Universidade Federal de Pernambuco
Graduação em Bacharelado em Ciências da Computação
Centro de Informática

# Construção Gráfica de Processos de Desenvolvimento e Geração de uma Ontologia de Processo de Software

Trabalho de Graduação

**Aluno:** Rodrigo Morais Araujo <rodrigomaraujo@gmail.com>

**Orientador:** Prof. Ph.D. Alexandre Marcos Lins de Vasconcelos <amlv@cin.ufpe.br>

**Co-orientador:** Prof. M.Sc. Sandro Ronaldo Bezerra Oliveira <srbo@cin.ufpe.br>

**Recife, Agosto de 2005**

Universidade Federal de Pernambuco
Centro de Informática

RODRIGO MORAIS ARAUJO

# Construção Gráfica de Processos de Desenvolvimento e Geração de uma Ontologia de Processo de Software

Monografia apresentada ao Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para obtenção do Grau de Bacharel em Ciência da Computação sob orientação do Prof. Ph.D. Alexandre Vasconcelos e co-orientação do aluno de doutorado M. Sc. Sandro Ronaldo Bezerra Oliveira.

Recife, Agosto 2005

# Agradecimentos

Em primeiro lugar gostaria de agradecer à minha família que me deu condições necessárias para alcançar esse sonho de concluir minha graduação.

Aos meus amigos e colegas de estudo, por me ensinarem lições humanas essenciais para qualquer profissional e cidadão, e que livro nenhum consegue ensinar.

À minha noiva, Ana Beatriz, que me apoiou nos momentos difíceis dessa vitória que estou alcançando.

Aos professores, que se dedicam nobremente a passar um pouco de seus conhecimentos e experiências a nós alunos. Em especial Prof. Alexandre Vasconcelos que me orientou nesse trabalho de conclusão, e a Sandro Oliveira, que sem dúvida me ajudou bastante nessa reta final sendo meu co-orientador.

# Resumo

A experiência vem mostrando que os projetos de desenvolvimento de software, em sua maioria, encontram problemas em seu gerenciamento de tempo e custo, na grande maioria das vezes esses problema gerenciais são originados pela falta de um processo de software disciplinado. Desde que essas constatações foram feitas surgiram esforços como o da OMG para definir metamodelos para definição de processos de software, o que resultou no SPEM. Fazendo uso dessas notações observa-se muitos ambientes voltados ao processo sendo propostos na literatura especializada, e entre eles podemos observar o ImPProS. Este trabalho está inserido como uma parte dos estudos para definição do ImPProS, no que diz respeito a modelagem gráfica de processos de software, assim como representar processos de software de forma lógica para que máquinas automáticas de inferência possam através de uma base de conhecimento de processos, sugerir processos e melhorias para os já existentes. O resultado apresentado será utilizado para compor o ambiente voltado ao processo ImPProS.

# Índice

# Lista de Figuras

# Lista de Tabelas

# 1. Introdução

A experiência da indústria de software mostra que o insucesso dos projetos tem sua principal razão na falta de um processo de software disciplinado, ou seja, na falta de um mecanismo que habilite o gerenciamento e controle da qualidade dos produtos. Seguindo a mesma tendência, já é amplamente aceito que a qualidade de um produto de software seja fortemente determinada pela qualidade do processo utilizado durante o seu desenvolvimento e manutenção [PFLEEGER98]. Afinal, não se consegue ter certeza da manutenibilidade ou fidedignidade apenas usando a ferramenta.

No entanto, a definição de um processo de software não é uma atividade simples; exige experiência e envolve o conhecimento de muitos aspectos da engenharia de software. A dificuldade em definir processos encontra-se na ausência de um processo de software possível de ser genericamente aplicado. Os processos variam porque são diferentes os tipos de sistemas, os domínios de aplicação, as equipes, as organizações e as próprias restrições de negócio, tais como, cronograma, custo, qualidade e confiabilidade [MACHADO00].

## 1.1. Contexto do Trabalho

Neste contexto Oliveira propôs, em [OLIVEIRA05], a definição de um ambiente para a implementação de processo de software, o ImPProS (Implementação Progressiva de Processos de Software), com os objetivos de possibilitar: a especificação dos processos de acordo com o domínio do projeto específico e das características da organização; a instanciação do processo de software de acordo com as propriedades de cada projeto; sua simulação a partir dos parâmetros de configuração (prazo, pressões, custo, recursos, etc.); uma execução (automação) mais próxima do que se espera para

um processo organizacional; e uma avaliação a partir da coleta de métricas desta execução.

## 1.2. Motivação

O presente trabalho é motivado por uma visão do mercado brasileiro atual que aposta cada vez mais na qualificação dos processos utilizados pela indústria de software em iniciativas de certificações internacionais como CMM/CMMI e ISO, assim como recentes esforços nacionais para construir um processo de qualidade a um custo mais baixo para a realidade financeira brasileira, o MPS.Br.

## 1.3. Objetivos

O objetivo deste trabalho é colaborar com esses esforços por adoção de processos de qualidade definindo um ambiente gráfico de modelagem destes processos assim como uso de regras lógicas para inferência automática com base em processos da natureza do desenvolvimento de software. É fato perceber que a real importância desta iniciativa está em prover ao usuário, no momento da definição de processo de software, a manipulação dos componentes (atividades, artefatos, recursos, procedimentos, etc.) do processo de software a partir de representações diagramáticas (notações gráficas). Além disso, no contexto da execução de processos de software, este trabalho possui a sua real utilidade no que tange o acompanhamento da automação do processo a partir do uso de uma linguagem que possa ser interpretada por uma máquina de processo (mecanismo que captura a definição do processo em uma linguagem de baixo nível e interpreta a sua execução a partir dos termos definidos).

## 1.4. Metodologia de Trabalho

Neste trabalho será utilizado o estudo bibliográfico de teses anteriores que seguem a linha de definição e automatização de processos de software, definindo modelando e comparando estes trabalhos anteriores às necessidades do ambiente proposto por [OLIVEIRA05]. Será também feito o uso de ferramentas de análise e modelagem de software para garantir a qualidade do trabalho final apresentado.

## 1.5. Estrutura do Trabalho

Além desse capítulo introdutório, o estudo vai contemplar no capítulo 2 uma descrição mais ampla do ambiente onde está inserido o objetivo desse trabalho. No capítulo 3 será apresentado o meta-modelo SPEM, definido pela OMG com apoio de um consórcio de industria de software, no qual a modelagem gráfica desse trabalho se baseará, assim como uma ferramenta de modelagem existente que faz uso da notação proposta pela OMG. Em seguida, no capítulo 4 veremos as modificações sugeridas sobre a ferramenta escolhida para que a mesma suporte melhor o ambiente descrito no capítulo 2, e também o mapeamento de regras lógicas encontradas na literatura especializada para as necessidades do ambiente onde o trabalho está inserido.

No capítulo 5 será apresentado um estudo de caso para confirmar que as modificações analisadas e propostas realmente satisfazem um processo de software real. E no último no capítulo 6 as conclusões e indicação de trabalhos futuros deste trabalho.

# 2. Ambiente de Implementação de Processo de Software

Neste capítulo será apresentado o que é um processo de desenvolvimento de software, o porquê do uso de ambientes de desenvolvimento centrados no processo e por último será descrito o ambiente de desenvolvimento de software proposto por [OLIVEIRA05], denominado ImPProS.

## 2.1. Processo de Software

Por processo se entende por uma sucessão de estados e mudanças, maneira como se realiza alguma coisa, de acordo com normas, métodos e/ou técnicas. Processo de software seria o conjunto de estados e suas transições de acordo com métodos, métodos e/ou técnicas que compõem atividades com a finalidade de desenvolver algum produto de software.

No processo de software podemos decompor a tarefa principal, que é desenvolver o software em questão, em tarefas menores até que se alcance um bom nível de detalhamento que possa ser melhor gerenciado. Estas tarefas ainda podem ser desmembradas em unidades ainda menores que serão chamadas de atividades. Essas atividades é que serão executadas pelos recursos para que seja alcançado o objetivo maior que é o software requerido, funcionando de acordo com as necessidades do usuário.

Os recursos por sua vez podem ser de natureza humana ou ferramentas apropriadas para automatizar tarefas. Como é o caso de scripts de testes que executam a atividade de testar o sistema de forma automática e automatizada, cabendo ao engenheiro de testes apenas programar uma única vez o teste que será realizado de forma automática nas vezes subseqüentes.

Outro conceito importante em processos de software é o conceito de artefato, que é um produto elaborado ou modificado por uma atividade. Um artefato também pode ser entrada de conhecimento para uma dada atividade do processo.

## 2.2. Ambiente de Desenvolvimento Centrado no Processo

O surgimento da tecnologia CASE (Computer Aided Software Engineering) - Engenharia de Software Auxiliada por Computador, exerceu um enorme impacto sobre a área de engenharia de software. A idéia de utilizar software para auxiliar a produção de software foi bem recebida pelos desenvolvedores. As ferramentas CASE proporcionam uma sólida estrutura às metodologias e métodos de desenvolvimento de software.

A definição de Ambiente de Desenvolvimento de Software veio em decorrência ao reconhecimento de que a comunicação e a coordenação entre todas as ferramentas usadas no desenvolvimento e manutenção de software são essenciais para a produção eficiente de software de qualidade.

Ambientes de Desenvolvimento de Software, ou simplesmente ADS, são considerados uma evolução do conceito de ferramentas CASE e têm como principal objetivo prover um ambiente no qual produtos de software de grande porte possam ser desenvolvidos através da integração de um conjunto de ferramentas que suportam métodos de desenvolvimento, apoiados por uma estrutura que permite a comunicação e cooperação entre as ferramentas [OLIVEIRA05].

Uma evolução significativa nos ADSs foi detectada com a tecnologia de processos de software. A automação do processo de software foi incorporada aos ADSs mais recentes tornando-os ADSs centrados em processo (ou orientados a processo), também conhecido na literatura como PSEE - Process-Centered Software Engineering Environment. Estes ambientes constituem uma nova geração de ADS que suportam

além da função de desenvolvimento de software, também as funções associadas de gerência e garantia da qualidade durante o ciclo de vida do software.

Um ADS centrado em processo baseia-se em uma definição explícita do processo de desenvolvimento de software. Por isso o processo de software utilizado na organização deve estar formalizado e ser obedecido. O ambiente, então, usa o modelo de processo de software descrito em uma linguagem de modelagem do processo para executar de forma independente aquelas tarefas que podem ser completamente automatizadas, coordenar tarefas mais complexas, e garantir informação de gerência atualizada, eliminando cargas administrativas desnecessárias dos usuários.

Os ADSs centrados em processo atuam de forma mais abrangente no desenvolvimento de software. As funções genéricas que podem ser suportadas por um ADS centrado em processo são, extraído de [REIS 2000]:

- Engenharia de Processos: definição e manutenção de modelos de processo de software. O ADS deve prover facilidades de definição, análise e simulação de processos;
- Engenharia de Software: desenvolvimento e manutenção de um produto de software através do seguimento de um processo de software;
- Gerência de Projetos: coordenação e monitoramento das atividades da engenharia de software a fim de garantir que o processo está sendo seguido.

### *III.1.1. Características dos ADSs Atuais*

Como já tratado na seção anterior, os ambientes de desenvolvimento de software mais recentes são os ADSs orientados ao processo. Através da observação dos ambientes propostos recentemente na literatura é possível destacar as características principais que influenciam na sua utilização. Cabe ressaltar que nem todos os ambientes orientados a processo possuem todas as características que serão apresentadas.

O uso de ambientes orientados a processos acaba resultando na definição rigorosa da execução do processo. Esta característica traz benefícios, tais como:

- Melhorar a comunicação entre as pessoas envolvidas;
- Garantir consistência do que se está sendo feito;
- Facilitar treinamento de novos usuários.

Além disso, enquanto o processo está em execução, o ambiente interage com os usuários com a finalidade de facilitar o seu trabalho e prover informações sempre que necessário. Algumas das ações realizadas pelo ambiente são:

- Prover informações que guiam o desenvolvedor a realizar seu trabalho com mais eficiência;
- Realizar ações automáticas, "liberando" assim os seus usuários de tarefas repetitivas;
- Fornecer informações sobre o processo quando necessário.

O fato de que as definições de processo podem ser reunidas em uma biblioteca para reutilização é uma das vantagens principais do uso de ambientes orientados a processo. Assim, um processo realizado com sucesso pode ser reusado por outros sem muito esforço, respeitando as características usadas na sua definição. Além disso, outro recurso de grande contribuição é a coleta automática de métricas que pode ser obtido com o uso de ambientes orientados a processo. As características de ambientes orientados a processo são apresentadas a seguir, extraídas de [OLIVEIRA05]:

- Suporte a múltiplos usuários: o sistema deve ter mecanismos para atender vários usuários ao mesmo tempo requisitando serviços do ambiente concorrentemente e mantendo a consistência do ambiente;
- Gerência de objetos: o ambiente deve prover um módulo que seja responsável por controlar o acesso e a evolução de objetos compartilhados. Geralmente este módulo

é conhecido como Repositório do ambiente e é implementado através de um sistema de gerência de banco de dados (SGBD). As versões dos documentos e produtos de software devem ser gerenciadas para permitir cooperação e consistência;

- Gerência de comunicação entre pessoas: as pessoas que estarão envolvidas no desenvolvimento de software devem ter acesso a mecanismos de comunicação, tais como mensagens eletrônicas e conferência eletrônica. Além disso, a interface disponível para comunicação entre pessoas, e entre pessoas e o ambiente deve ser adequada e dirigida a tarefas específicas para reduzir o excesso de informação contido no ambiente;
- Gerência de cooperação: a edição cooperativa de documentos e itens de software deve ser gerenciada pelo ambiente de forma que os usuários envolvidos obtenham comunicação síncrona sobre os produtos que estão manipulando. Esta característica vem do fato de alguns ambientes apoiarem o trabalho de equipes de desenvolvimento geograficamente dispersas;
- Gerência de Processo: o suporte a modelagem e execução do processo de software têm sua importância na gerência das atividades desenvolvidas pelas pessoas e ferramentas durante a construção de software. Dentro desta característica, encontra-se a necessidade de um formalismo de modelagem de processo e uma máquina de execução das definições de processo. Alguns ambientes também fornecem um módulo de simulação de processos de software, para que o modelo desenvolvido com a linguagem de modelagem possa ser validado e refinado antes de sua execução real, prevenindo, desta forma, problemas com cronograma e orçamento do projeto;
- Extensibilidade: permitir extensão do ambiente através da inclusão de novas ferramentas, sejam elas de apoio ao desenvolvimento de software ou com outras

funções. Esta característica implica a existência de um mecanismo de integração eficiente no ambiente;

- Integração entre todos os módulos: todos os níveis de integração devem estar disponíveis para viabilizar as características apontadas. As ferramentas do ambiente devem concordar sobre os tipos de dados, operações, métodos utilizados e o processo de desenvolvimento sendo seguido.

