APSEE-StaticPolicy: Verificação de Políticas Estáticas em Modelos de Processos de Software

Rodrigo Quites Reis[*, #] Carla Alessandra Lima Reis[*, #] Daltro José Nunes[*]
[*]Programa de Pós-Graduação em Computação - Universidade Federal do Rio Grande do Sul
[#]Departamento de Informática - Universidade Federal do Pará
E-mail: quites@computer.org / clima@ufpa.br / daltro@inf.ufrgs.br

Resumo

Recentemente diversos trabalhos da Tecnologia de Processo de Software vem sendo propostos no sentido de descrever estratégias para auxiliar a automação da gerência do processo de desenvolvimento de software. Este artigo apresenta uma contribuição neste contexto, descrevendo um mecanismo para a definição formal de Políticas Estáticas, um construtor que é útil na verificação de propriedades sintáticas de modelos de processos de software. O mecanismo proposto atua durante a modelagem de processos de software, permitindo a reutilização de Políticas em diferentes processos (e seus componentes) e sua verificação automática em ambientes de desenvolvimento de software orientados ao processo (Process-Centered Software Engineering Environments - PSEEs). Neste texto, a linguagem para definição de políticas é apresentada através da sua descrição informal, seguida de exemplos da sua utilização. Além disso, o artigo ainda discute alguns dos aspectos principais da semântica formal (algébrica) do mecanismo de interpretação de Políticas, da qual foi derivada a implementação de um protótipo integrado a um PSEE.

.

Abstract

Software Process Technology evolved to support software processes management. This paper presents a contribution to this field, describing a mechanism to formally model Static Policies, which are useful to automate the verification of processes properties. The proposed mechanism acts during software process modeling, allowing Policy reuse across different processes under enaction of a Process Centered Software Engineering Environment (PSEE). In this text, the language for StaticPolicy definition is presented first though an informal description, followed by a couple of examples. This paper also discusses some of the main issues related to the formal (algebraic) semantics defined for the Policy interpreter which was used as a base to implement a Java-based prototype in a PSEE.

1. Introdução

Software tornou-se a base de sustentação de inúmeras organizações dos mais diversos ramos de atuação espalhados pelo planeta, consistindo de um elemento estratégico na diferenciação de produtos e serviços atuais. Atualmente, o software está embutido em sistemas de uma infindável lista de diferentes ciências e tecnologias, e segundo [20], será o propulsor dos novos avanços que influenciam uma ampla gama de indústrias, envolvendo desde a Educação Elementar até corporações envolvidas com a Engenharia Genética.

Apesar dos inúmeros avanços recentes nesta área, muito ainda é discutido acerca da baixa qualidade e produtividade da indústria mundial de software, o que se reflete na insatisfação dos seus usuários e em prejuízos financeiros de enormes proporções. Adicionalmente, os computadores estão tornando-se rapidamente componentes comuns do dia-a-dia de um número maior de pessoas que, por sua vez, apontam necessidades com requisitos de complexidade cada vez maiores. Assim, como resultado, o atual processo de desenvolvimento de software (processo de software, doravante) é uma atividade que exige a participação de uma quantidade cada vez maior de profissionais com formações e expectativas diferenciadas que, através da Internet, trabalham de forma distribuída em diferentes localidades geográficas, em um ambiente com inúmeros desafios técnicos, sociais e culturais [10].

No desenvolvimento de software de alta complexidade que exija o envolvimento de muitos profissionais vem sendo proposta a utilização da tecnologia de processo de software para automatizar a gerência do processo [6]. Neste sentido, diversas ferramentas, linguagens e metodologias vem sendo desenvolvidas e apresentadas na literatura especializada com o objetivo de automatizar e aperfeiçoar o processo de desenvolvimento adotado pelas organizações.

A tecnologia de processo de software é fortemente influenciada pelas linguagens utilizadas para a modelagem do processo, descritas na literatura genericamente como PMLs (*Process Modelling Languages*) [9]. PMLs são utilizadas para descrever formalmente o processo de desenvolvimento de software através da especificação dos seus componentes, etapas, e a influência da atividade de pessoas na condução deste processo. As PMLs são freqüentemente associadas a PSEEs (*Process-Centered Software Engineering Environments*), que permitem automatizar a gerência do processo de software.

Este artigo apresenta uma proposta que complementa o papel desempenhado por PMLs, através de operadores para a modelagem e verificação automatizada de Políticas Estáticas em modelos de processos de software. Uma Política Estática descreve uma seqüência de propriedades estáticas para programas escritos em PMLs, podendo ser reutilizada em diferentes modelos adotados na organização de desenvolvimento de software.

O artigo é apresentado como segue. Na seção 2, alguns dos principais conceitos que norteiam a área de processos de software são apresentados como base contextual do restante do texto. A seção 3 apresenta o meta-modelo de processos de software APSEE (adotado como base para experimentação do modelo de Políticas proposto). A seção 4 apresenta a linguagem APSEE-StaticPolicy inicialmente através da sua descrição informal, seguida de exemplos, especificação da estrutura de controle para realizar a sua verificação automática, incluindo as limitações do modelo e implementação atual. Na seção 5 são discutidos os trabalhos relacionados encontrados na literatura e, finalmente, na seção 6 são apresentadas as conclusões e atividades futuras.

2. Automação do Processo de Software

A qualidade na definição de processos de desenvolvimento de software é um dos elementos-chave para que uma organização possa atingir melhores níveis de maturidade na avaliação dos modelos CMM [17] e SPICE [5]. A Tecnologia de Processos de Software [9] surgiu neste contexto, propondo o desenvolvimento e adoção de mecanismos para automatizar a gerência dos processos, tendo como objetivos principais:

- Definir ferramentas para descrever os processos e acompanhar a sua execução, isto é, controlar a realização de atividades que ocorrem no desenvolvimento de software, registrando as ocorrências e detectando anomalias que podem ser corrigidas em projetos correntes e futuros;
- Facilitar a adoção de uma estratégia de melhoria da maturidade dos processos de uma organização, a partir da prescrição, monitoração e registro automatizado de eventos;
- Permitir a reutilização do conhecimento produzido acerca processos bem sucedidos na organização para contextos similares no futuro;
- Proporcionar um controle preciso na alocação e consumo de recursos durante o desenvolvimento de software;
- Coletar métricas dos projetos da organização e torná-las disponíveis para consultas posteriores.

Deste modo, inúmeros ambientes e metodologias propostos nos últimos dez anos na literatura em Engenharia de Software constituem o que hoje denominamos Tecnologia de

Processo de Software [4]. As sub-seções a seguir aprofundam-se na terminologia utilizada na definição do modelo de políticas de processos de software proposto neste artigo.

2.1. Modelagem do Processo de Software

O processo de software envolve atividades complexas desempenhadas por pessoas (agentes) com as mais diversas capacidades. Um modelo de processo de software é uma descrição abstrata do processo de software. Vários tipos de dados são integrados em um modelo de processo de software para indicar quem, quando, onde, como e por que os passos são realizados [8]. Para representar um modelo de processo de software é freqüentemente adotada uma PML, a qual deve oferecer recursos para descrever e manipular elementos do processo.

Um modelo de processo instanciado ou processo executável é um modelo de processo pronto para execução. Um projeto é a instância de um processo, com objetivos e restrições específicos, envolvendo agentes, prazos, orçamentos, recursos e um processo de desenvolvimento (o encadeamento de atividades necessárias para atingir o objetivo). Assim, a modelagem de processos de software é freqüentemente associada à sua execução, descrita na seção a seguir.

2.2. Execução de Modelos de Processo

Com um modelo de processos definido é possível controlar a execução dos passos de forma automatizada. Na fase de execução de modelos de processo de software devem ser levadas em consideração as questões de coordenação de múltiplos usuários e a interação entre as ferramentas automatizadas e os agentes (profissionais envolvidos no processo).

