Gerenciamento da Integração de Processos de Software no APSEE-Integrate

Ana Vitoria Piaggio de Freitas, Anderson Baia Maia, Daltro José Nunes
Universidade Federal do Rio Grande do Sul, Instituto de Informática
Porto Alegre, Brasil, 91501-970
{avpfreitas, abmaia, daltro}@inf.ufrgs.br

Abstract
Software processes can be formally defined through process models, and executed by PSEEs - Process-Centered Software Engineering Environments. When the software development involves autonomous organizations, it is undesirable to use an unique process model to reflect the whole scope of the software process. PSEEs should provide infrastructure for processes that involve teams dispersed geographically. Such processes are usually quite extensive, consisting of several sub-processes, that can be defined in different modelling processes notations and executed in different PSEEs. PSEEs should be capable to interact, allowing interoperability among process models, in the modelling and execution levels. The goal of this article is to present the approach of the APSEE-Integrate environment for software processes integration management, in the modelling and execution levels. This approach brings, as main contributions, the flexibility during execution allied to the automated support to the integration of process models. The components of the environment are specified formally through Graph Grammars. This article discusses the components directly related to the management of processes integration models, that are: the processes interaction modeling language and the execution mechanism.

Keywords: Software Processes, Process-Centered Software Engineering Environments, Software Processes Integration.

Resumo
Processos de software podem ser definidos formalmente através de modelos de processo, e executados por ambientes de engenharia de software centrados no processo (PSEEs - *Process-Centered Software Engineering Environments*). Quando o desenvolvimento de software envolve organizações autônomas, é inviável utilizar um único modelo de processo para refletir todo o escopo do processo de software. PSEEs devem prover infra-estrutura para processos que envolvem equipes dispersas geograficamente. Tais processos são geralmente bastante extensos, consistindo de vários sub-processos, que podem ser definidos em diferentes notações de modelagem de processos e executados em diferentes PSEEs. PSEEs devem ser capazes de interagir, permitindo interoperabilidade entre modelos de processo, nos níveis de modelagem e de execução. Este artigo tem como objetivo apresentar a abordagem do ambiente APSEE-Integrate para gerência da integração de processos de software, nos níveis de modelagem e execução. Esta abordagem traz, como contribuições principais, a flexibilidade durante a execução aliada ao suporte automatizado à integração de modelos de processo. Os componentes do ambiente são especificados formalmente através de Gramáticas de Grafos. Este artigo discute os componentes diretamente relacionados à gerência de modelos de integração entre processos, que são: a linguagem de modelagem de interações entre processos e o mecanismo de execução.

Palavras-Chave: Processos de Software, Ambientes de Engenharia de Software Centrados no Processo, Integração de Processos de Software.

1. INTRODUÇÃO

O processo de desenvolvimento de software (ou simplesmente processo de software) pode ser compreendido como o conjunto de atividades a serem realizadas desde a concepção até a implantação do produto de software. Processos de software bem definidos reduzem o tempo de desenvolvimento e os custos de produção, além de possibilitarem melhor acompanhamento do desenvolvimento de software [5]. A experiência tem mostrado que processos de software têm grande influência na qualidade dos produtos de software; controlando-se o processo de software pode-se atingir melhor controle sobre a qualidade do produto final.

Processos de software podem ser definidos formalmente através de modelos de processo. O modelo de processo define a seqüência de atividades que devem ser realizadas, as ferramentas a serem utilizadas, os produtos e documentos que serão criados e/ou manipulados e os papéis das pessoas envolvidas no desenvolvimento de software [9]. Se notações formais forem utilizadas para a modelagem do processo, é possível automatizar partes do processo, suportar a interação e a cooperação entre os desenvolvedores e guiá-los durante a execução do processo. Os formalismos utilizados para descrever modelos de processo são chamados de linguagens de modelagem de processos.

A modelagem e a execução de processos de software é suportada por PSEEs (*Process-Centered Software Engineering Environment*s), que são ambientes de desenvolvimento de software orientados ao processo. A utilização de PSEEs traz benefícios, como melhor comunicação entre as pessoas envolvidas no desenvolvimento de software, realização de algumas atividades de forma automática, coleta de métricas, suporte à reutilização de modelos de processo e disponibilidade de informações sobre o andamento do processo de software [12]. Cada PSEE é caracterizado pela sua linguagem de modelagem, que deve suportar a descrição dos vários conceitos que caracterizam o processo de desenvolvimento de software, como [5]: atividades, artefatos de software, papéis, agentes humanos, recursos e ferramentas.

Devido à globalização da economia e ao rápido crescimento da Internet, cada vez mais processos de software se expandem por múltiplas organizações. Quando o desenvolvimento de software envolve organizações autônomas, é inviável utilizar um único modelo de processo centralizado para refletir todo o escopo do processo de software [13]. Segundo [9], PSEEs devem prover infra-estrutura para processos que envolvem equipes dispersas geograficamente, possuindo práticas de desenvolvimento distintas, usando diferentes ferramentas e linguagens de programação. Tais processos são geralmente bastante extensos, consistindo de vários sub-processos, que podem ser definidos em diferentes notações de modelagem de processos e executados em diferentes PSEEs. Embora os sub-processos sejam autônomos, eles precisam interagir. Atividades modeladas em um processo podem depender do estado de execução das atividades definidas em outros modelos de processo. PSEEs devem ser capazes de interagir, permitindo interoperabilidade entre modelos de processo, nos níveis de modelagem (integração entre modelos de processo especificados através de diferentes formalismos) e de execução (integração na execução realizada por diferentes PSEEs) [8].