## 2.3. Visão Geral do Ambiente ImPProS

Como mostrado anteriormente, os ambientes de desenvolvimento de software centrados no processo são propostos com o intuito de apoiar a modelagem e a execução de processo de desenvolvimento de software. No entanto, em alguns casos, percebe-se ao longo da execução do processo a partir destes ambientes de desenvolvimento que sua implementação nem sempre perfaz a realidade das características da organização ou do projeto desenvolvido por esta. Isto se deve ao fato de que os responsáveis pela definição do processo não dispõem de um guia contendo as suas reais necessidades de execução e estes, por si, indiquem as melhores práticas a serem instanciadas a partir de um processo padrão.

Assim, para ajudar uma organização na implementação progressiva de um processo de software, é útil fornecer apoio automatizado por meio de um ambiente capaz de suportar as fases que a literatura especializada propõe como necessárias. O termo "progressiva" decorre do fato de que a implementação do processo é aperfeiçoado com as experiências aprendidas na sua definição, simulação, execução e avaliação.

O ambiente ImPProS (Implementação Progressiva de Processo de Software), proposto por Oliveira em [OLIVEIRA05], em seu projeto de Tese de Doutorado no Cin/UFPE, está sendo concebido com o objetivo principal de apoiar a implementação de

um processo de software em uma organização. Dentro deste contexto podem ser caracterizados como seus objetivos específicos:

- Especificar um meta-modelo de processo de software a fim de definir uma terminologia única entre os vários modelos de qualidade de processo de software existentes, para uso do ambiente em seus serviços providos.
- Apoiar a definição de um processo de software para organização;
- Permitir a modelagem e instanciação deste processo;
- Permitir a simulação do processo a partir das características instanciadas para um projeto específico;
- Dar apoio à execução do processo de software tomando como base uma máquina de inferência;
- Possibilitar a avaliação dos critérios do processo de software;
- Apoiar a melhoria contínua do processo de software e o reuso através da realimentação e coleta das experiências aprendidas.

Para alcançar estes objetivos o ambiente foi concebido para adotar a arquitetura apresentada na Figura 2.1.

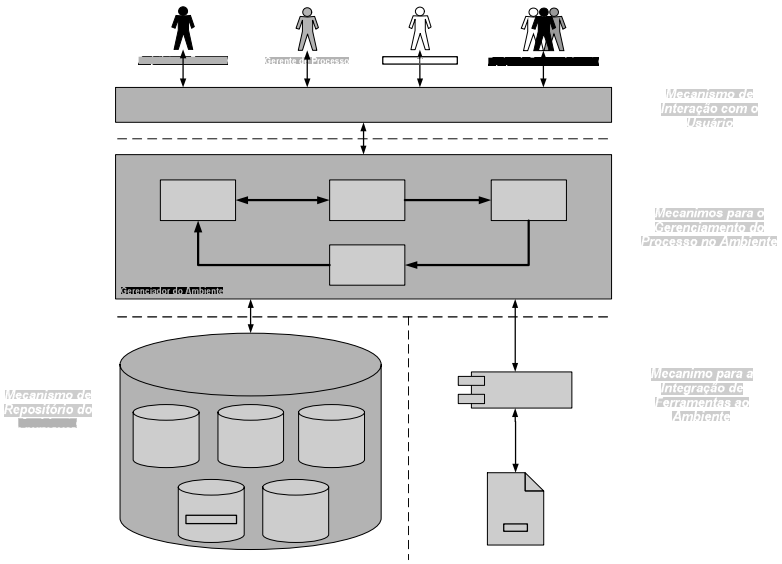

**Figura 2.1 Arquitetura do Ambiente ImPProS**

Pode-se notar que a arquitetura contempla quatro tipos de usuários para interação com o Ambiente:

- Projetista do Processo: responsável pela definição do processo, coleta e triagem de experiência acerca da execução de projetos. Este tipo de usuário interage com o ambiente recebendo orientações e identificando melhorias para processos existentes ou em concepção;
- Gerente de Processo: este tipo de usuário acompanha a simulação e a avaliação do processo a fim de prover conhecimentos formal e informal (lições aprendidas) para possibilitar o reuso e a melhoria contínua dos processos de software;
- Gerente de Projetos: este usuário atua nas fases de instanciação do processo para um projeto específico, acompanhando a execução do processo e a sua avaliação para posterior coleta de experiências;
- Equipe de Desenvolvimento: agrupa todos os perfis relacionados à execução de um projeto de software (Gerentes, Analistas, Engenheiros de Software, Arquitetos, etc.).

A definição de cada um dos componentes definidos na arquitetura do ambiente pode ser encontrada em [OLIVEIRA05]. No presente trabalho, será tratada a questão da representação diagramática do processo definido, sua representação em XML, para um intercâmbio de informações entre ferramentas, e sua especificação em uma ontologia, para que o processo possa ter seus componentes automatizados.

# 3. SPEM: Uma Linguagem de Modelagem de Processo de Software

No capítulo anterior foram explanados diversos conceitos a cerca de processos de software, assim como apresentado o ambiente ImPProS onde este trabalho está inserido. E em um dos pontos que esse trabalho quer alcançar é justamente a modelagem gráfica como linguagem universal para representação dos processos de software descritos no capítulo 2.

Nesse contexto iremos utilizar o SPEM [SPEM05], como meta-modelo para representação gráfica de um processo de software. A escolha do SPEM, em face a outros modelos gráficos de representação de processos, se dá pela importância que o mesmo vem tomando no mercado da engenharia de software atual. É um padrão definido pela OMG, organização conhecida por outros padrões como o UML, com o apoio das maiores industrias que trabalham com engenharia de software do mundo, como pode ser visto na própria definição do [SPEM05]

## 3.1. Visão Geral

O Software Process Engineering Metamodel (SPEM) [SPEM05] é um meta-modelo que pode ser usado para descrever um processo concreto de desenvolvimento de software ou uma família de processos relacionados. SPEM adota uma abordagem orientada a objetos para modelar processos e usa UML como notação. A Figura 3.1 descreve a arquitetura de quatro níveis de modelagem definida pela OMG e respeitada por SPEM.

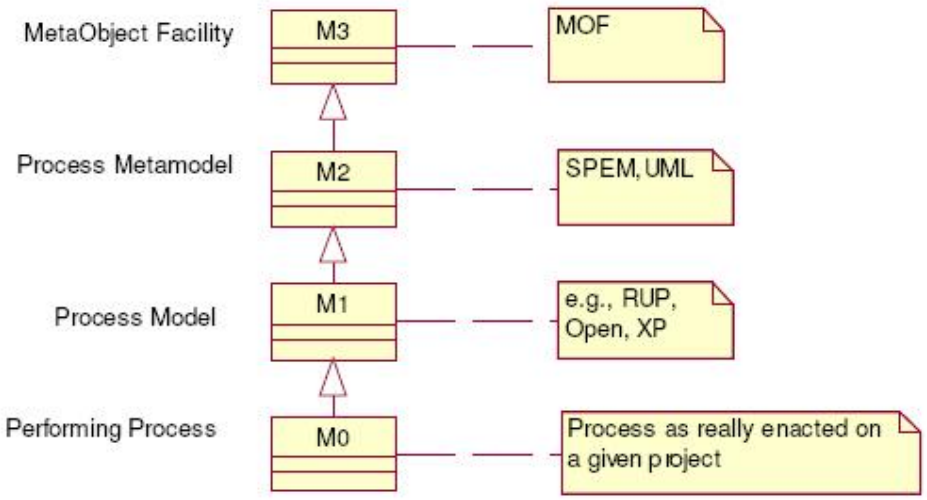


**Figura 3.1 Níveis de modelagem em [SPEM05]**

O processo real de produção, conforme instanciado para um projeto específico, está no nível M0. No nível M1 encontra-se a definição do processo que foi instanciado em M0 como, por exemplo, RUP, OPEN ou XP. O foco da definição de SPEM está em criar um meta-modelo que se encontra no nível M2 que possui estereótipos para a modelagem dos processos que estão no nível M1. A especificação de SPEM está estruturada como um perfil UML (UML profile, que é uma extensão do UML para acustomizá-la paraum domínio em particular, como,por exemplo, a modelagem de processos no caso do SPEM) e também através de um meta-modelo baseado no MOF (tecnologia adotada pela OMG para definir metadados) que se encontra no nível M3. Essa abordagem facilita a utilização e construção de ferramentas para UML e ferramentas baseadas em MOF.

O objetivo do padrão SPEM é abranger uma vasta gama de processos de desenvolvimento de software já existentes ao invés de excluí-los devido ao excesso de elementos e restrições que poderiam existir na sua definição. SPEM permite que seus usuários (modeladores de processo) usem a UML, e define estereótipos que podem ser usados nos modelos produzidos.

O meta-modelo SPEM é construído pela extensão de um subconjunto do metamodelo da UML 1.4. Esse subconjunto de UML é denominado na definição de SPEM como “SPEM_Foundation” . Além disso, o modelo SPEM possui o pacote “SPEM Extensions” que adiciona as construções e semânticas requeridas para a engenharia de processos de software. A Figura 3.2 abaixo mostra esses dois pacotes.

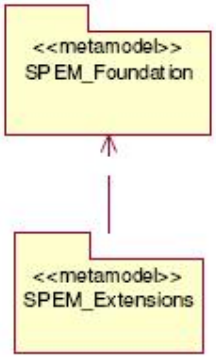


**Figura 3.2 Estrutura de pacotes em [SPEM05]**

Um maior detalhamento do SPEM pode ser obtido diretamente do site da OMG na internet. Os principais elementos definidos no SPEM será apresentado nas próximas seções desse trabalho juntamente com a descrição de cada um e sua ligação no ImPProS.

## 3.2. Ferramenta de Automação do SPEM

Para fazer uso da notação proposta pelo SPEM, torna-se necessária o uso de uma ferramenta que automatize o seu serviço e valide o processo projetado, segundo as regras semânticas propostas pelo SPEM.

## 3.3. Critérios de Seleção

Dessa forma, pesquisou-se algumas iniciativas comerciais, como visto o resultado de sua análise na Tabela 3.1, que possibilitassem esta automação e que atendessem aos requisitos solicitados pelo ambiente ImPProS, a saber:

- A ferramenta deve ser free e open-source, já que a estrutura do ambiente está sendo contemplada sobre esta vertente a fim de alavancar o uso de processos de software nas empresas de pequeno e médio porte, e propiciar a integração de tecnologias existentes no mercado com o caráter de automatizar o ciclo de vida da implementação de um processo de software;
- A ferramenta deve além de automatizar o uso da notação do SPEM, validar semanticamente os conceitos providos pelo meta-modelo;
- A ferramenta deve prover na sua estrutura a leitura e/ou geração de arquivos XML a fim de possibilitar o intercâmbio dos componentes definidos ao processo de software;
- A ferramenta deve estar bem documentada na sua arquitetura e codificação, para um melhor entendimento do que será adaptado na mesma a fim da melhor aderência da uso da mesma a partir do ambiente ImPProS.

**Tabela 3.1 Análise de Ferramentas de Modelagem do SPEM**

| Ferramenta | Fornecedor | Características requeridas pelo ImPProS | | | |
|---|---|---|---|---|---|
| | | Free e Open-source | Automação e Validação do SPEM | Leitura e/ou Geração de XML | Documentação |
| APES | IUP ISI – Universidade de Toulouse na França | Sim | Parcialmente (Não valida e não contempla todas as características do processo) | Sim | Sim |
| APES2 | IUP ISI – Universidade de Toulouse na França | Sim | Sim | Sim | Sim |
| Estêncil do Visio | Microsoft | Não | Não | Não | Não |
| RUP Builder | IBM/Rational | Não | Sim (Adaptação do RUP) | Sim | Sim |

Durante o processo de busca e seleção pela ferramenta que precisávamos no ImPProS vale citar que o APES2 se destacou por ser uma ferramenta que é livre, qualquer pessoa pode ver, estudar e expandir seu código, e que foi desenvolvida de forma bem estruturada e tendo uma vasta documentação do processo por meio de documentos, como documento de visão, arquitetura, casos de uso. Embora esses documentos estejam originalmente disponibilizados em francês, optamos por traduzir a parte desses documentos que nos interessa, o que pode ser visto no Apêndice A

## 3.4. Ferramenta APES2

A ferramenta APES2 surgiu em 2003 como fruto de um projeto anual feito pelos estudantes do IUP ISI na universidade de Toulouse e atualmente mantido pelo grupo open-source IPSquad. Em 2004 em um novo ciclo de desenvolvimento na mesma universidade onde o objetivo era expandir o trabalho feito no ano anterior e adicionar novas funcionalidades, destacando o módulo de validação de processo, surgindo assim o APES2.

O APES2 segue uma arquitetura em camadas composta de uma camada de apresentação denominada **Interface** que é a interface gráfica que o usuário vê quando está interagindo com a aplicação. Uma camada denominada **Aplicação** onde todas as regras de negócios foram colocadas. Uma camada de **Domínio** onde estão classes que modelam o domínio onde o APES está inserido, como as notações do SPEM estão representadas nessa camada. E por último uma camada de **Infra-estrutura** onde estão componentes feitos por terceiro que as demais camada do APES2 fazem uso.

A tradução na integra do documento de arquitetura que se encontra originalmente em Francês na documentação do ferramenta pode ser encontrada no Apêndice A.

