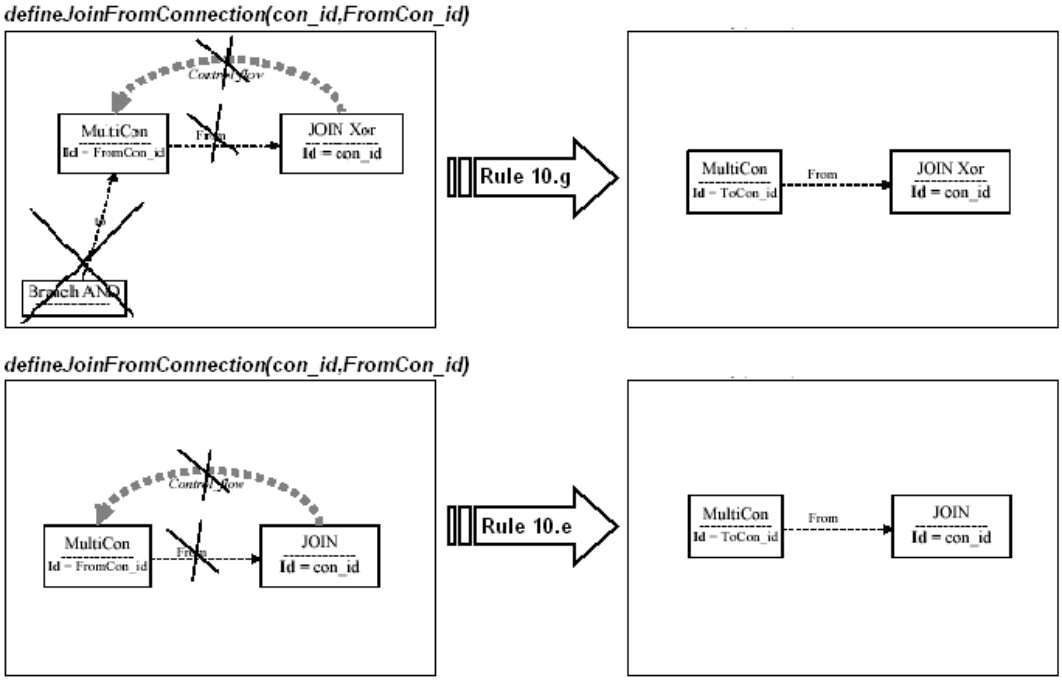
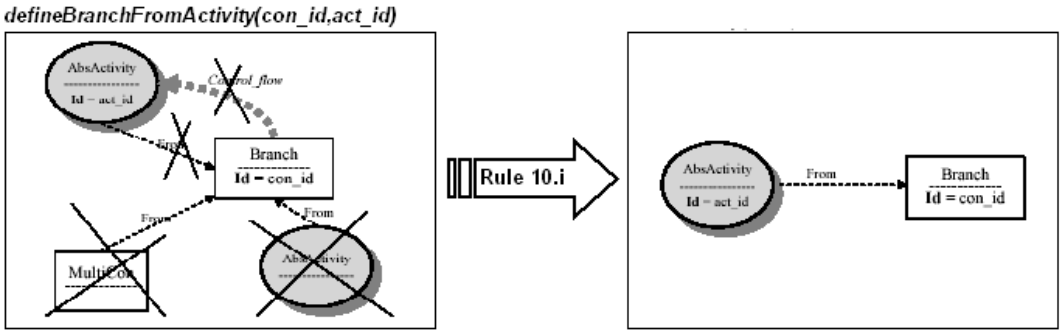


Figura C.29: Regras para definir uma conexão múltipla como origem da conexão Join (REIS,2002)