PSEEs (*Process-Centered Software Engineering Environments* ou ambientes de desenvolvimento de software orientados ao processo [14]) constituem um tipo especial de ambientes de desenvolvimento de software que surgiram na última década para apoiar a definição rigorosa de processos de software com o objetivo de automatizar a gerência do desenvolvimento. Tais ambientes geralmente provêem serviços para análise, simulação, execução e reutilização das definições de processos, que cooperam no aperfeiçoamento contínuo de processos.

A arquitetura de um PSEE usualmente define como componente central a Máquina de Processo (*process enaction engine* [4]) que auxilia na coordenação das atividades realizadas por pessoas e por ferramentas automatizadas, sendo responsável pela interpretação/execução dos modelos de processos descritos com PMLs. Segundo [14], uma máquina de processos pode: ativar automaticamente atividades sem intervenção humana através de uma integração com as ferramentas do ambiente; apoiar o envolvimento cooperativo dos desenvolvedores; monitorar o andamento do processo; e, registrar histórico da sua execução. Ainda segundo [14], a máquina de processo também deve garantir a execução das atividades na seqüência definida no modelo de processo (fluxo de controle); a repetição de atividades; a informação de *feedback* sobre o andamento do processo; a gerência das informações de processo (incluindo gerência de versões); a coleta automática de métricas; a mudança do processo durante sua execução; a interação com as ferramentas do ambiente e a gerência de alocação de recursos.

3. O Modelo APSEE

Esta seção apresenta resumidamente o meta-modelo APSEE utilizado como base para a descrição dos mecanismos descritos neste artigo. O meta-modelo APSEE define um conjunto de serviços para a gerência automatizada de processos de software, tendo sido especificado para o ambiente PROSOFT [15].

PROSOFT é um ambiente integrado de desenvolvimento de software construído para apoiar o uso de métodos formais na construção de software de alta complexidade [16]. Os componentes do PROSOFT são os ATOs - Ambientes de Tratamento de Objetos. Todo ATO especifica algebricamente um tipo abstrato de dados e a ICS (Interface de Comunicação de Sistema), provê o mecanismo básico de comunicação dos ATOs PROSOFT. O desenvolvimento de software, sob a ótica PROSOFT, consiste na definição de um ou mais ATOs. Todo ATO possui um nome, e na sua definição são especificados a sua **classe** (representada graficamente e definida através da composição de tipos de dados), a **interface** das operações e a especificação (semântica) das **operações**.

Recentemente o ambiente PROSOFT vem sendo utilizado para experimentação e validação de diferentes técnicas, ferramentas e metodologias para apoiar de forma integrada os aspectos de interação cooperativa de profissionais no desenvolvimento de produtos de alta complexidade. Deste modo, a última versão do ambiente está implementada em Java, beneficiando-se dos mecanismos de distribuição e portabilidade fornecidos por esta linguagem.

Um primeiro mecanismo de gerenciamento de processos de software para o PROSOFT foi desenvolvido em 1998 [14]. Nos últimos anos este mecanismo foi estendido, com a introdução de novos recursos para apoiar de forma mais adequada as características evolucionárias e dinâmicas do processo de software, dando origem ao modelo APSEE. Um resumo das características principais do meta-modelo APSEE é apresentado abaixo:

- É baseado em um paradigma baseado em atividades, e sua PML descreve um processo de software como um conjunto de atividades orientadas para um objetivo específico;
- Provê uma taxonomia hierárquica para os tipos de processos em um estilo orientado a objetos (O-O) para todos os componentes dos processos (*roles, artifacts, resources, tools, policies, activities,* entre outros) de forma similar ao ambiente E3 [11];
- Foi especificado com o PROSOFT-Algébrico [16] sendo, posteriormente, implementado um protótipo no Java-PROSOFT (implementação atual do ambiente), utilizando os serviços de gerência de trabalho cooperativo e controle de versões fornecidos pelo repositório do ambiente;
- Permite a definição tardia de estruturas de processos, adiando a definição de estruturas concretas até o momento de execução do processo;
- Fornece *ProcessTemplates* como estruturas reutilizáveis para definição de modelos de processos, permitindo a consulta de processos a partir de um motor de busca que realiza o cômputo de similaridade através do raciocínio baseado em casos [21];
- Provê um conjunto dinâmico de mecanismos de controle para conexão de atividades em um processo. Atualmente, quatro tipos básicos de precedência na execução de atividades estão disponíveis (*Strong*, *Weak*, *Synchronization* e *Feedback*, cuja semântica é descrita por [7]).

Para os propósitos deste texto vale destacar a existência no modelo APSEE de quatro tipos principais de processo, representando o diferente nível de detalhamento necessário para o modelo. *ProcessTemplate* fornece uma descrição genérica e abstrata, apropriada para reutilização futura (podendo ser reutilizado em contextos organizacionais e/ou tecnológicos diferentes). O *AbstractProcessModel* fornece uma descrição adaptada para uma organização específica, porém sem uma atribuição precisa de quais elementos do processo serão alocados durante o desenvolvimento (o *AbstractProcessModel* também é dito "não-instanciado"). O *InstantiatedProcessModel* consiste em um modelo com informações já instanciadas, definindo a alocação de recursos e agentes para as atividades dos processos. Por fim, o *EnactedProcessModel* é um modelo que representa informações sobre a real execução do

processo. É importante observar que um projetista de processos pode, com as ferramentas disponíveis, descrever um modelo de processo de software em qualquer um destes níveis de detalhe (desde descrições essencialmente abstratas como os *ProcessTemplates* até descrições executáveis na forma de modelos instanciados), adicionando informações no modelo de processo de acordo com a necessidade.

4. Políticas Estáticas no modelo APSEE

Segundo [6], políticas de processos de software consistem em "princípios que conduzem o desenvolvimento e/ou a execução de processos de software". No modelo APSEE, uma política pode ser informalmente descrita como um conjunto de propriedades que atuam na formação e execução de modelos de processos de software, representando um conhecimento gerencial genérico e reutilizável para diferentes contextos. Uma política pode estar habilitada em uma atividade de um processo, em um fragmento de processo (estando habilitada em todas as atividades componentes), ou em uma organização (isto é, em todos os processos existentes na organização), permitindo a sua reutilização em diferentes contextos.

De fato, no modelo APSEE, Políticas foram propostas com o objetivo de complementar os *Templates* na descrição formal de elementos reutilizáveis para processos de software, podendo ser úteis para expressar conhecimento relacionado com a gerência de processos de software de uma maneira computável e compacta. No modelo APSEE, as Políticas estendem a PML do ambiente e estão disponíveis em três tipos, com diferentes finalidades [22]:

- **Políticas Estáticas.** Apóiam os projetistas/programadores de modelos de processo de software, definindo regras sintáticas que atuam na formação dos modelos, podendo ser úteis para definir formalmente que boas práticas de gerenciamento de projetos são adotadas nos modelos de processos da organização;
- **Políticas de Instanciação.** Definem critérios para a alocação de recursos e pessoas utilizadas por uma organização em contextos ou processos específicos, podendo ser baseadas no histórico da execução de processos anteriores;
- **Políticas Dinâmicas.** São baseadas no *log* de eventos gerado durante a execução de processos de software.

As Políticas Estáticas, objeto deste texto, são apresentadas nas sub-seções a seguir.

4.1. Semântica informal das Políticas Estáticas e do Mecanismo de Verificação

Políticas Estáticas (no modelo de dados descrito a seguir sendo referenciadas como *StaticPolicies*) atuam na modelagem, auxiliando o projetista/programador de processo através da verificação de propriedades sintáticas habilitadas para elementos do processo. Assim, uma política estática descreve uma lista ordenada de propriedades dos processos que, quando avaliadas, devem ser satisfeitas (retornando *True*, na sua avaliação).