# 4. Adaptação de uma Ferramenta de Modelagem usando o SPEM em um Ambiente de Implementação de Processo de Software

Este capítulo mostra o estudo que foi feito da ferramenta selecionada de modelagem SPEM, comparando o resultado obtido com as necessidades que o processo ImPProS.

## 4.1. Necessidade da Adaptação

Da comparação das necessidades do ImPProS e dos recursos oferecidos pelo *APES2* foi levantado uma tabela de mapeamento de cada conceito constante no modelo de processo usado pelo ImPProS com seus equivalentes no SPEM e as facilidades já oferecidas pela versão atual do *APES2*. O resultado pode ser visto abaixo na Tabela 4.1 Mapeamento ImPProS x SPEM x APES2.

**Tabela 4.1 Mapeamento ImPProS x SPEM x APES2**

| ImPProS | SPEM | APES2 |
|---|---|---|
| Processo | Process | SProcess |
| | ProcessComponent | ProcessComponent |
| Modelo de Ciclo de Vida | LifeCycle | N/A |
| | Iteration | |
| Combinação | Phase | N/A |
| | ProcessPackage | SPackage |
| Atividade | WorkDefinition | WorkDefinition |
| | Discipline | N/A |
| | Activity | Activity |
| | Step | N/A |

| ImPProS | SPEM | APES2 |
|---|---|---|
| Artefato | WorkProduct<br>Document<br>UMLModel | WorkProduct |
| Procedimento | Guidance<br>Guideline<br>Technique<br>UMLProfile<br>ToolMentor<br>CheckList<br>Template | Guidance |
| Recurso | ProcessPerformer<br>ProcessRole | ProcessPerformer<br>ProcessRole |
| Padrão de Atividades | Step | N/A |
| Paradigma de Desenvolvimento | | N/A |
| Tecnologia de Desenvolvimento | | N/A |
| Restrições | ExternalDescription<br>Goal<br>Precondition<br>ActivityParameter<br>trace<br>refersTo<br>categorizes<br>precedes<br>impacts<br>import<br>governs<br>assist<br>perform | ExternalDescription |

Partindo desse mapeamento foram identificados alguns pontos onde o *APES2* precisa ser adaptado para que satisfaça o ImPProS, são eles:

### *III.1.2. Extensão do Meta-Modelo SPEM*

Notamos pela Tabela 4.1 Mapeamento ImPProS x SPEM x APES2 que o modelo de ciclo de vida definido pelo SPEM não é contemplado pela versão atual do APES. Isto demanda uma reestruturação no pacote `apes.model.spem` e seus sub-pacotes com a inserção de duas novas classes para representar os conceitos de **Phase** e **Lifecycle**.

Outra particularidade da implementação do *APES* é que a granularidade da definição de metodologias só chega ao nível de atividades, enquanto o modelo proposto na ImPProS precisa descer no nível de **Steps** associado às atividades.

Para contemplar estes conceitos do SPEM que não estão contemplados na ferramenta, propomos a seguinte remodelagem no relacionamento encontrado no documento de arquitetura do APES (ver detalhes em Apêndice A:), conforme visto na Figura 4.1. Assim, as seguintes classes devem ser adicionadas:

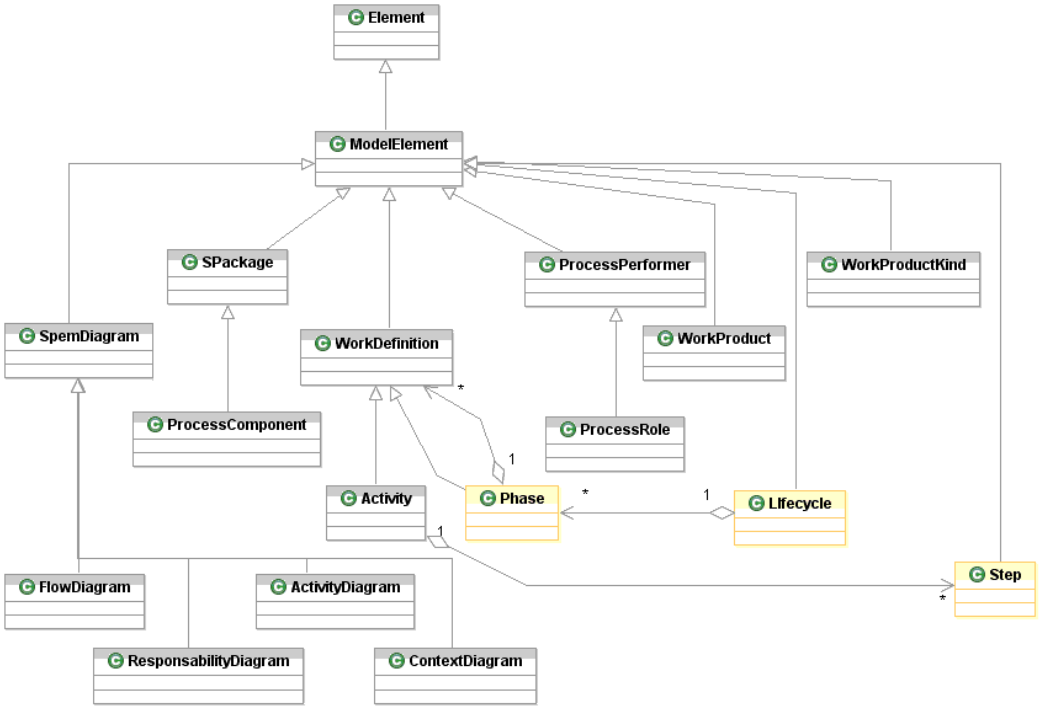


**Figura 4.1 Hierarquia das classes do pacote apes.model.spem**

**Lifecycle**

Classe que representa a seqüência de fases para compor o ciclo de vida do processo.

**Phase**

Classe que estende WorkDefinition porém contemplando uma ou mais pré-condições para seu início e um ou mais objetivos para seu término. No contexto desta extensão, a Phase caracteriza a definição das combinações de WorkDefinition (descrição de macro-atividades) que formam uma iteração do modelo de ciclo de vida.

**Step**

Descrição textual de passos que devem ser seguidos para execução correta de uma atividade.

Notem que mudanças arquiteturais não foram demandadas, apenas inserção de novas classes na estrutura.

Porém a simples inserção de novas classes no pacote `apes.model.spem` também irá afetar outras camadas:

- Na camada de Interface sofrerá mudanças estruturais no `ApesTree` para contemplar a visão de ciclo de vida dividida em fases, e no `ToolPalette` para adicionar os novos conceitos na paleta de instrumentos;
- Ainda na camada de Interface o pacote `apes.ui.tools` terá de ser revisado para contemplar os novos conceitos;
- Na camada de aplicação também será revisto o tratamento do modelo SPEM e o fluxo de percurso do mesmo com as novas classes;
- O pacote apes.adapters terá que entender os novos relacionamentos e como adapta-los a camada de Interface;

### *III.1.3. Demais extensões*

O ImPProS contém algumas definições que não fazem parte do meta-modelo SPEM e que também precisam ser contempladas pela ferramenta APES para que a mesma possa ser útil para o processo definido. Como pode ser visto na Tabela 4.1, são os atributos: Paradigma de desenvolvimento; e Tecnologia adotada.

Para contemplar estas características propomos que seja alterada a classe `Project` do pacote `apes`, pois ela representa o projeto como um todo e tais atributos são exatamente relativos ao projeto.

Devido a essa modificação a camada de Interface precisa sofrer alterações para contemplar a inserção de um diálogo para preenchimento de tais características associados ao projeto.

## 4.2. Instanciação do Processo de Software em XML

Na analise da arquitetura atual do *APES2* foi levantado que ele faz uso na camada de **infra-estrutura** da tecnologia *JSX* que faz a serialização e desserialização automática de classes *Java* em *XML*. Porém isto deixa o projeto amarrado a tecnologia dificultando: o entendimento do projeto *APES2* persistido em arquivo; a importação dos dados gerados na ferramenta para outras ferramentas que entendem o *SPEM*; a tradução do projeto em *XML* para outros formatos com as ontologias necessárias para o processo proposto (ver 4.3).

Portanto com a finalidade de resolver estas limitações levantadas e outras que não foram analisadas, mas que podem surgir com a definição da persistência de dados a uma tecnologia, proponho a modelagem de um padrão de estrutura *XML* baseado na definição da *OMG* para o *SPEM* e da especificação de um subsistema que use esta definição para persistir os projetos do *APES2*, uma vez que estes são definidos por uma representação diagramática da notação do SPEM, em arquivos *XML* substituindo a atual *JSX*, que define o modelo gerado pelo *APES2* como uma estrutura de processo fortemente "amarrada" ao contexto arquitetural proposto pela ferramenta e não baseada na estrutura de em um processo de software comum na literatura especializada.

Após um estudo levantando os dados que devem ser persistidos chegamos a um *DTD* (Data Type Definition) ao qual a ferramenta deve seguir para armazenar um projeto de definição de metodologia, veja esta definição no Tabela 4.2.

**Tabela 4.2 Definição do DTD para validação do XML proposto.**

```
<!--                                                                -->
<!--                                                                -->
<!-- process e o elemento raiz                                      -->
<!-- ______________________________________________________________ -->

<!ELEMENT process (workdefinition*, workproduct*, lifecycle)>
<!ATTLIST process
      version     CDATA #FIXED      "1.0"
      name  CDATA #REQUIRED>
```

```
<!ELEMENT workdefinition (activity*)>
<!ATTLIST workdefinition
      id          ID
      name  CDATA #REQUIRED>

<!ELEMENT activity (step*)>
<!ATTLIST activity
      id          ID
      name  CDATA #REQUIRED
      inputs      IDREF #IMPLIED
      outputs     IDREF #IMPLIED>

<!ELEMENT step CDATA>

<!ELEMENT lifecycle (phase+)>
<!ATTLIST lifecycle
      type  (type1|type2)     "type1">

<!ELEMENT phase (diagrams)>
<ATTLIST phase
      workdefinitions   IDREFS      #REQUIRED>

<!ELEMENT workproduct EMPTY>
<!ATTLIST workproduct
      id          ID          #REQUIRED
      name  CDATA #REQUIRED>

<!ELEMENT diagrams (activity.diagram?, responsability.diagram?)>

<!ELEMENT activity.diagram (activity.element*, transition*)>

<!ELEMENT responsability.diagram
(responsability.element*,responsability*)>

<!ELEMENT activity.element EMPTY>
<!ATTLIST activity.element
      id                      ID          #REQUIRED
      name              CDATA #REQUIRED
      type              (initialpoint|finalpoint|activity) "activity"
      activity          IDREF #IMPLIED>

<!ELEMENT transition EMPTY>
<!ATTLIST transition
      from  IDREF #REQUIRED
      to          IDREF #REQUIRED>

<!ELEMENT responsability.element EMPTY>
<!ATTLIST responsability.element
      id          ID    #REQUIRED
      type  (processrole|worproduct)
      element     IDREF #REQUIRED>


<!ELEMENT responsability EMPTY>
<!ATTLIST responsability
      role        IDREF #REQUIRED
      workproduct IDREF #REQUIRED>
```

## 4.3. Instanciação do Processo de Software em Ontologias

Por se tratar de um ambiente automatizado para definir, analisar e executar processos de desenvolvimento de software, o ImPProS precisa alguma forma simples, auto explicável e livre de dupla interpretação. Para isso uma linguagem lógica seria a ideal.

Pesquisando por trabalhos dessa magnitude encontramos [FALBO98] que define através de uma notação lógica um processo metodológico de desenvolvimento de software, onde a partir do mesmo, alguns trabalhos na mesma linha de estudo fazem uso de sua notação e podem ser detectados na literatura especializada [BERGER03], [MACHADO00] [OLIVEIRA99]. O trabalho seguinte foi analisar a definição dos axiomas da ontologia definida por Falbo e mapear as necessidades ao ImPProS, o que resultou na Tabela 4.3 abaixo.

**Tabela 4.3 Mapeamento ImPProS x Ontologias**

| ImPProS | Ontologias |
|---|---|
| Processo | $(\forall a,pr)\ (\ proc\text{-}composição(a,pr) \rightarrow processo(pr) \wedge atividade(a))$ |
| Modelo de Ciclo de Vida | $(\forall c,mcv)\ (mcv\text{-}composição(c,mcv,n) \rightarrow mciclovida(mcv,*) \wedge combinação(c,*) \wedge n \in N^{+})$ |
| Combinação | $(\forall a,c\ n)\ (comb\text{-}composição(a,c,n) \rightarrow macroatividade(a) \wedge combinação(c,*) \wedge n \in N^{+})$ |