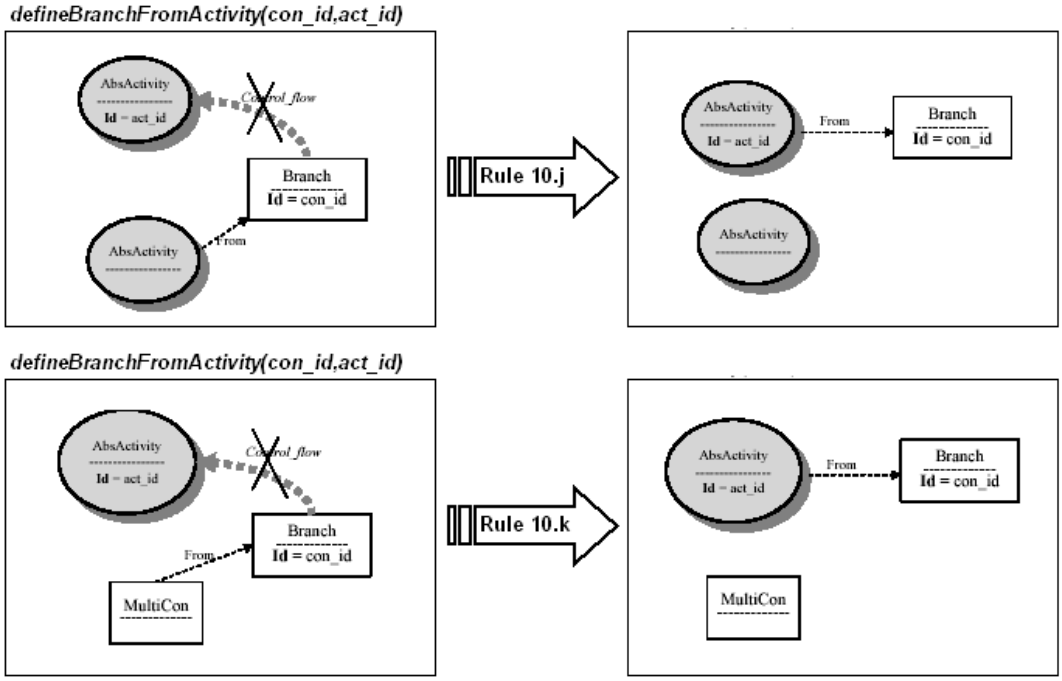
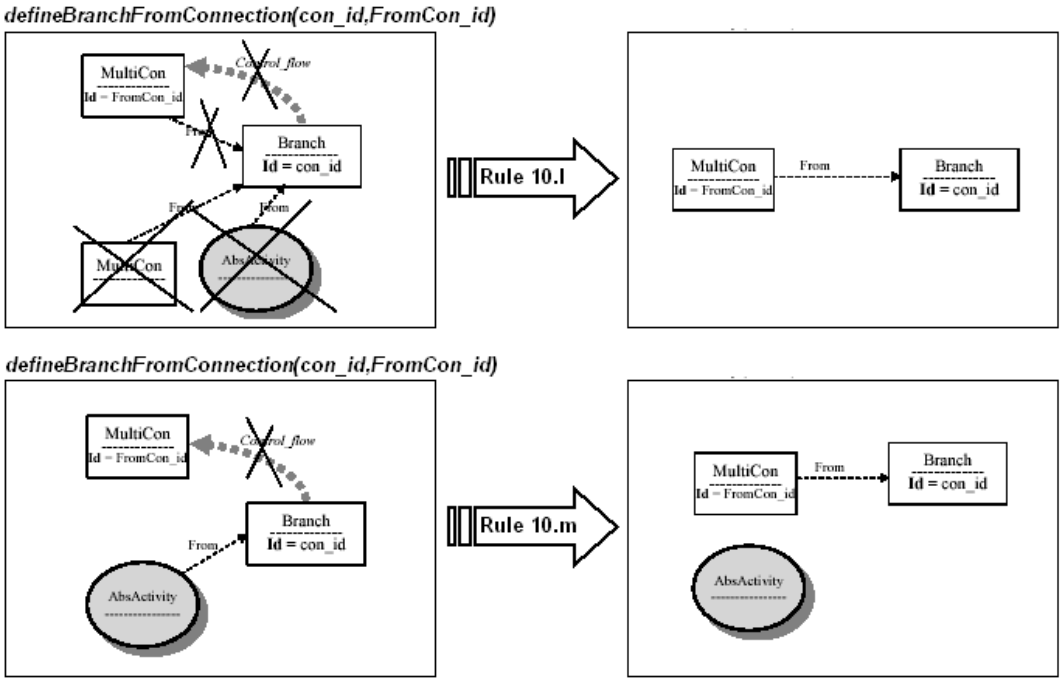


Figura C.30: Regras para definir uma atividade abstrata como origem da conexão Branch (REIS,2002)