Políticas Estáticas são verificadas durante a modelagem de processos, quando o projetista de processo solicita a verificação do modelo (um passo obrigatório no ciclo de vida de processos APSEE para habilitar execução de um modelo de processo instanciado).

Os principais componentes de uma política estática são:

- Uma **identificação** única;
- **Nome** da política;
- **Descrição** textual da política;
- **Tipo da política**, indica o tipo da Política Estática na sua hierarquia de tipos;
- **Tipo de reação**. Determina se a política é obrigatória (*mandatory* - impede o prosseguimento da verificação enquanto a política não for satisfeita), ou

desejável (*desirable* – fornecendo somente um aviso para o usuário durante a verificação do modelo quando a política não é satisfeita);

- **Interface da Política.** Especifica o tipo de objeto-APSEE (objetos das classes *Activity, Role, Agent, Artifact, Process* ou *Resource*) que será tratado pela política. Por exemplo, se uma política tem interface definida para o tipo *Activity*, isto significa que a política pode atuar sobre atividades (como no exemplo "*External Review Policy*" da seção 4.3 – figura 3). Ainda, se a política citada for aplicada a um fragmento de processo, isto indica que todas as suas atividades constituintes do processo são tratadas por esta política;
- **Propriedades para habilitação (*Enabling Properties*).** Constituem as pré-condições lógicas que devem ser satisfeitas para que uma política seja executada (relacionadas ao tipo definido na interface da política).
- **Propriedades (*Properties*).** Sempre que as propriedades para habilitação são satisfeitas, as propriedades descritas no corpo de uma política estática são verificadas. Caso alguma propriedade não seja satisfeita, o mecanismo de verificação de processos adota o comportamento especificado pelo tipo de reação atribuído à política. As propriedades correspondem essencialmente às chamadas de métodos disponíveis nos tipos definidos no APSEE.

A figura 1 apresenta um diagrama de atividades SADT [23] que descreve de forma simplificada o encadeamento e fluxo de dados das etapas de modelagem e verificação de processos de software com Políticas Estáticas. Inicialmente, o processo de software é modelado (com a PML adotada pelo ambiente) e instanciado (com auxílio do mecanismo de instanciação de processos denominado *APSEE-Planner* [21]), obtendo como resultado um modelo de processo de software instanciado (objeto de *InstantiatedProcess*). O modelo produzido pela atividade "Modelagem e Instanciação de Processos de Software" é então verificado quanto à correta formação do seu grafo de precedência de atividades (etapa "Verificação de consistência do modelo" na figura 1) e, posteriormente, o modelo é verificado quanto a sua conformidade em relação às políticas estáticas habilitadas (etapa "Verificação das Políticas Estáticas"). Nesta última etapa, a verificação é realizada para todas as atividades que compõem o modelo. Como resultado, são gerados uma lista de ocorrências (com *warnings* - para políticas *desirable* que não foram satisfeitas, e *errors* - para políticas obrigatórias não satisfeitas) e o modelo de processo de software propriamente dito, pronto para execução (se não houver erros).

Para os propósitos deste texto, convém detalhar a etapa de "Verificação das Políticas Estáticas" do modelo. Esta etapa é responsável por varrer o grafo de precedência de atividades verificando as políticas habilitadas para cada atividade (isto é, se alguma política habilitada possui como interface o tipo de atividade em questão e se as propriedades para habilitação são satisfeitas). Ainda, para cada atividade analisada, todos os seus componentes associados são verificados de forma análoga. O procedimento de verificação se encerra quando todas as atividades que constituem o fragmento de processo sob análise tenham sido verificadas. Uma descrição mais rigorosa deste algoritmo é apresentada na seção 4.6 a seguir.

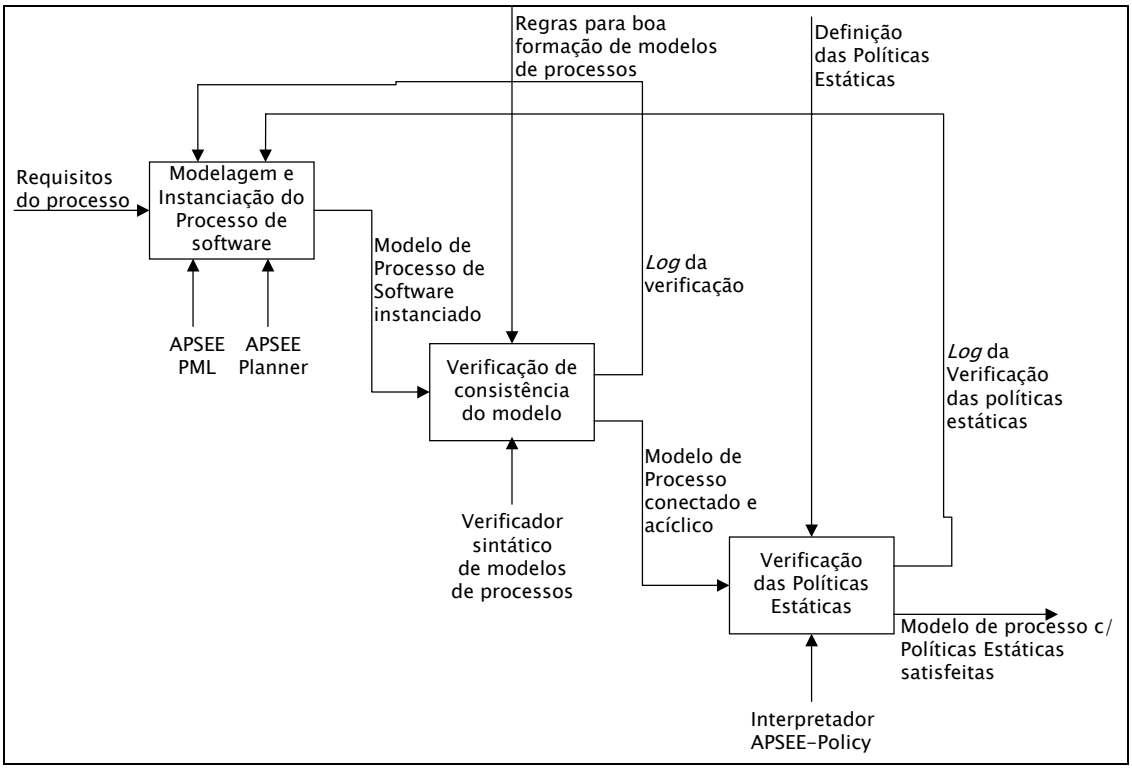

Figura 1 Procedimento de verificação de Políticas Estáticas (simplificado)

4.2. Gramática da linguagem de definição de Políticas Estáticas

Os programas para definição de Políticas Estáticas seguem o formato descrito pela gramática apresentada na figura 2, correspondendo aos elementos descritos na seção anterior.