| ImPProS | Ontologias |
|---|---|
| Atividade | $(\forall a)$ (atconstrução(a) $\rightarrow$ atividade(a))<br>$(\forall a)$ (atgerência(a) $\rightarrow$ atividade(a))<br>$(\forall a)$ (atavqualidade(a) $\rightarrow$ atividade(a))<br>$(\forall a_1, a_2)$ ( subatividade($a_1$, $a_2$) $\leftrightarrow$ superatividade($a_2$, $a_1$))<br>$(\forall a_1, a_2)$ ( subatividade($a_1$, $a_2$) $\rightarrow$ atividade($a_1$) $\wedge$ atividade($a_2$))<br>$(\forall a_1, a_2, a_3)$ (subatividade($a_1$, $a_2$) $\wedge$ subatividade($a_2$, $a_3$) $\rightarrow$ subatividade($a_1$, $a_3$))<br>$(\forall a_1, a_2)$ ( subatividade($a_1$, $a_2$) $\rightarrow \neg$ subatividade($a_2$, $a_1$))<br>$(\forall a)$ ( atividadeelementar(a) $\leftrightarrow \neg (\exists a_1)$ (subatividade($a_1$, a)))<br>$(\forall a)$ ( macroatividade(a) $\leftrightarrow (\exists a_1)$ (subatividade($a_1$, a)))<br>$(\forall a_1, a_2)$ ( preatividade($a_1$, $a_2$) $\leftrightarrow$ posatividade($a_2$, $a_1$))<br>$(\forall a_1, a_2, a_3)$( preatividade($a_1$, $a_2$) $\wedge$ preatividade($a_2$, $a_3$) $\rightarrow$ preatividade($a_1$, $a_3$))<br>$(\forall a_1, a_2)$ ( preatividade($a_1$, $a_2$) $\rightarrow \neg$ preatividade($a_2$, $a_1$))<br>$(\forall a_1, a_2)$ ( preatividade($a_1$, $a_2$) $\rightarrow$ atividade($a_1$) $\wedge$ atividade($a_2$))<br>$(\forall a)$ (atividade (a) $\rightarrow$ $(\exists s)$ (produto(s,a)) |
| Artefato | $(\forall a, s)$ ( insumo(s, a) $\rightarrow$ artefato(s,*) $\wedge$ atividade(a))<br>$(\forall a, s)$ ( produto(s, a) $\rightarrow$ artefato(s,*) $\wedge$ atividade(a))<br>$(\forall a_1, a_2)$ (preatividade($a_1$, $a_2$) $\leftrightarrow$ $(\exists s)$ (insumo(s,$a_2$) $\wedge$ produto(s,$a_1$))<br>$(\forall a, a_1, \ldots, a_n, s)$ (subatividade($a_1$, a) $\wedge \ldots \wedge$ subatividade($a_n$, a) $\wedge$ insumo(s,$a_i$) $\wedge$ (($\neg \exists a_k$) produto(s, $a_k$)) $\rightarrow$ insumo(s,a))<br>$(\forall a, a_1, \ldots, a_n, s)$ (subatividade($a_1$, a) $\wedge \ldots \wedge$ subatividade($a_n$, a) $\wedge$ produto(s,$a_i$) $\wedge$ (($\exists b$) (insumo(s,b) $\wedge b \notin \{a_1, a_2, \ldots, a_n\}$)) $\rightarrow$ produto(s,a)) |

| ImPProS | Ontologias |
|---|---|
| Procedimento | $(\forall p,a)\ (\text{possíveladoção}(p,a) \rightarrow \text{procedimento}(p) \wedge \text{atividade}(a))$<br>$(\forall a, r)\ (\text{possíveladoção}(r,a) \wedge \text{roteiro}(r) \rightarrow (\exists\ s)\ (\text{produto}(s,a) \wedge \text{artefato}(s,\text{Documento}))$<br>$(\forall a, p)\ ((\text{metconstrução}(p) \vee \text{tecconstrução}(p)) \wedge \text{possíveladoção}(p,a) \rightarrow \text{atconstrução}(a))$<br>$(\forall a, p)\ ((\text{metgerência}(p) \vee \text{tecgerência}(p)) \wedge \text{possíveladoção}(p,a) \rightarrow \text{atgerência}(a))$<br>$(\forall a, m)\ (\text{metavqualidade}(p) \vee \text{tecavqualidade}(p)) \wedge \text{possíveladoção}(p,a) \rightarrow \text{atavqualidade}(a))$<br>$(\forall a, m)\ (\text{metódo}(m) \wedge \text{possíveladoção}(m,a) \rightarrow \text{macroatividade}(a))$<br>$(\forall a, t)\ (\text{técnica}(t) \wedge \text{possíveladoção}(t,a) \rightarrow \text{atividadeelementar}(a))$ |
| Recurso | $(\forall a, r)\ (\text{uso}(r,a) \rightarrow \text{recurso}(r) \wedge \text{atividade}(a))$<br>$(\forall a_1, a, r)\ ((\text{uso}(r,a_1) \wedge \text{subatividade}(a_1,a)) \rightarrow \text{uso}(r,a))$<br>$(\forall f,p)\ (\text{possívelautomatização}(f,p) \rightarrow \text{ferramenta}(f) \wedge \text{procedimento}(p))$<br>$(\forall f, p)\ ((\text{metconstrução}(p) \vee \text{tecconstrução}(p)) \wedge \text{possívelautomatização}(f,p) \rightarrow (\text{ferconstrução}(f) \vee \text{ferpropgeral}(f)))$<br>$(\forall f, p)\ ((\text{metgerência}(p) \vee \text{tecgerência}(p)) \wedge \text{possívelautomatização}(f,p) \rightarrow (\text{fergerência}(f) \vee \text{ferpropgeral}(f)))$<br>$(\forall f, p)\ ((\text{metavqualidade}(p) \vee \text{tecavqualidade}(p)) \wedge \text{possívelautomatização}(f,p) \rightarrow (\text{feravqualidade}(f) \vee \text{ferpropgeral}(f)))$<br>$(\forall a, f)\ (\text{uso}(f,a) \rightarrow (\exists p)\ (\text{possíveladoção}(p,a) \wedge \text{possívelautomatização}(f,p)))$ |
| Padrão de Atividades | $(\forall a,m,pa)\ (\text{descrição}(m,a,pa) \rightarrow \text{possíveladoção}(m,a) \wedge$ |

| ImPProS | Ontologias |
|---|---|
| | *padrãoat(pa ))*<br>*(∀ a,a1,pa) (padrão-composição(a1,pa) ∧ descrição(m,a,pa) → (a1 ≠ a*<br>*∧ ¬ superatividade(a1,a))* |
| Paradigma de Desenvolvimento | *(∀ p,pd) (proced-conformidade(p,pd) → procedimento(p) ∧*<br>*paradigma(pd))*<br>*(∀ pr,pd) (processo-conformidade(pr,pd) → processo(pr) ∧*<br>*paradigma(pd))*<br>*(∀ a, p, pr,pd) (possíveladoção(p,a) ∧ proc-composição(a,pr) ∧*<br>*processo-conformidade(pr,pd) → proced-conformidade(p,pd)* |
| Tecnologia de Desenvolvimento | *(∀ p,td) ( proced-adequação(p,td) → procedimento(p) ∧*<br>*tecnologia(td))*<br>*(∀ pr,td) ( processo-adequação(pr,td) → processo(pr) ∧*<br>*tecnologia(td))*<br>*(∀ a, p, pr,td) (possíveladoção(p,a) ∧ proc-composição(a,pr) ∧*<br>*processo-adequação(pr,td) → proced-adequação(p,td)* |
| Restrições | |

Durante o processo de análise e mapeamento foi comprovado que a definição existente encaixou-se muito bem (dentre o conjunto de axiomas definido por Falbo, detectamos um ajuste em apenas dos componentes do processo) às necessidades com a exceção da definição de macro-atividade, que para adequar-se a definição dos modelos de qualidade CMMI [CMMI02] e ISO (12207 [ISO95] e 15504 [ISO03]) precisou sofrer uma pequena modificação. Para Falbo uma macro-atividade é uma atividade que não tem super-atividades, enquanto para os modelos de qualidade uma macro-atividade é uma atividade que tem sub-atividades, por essa razão a regra ontologica: *(∀ a) ( macroatividade(a) ↔ ¬ (∃ a1 ) (superatividade(a1 , a)))* foi modificada para: *(∀ a) ( macroatividade(a) ↔ (∃ a1 ) (subatividade(a1 , a)))*.

Note que não temos nenhuma ontologia ligada diretamente a restrições, porém é importante notar que restrições já estão presentes indiretamente em outras ontologias, quando por exemplo restringimos o que é uma macro-atividade.

# 5. Estudo de Caso

No capítulo anterior tratamos das modificações propostas nesse trabalho com a finalidade de suportar as necessidades do ambiente ImPProS, portanto neste faremos um estudo de caso para validar se o que está sendo proposto irá realmente ser válido na definição de um processo de software. Inicialmente será feita uma modelagem de uma disciplina do *framework* de processo do RUP, a partir da notação definida pelo *SPEM*, a fim de visualizarmos a sua representação diagramática. Posteriormente, será simulada a definição do arquivo XML que represente a estrutura da disciplina do processo modelada. Por fim, o mapeamento para os axiomas da ontologia de processo de software serão usados para compor uma representação do processo pronto para ser executado.

Para isso usaremos a disciplina de *Environment* uma das mais conhecidas e utilizadas metodologias atuais, o *RUP* [RUP05].

## 5.1. Modelagem Gráfica

Assim como está definido no *RUP*, ele segue um ciclo de vida iterativo e incremental, com quatro fases distintas: Concepção, Elaboração, Construção e Transição; e cada uma delas pode ser dividida em uma ou mais iterações. Então vamos analisar a Figura 4.1 representando a disciplina *Environment*.

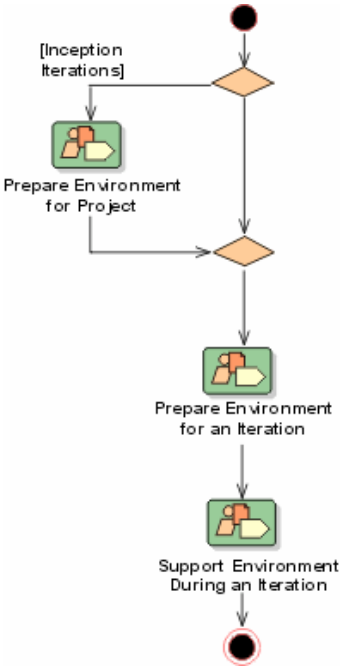


**Figura 5.1 Workflow da disciplina *Environment* (original do RUP)**

Na Figura 5.1 vemos três macro-atividades sendo que a primeira *Prepare Environment for Project* fazendo parte da fase Concepção, a segunda *Prepare Environment for Iteration* no início de cada iteração, e a terceira *Support Environment During na Iteration* durante o decorrer de todas as iterações do projeto.

As Figura 5.2, Figura 5.3 e Figura 5.4 permitem uma visualização do detalhamento de cada uma dessas macro-atividades.

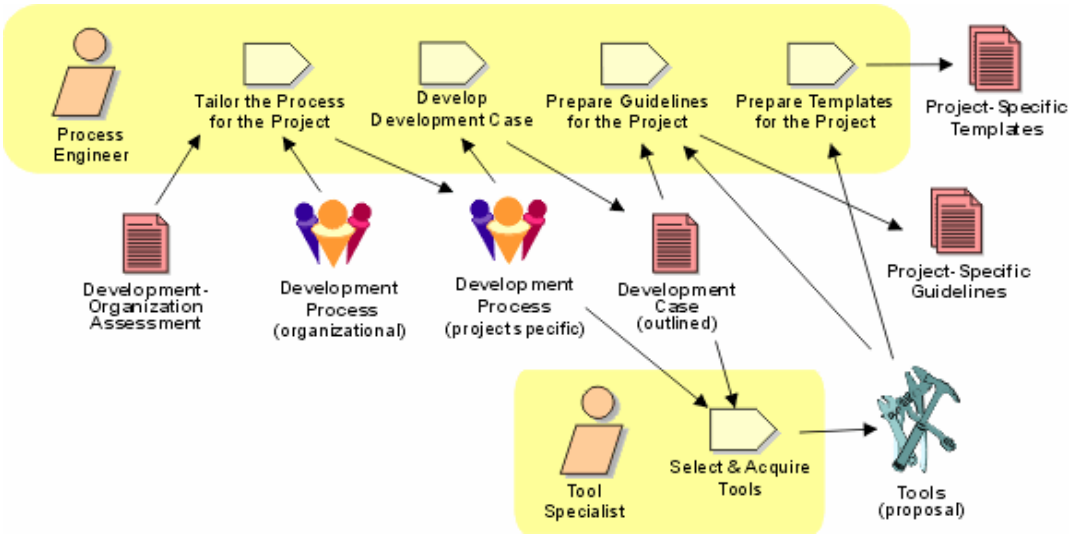

**Figura 5.2 Macro-atividade *Prepare Environment for Project* (original do RUP)**

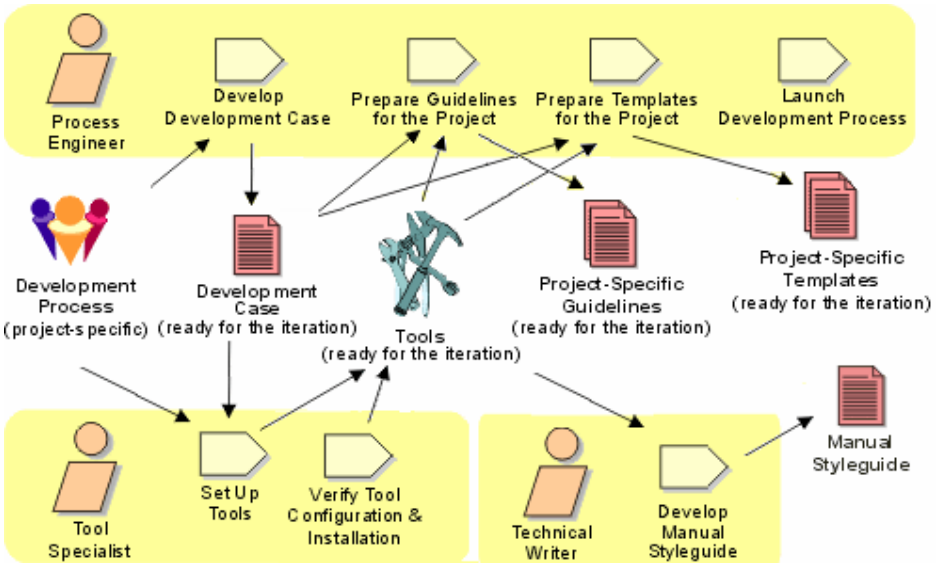


**Figura 5.3 Macro-atividade *Prepare Environment for an Iteration* (original do RUP)**

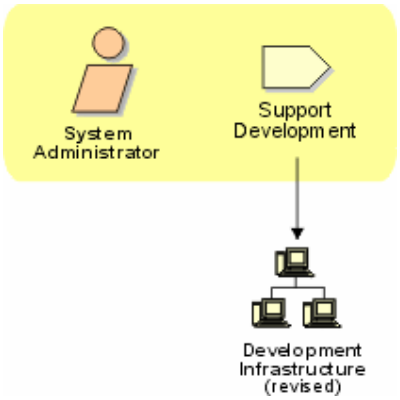


**Figura 5.4 Macro-atividade *Support Environmento During an Iteration* (original do RUP)**

A seguir, cada uma das atividades representadas nas Figura 5.2, Figura 5.3 e Figura 5.4 apresenta um detalhamento de sua composição, extraído do texto original do RUP.