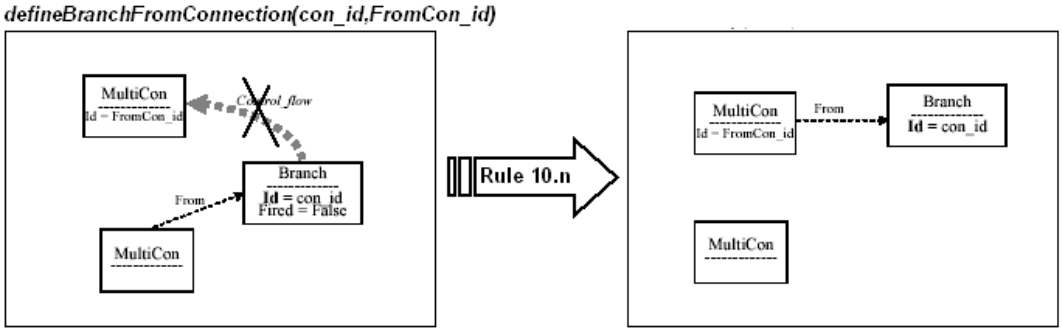


Figura C.31: Regras para definir uma conexão múltipla como origem da conexão Branch (REIS,2002)

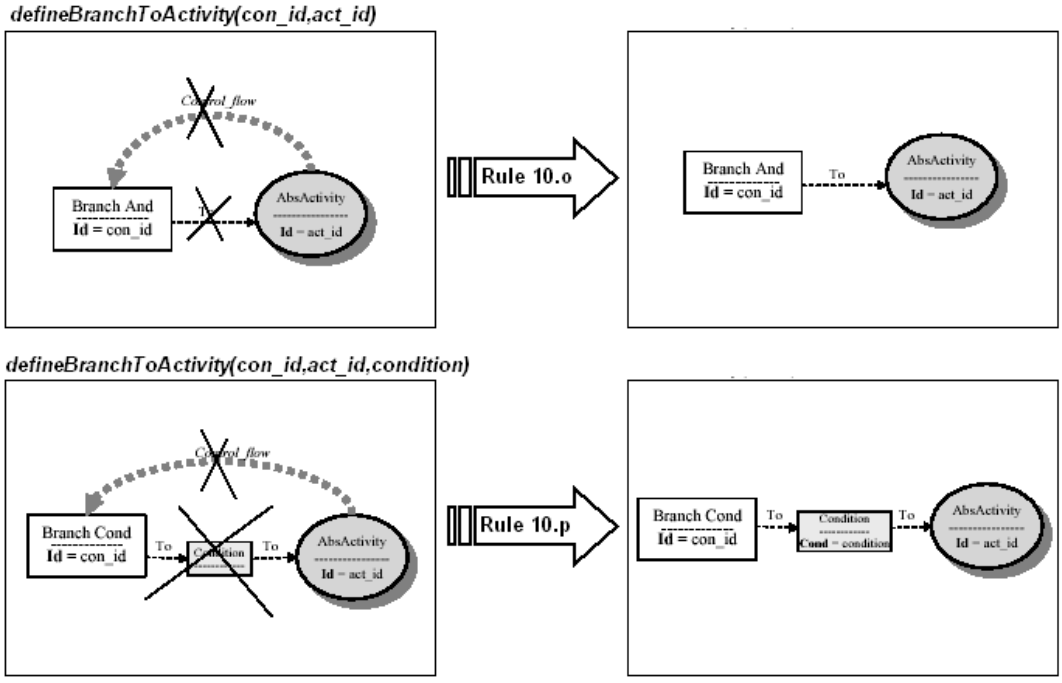
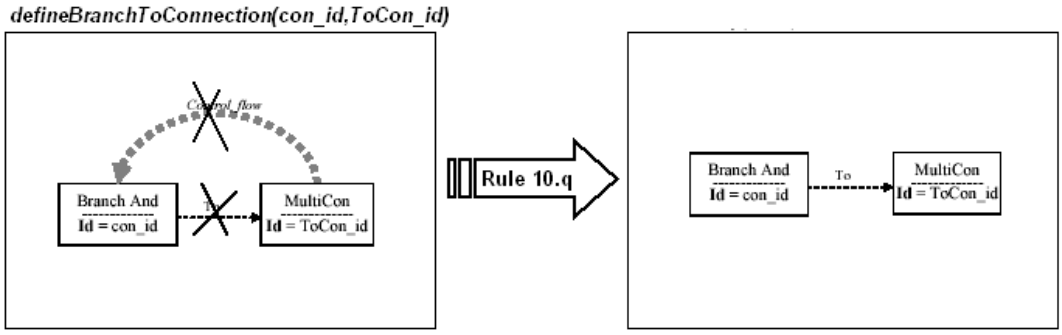