<policy>	→ **Id** <Stringue> **(**[**Name** <Stringue>] [**; Description** <Stringue>] **;**
	TypeID <type_id> **;**
	Mandatory <boolean> **;**
	Interface <pol_interface> **;**
	Enabling_Properties <condition> **;**
	Properties <condition>**) ;**
<boolean>	→ **true** | **false**
<pol_interface>	→ <label> **:** <type>
<condition>	→ <policy_operand> [<opt_relation>] [<opt_connection>]
<opt_relation>	→ <comparison> <policy_operand>
<comparison>	→ **>** | **<** | **>=** | **<=** | **=** | **<>** | **contains** | **not_contains** | **sub_type_of**
<opt_connection>	→ <conn_type> <condition>
<conn_type>	→ **and** | **or**
<policy_operand>	→ <policy_object> <operators>
<operators>	→ <policy_operator>*
<policy_operator>	→ <method_id> <parameters> | <reserved_word> | ∪ | ∩
<parameters>	→ <policy_operand>*
<reserved_word>	→ **any**| **all** | **no**
<type>	→ **Activity** | **Artifact** | **Process** | **Resource** | **Agent** | **Role**
<policy_object>	→ <Stringue>
<label>	→ <Stringue>
<method_id>	→ <Stringue>

Figura 2 Gramática da linguagem StaticPolicies

4.3. Exemplos

Esta seção apresenta dois exemplos de políticas estáticas descritas de acordo com a gramática especificada para a linguagem. A figura 3apresenta a Política de Revisão Externa

("*External Review*") utilizada para exigir que toda atividade do tipo "Development" (ou seus sub-tipos) tenha como sucessora uma atividade do tipo "Verification" (ou seus sub-tipos). Além disso, esta política exige que o conjunto de agentes que atue na atividade "Verification" não contenha agentes da atividade anterior, e que os artefatos consumidos e produzidos pela atividade de desenvolvimento sejam fornecidos para a verificação. Finalmente, a política exige que algum artefato produzido pela atividade de verificação seja do tipo "Review Report" (ou sub-tipos) e, ainda, que a conexão de feedback entre as atividades seja feita em função do conteúdo de "Review Report". Esta política é particularmente útil em atividades ou processos em que a complexidade do software em produção exige que as etapas de desenvolvimento e verificação sejam conduzidas por equipes independentes.

Name: "External Review"
Description: "Each development activity must be followed by a verification activity which, in turn, must use a different set of agents having access to the artifacts used and produced by the development activity".
Mandatory: True
Interface: a: Activity;
Enabling Properties: a.get_type() **sub_type_of** "Development"
Properties:
a.number_of_sucessors() = 1 **and**
a.get_sucessor().get_type() **sub_type_of** "Verification" **and**
a.get_sucessor().get_agents() **not_contains** a.get_agents() **and**
a.get_sucessor().get_input_artifacts() **contains** a.get_output_artifacts() ∪ a.get_input_artifacts() **and**
a.get_sucessor().get_output_artifacts().any.get_type() **sub_type_of** "Review Report" **and**
a.get_feedback_connection(a.get_sucessor()).get_artifact().get_type() **sub_type_of** "Review Report";

Figura 3 Política "External Review"

A figura 4, por sua vez, apresenta a definição para a política "Trace design to requirements", inspirada no princípio homônimo (número 62) definido em [3]. Tal política define que todo artefato da hierarquia "Software Artifact" produzido por uma atividade cujo tipo seja "Software Design" deve ser derivado de um objeto do tipo "ReqSpec".

Name: "Trace design to requirements"
Description: "The designer must know which requirements are being satisfied by each component".
Mandatory: True
Interface: a: Artifact;
Enabling Properties: a.get_type() **sub_type_of** "Software Artifact" **and**
a.is_produced_by().any.get_type() **sub_type_of** "Software Design"
Properties: a.is_derived_from.any **sub_type_of** "ReqSpec";

Figura 4 Definição da Política "Trace design to requirements"

A figura 5 apresenta um esquema com a descrição de um processo de software na PML adotada pelo ambiente APSEE. Nesta PML, atividades são representadas graficamente por elipses, artefatos com retângulos tridimensionais, conexão por arcos orientados, e políticas são representadas por retângulos brancos. À esquerda da figura é apresentado um modelo de processo de alto nível com três atividades-componentes: *Requirements Definition, Requirements Specification* e *Software Design*. Em *SoftwareDesign* (atividade decomposta, representada pela elipse com traço mais grosso) está habilitada a política estática *ExternalReview*. À direita da figura é apresentado o detalhamento do fragmento *Software Design*.

É importante ressaltar que a figura 5 ainda apresenta o resultado do processo de verificação da política *ExternalReview* no fragmento *Software Design.* No rodapé da tela colocada à direita da figura é apresenta a mensagem do sistema "*Error: "Database design" activity does not satisfy "External Review" policy*", que é acompanhada da identificação da atividade correspondente no modelo. O resultado da etapa de verificação de políticas estáticas

em processos APSEE é composto de uma lista ordenada de avisos (*warnings*) gerados por políticas não obrigatórias. No exemplo dado, somente a primeira ocorrência da lista é mostrada.

4.4. Funções embutidas na linguagem

A linguagem define um conjunto de funções aplicáveis aos tipos *Activity, Artifact, Process, Resource, Agent, Role* e *Connection*, consultando informações sintáticas acerca do modelo de processo em questão. O conjunto de funções disponíveis para o programador é fixo, determinado por primitivas disponíveis no mecanismo de definição e execução de processos do ambiente APSEE. A lista completa das funções disponíveis na linguagem e a sua semântica estão disponíveis em [22].

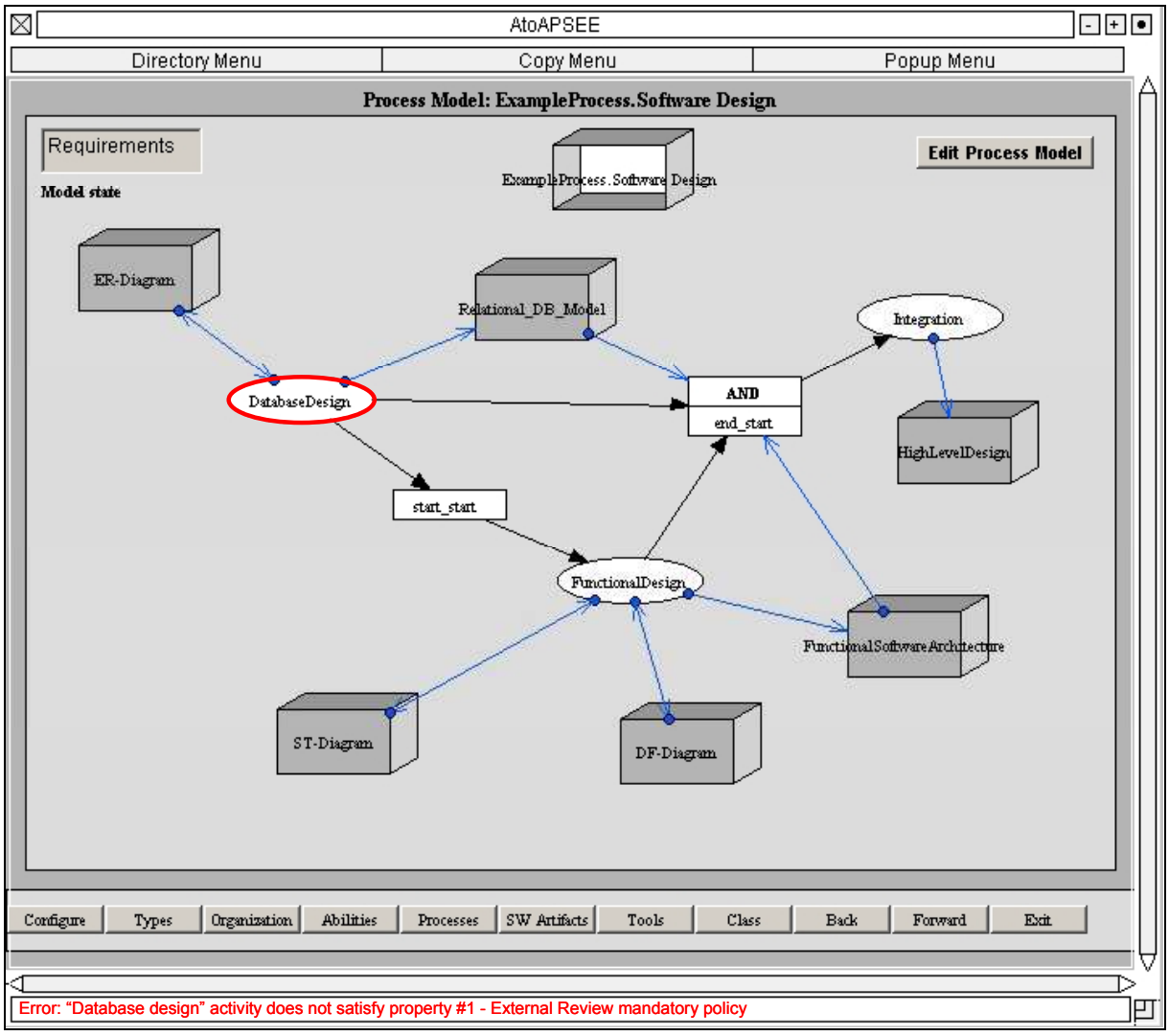

Figura 5 Exemplo de verificação de políticas estáticas em um modelo de processo de software

4.5. Tipos de dados PROSOFT para representação interna da linguagem

Esta seção apresenta alguns dos principais tipos de dados do PROSOFT-Algébrico [15] [16] criados para o armazenamento e processamento de Políticas Estáticas. Os tipos de dados PROSOFT são classificados como primitivos (*Longue, Stringue, Texto, Real, Caractere, Booleano* e *Coordenada*), compostos (*Set, List, Mapping, Record* e *Disjount*

union) e definidos pelo usuário, e a notação gráfica adotada para especificar a composição dos tipos é apresentada em [16].