### Activity: Tailor the Process for the Project

**Purpose**

- To right-size the software development process according to the specific needs of the project
- To provide a relevant and accessible process description for the members of the

project

**Role:** Process Engineer

**Frequency:** The bulk of the work takes part at the onset of the project. May repeat as necessary in any iteration

**Steps**

- Analyze the Project
- Define the Scope of the Process
- Extend the Process Framework (optional)
- Configure the Process
- Prepare the Process for the Project
- Introduce the Process to the Project Members
- Maintain the Process

| **Input Artifacts:** | **Resulting Artifacts:** |
|---|---|
| • Development-Organization Assessment<br>• Development Process | • Development Process |

**Activity: Develop Development Case**

**Purpose**

- To develop a development case that describes the software-development process for a project (or projects).
- To relate the development case to the organization-specific process product.

**Role:** Process Engineer

**Frequency:** Once every iteration in the software project.

**Steps**

- Decide How to Perform Each Discipline

- Tailor Artifacts per Discipline
- Modify Disciplines and Activities
- Choose Lifecycle Model
- Identify Stakeholders
- Map Roles to Job Positions
- Describe Sample Iterations
- Document the Development Case
- Maintain the Development Case

| **Input Artifacts:** | **Resulting Artifacts:** |
|---|---|
| • Development Case<br>• Development Process<br>• Software Development Plan<br>• Target-Organization Assessment | • Development Case |

**Activity: Launch Development Process**

**Purpose**

- To make the project members use the development process tailored for the project, together with the supporting tools.

**Role:** Process Engineer

**Frequency:** Once every iteration in the software project.

**Steps**

- Make the changes public
- Educate project members
- Collect feedback

| **Input Artifacts:** | **Resulting Artifacts:** |
|---|---|
| • Development Case<br>• Development Process<br>• Project Specific Guidelines<br>• Project-Specific Templates<br>• Requirements Management Plan | • Change Request |

| | |
|---|---|
| • Tools | |

**Activity: Prepare Guidelines for the Project**

| |
|---|
| **Purpose**<br><br>• To harvest existing or develop new guidelines for use by the project.<br>• To make the existing guidelines accessible for the project members when needed. |
| **Role:** Process Engineer |
| **Frequency:** The initial collection of guidelines is done during the inception phase, as part of tailoring the process for the project. The activity is performed again at the beginning of each iteration if necessary. |
| **Steps**<br><br>• Identify the Project's Needs for Guidelines<br>• Prepare Guidelines for Project Use<br>• Maintain Guidelines |

| **Input Artifacts:** | **Resulting Artifacts:** |
|---|---|
| • Development Case<br>• Project Specific Guidelines<br>• Tools | • Project Specific Guidelines |

**Activity: Prepare Templates for the Project**

| |
|---|
| **Purpose**<br><br>• To harvest existing or develop new templates for use by the project.<br>• To prepare the templates for project use by partially instantiating them with project-specific information. |

- To make the existing templates accessible to the project members when needed.

**Role:** Process Engineer

**Frequency:** The initial collection of templates is done during the inception phase, as part of tailoring the process for the project. The activity is performed again whenever there is a need for a new or changed template

**Steps**

- Identify Templates for the Project
- Prepare Templates for Project Use
- Maintain Templates

| **Input Artifacts:** | **Resulting Artifacts:** |
|---|---|
| • Development Case<br>• Project-Specific Templates<br>• Tools | • Project-Specific Templates |

**Activity: Select and Acquire Tools**

**Purpose**

- To select tools that fit the needs of the project.
- To acquire the tools for the project.
- Sometimes special tools have to be developed internally to support special needs, provide additional automation of tedious or error-prone tasks, and provide better integration between tools.

**Role:** Tool Specialist

**Frequency:** Most of the tools are acquired early in the project.

**Steps**

- Identify needs and constraints

- Collect information about tools
- Compare tools
- Select tools
- Acquire tools

| **Input Artifacts:** | **Resulting Artifacts:** |
| --- | --- |
| • Development Case<br>• Development-Organization Assessment<br>• Tools | • Tools |

**Activity: Set Up Tools**

**Purpose**

- To customize the tools.
- To install tools on servers and for users.

**Role:** Tool Specialist

**Frequency:** Most tool installation and customization is done early in the project's lifecycle.

**Steps**

- Install the Tool on the Server
- Customize the Tool (on the Server)
- Set up Multisite Support
- Integrate with Other Tools
- Install and Customize Tools on Clients

| **Input Artifacts:** | **Resulting Artifacts:** |
| --- | --- |
| • Development Case<br>• Project Specific Guidelines<br>• Tools | • Project Specific Guidelines<br>• Tools |

**Activity: Verify Tool Configuration and Installation**

**Purpose**

- To verify that the tools can be used to develop the system.

**Role:** Tool Specialist

**Frequency:** When new tools are introduced or changes have been made to existing tools.

**Steps:**

- Verify the environment
- Verify the tools
- Verify data
- Run the tools

| **Input Artifacts:** | **Resulting Artifacts:** |
|---|---|
| • Development Case<br>• Development Infrastructure<br>• Project Specific Guidelines<br>• Tools | |

**Activity: Develop Manual Styleguide**

**Purpose**

- To develop a styleguide for the end-user support material.

**Role:** Technical Writer

**Frequency:** As required, typically once per phase starting as early as Inception, and revisited as required.

| Input Artifacts: | Resulting Artifacts: |
|---|---|
| • Development Case<br>• Manual Styleguide<br>• Software Development Plan<br>• Tools | • Manual Styleguide |

**Activity: Support Development**

| Purpose |
|---|
| • To support the development with hardware and software. |
| **Role:** System Administrator |
| **Frequency:** On-going. |

| Input Artifacts: | Resulting Artifacts: |
|---|---|
| • Development Infrastructure<br>• Test Environment Configuration<br>• Tools | • Development Infrastructure |

## 5.2. Instância em XML

Nesta seção veremos como ficará representado no XML proposto a modelagem gráfica proposta na seção 5.1

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE process SYSTEM "apes2.dtd" >
<process name="Environment" version="1.0">
  <workdefinition id="PrepareEnvProject">
    <activity id="TailorProcessProject">
      <step>Analyze the Project</step>
      <step>Define the Scope of the Process</step>
      <step>Extend the Process Framework (optional)</step>
      <step>Configure the Process</step>
      <step>Prepare the Process for the Project</step>
      <step>Introduce the Process to the Project Members</step>
      <step>Maintain the Process</step>
```

```
    </activity>
    <activity id="DevelopDevCase">
      <step>Decide How to Perform Each Discipline</step>
      <step>Tailor Artifacts per Discipline</step>
      <step>Modify Disciplines and Activities</step>
      <step>Choose Lifecycle Model</step>
      <step>Identify Stakeholders</step>
      <step>Map Roles to Job Positions</step>
      <step>Describe Sample Iterations</step>
      <step>Document the Development Case</step>
      <step>Maintain the Development Case</step>
    </activity>
    <activity id="PrepGuidelinesProject">
      <step>Identify the Project's Needs for Guidelines</step>
      <step>Prepare Guidelines for Project Use</step>
      <step>Maintain Guidelines</step>
    </activity>
    <activity id="PrepTemplatesProject">
      <step>Identify Templates for the Project</step>
      <step>Prepare Templates for Project Use</step>
      <step>Maintain Templates</step>
    </activity>
    <activity id="SelectAcquireTools">
      <step>Identify needs and constraints</step>
      <step>Collect information about tools</step>
      <step>Compare tools</step>
      <step>Select tools</step>
      <step>Acquire tools</step>
    </activity>
  </workdefinition>
  <workdefinition id="PrepareEnvIteration">
    <activity id="DevelopDevCase">
      <step>Decide How to Perform Each Discipline</step>
      <step>Tailor Artifacts per Discipline</step>
      <step>Modify Disciplines and Activities</step>
      <step>Choose Lifecycle Model</step>
      <step>Identify Stakeholders</step>
      <step>Map Roles to Job Positions</step>
      <step>Describe Sample Iterations</step>
      <step>Document the Development Case</step>
      <step>Maintain the Development Case</step>
    </activity>
    <activity id="PrepGuidelinesProject">
      <step>Identify the Project's Needs for Guidelines</step>
      <step>Prepare Guidelines for Project Use</step>
      <step>Maintain Guidelines</step>
    </activity>
    <activity id="PrepTemplatesProject">
      <step>Identify Templates for the Project</step>
      <step>Prepare Templates for Project Use</step>
      <step>Maintain Templates</step>
    </activity>
    <activity id="LaunchDevProcess">
      <step>Make the changes public</step>
      <step>Educate project members</step>
      <step>Collect feedback</step>
    </activity>
    <activity id="SetUpTools">
      <step>Install the Tool on the Server</step>
      <step>Customize the Tool (on the Server)</step>
      <step>Set up Multisite Support</step>
```

```
      <step>Integrate with Other Tools</step>
      <step>Install and Customize Tools on Clients</step>
    </activity>
    <activity id="VerifyToolConfInst">
      <step>Verify the environment</step>
      <step>Verify the tools</step>
      <step>Verify data</step>
      <step>Run the tools</step>
    </activity>
    <activity id="DevelopManualStyleguide">
      <step></step>
    </activity>
  </workdefinition>
  <workdefinition id="SupportIteration">
    <activity id="SupportDevelopment">
      <step></step>
    </activity>
  </workdefinition>
  <workproduct id="DevelopmentCase" name="Development Case"/>
  <workproduct id="ProjectSpecificGuidelines" name="Project Specific
Guidelines"/>
  <workproduct id="ProjectSpecificTemplates" name="Project Specific
Templates"/>
  <workproduct id="ManualStyleguide" name="Manual Styleguide"/>
  <workproduct id="DevelopmentProcess" name="Development Process"/>
  <workproduct id="Tools" name="Tools"/>
  <lifecycle name="Incremental" type="Iter">
    <phase name="Concepcao"
workdefinitions="PrepareEnvProject,PrepareEnvIteration,SupportIteration"/>
    <phase name="Elaboracao"
workdefinitions="PrepareEnvIteration,SupportIteration"/>
    <phase name="Construcao"
workdefinitions="PrepareEnvIteration,SupportIteration"/>
    <phase name="Transicao"
workdefinitions="PrepareEnvIteration,SupportIteration"/>
  </lifecycle>
  <diagrams>
    <activity.diagram/>
    <responsability.diagram/>
    <flow.diagram/>
    <context.diagram/>
  </diagrams>
</process>
```

## 5.3. Instância em Ontologia

Veremos nessa seção como dar-se-á o estudo de caso representado pelos axiomas da ontologia proposta por [FALBO98] mapeadas nesse trabalho para uso no ImPProS, a partir da disciplina *Environment* do RUP, detalhada da seção 5.1.

```
    paradigma(OO)
    mciclovida(Incremental,Iter)
    mcv-composicao(Concepção,Incremental,1)
    mcv-composicao(Elaboração,Incremental,2)
```

```
mcv-composicao(Construção,Incremental,2)
mcv-composicao(Transição,Incremental,2)
comb-composição(PrepareEnvProject,Concepção,1)
comb-composição(PrepareEnvIteration,Concepção,1)
comb-composição(SupportIteration,Concepção,2)
comb-composição(PrepareEnvIteration,Elaboração,1)
comb-composição(SupportIteration,Elaboração,2)
comb-composição(PrepareEnvIteration,Construção,1)
comb-composição(SupportIteration,Construção,2)
comb-composição(PrepareEnvIteration,Transição,1)
comb-composição(SupportIteration,Transição,2)
macroatividade(PrepareEnvProject)
macroatividade(PrepareEnvIteration)
macroatividade(SupportIteration)
macroatividade(TailorProcessProject)
macroatividade(DevelopDevCase)
macroatividade(LaunchDevProcess)
macroatividade(PrepGuidelinesProject)
macroatividade(PrepTemplatesProject)
macroatividade(SelectAcquireTools)
macroatividade(SetUpTools)
macroatividade(VerifyToolConfInst)
atividadeelementar(DevelopManualStyleguide)
atividadeelementar(SupportDevelopment)
atividadeelementar(AnalyzeProject)
atividadeelementar(DefineScopeProcess)
atividadeelementar(ExtendProcessFramework)
atividadeelementar(ConfigureProcess)
atividadeelementar(PrepareProcessForProject)
atividadeelementar(IntroduceProcessToProjectMembers)
atividadeelementar(MaintainProcess)
atividadeelementar(DecideHowToPerformEachDiscipline)
atividadeelementar(TailorArtifactsDiscipline)
atividadeelementar(ModifyDisciplinesActivities)
atividadeelementar(ChooseLifecycleModel)
atividadeelementar(IdentifyStakeholders)
atividadeelementar(MapRolesJobPositions)
atividadeelementar(DescribeSampleIterations)
atividadeelementar(DocumentDevelopmentCase)
atividadeelementar(MaintainDevelopmentCase)
atividadeelementar(MakeChangesPublic)
atividadeelementar(EducateProjectMembers)
atividadeelementar(CollectFeedback)
atividadeelementar(IdentifyProjectNeedsGuidelines)
atividadeelementar(PrepareGuidelinesProjectUse)
atividadeelementar(MaintainGuidelines)
atividadeelementar(IdentifyTemplatesProject)
atividadeelementar(PrepareTemplatesProjectUse)
atividadeelementar(MaintainTemplates)
atividadeelementar(IdentifyNeedsAndConstraints)
atividadeelementar(CollectInformationAboutTools)
atividadeelementar(CompareTools)
atividadeelementar(SelectTools)
atividadeelementar(AcquireTools)
atividadeelementar(InstallToolServer)
atividadeelementar(CustomizeToolServer)
atividadeelementar(SetUpMultisiteSupport)
atividadeelementar(IntegrateWithOtherTools)
atividadeelementar(InstallCustomizeToolsClients)
atividadeelementar(VerifyEnvironment)
atividadeelementar(VerifyTools)
```