Figura C.32: Regras para definir uma atividade abstrata como destino da conexão Branch (REIS,2002)



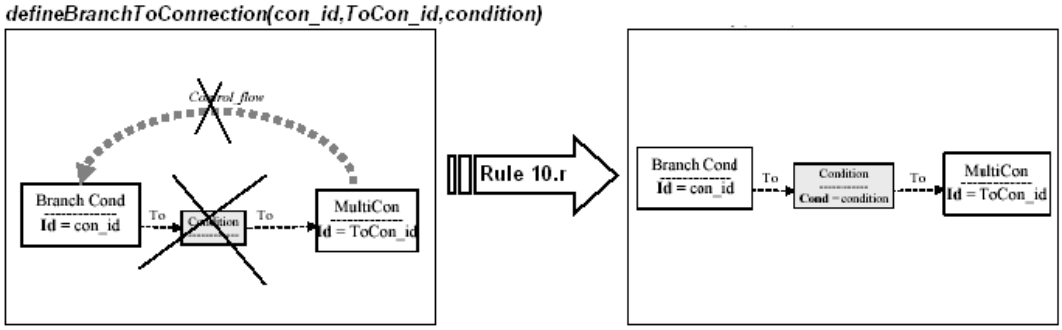


Figura C.33: Regras para definir uma conexão múltipla como destino da conexão Branch (REIS,2002)

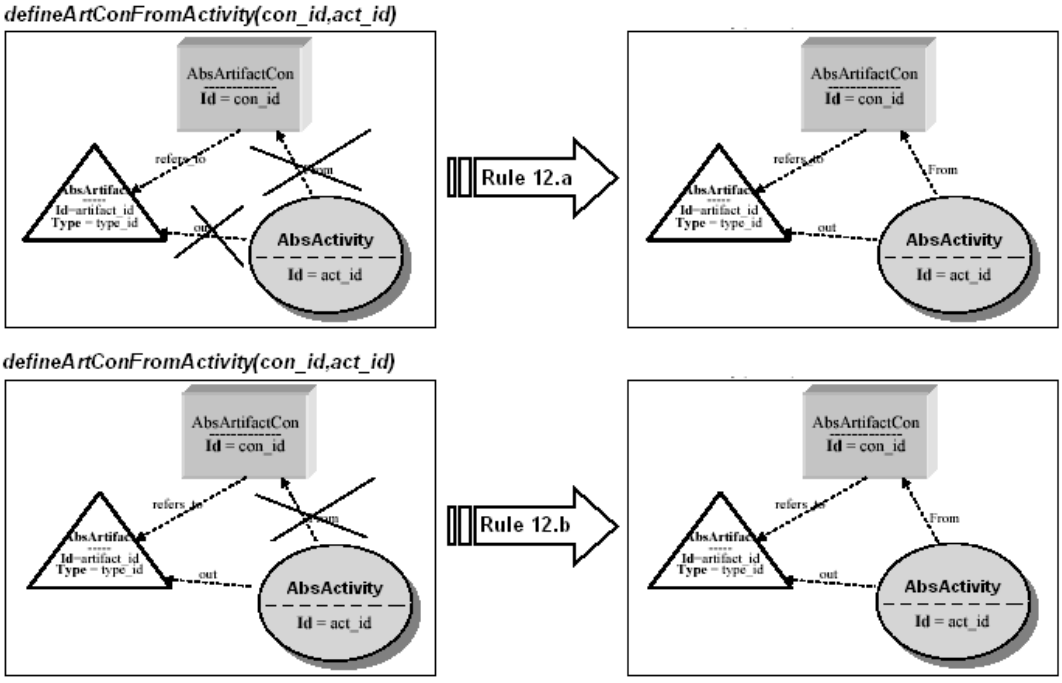
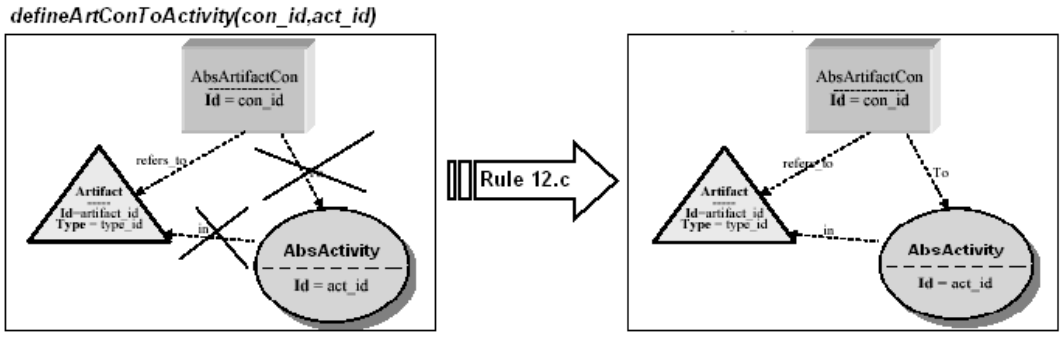