A definição de Política Estática é realizada pelo usuário em um editor de textos do ambiente PROSOFT (ATO Texto). Neste texto de entrada é realizada análise sintática e léxica que gera como resultado objetos das classes PROSOFT descritas nesta seção, conforme ilustrado pelo esquema da figura 6. Assim, a descrição de Políticas Estáticas segue o padrão definido pelo tipo de dados *StaticPolicy* descrito na figura 7. Políticas Estáticas são individualizadas por um identificador único e consistem dos atributos *Name, Description, Type-id, Mandatory, Interface, EnablingProperty e Property*, correspondentes aos itens descritos na seção 4.1, definidos na linguagem.

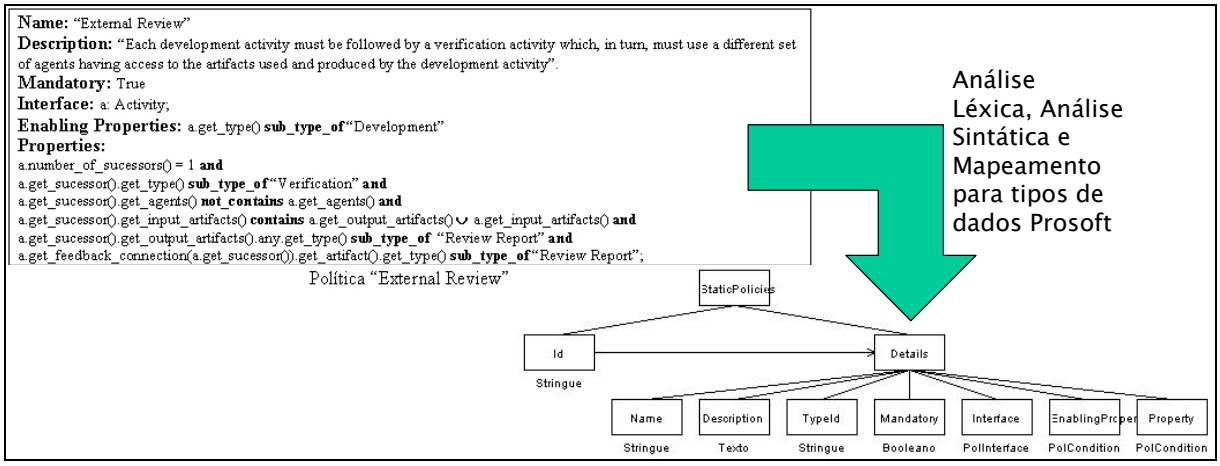

Figura 6 Esquema do mapeamento da definição de políticas para tipos de dados internos do ambiente

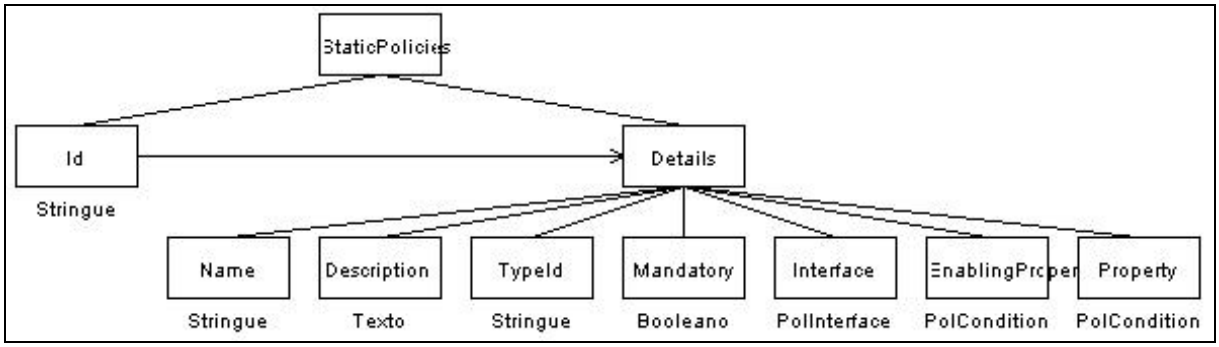

Figura 7 Classe para definição de políticas estáticas (ATO StaticPolicy)

A interface de uma política (*PolInterface*) é definida por uma tupla com o rótulo (*Label*) e a identificação do tipo APSEE correspondente (*Type*), como na figura 8.

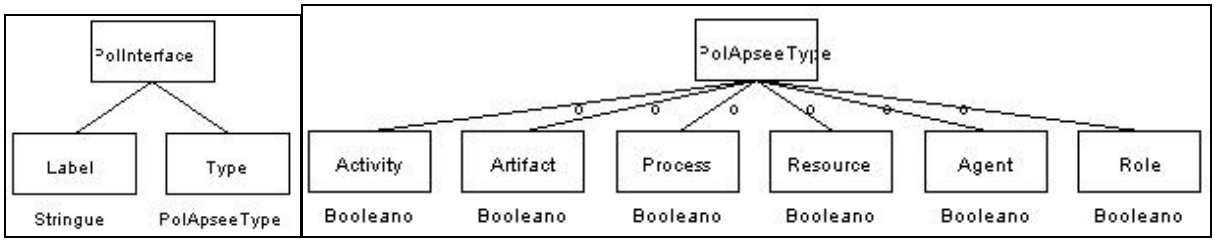

Figura 8 Classes para definição da Interface (ATO PolicyInterface) e tipo de objeto APSEE (PolApseeType)

O tipo *PolCondition* (à esquerda da figura 9) descreve as propriedades de uma política, com um Operando (objeto de *PolOperand*), uma relação opcional (*Opt_relation*), e uma conexão opcional (*Opt_conn*, que expressa a conexão *and* ou *or* com outra propriedade). Um operando de política é definido pelo tipo *PolOperand* (à direita da figura 9), com

referência a um objeto APSEE (no caso da política estática, este deve ser o item definido como interface) e uma lista ordenada de operadores (*Operator*). *ComparisonType* (figura 10) determina o tipo de comparação usado para uma relação entre dois operandos (igualdade, desigualdade, maior, maior ou igual, menor, menor ou igual, contém e não-contém).

PolicyOperator (figura 11) é especificado como uma alternativa entre uma chamada de método (*MethodCall*) ou uma palavra reservada (*Any, All* e *No*) da linguagem (*ReservedWord*).