```
atividadeelementar(VerifyData)
atividadeelementar(RunTools)
subatividade(TailorProcessProject,PrepareEnvProject)
subatividade(DevelopDevCase,PrepareEnvProject)
subatividade(PrepGuidelinesProject,PrepareEnvProject)
subatividade(PrepTemplatesProject,PrepareEnvProject)
subatividade(SelectAcquireTools,PrepareEnvProject)
subatividade(DevelopDevCase,PrepareEnvIteration)
subatividade(PrepGuidelinesProject,PrepareEnvIteration)
subatividade(PrepTemplatesProject,PrepareEnvIteration)
subatividade(LaunchDevProcess,PrepareEnvIteration)
subatividade(SetUpTools,PrepareEnvIteration)
subatividade(VerifyToolConfInst,PrepareEnvIteration)
subatividade(DevelopManualStyleguide,PrepareEnvIteration)
subatividade(SupportDevelopment,SupportIteration)
subatividade(AnalyzeProject,TailorProcessProject)
subatividade(DefineScopeProcess,TailorProcessProject)
subatividade(ExtendProcessFramework,TailorProcessProject)
subatividade(ConfigureProcess,TailorProcessProject)
subatividade(PrepareProcessForProject,TailorProcessProject)
subatividade(IntroduceProcessToProjectMembers,TailorProcessProject)
subatividade(MaintainProcess,TailorProcessProject)
subatividade(DecideHowToPerformEachDiscipline,DevelopDevCase)
subatividade(TailorArtifactsDiscipline,DevelopDevCase)
subatividade(ModifyDisciplinesActivities,DevelopDevCase)
subatividade(ChooseLifecycleModel,DevelopDevCase)
subatividade(IdentifyStakeholders,DevelopDevCase)
subatividade(MapRolesJobPositions,DevelopDevCase)
subatividade(DescribeSampleIterations,DevelopDevCase)
subatividade(DocumentDevelopmentCase,DevelopDevCase)
subatividade(MaintainDevelopmentCase,DevelopDevCase)
subatividade(IdentifyProjectNeedsGuidelines,PrepGuidelinesProject)
subatividade(PrepareGuidelinesProjectUse,PrepGuidelinesProject)
subatividade(MaintainGuidelines,PrepGuidelinesProject)
subatividade(IdentifyTemplatesProject,PrepTemplatesProject)
subatividade(PrepareTemplatesProjectUse,PrepTemplatesProject)
subatividade(MaintainTemplates,PrepTemplatesProject)
subatividade(IdentifyNeedsAndConstraints,SelectAcquireTools)
subatividade(CollectInformationAboutTools,SelectAcquireTools)
subatividade(CompareTools,SelectAcquireTools)
subatividade(SelectTools,SelectAcquireTools)
subatividade(AcquireTools,SelectAcquireTools)
subatividade(MakeChangesPublic,LaunchDevProcess)
subatividade(EducateProjectMembers,LaunchDevProcess)
subatividade(CollectFeedback,LaunchDevProcess)
subatividade(InstallToolServer,SetUpTools)
subatividade(CustomizeToolServer,SetUpTools)
subatividade(SetUpMultisiteSupport,SetUpTools)
subatividade(IntegrateWithOtherTools,SetUpTools)
subatividade(InstallCustomizeToolsClients,SetUpTools)
subatividade(VerifyEnvironment,VerifyToolConfInst)
subatividade(VerifyTools,VerifyToolConfInst)
subatividade(VerifyData,VerifyToolConfInst)
subatividade(RunTools,VerifyToolConfInst)
artefato(DevelopmentCase,Documento)
artefato(ProjectSpecificGuidelines,Documento)
artefato(ProjectSpecificTemplates,Documento)
artefato(ManualStyleguide,Documento)
procedimento(DevelopmentProcess)
ferramenta(Tools)
rechumano(ProcessEngineer)
```