Figura C.34: Regras para definir uma atividade abstrata como origem da conexão de artefato (REIS,2002)



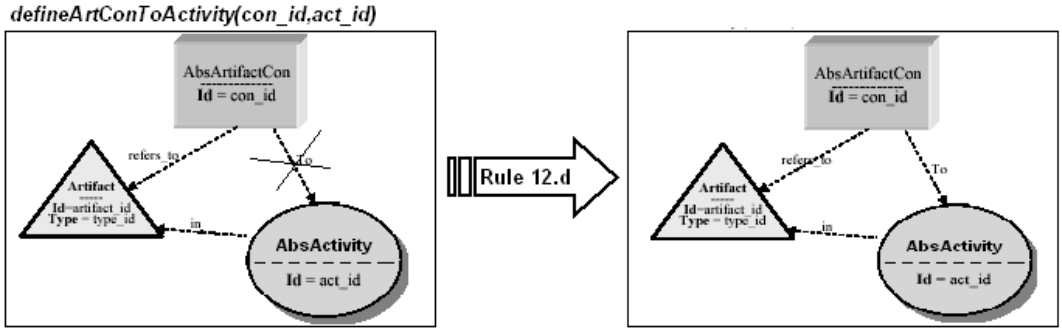


Figura C.35: Regras para definir uma atividade abstrata como destino da conexão de artefato (REIS,2002)

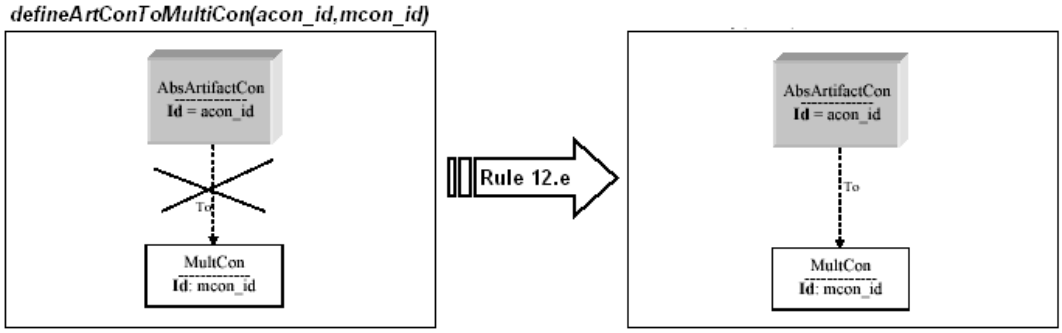
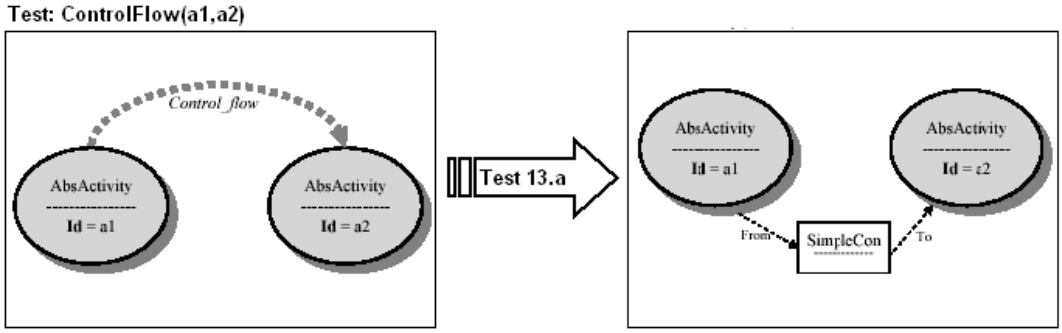


Figura C.36: Regra para definir uma conexão múltipla como destino da conexão de artefato (REIS,2002)

C.7 Regras para detecção de fluxo de controle

A função a seguir é utilizada na detecção de fluxo de controle somente entre atividades abstratas, entre atividades abstratas e conexões múltiplas ou somente entre conexões múltiplas. É definida pelo conjunto de regras 13.*