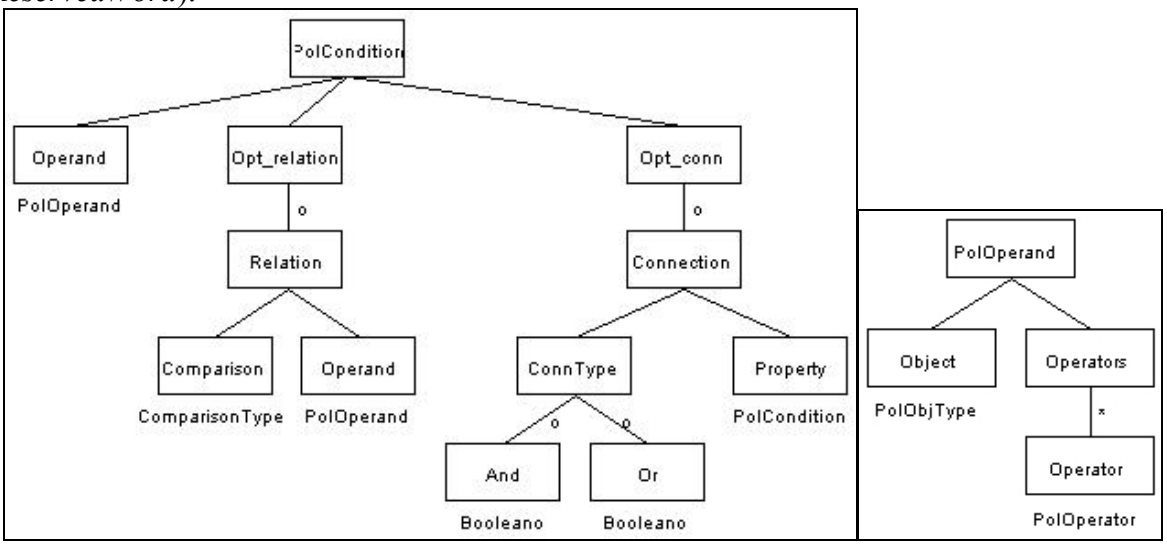

Figura 9 Classes para definição de condições e para definição de operandos

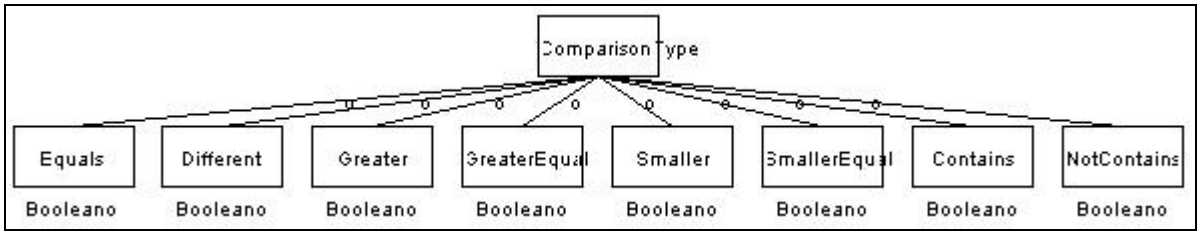

Figura 10 ATO ComparisonType

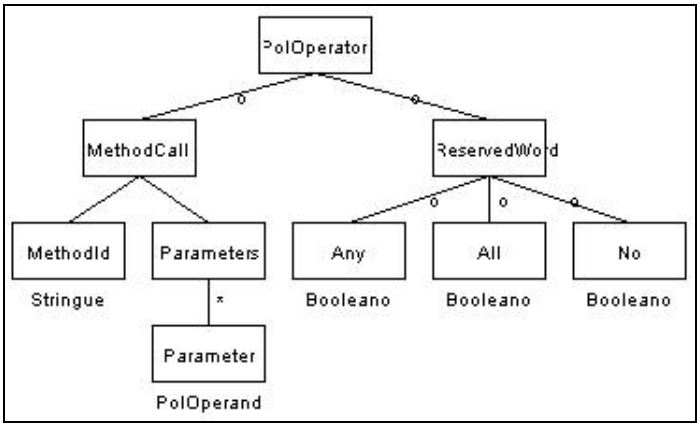

Figura 11 Classe para definição de chamada de métodos ou palavras reservadas (ATO PolOperator)

4.6. Mecanismo de interpretação de StaticPolicies

A semântica do mecanismo de interpretação de objetos da classe *StaticPolicies* foi especificada algebricamente, ocupando cerca de 40 axiomas em PROSOFT-Algébrico[1]. Por questão de espaço, esta seção apresenta somente uma visão geral do algoritmo principal do

[1] No estilo de especificação adotado neste paradigma, é utilizado um mecanismo para a troca de mensagens para os ATOs chamado ICS (Interface de Comunicação do Sistema) cuja sintaxe segue o formato: **ICS**(nome-do-ato, nome-da-função, seletor, <lista de argumentos>) [16]

mecanismo, como detalhamento da etapa "Verificação das Políticas Estáticas" da figura 1 e em [22] é apresentada a sua especificação completa, incluindo exemplos adicionais.

O procedimento de verificação de políticas para um (fragmento de) processo de software gera como resultado um objeto da classe *StaticPolEval* (figura 15 - esquerda). No campo *Log* da classe *StaticPolEval* são armazenadas ocorrências (*warnings* ou *errors*) para cada uma das atividades que compõem o fragmento de processo em avaliação: objetos da classe *ActEvalLog* (à direita da figura 15) são criados para registrar especificamente a ocorrência gerada pela avaliação de uma atividade (se houver).

A função *verify_static_policies* (figura 12) é a principal da interpretação, tratando um objeto do tipo *APSEE* (o meta-gerenciador de processos) e um *Stringue* que denota o fragmento a ser verificado, gerando como resultado um objeto do tipo *StaticPolEval*. Para tanto, é criado um objeto inicial de *StaticPolEval* (item {1}), e obtidas as políticas habilitadas em todo o processo {2}, e em toda a organização {3}. Em {4}, são obtidas todas as atividades que compõem o fragmento de processo. Todos estes objetos construídos são passados como argumentos para a função *verify_static_policies_activities*, descrita no parágrafo a seguir.

```
Definição: verify_static_policies: APSEE, Stringue → StaticPolEval
verify_static_policies(apsee, proc-id)
=       verify_static_policies_activities(apsee,
            ICS(StaticPolEval, create, proc-id, date, time),                                  {1}
            ICS(Processes, get_enabled_static_policies, select-Processes(apsee), <proc-id>) ∪  {2}
            ICS(Organization, get_enabled_static_policies, select-Organizaton(apsee)),        {3}
            ICS(Processes, get_activities, select-Processes(apsee), <proc-id>), proc-id))     {4}
```

Figura 12 Função principal do interpretador de políticas

A função *verify_static_policies_activities* (figura 13) realiza a avaliação de uma atividade específica do processo, adicionando um novo item de *Log* no objeto *staticPolEval* se tal avaliação gerar alguma ocorrência (*warning* ou *error*). Esta função se baseia em chamadas recursivas {1}, tratando cada uma das atividades que compõem o processo. O item {2} mostra a transformação que ocorre no objeto *staticPolEval* recebido: a este será adicionado o resultado da avaliação de uma atividade específica ao *Log* de ocorrências do processo. O conjunto de políticas que precisam ser verificadas (que já possuía as políticas habilitadas para a organização e para o processo {3}) é complementado com as políticas habilitadas especificamente para a atividade em questão ({4}). Finalmente, são determinadas as condições de parada para a recursão: quando não existem mais atividades a serem verificadas {5} ou quando o conjunto de políticas a serem verificadas é vazio{6}.

```
Definição:
verify_static_policies_activities: APSEE, StaticPolEval, SetOfString, SetOfString, Stringue → StaticPolEval

verify_static_policies_activities(apsee, staticPolEval, setOfPolicyIds, setOfActivityIds, proc-id)
=   verify_static_policies_activities(apsee,                                                  {1}
        ICS(StaticPolEval, include_activity_log, staticPolEval,                               {2}
            <act-id,verify_static_policies_activity(apsee, proc-id,
            ICS(SetOfString, get_first, setOfActivityIds)
            setOfPolicyIds ∪                                                                  {3}
            ICS(Processes, get_enabled_static_policies, select-Processes(apsee),              {4}
                <proc-id, act-id>))>),
            setOfPolicyIds, ICS(SetOfString, get_tail, setOfActivityIds), proc-id))

verify_static_policies_activities(_, staticPolEval, _, empty-set, _) = staticPolEval          {5}

verify_static_policies_activities(_, staticPolEval, empty-set, _, _) = staticPolEval          {6}
```