```
    rechumano(ToolSpecialist)
    rechumano(TechnicalWriter)
    rechumano(SystemAdministrator)
    uso(ProcessEngineer,TailorProcessProject)
    uso(ProcessEngineer,DevelopDevCase)
    uso(ProcessEngineer,LaunchDevProcess)
    uso(ProcessEngineer,PrepGuidelinesProject)
    uso(ProcessEngineer,PrepTemplatesProject)
    uso(ToolSpecialist,SelectAcquireTools)
    uso(ToolSpecialist,SetUpTools)
    uso(ToolSpecialist,VerifyToolConfInst)
    uso(TechnicalWriter,DevelopManualStyleguide)
    uso(SystemAdministrator,SupportDevelopment)
```

# 6. Conclusões

Concluímos com o atual trabalho que a cada dia o desenvolvimento de software vem buscando soluções para os seus principais problemas atuais, como aumentar o percentual de projetos bem sucedidos respeitando o prazo, custo e principalmente alcançando o resultado esperado pelo cliente, através da implantação e uso de processos de desenvolvimento de software, facilitando assim o gerenciamento do mesmo.

E para essas empresas que buscam implantar processos definidos de software notamos que existe um esforço manual muito grande do time de desenvolvimento para seguir tais processos. Com isso vem ganhando destaque os ambientes para automação desses processos de software, que irão permitir as empresas tornar a adoção e aperfeiçoamento de seus processos algo menos custoso do ponto de vista de trabalho dependido pelos membros da equipe em controlar se o processo está sendo seguido e em que pontos o processo precisa de melhora.

## 6.1. Sumário do Trabalho

Com isso, no trabalho atual vimos que para tornar possível a implantação de tal ambiente se torna necessário o uso de notações universais para representação do processo e apontamos o SPEM, padrão proposto pela OMG com o apoio de algumas das mais importantes empresas relacionadas ao desenvolvimento de software, e baseado na notação do UML, como uma boa oportunidade para assumir essa notação universal. Feito isso foi levantado as ferramentas existentes no mercado que fazem uso de tal notação para modelagem de processos, onde além de outras opções foi escolhida a ferramenta APES2 por ser uma ferramenta open-source, o que além de torná-la de fácil expensão também permite que empresas de um porte menor possa adotar o ambiente se elevar os custos de licenças de software.

Estudamos o APES2 mais a fundo e levantamos a necessidade de pequenas modificações em sua implementação, visto que no estado atual ela já suporta a maioria das notações do SPEM a qual o ambiente proposto por [OLIVEIRA05] faz uso, como indica a seção 4.1. Ainda levantamos a necessidade de uma reestruturação maior na funcionalidade de gravar o processo em um XML para permitir uma melhor comunicação com ferramentas externas ao APES2, como visto na seção 4.2.

Uma vez visto na seção 2.3, o ImPProS precisa de uma forma lógica para representar o processo que permita o ambiente fazer inferências sobre elas de forma automática, sem intervenção humana. Encontramos nos estudos de [FALBO98] uma ontologia de processo de software adotada com sucesso em alguns outros trabalhos científicos, então na seção 4.3 mapeamos as necessidades do ImPProS com essa ontologia construída por Falbo.

Por último no capítulo 5 apresentamos um estudo de caso que demonstre que o trabalho elaborado no capítulo 4 é válido para um processo real.

## 6.2. Trabalhos Relacionados

Este trabalho de conclusão de curso faz parte dos estudos liderados por Sandro Oliveira e descritos em [OLIVEIRA05], orientado pelo Prof. Ph.D. Alexandre Vasconcelos.

## 6.3. Trabalhos Futuros

Como continuidade deste trabalho já está começando a ser feita a implementação das modificações sugeridas na seção 4.1 e 4.2 por trabalhos de iniciação cientifica do Cin/UFPE.

Ainda podemos indicar como trabalho futuro que fará uso do que foi definido na seção 4.3, a implementação da base de conhecimento e dos processos para permitir inferências em cima do mapeamento proposto.

# Referências Bibliográficas

[BERGER03]: BERGER, PATRICIA MACHADO – Instanciação de Processos de Software em Ambientes Configurados na Estação TABA [Rio de Janeiro] 2003

[FALBO98]: FALBO, RICARDO DE ALMEIDA – Integração de Conhecimento em um Ambiente de Desenvolvimento de Software [Rio de Janeiro] 1998

[MACHADO00]: MACHADO, LUIS FILIPE DIONISIO CAVALCANTI – Modelo para Definição de Processos de Software na Estação TABA. [Rio de Janeiro] 2000

[OLIVEIRA99]: OLIVEIRA, KATHIA MARCAL DE – Modelo para Construção de Ambiente de Desenvolvimento de Software Orientado a Domínio [Rio de Janeiro] 1999

[OLIVEIRA05] Oliveira, S., Vasconcelos, A., Rouiller, A. C., *Uma Proposta de um Ambiente de Implementação de Processo de Software*, Artigo publicado na Revista InfoComp – Revista de Ciência da Computação da UFLA – vol. 4, n.1, Lavras, MG, 2005.

[RUP05] Rational Unified Process® Version 2003.06.15

[CMMI02]: Software Engineering Institute, "Capability Maturity Model Integration for Software Engineering (CMMI)," Software Engineering Institute, Carnegie Mellon University CMU/SEI-2002-TR-028, ESC-TR-2002-028, 2002

[ISO95]: ISO/IEC TR 12207

[ISO03]: ISO/IEC TR 15509

[SPEM05]: Object Management Group, Software Process Engineering Metamodel Specification, Versão 1.1, Especificação adotada como padrão da OMG em 2005.

# Apêndice A: Documento de Arquitetura APES2

*Tradução do francês para português do documento de arquitetura na versão 4.0 escrita por Petit, Lionel.*

# 1 Introdução

## 1.1 Objetivo

Este documento tem por objetivo de libertar e explicar a organização e a concepção interna do software.

## 1.2 Público Alvo

O documento de arquitetura de software é destinado aos membros da equipe e os supervisores do projeto.

## 1.3 Referências

- Documento de Visão *(não traduzido)*
- Documento de Casos de Uso *(não traduzido)*
- Glossário *(não traduzido)*
- Documento de Arquitetura da versão anterior do Apes *(não traduzido)*

Estrutura

# 2 Estrutura

## 2.1 Apresentação Geral

A estrutura global da aplicação é uma organização do tipo MVC (Model/View/Controller). O principal objetivo desta arquitetura é desacoplamento entre o modelo e a apresentação deste modelo na aplicação. Além disso, tentamos focar sobre

uma extensão fácil dos tratamentos efetuados sobre o modelo e as interações possíveis entre o utilizador e a aplicação.

## 2.2 Organização em Camadas

Esta seção apresenta a organização em camadas do software. A figura 2.1 mostra as dependências entre estas camadas. Estão ao número de quatro e serão detalhadas nas secções seguintes. Esclarecerá o seu papel e as razões da sua presença.

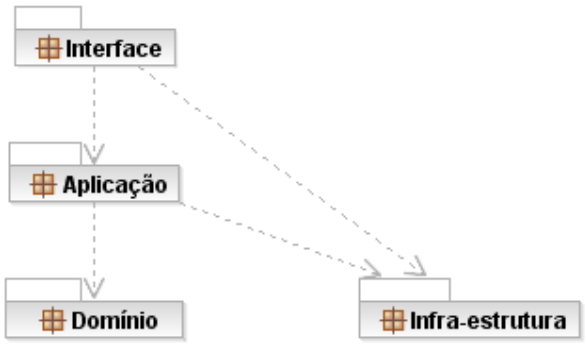


**Figura 2.1:** Organização em Camadas

### 2.2.1 Interface

Esta camada tem por objetivo a apresentação dos dados ao usuário. Permite-lhe também agir sobre o modelo subjacente. Graças a ela, pode editar o processo e lançar as operações disponíveis na camada Aplicação.

A camada Interface está coberta inteiramente pelas nossas classes personalizadas IHM (classes derivadas do API Swing).

### 2.2.2 Aplicação

Esta camada é responsável pelas regras de coerência da aplicação e a pilotagem da aplicação. A camada Aplicação comporta:

- A validação do modelo
- As classes necessárias para a aplicação das interações dos usuários sobre o modelo
- As ações acessíveis desde a camada Interface (abrir, salvar...)

É esta camada que é mais ligada aos casos de uso do projeto.

### 2.2.3 Domínio

Esta camada agrupa as classes de negócio da aplicação e a implantação das regras de gestão específicas.

A camada Domínio está coberta inteiramente pela nossa implementação do meta-modelo SPEM. A extensão da nossa implementação (em relação ao SPEM completo) é ponderada pelos casos de uso da aplicação. Se o cliente tem em mente entidades diferentes nos seus casos de uso, então é necessário estender o nosso modelo.

### 2.2.4 Infra-estrutura

Esta camada refere-se aos componentes reutilizados. Facilita o desenvolvimento de três aspectos importantes do projeto:

- A afixação e a interação com o diagrama, graças a JGraph
- O acesso e o armazenamento dos dados persistentes, graças a JSX (que deriva o API de serialização Java).
- A importação de componente, graças a SAX.

## 2.3 Subsistemas e pacotes

### 2.3.1 Organização de pacotes e componentes

Para cada camada encontra-se um ou vários pacotes:

| Camada | Pacote |
|---|---|
| Interface | apes.ui |
| Aplicação | utils, apes, apes.ui.tools, apes.ui.actions, apes.processing,apes.adapters |
| Domínio | apes.model (e qualquer de seus sub-pacotes) |
| Infra-estrutura | JGraph, JSX, SAX |

Cada um destes pacotes apresenta dependências, são representadas na figura 2.2.

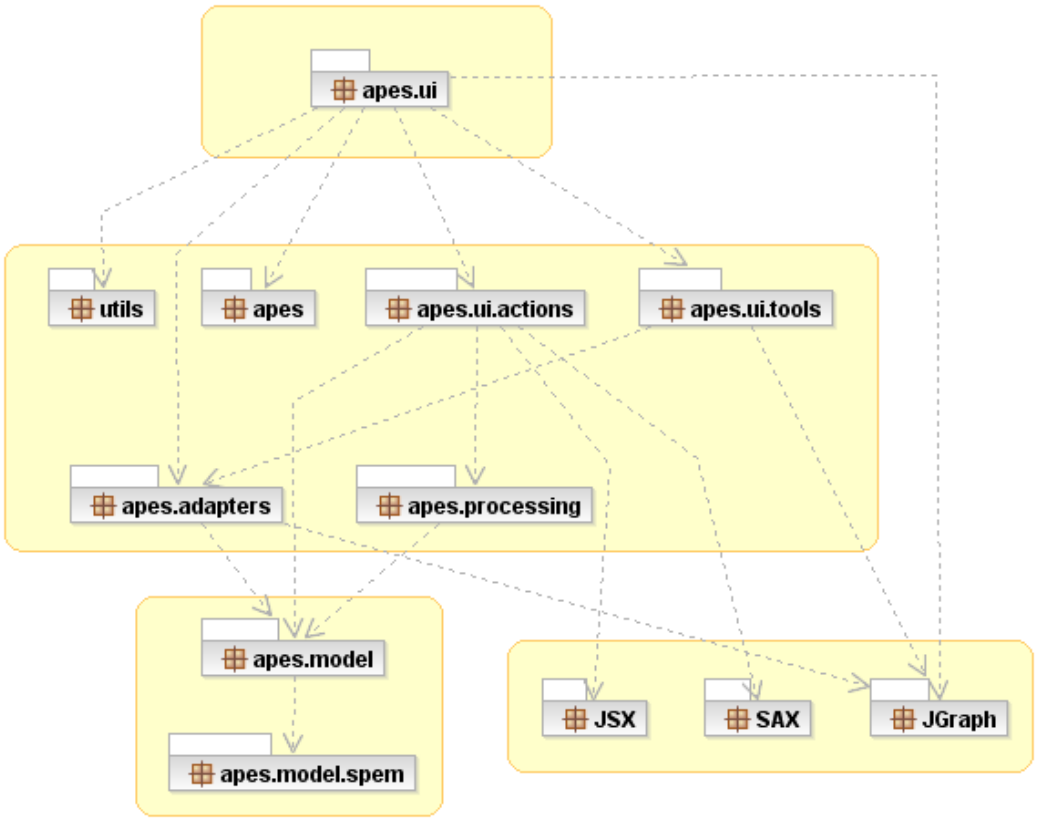


**Figura 2.2:** Dependência de Pacotes

**Obs.:** Paralelamente às dependências presentes sobre a figura 2.2, todos os pacotes da camada Aplicação são dependentes pacotes utils.

## 2.3.2 Pacotes desenvolvidos

### 2.3.2.1 utils

Este pacote fornece classes utilitárias não específicas à aplicação. Consequentemente, o conjunto das classes presentes poderá ser reutilizado em outras aplicações.

**Debug**

Classe permitindo conservar mensagens de eliminação de erros na aplicação. Permite na compilação, ativar ou não as suas funções de análise.

**ResourceManager**

Classe que facilita a internacionalização. Permite carregar automaticamente um arquivo de mensagens adaptado à língua do usuário. Dispõe de uma instância única na aplicação, é por isso que o modelo de concepção do *Singleton* particularmente é indicado no seu caso.

**IconManager**

Classe que facilita o carregamento e a manipulação de recursos gráficos. Dispõe de uma instância única na aplicação, é por isso que o modelo de concepção do *Singleton* particularmente é indicado no seu caso.

#### 2.3.2.2 apes

Este pacote fornece duas classes importantes.

**Context**

Classe representando permanentemente o estado interno da aplicação (principalmente a conversão). Dispõe de uma instância única na aplicação, é por isso que o modelo de concepção do *Singleton* particularmente é indicado no seu caso.

**Project**

Classe representando um projeto completo da aplicação. É ela que permite fazer a relação entre o modelo SPEM e a vista deste modelo através dos diagramas.

#### 2.3.2.3 apes.ui

Este pacote refere-se IHM da aplicação. Contem as classes que organizam o aspecto gráfico da aplicação e as interações do usuário com celle?ci.

**ApesFrame**

Janela principal da aplicação.

**GraphFrame**

Janela fixando uma representação gráfica. Dispõe de dois métodos segundo o modelo de concepção de fabrica para a construção da paleta de instrumentos associada e

a zona de fixação da representação gráfica. É, por conseguinte derivável seguinte os tipos de representações gráficas apresentados.

**ApesTree**

Zona de fixação do modelo completo sob a forma de árvore (só a estrutura pacotes é fixa aqui, as outras relações entre elementos serão visível em GraphFrame).

**ToolPalette**

Paleta de instrumentos de manipulação de uma representação gráfica.

#### 2.3.2.4 apes.ui.tools

Este pacote contém qualquer classe que é necessária à manipulação dos instrumentos utilizados para a edição dos diagramas.

**Tool**

Classe básica para qualquer instrumento aplicação. Aplica as operações necessárias para a gestão dos *listeners* ToolListener.

**ToolListener**

Conversão observador adaptada à classe Tool.

**DefaultTool**

Instrumento padrão da aplicação. Permite a deslocação dos nós da representação gráfica, a supressão de elementos da representação gráfica...

**CellTool**

Instrumento dedicado à adição de nós numa representação gráfica. Não é necessário derivá-lo porque funciona por prototipificação.

**EdgeTool**

Instrumento dedicado à adição de arcos numa representação gráfica. Não é necessário derivá-lo porque funciona por prototipificação.

#### 2.3.2.5 apes.ui.actions

Este pacote contém a aplicação das ações acessíveis desde IHM. Aquilo que se refere todas as operações do tipo "abrir um arquivo", "colar", "copiar"... Para cada uma destas ações, pode-se associar um ícone e um atalho teclado.

Todos dos casos de uso que necessitam o apoio de um simples botão ou a utilização de um atalho teclado têm uma classe neste pacote

#### 2.3.2.6 apes.processing

Este pacote contém as classes necessárias aos tratamentos sobre o modelo SPEM. Encontra-se duas famílias de classes neste pacote, cada uma inspirada por um modelo de concepção:

- As estratégias que determinam como o percurso do modelo será efetuado
- Os visitantes que fixam as operações efetuadas para cada tipo de elemento do modelo tratado por combinações de objetos destas duas famílias, pode-se simplesmente obter novos tratamentos a efetuarem sobre o modelo.

#### 2.3.2.7 apes.adapters

Este pacote contém simplesmente as classes necessárias para adaptar o modelo às necessidades da camada Interface.

**SpemTreeAdapter**

Classe que manipula a estrutura em árvore do modelo (pacotes e elementos de modelo).

**SpemGraphAdapter**

Classe de base necessária à adaptação do modelo para um tipo de diagrama.

#### 2.3.2.8 apes.model

##### 2.3.2.8.1 apes.model.spem

Este pacote contém sub-pacotes aderentes com padrão SPEM e as duas classes de acordo com o padrão de projeto *Visitors*. As duas classes em primeiro lugar são apresentadas, sub-pacote de `apes.model.spem` serão detalhados seguidamente.

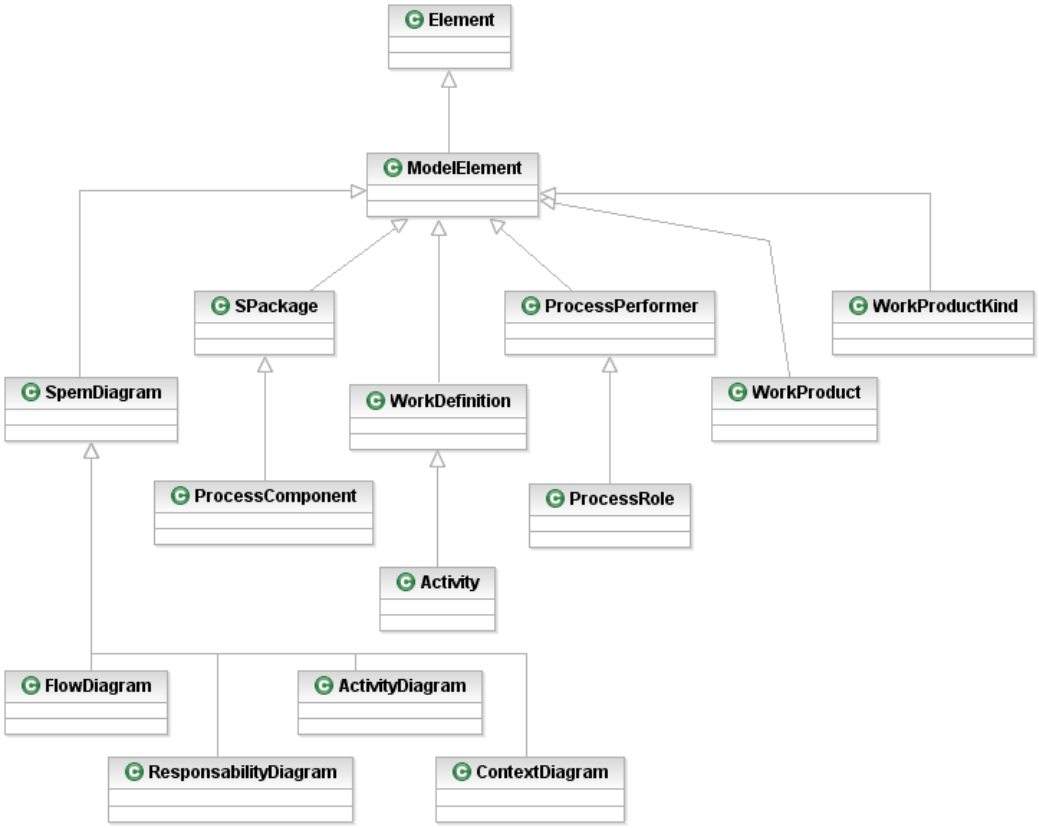


**Figura 2.3:** Hierarquia de classes

**SpemVisitor**

**DefaultSpemVisitor**

**apes.model.spem.basic**

**ExternalDescription**

Classe que representa a descrição externa de um elemento do modelo.

**Guidance**

Classe cuja instancia representa os guias do modelo.

**GuidanceKind**

Classe que permite definir grupos de guias.

**apes.model.spem.core**

Classe que permite criar pacotes e permitindo, por conseguinte agrupar elementos do modelo.

**apes.model.spem.process.components**

**ProcessComponent**

Classe que permite representar componentes do projeto.

**Process**

Classe cuja instancia representa um processo completo.

**apes.model.spem.process.structure**

**Activity**

Classe que permite representar as atividades do modelo.

**ProcessPerformer**

Classe que permite representar os executores de processo do modelo.

**ProcessRole**

Classe que permite representar os papéis de processos do modelo.

**WorkDefinition**

Classe que permite representar as definições de trabalho do modelo.

**WorkProduct**

Classe que permite representar os produtos do trabalho do modelo.

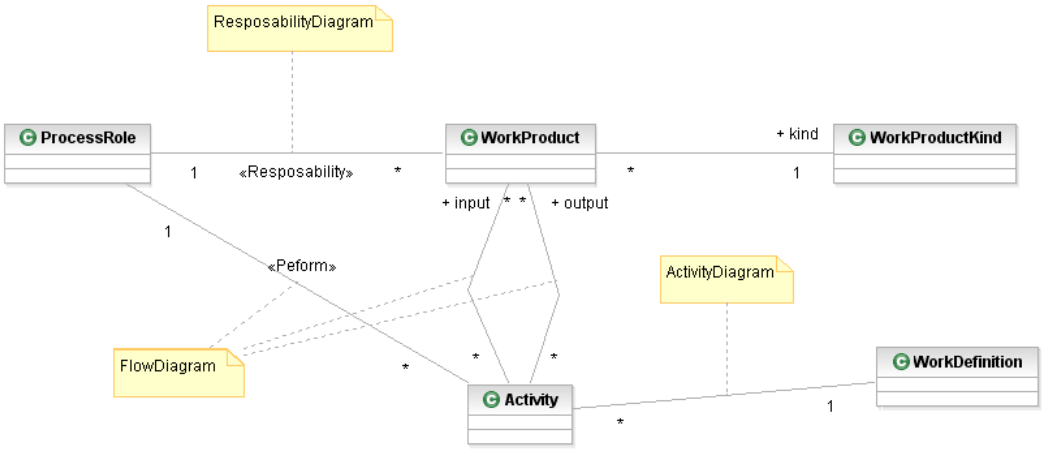


**Figura 2.4:** Estrutura

#### 2.3.2.8.2 apes.model.extension

Este pacote contém o conjunto das classes que permitem estender o *SPEM*. Há também as classes que permitem representar diagramas adaptados ao *SPEM*.

**ApesProcess**

A raiz do modelo. Contém o componente e as conversões.

**WorkProductRef**

Representa um produto de trabalho numa conversão.

**ApesWorkDefinition**

Extensão da definição de trabalho do *SPEM* para acrescentar-lhe um diagrama de seqüência e um diagrama de atividades.

**SpemDiagram**

Classe básica dos diagramas adaptados ao *SPEM*.

**FlowDiagram**

Classe permitindo representar diagramas de seqüência que permitem modelar as relações entre atividades, produtos de trabalho e papéis.

**ActivityDiagram**

Classe que permitem representar diagramas de atividades que permitem modelar o desenrolar das atividades e o processo.

**ResponsabilityDiagram**

Classe representando os papéis responsáveis dos produtos de trabalho.

**ContextDiagram**

Classe permitindo ver os produtos de trabalho requeridos pelo componente e aqueles que ele fornecidos.

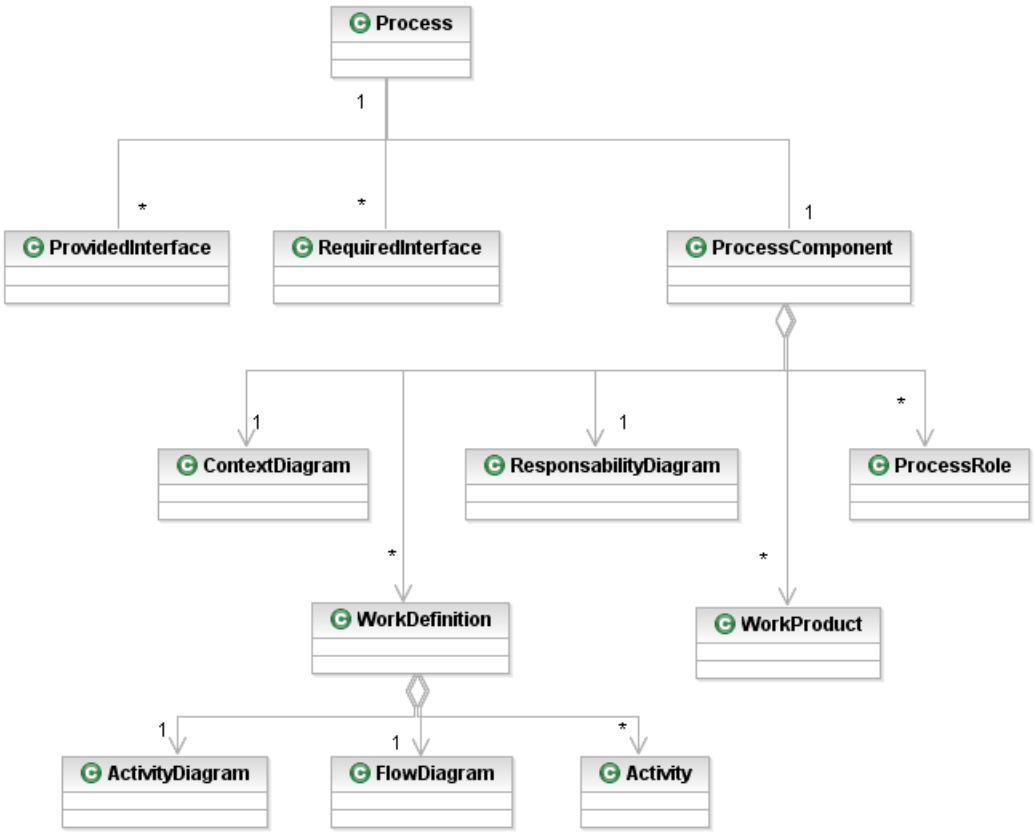


**Figura 2.5:** Processo raiz

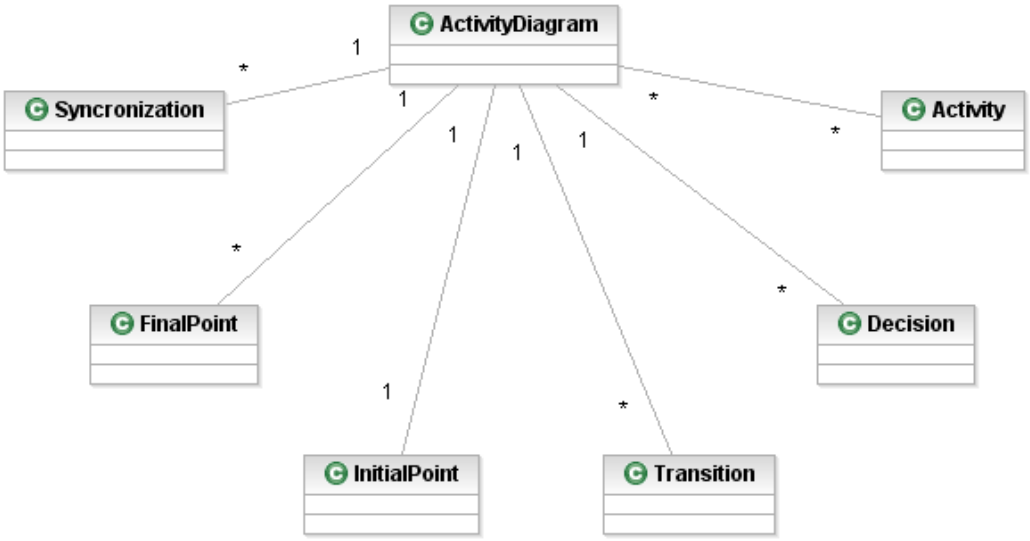


**Figura 2.6:** Diagrama de Atividade

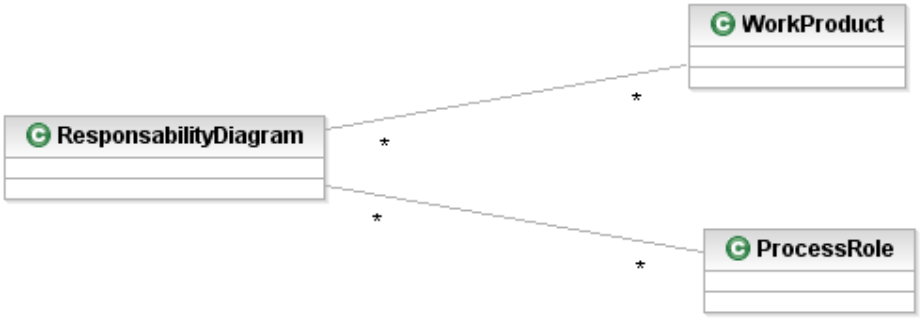


**Figura 2.7:** Diagrama de Responsabilidade

#### 2.3.2.8.3 apes.model.frontEnd

Este pacote serve ao instaurado de um mediador que centraliza as chamadas da camada controladora para a camada exemplar.

**ApesMediator**

Mediador que recebe as chamadas que provêm adaptadores, altera a camada exemplar e envia uma resposta aos objetos que ouve.

### 2.3.3 Componentes Reusáveis

#### 2.3.3.1 JGraph

É o componente da apresentação e edição das representações gráficas do modelo. É reutilizado tal qual nenhuma adaptação é necessária, é suficiente aplicar as nossas próprias classes personalizadas conformes com as interfaces especificadas no componente.

**JGraph**

Zona de apresentação de uma representação gráfica. É a classe central do componente.

**GraphModel**

Interface definindo um modelo compatível para um JGraph.

**GraphView**

Interface definindo uma vista compatível para um JGraph. Permite a associação entre os elementos do modelo e os seus representantes apresentados no JGraph.

**BasicMarqueeHandler**

Aplicação simples para gerir as interações do utilizador com um JGraph. Trata-se de um acesso privilegiado às interações com o mouse.

#### 2.3.3.2 JSX

É a biblioteca de serialização em XML. É baseada na serialização presente em API Java. É por conseguinte diretamente utilizável por qualquer objecto Java serializável sem modificação prévia.

**ObjOut**

Classe permitindo a serialização XML. Poderia ser nomeada XMLObjectOutputStream.

**ObjIn**

Classe permitindo o deserialização XML. Poderia ser nomeada XMLObjectInputStream.

#### 2.3.3.3 SAX

Esta biblioteca permite explorar o conteúdo de um arquivo XML.

# 3 Mecanismos

## 3.1 Padrão Visitor

### 3.1.1 Motivação

Suponham que se dispõe na tela de uma lista de elementos aparentados. Quando o usuário clica em um dos elementos da lista, deve-se apresentar uma lista contextual. Se dois elementos diferem unicamente pela sua profundidade na árvore de herança, as listas que resultam devem ter um número de entradas comuns. Com efeito, as operações realizáveis sobre um objeto de uma classe são também realizáveis sobre os objetos de um das suas filhas. Pode-se, por conseguinte utilizar um visitante para preencher esta tarefa dado que permite uma identificação de tipo em execução. Mas tendo em conta além das particularidades da árvore de herança, pode-se fatorizar uma parte do comportamento do visitante.

### 3.1.2 Implementação

Parte-se de uma aplicação clássica do visitante. É suficiente acrescentar um visitante concreto aplicado do seguinte modo:

- Acrescenta-se para cada classe abstrata da árvore de herança completa um método de visita protegido;
- Todos os métodos do visitante fazem apenas só uma coisa, chamar o método que corresponde à classe parente, exceto o método que corresponde à raiz da árvore de herança que não faz nada. Assim é suficiente extender a classe obtida e sobrecarregar os métodos *ad hoc* para obter o comportamento esperado, ou seja fatorizar comportamentos em função do ramo de herança.

### 3.1.3 Exemplo de Código

Eis um exemplo no Java deste modelo.

Supor que nós dispor classe seguinte:

```
abstract class Element
{
    public abstract void accept(Visitor v);
}
class ElementA extends Element
{
    public void accept(Visitor v) { v.visitElementA(this); }
}
class ElementB extends Element
{
    public void accept(Visitor v) { v.visitElementB(this); }
}
class ElementC extends ElementB
{
    public void accept(Visitor v) { v.visitElementC(this); }
}
```

Então a aplicação de um visitante dará isto:

```
interface Visitor
{
    public void visitElementA(ElementA a);
    public void visitElementB(ElementB b);
    public void visitElementC(ElementC c);
}
class ConcreteVisitor implements Visitor
{
    public void visitElement(Element e) { }
    public void visitElementA(ElementA a) { visitElement(a); }
    public void visitElementB(ElementB b) { visitElement(b); }
    public void visitElementC(ElementC c) { visitElementB(c); }
}
```

### 3.1.4 Utilizações notáveis em APES

A classe `DefaultSpemVisitor` do pacote `apes.spem` é conforme com este padrão. As suas filhas são utilizadas nomeadamente na árvore para apresentar os ícones associados aos seus nós e para associar uma lista contextual a cada nó.

# 4 Qualidade da Arquitetura

## 4.1 Vantagens

A principal vantagem desta arquitetura é reutilização forte de componentes provados no seu domínio. Além disso, estes componentes são mantidos ativamente, e utilizados em outras aplicações. É importante notar a constituição de um patrimônio das

classes reusáveis por meio pacote `utils`. Seguidamente, novos tratamentos sobre o modelo podem ser escritos facilmente graças à organização pacotes `apes.processing` e a utilização do padrão de projeto visitor na nossa aplicação do *SPEM*. Por último, uma das principais vantagens da nossa arquitetura é o acoplamento fraco entre a nossa aplicação do *SPEM* e o resto da aplicação.

## 4.2 Desvantagens

Esta arquitetura apresenta alguns inconvenientes. Mais flagrante provem de uma limitação do componente *JGraph*. Com efeito, este último impõe uma relação 1/1 entre os elementos do `GraphModel` e os elementos afixados no *JGraph*. O nosso modelo *SPEM* não pode, por conseguinte aplicar diretamente a interface `GraphModel`. No entanto, este inconveniente tem uma importância limitada dado que a aplicação direta da interface `GraphModel` aumentaria o acoplamento entre o nosso modelo e *JGraph*. Seguidamente, devemos escrever-nos adaptadores entre o modelo *SPEM* e *JGraph* por um lado, e entre o modelo *SPEM* e *JTree* por outro lado. Estes adaptadores são um trabalho importante da arquitetura e são pesados a escrever se o modelo sub jacente é afastado da interface a obter. Por último, a utilização do padrão de projeto visitor no pacote `apes.processing` pode provocar o aparecimento de classes fastidiosas a escrever se o modelo *SPEM* aplicado fica demasiado vasto.

## 4.3 Extensões Possíveis

O modelo *SPEM* aplicado em *APES* pode ser estendido e alterado. Pode ser importante torná-lo perceptível (além visível), a fim de reduzir mais ainda o seu acoplamento com o resto da aplicação. Será necessário, no entanto, avaliar o impacto tal remanescente nos desempenhos globais da aplicação (o fato de os adaptadores do modelo sejam eles mesmo perceptíveis a utilização da prototipificação nas classes

`EdgeTool` e `CellTool` bem como a presença de métodos fabricos em `GraphFrame` permite o número facilmente aumentar de tipos de diagramas diferentes.

# 5 Principio da Evolução

Este capítulo apresenta as evoluções essenciais efetuadas sobre a arquitetura de *Apes*.

## 5.1 JGraph

A versão utilizada de *JGraph* passou do 1. * ao 3. isto gerou numerosas mudanças a nível da utilização das representações gráficas.

## 5.2 Comunicação entre as camadas controlador e modelo

Para simplificar as comunicações entre a árvore, as representações gráficas e o modelo, um mediador foi instaurado. Para efetuar uma ação sobre o modelo, o adaptador deve enviar uma chamada que contem o seu pedido ao mediador. Por sua vez verifica que a ação é possível e, se for caso disso, envia uma mensagem aos diferentes objetos que são à sua escuta. Resulta que os adaptadores não podem alterar o modelo diretamente. As vantagens:

- Desacoplamento entre a árvore e as representações gráficas
- Centralização das ações
- Esclarecimento da comunicação

Inconvenientes:

- Uma má aplicação poderia tornar a manutenção do mediador difícil.

Diagrama de seqüência da inserção de um elemento na árvore:

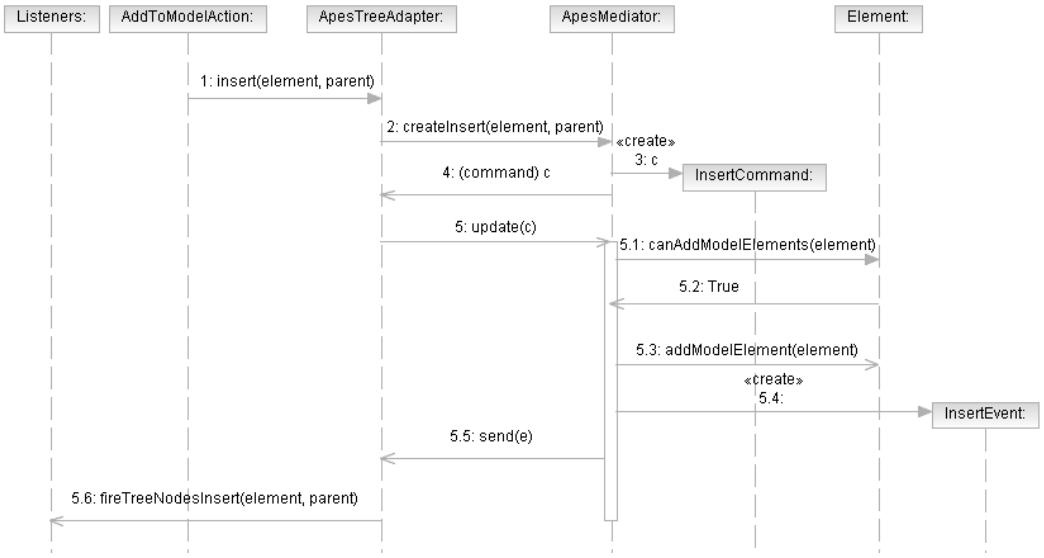


**Figura 5.1:** Digrama de seqüência