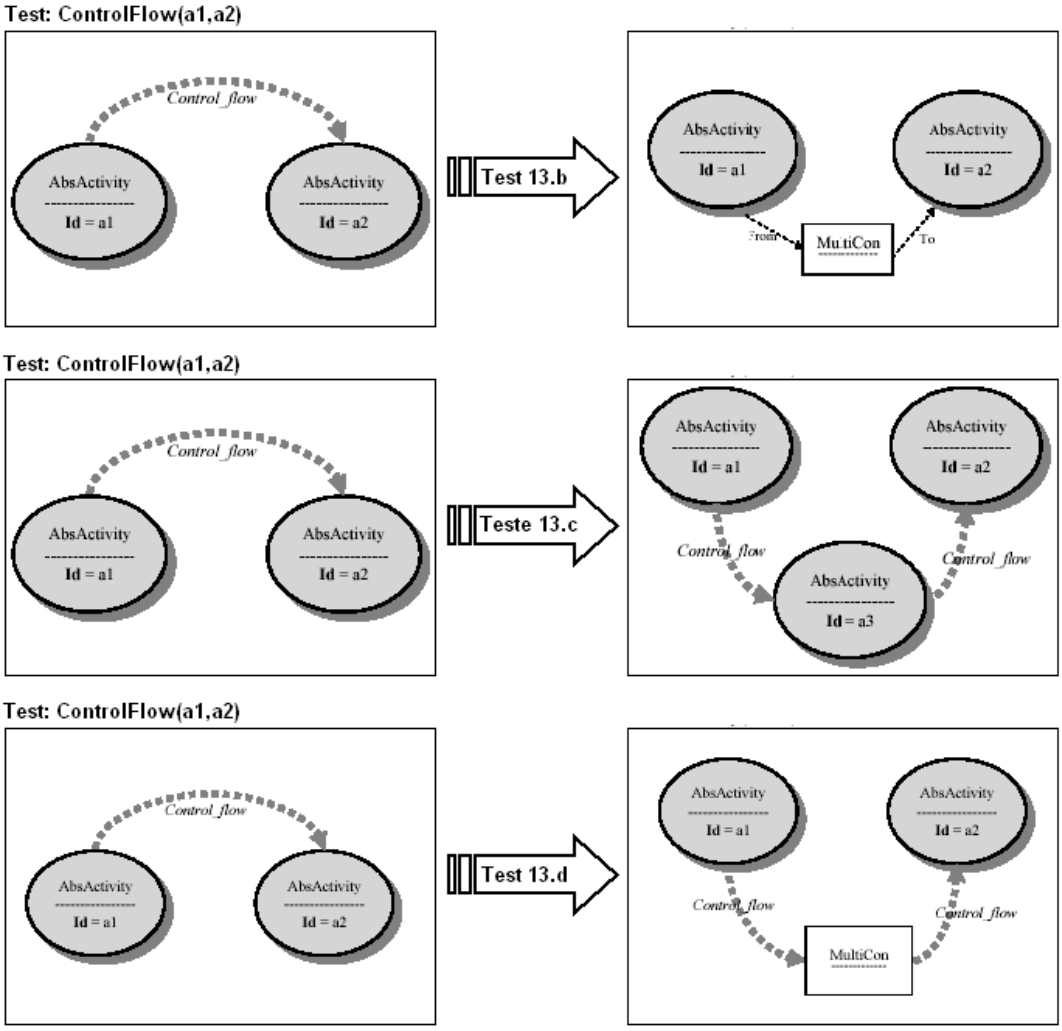
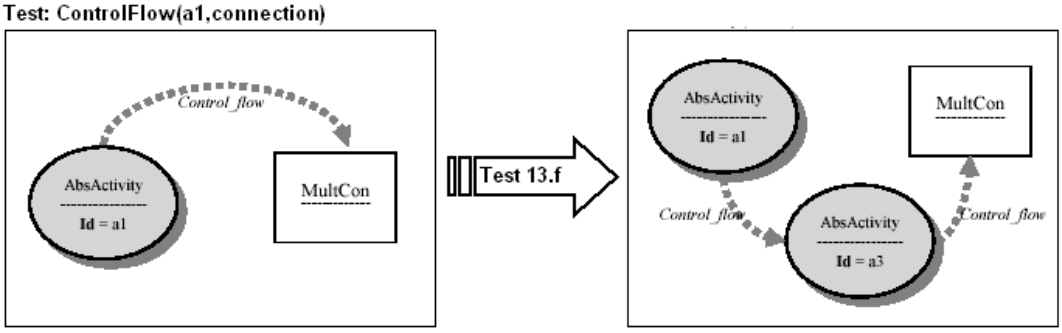