Figura 13 Função que avalia todas as atividades que compõem o fragmento de processo

O mecanismo de verificação de políticas habilitadas para uma atividade específica é determinado pela função *verify_static_policies_activity* apresentado na figura 14. Esta função verifica inicialmente se as propriedades de habilitação (pré-condições) da política na atividade em questão são satisfeitas {1}. Se as pré-condições forem atendidas, isto significa que a

política em questão será avaliada em função da atividade associada (função *evaluate_policy_activity* em {2}) e o resultado desta avaliação será agrupado com o resultado da avaliação das políticas restantes (item {3}). A condição de parada de recursão é definida pela inexistência de políticas a serem avaliadas (item {4}).

```
Definição: verify_static_policies_activity: APSEE, Stringue, Stringue, Set → ActEvalLog

verify_static_policies_activity(apsee, proc-id, act-id, add(policies, pol-id))
=       if      enabling_properties_satisfied(apsee, proc-id, act-id, pol-id)                    {1}
        then    composition(evaluate_policy_activity(apsee, proc-id, act-id, pol-id),           {2}
                                verify_static_policies_activity(apsee, proc-id, act-id, policies))  {3}
        else    verify_static_policies_activity(apsee, proc-id, act-id, policies)

verify_static_policies_activity(_, _, _, empty-set) = empty-mapping                              {4}
```

Figura 14 Função que avalia todas as políticas habilitadas para uma atividade específica

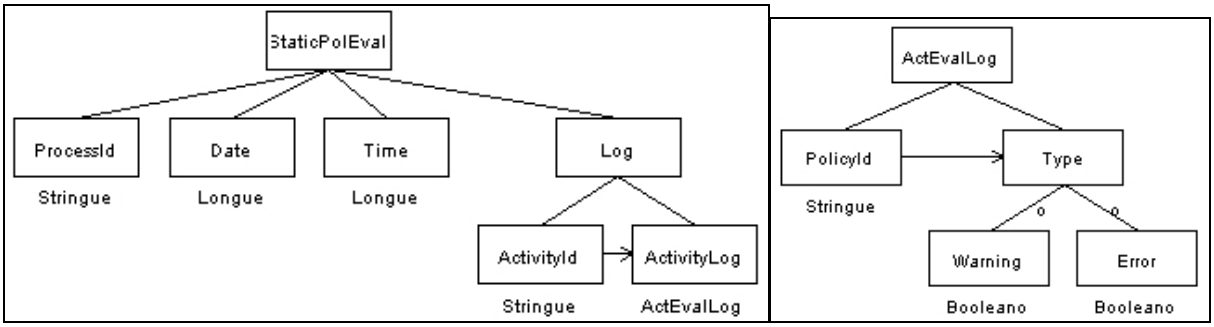

Figura 15 Registro da avaliação de um fragmento de processo de software (ATOs StaticPolEval e ActEvalLog)

4.7. Limitações do modelo e da implementação atual

A linguagem para definição de Políticas Estáticas, o mecanismo de interpretação especificado e o protótipo implementado possuem limitações importantes as quais são descritas a seguir.

- O "leque" de funções definido para a linguagem é restrito, sendo determinado pelas primitivas disponíveis de antemão no ambiente de modelagem e execução de processos;
- Não está disponível um mecanismo para prevenir a habilitação de políticas contraditórias que podem levar o modelo de processo para um estado incoerente. A detecção de inconsistências é uma questão ainda em aberto no projeto de sistemas baseados em regras [2];
- Políticas podem ser burladas, por exemplo, com o projetista alterando tipo de uma atividade somente para atender uma propriedade (sem que esta alteração seja de fato, semanticamente coerente);
- A implementação atual do protótipo desenvolvido no Java-PROSOFT do interpretador de políticas permite que políticas estáticas sejam definidas somente com interface do tipo *Activity*. Além disso, o interpretador não fornece para usuário informação exata acerca de qual propriedade falhou na avaliação de uma política específica;
- A implementação atual do interpretador de políticas é funcional somente com o ambiente APSEE e a PML adotada pelo ambiente. A interoperabilidade com outros ambientes PSEEs e de *Workflow* é uma tarefa futura do nosso grupo, para avaliar a utilização dos construtores aqui definidos em ambientes heterogêneos.

5. Trabalhos correlatos

Diversos trabalhos estão disponíveis na literatura especializada tratando de mecanismos de reutilização e aperfeiçoamento de processos de software. Descrições genéricas e reutilizáveis de processos e metodologias de desenvolvimento de software são encontradas, por exemplo, nos Padrões de Processo de Ambler [1] e nos Princípios de desenvolvimento de software de Davis [3]. Entretanto, tais descrições não possuem o compromisso de serem computáveis, o que limita sobremaneira sua adoção em ambientes automatizados como PSEEs.

Representações computáveis para Padrões de Processo tais como a proposta de Vasconcelos [24] representam um importante avanço nesta área ao adicionar maior rigor nas descrições de processos reutilizáveis, aproximando dos objetivos procurados pela automação do processo de software. O conceito *ProcessTemplate* adotado pelo ambiente APSEE segue uma abordagem similar, representando processos abstratos e fornecendo mecanismos para adaptá-los para contextos tecnológicos e organizacionais específicos.

As Políticas Estáticas descritas neste trabalho complementam os *Templates* armazenando políticas de processos genéricas e reutilizáveis, sendo especialmente úteis para controlar a adaptação (*tailoring* [6]) de *Templates* para contextos específicos. Embora o conceito de políticas de processo para determinar propriedades a serem respeitadas na modelagem e execução de processos de software não seja novo [6], a maioria dos trabalhos nesta área preocupa-se somente com a verificação de propriedades dinâmicas da execução de processos de software. Assim, diversos PSEEs e ambientes *Workflow* possuem mecanismos baseados em regras para verificar propriedades dinâmicas de processos, tais como o ambiente TrigsFlow [12]. Em Trigsflow, regras "Evento-Condição-Ação" armazenadas em bancos de dados ativos são disparadas a partir da ocorrência de eventos durante a execução de um processo (determinando o comportamento do ambiente em resposta a eventos pré-determinados).

Poucos trabalhos na literatura especializada preocupam-se com a descrição de Políticas Estáticas como mecanismo para automatizar a verificação de propriedades estáticas (sintáticas) em modelos de processos de software. Acredita-se que isto se deva ao fato de que poucos ambientes baseados em hierarquia de tipos estão disponíveis atualmente (tais como APSEE e E3 [11]). Com tais meta-modelos de processos, surge a possibilidade de se definir um modelo de processo de software com algum conhecimento semântico associado, podendo-se definir verificadores de propriedades tais como o descrito neste texto.

Dewayne E. Perry propõe em [18] e [19] um construtor para definição de *Policies* na PML Interact (associada ao ambiente Intermediate). Interact é uma linguagem baseada em regras e os seus modelos de processos consistem na definição de objetos, políticas e atividades. Políticas são definidas para estabelecer relacionamentos entre objetos e, dependendo da avaliação de expressões lógicas descritas no corpo de uma política, fatos sobre o estado de produtos ou de projetos podem ser asseridos (*assert*). Apesar de, à primeira vista, as abordagens possuírem denominações idênticas e objetivos semelhantes, APSEE-StaticPolicies e Interact-PolicyDefinitions adotam soluções diferentes, decisivamente influenciadas pelos paradigmas de modelagem adotados nas respectivas linguagens.