Figura C.37: Regras para detecção de fluxo de controle entre atividades (REIS,2002)



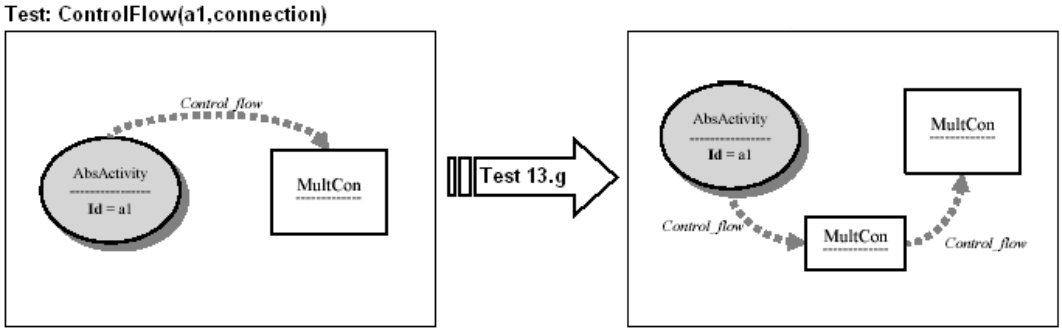


Figura C.38: Regras para detecção de fluxo de controle entre atividade e conexões (REIS,2002)

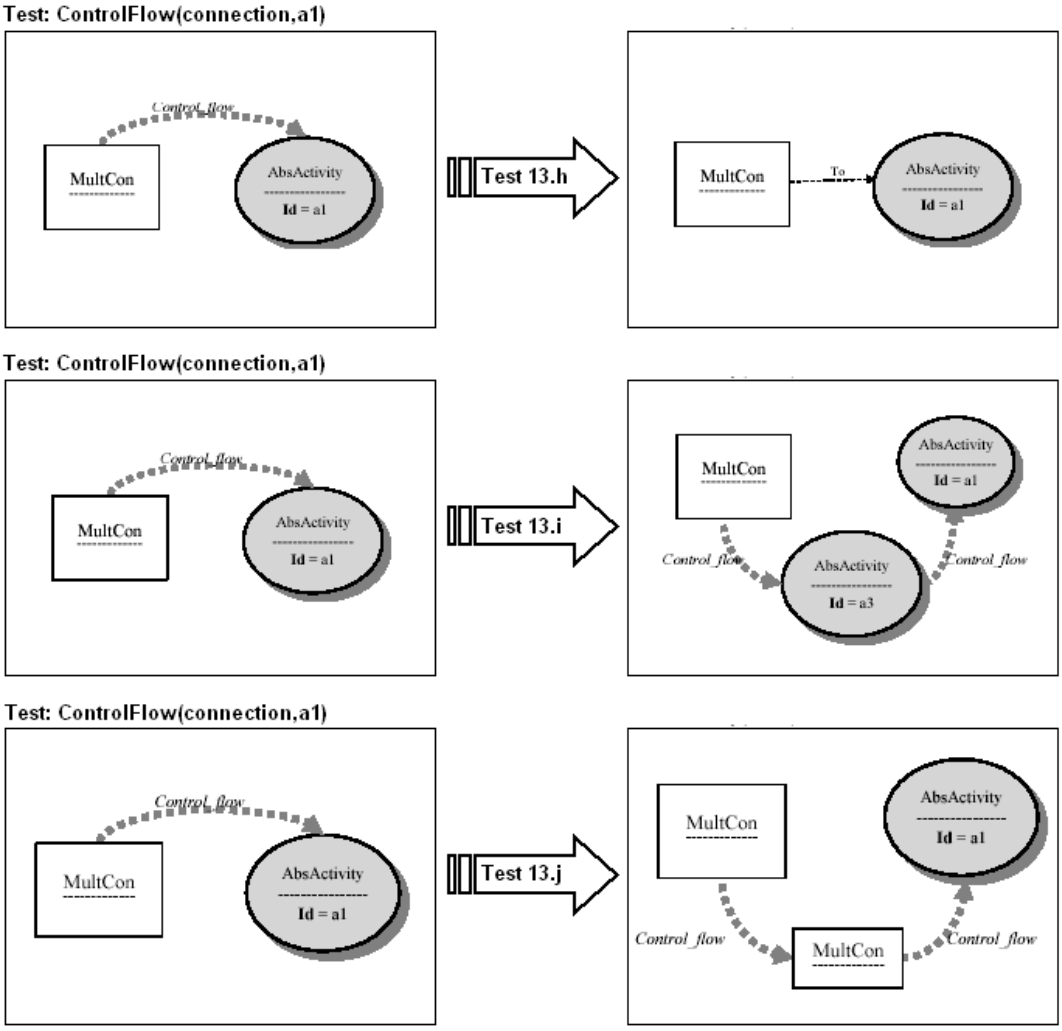


Figura C.39: Regras para detecção de fluxo de controle entre conexões e atividades (REIS,2002)

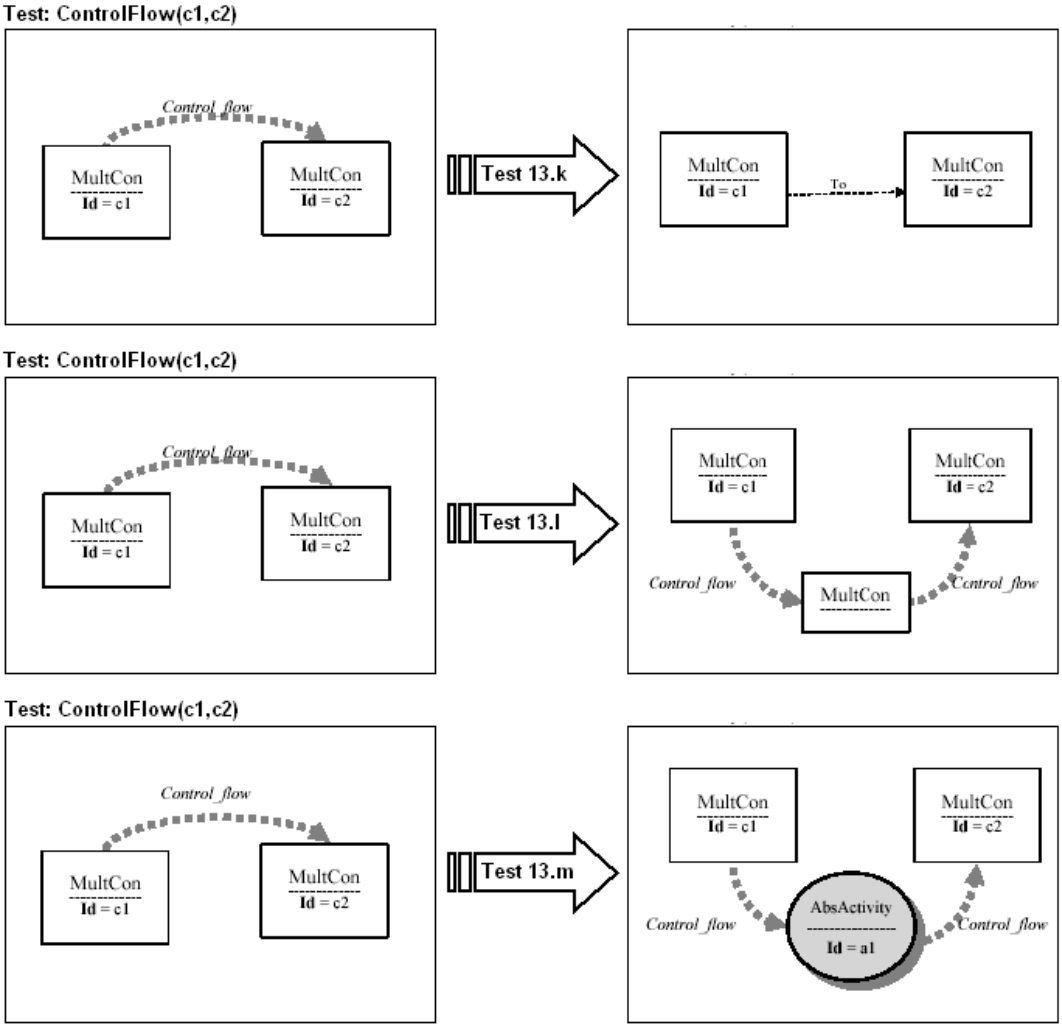


Figura C.40: Regras para detecção de fluxo de controle entre conexões múltiplas (REIS,2002)