Finalmente, os autores acreditam que são duas as principais contribuições deste trabalho. Primeiro, a maneira como informações gerenciais são incorporadas em um modelo de processo de software é inovadora, através de uma linguagem compacta. A segunda contribuição consiste da forma como tal mecanismo foi especificado: a linguagem possui base semântica formal, permitindo que novas extensões e mecanismos de verificação formal de processos possam ser desenvolvidos no futuro. De fato, a especificação formal do

interpretador de políticas norteou a implementação de um protótipo funcional que foi integrado ao ambiente APSEE e estabelece um dos pilares básicos do mecanismo de adaptação de processos de software disponível no ambiente APSEE [21].

6. Conclusões e trabalhos futuros

Este texto apresentou uma descrição funcional das Políticas Estáticas, mecanismo incorporado no ambiente APSEE para permitir a verificação de propriedades estáticas (sintáticas) em modelos de processos de software. A experiência dos autores no uso de Políticas Estáticas [22], em especial na experimentação com exemplos extraídos de diversos textos com estratégias para gerência de projetos (tais como [1] e [3]), vem obtendo bons resultados, e indica que é possível descrever com este construtor muitas das estratégias gerenciais descritas na literatura. Políticas Estáticas exercem ainda um papel fundamental na reutilização de processos no ambiente, pois uma política habilitada no modelo original pode ser utilizada para restringir a adaptação de um processo recuperado para um contexto específico. Entretanto, a modelagem de processos de software ainda é uma atividade extremamente complexa, exigindo muita experiência dos projetistas envolvidos. Assim, uma extensão natural deste trabalho é avaliar a viabilidade de fornecer o construtor aqui descrito para utilização com diferentes PMLs disponibilizadas na literatura, permitindo que projetistas de processos experientes com outras notações possam se beneficiar do conceito proposto.

Atualmente, um protótipo de todo o ambiente APSEE vem sendo implementado sobre o ambiente Java-PROSOFT a partir das especificações construídas em PROSOFT-Algébrico, integrando as diferentes ferramentas definidas isoladamente e citadas neste texto. Além disso, um editor de políticas baseado em formulário vem sendo projetado de tal forma que a interface gráfica do ambiente forneça algum auxílio para o usuário durante a definição de Políticas Estáticas. Estudos vêm sendo realizados com o objetivo de permitir que a linguagem seja estendida com funções definidas pelo usuário. Finalmente, um grande trabalho adicional é vislumbrado, incluindo a investigação da viabilidade do uso de Políticas Estáticas em ambientes *Workflow* e a extensão da linguagem para permitir maior flexibilidade no seu processo de desenvolvimento.

7. Agradecimentos

Este trabalho obteve apoio parcial do programa PICDT-CAPES da Universidade Federal do Pará (para Rodrigo Quites Reis e Carla Alessandra Lima Reis), e do CNPq (para Daltro José Nunes).

8. Referências

[1] AMBLER, S.W. **Process Patterns:** Building Large-Scale Systems Using Object Technology, SIGS/Cambridge, 1998.

[2] BERTINO E.; FERRARI, E. The specification and enforcement of authorization constraints in *Workflow* management systems. **ACM Transactions on Information and System Security.** v. 2 , n. 1, 1999. p. 65-104

[3] DAVIS, A.M. **201 Principles of Software Development.** New York: McGraw-Hill, 1995.

[4] DERNIAME, J.; KABA, B.; WASTELL, D. (eds.) Software Process: Principles, Methodology and Technology. Lecture Notes in Computer Science, vol. 1500. Springer-Verlag, 1998.

[5] EMAN, K.; DROUIN, J.N.; MELO, W. (eds.) **SPICE**: The theory and practice of software process improvement and capability determination. IEEE, 1998.

[6] FEILER, P.; HUMPHREY, W. Software Process Development and Enactment: Concepts and Definitions. In: Int'l. Conference on Software Process, 2. (ICSP2). **Proceedings…** Berlin, Germany. Feb, 1993.

[7] FRANCH, X.; RIBÓ, J. Some Reflexions in Modeling of Software Process. Int'l. Process Techonology Workshop, 1. (IPTW'99). 1999 Villars de Lans (France). **Proceedings...** 1999.

[8] FUNK, P.J.; CRNKOVIC, I. Reuse, Validation and Verification of System Development Processes. In: Int'l Workshop on the Requirements Engineering Processes (DEXA'99). **Proceedings...** Florence, Sept. 1999.

[9] GIMENES, I.M. **Uma Introdução ao Processo de Engenharia de Software:** Ambientes e Formalismos. Trabalho apresentado na Jornada de Atualização em Informática, 13., Caxambu: SBC, 1994.

[10] HERBSLEB, J.; MOITRA, D. Global Software Development. **IEEE Software**, March/April 2001.

[11] JACCHERI, M.L.; PICCO, G.P.; LAGO, P. **Eliciting Software Process Models with the E3 language.** Politécnico de Torino, Itália, 1998.

[12] KAPPEL, G.; RAUSCH-SCHOTT, S.; RETSCHITZEGGER, W. **Coordination in *Workflow* Management Systems – A Rule-Based Approach**. In: CONEN, W.; NEUMANN, G. (Eds.) Coordination Technology for Collaborative Applications: Organizations, Processes and Agents. Lecture Notes in Computer Science, vol. 1364, Springer-Verlag, 1998.

[13] KRANSER, H. et al. Lessons learned from a Software Process Modeling System. **Communications of the ACM**, v. 35, n.9, Sept. 1992.

[14] LIMA REIS, C.A.; REIS, R.Q.; NUNES, D.J. Gerenciamento do Processo de Desenvolvimento Cooperativo de Software no Ambiente PROSOFT. In: Simpósio Brasileiro de Engenharia de Software, 12. (SBES98). **Anais...** Maringá: SBC. 1998.

[15] NUNES, D.J. Estratégia Data-driven no Desenvolvimento de Software. In: Simpósio Brasileiro de Engenharia de Software, 6. **Anais...** p. 81-95, Gramado: SBC, 1992.

[16] NUNES, D.J. **PROSOFT:** Um Ambiente de Desenvolvimento de Software Baseado no Método Algébrico. Porto Alegre: UFRGS, 1995. (http://www.inf.ufrgs.br/~prosoft)

[17] PAULK, M.; WEBER, C.; CURTIS, B. **The Capability Maturity Model:** Guidelines for Improving the Software Process. Addison-Wesley Publishing Co., 1994.

[18] PERRY, D.E. Policy-Directed Coordination and Cooperation. Int'l. In: Software Process Workshop, 7. (ISPW'7) **Proceedings…** Yountville, California, 1991.

[19] PERRY, D.E. Using Process Modeling for Process Understanding. In: Software Process Improvement. **Proceedings…** Barcelona, Dec. 1997.

[20] PRESSMAN, R.S. Software Engineering. PRESSMAN, R.S. **Software Engineering:** A Practitioner's Approach. McGraw-Hill, 2001.

[21] REIS, R.Q.; LIMA REIS, C.A.; NUNES, D.J. Automated Support for Software Process Reuse: Requirements and Early Experiences with the APSEE model. In: International Workshop on Groupware, 7. (CRIWG'2001). **Proceedings…** IEEE Computer Society. Darmstadt, Alemanha. Sept. 2001.

[22] REIS, R.Q.; NUNES, D.J. **APSEE-StaticPolicy**: Sintaxe, semântica algébrica e exemplos de uma linguagem para verificação automática de políticas estáticas em modelos de processos de software. Porto Alegre: PPGC-UFRGS. Relatório de Pesquisa nº 311. Jul. 2001. (http://www.inf.ufrgs.br/~prosoft)

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

[24] VASCONCELOS JUNIOR, F.M.; WERNER, C.M.L. Suporte dos Padrões à Evolução de Processos de Desenvolvimento de Software. In: Simpósio Brasileiro de Engenharia de Software, 11., **Anais...** Fortaleza: SBC. p.131-146, 1997.