Este artigo tem como objetivo apresentar a abordagem do ambiente APSEE-Integrate para gerência da integração entre processos de software, nos níveis de modelagem e execução. Esta abordagem traz, como contribuições principais, a flexibilidade durante a execução aliada ao suporte automatizado à integração de modelos de processo. O APSEE-Integrate é baseado em um meta-modelo, que especifica as informações dos modelos de processo envolvidos na integração, e em uma arquitetura que provê serviços para integração de PSEEs, com base no meta-modelo. É proposta uma linguagem visual de modelagem, um mecanismo para execução das relações de dependência entre processos e mecanismos para tradução de formalismos de modelagem dos PSEEs para o formalismo do APSEE-Integrate. A linguagem de modelagem utilizada no ambiente serve como base para o mecanismo de execução proposto. São utilizados métodos formais para especificação dos componentes do ambiente, sendo que serão destacados neste artigo apenas os componentes diretamente relacionados à gerência de modelos de integração entre processos, que são: a linguagem de modelagem de interações entre processos e o mecanismo de execução. O uso de Gramáticas de Grafos para especificação formal será discutido.

A seção 2 apresenta o APSEE-Integrate detalhando os componentes relacionados à modelagem e à execução de modelos de integração entre PSEEs. A seção 3 especifica formalmente a linguagem de modelagem e o mecanismo de execução. A seção 4 apresenta trabalhos relacionados encontrados na literatura. Conclusões são discutidas na seção 5.

2. APSEE-INTEGRATE

Esta seção apresenta a proposta para interoperabilidade entre PSEEs heterogêneos do APSEE-Integrate. Será apresentada uma visão geral da arquitetura do APSEE-Integrate, e em seguida o meta-modelo, a linguagem de modelagem e o mecanismo de execução serão detalhados informalmente.

2.1. Visão Geral

O objetivo principal do APSEE-Integrate é propor uma arquitetura para interoperabilidade de PSEEs heterogêneos. A arquitetura deve permitir comunicação e sincronização entre diferentes PSEEs, com comportamento consistente. O foco do trabalho é permitir que atividades pertencentes a diferentes modelos de processo, executando em PSEEs distintos, sejam interligadas através da definição de dependências entre as mesmas. Uma visão geral do APSEE-Integrate é apresentada na Figura 1, ilustrando as três camadas principais: meta-modelo, mecanismos para gerência da integração e tradução de formalismos.

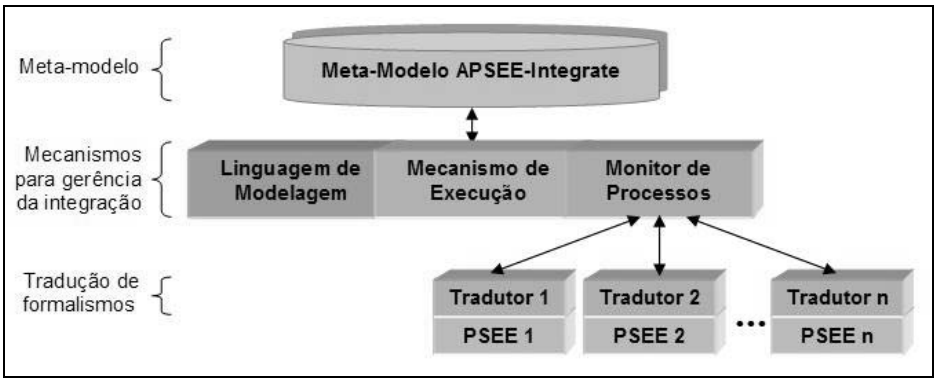

Figura 1 - Visão Geral do APSEE-Integrate

Uma breve descrição dos componentes do modelo é apresentada a seguir:

- **Meta-modelo**: define um formalismo comum, através do qual os modelos de processo poderão interagir.
- **Linguagem de modelagem**: linguagem para modelagem das dependências existentes entre atividades pertencentes a diferentes PSEEs.
- **Mecanismo de execução**: coordena a ativação das dependências entre atividades através da interpretação da linguagem de modelagem.
- **Monitor de processos**: monitora as alterações realizadas nos modelos de processo envolvidos na integração.
- **Tradutores**: são responsáveis pelo mapeamento dos formalismos de modelagem dos PSEEs para o meta-modelo do APSEE-Integrate. A cada formalismo de PSEE envolvido na integração está associado um tradutor específico, permitindo que o modelo de processo em execução no PSEE seja representado de acordo com o meta-modelo do APSEE-Integrate.

2.2. Meta-Modelo

Nesta seção serão apresentados os componentes do meta-modelo e seus relacionamentos através da notação UML. Esta notação é utilizada por ser largamente conhecida e facilitar o entendimento do modelo em alto nível de abstração.

O meta-modelo do APSEE-Integrate é baseado fortemente no meta-modelo de processos do APSEE [12]. Tal escolha foi realizada pelo fato de que a representação de dependências entre atividades do APSEE é bastante completa, permitindo expressar diversos tipos de dependências encontradas em processos reais. A Figura 2 mostra o diagrama de classes do meta-modelo APSEE-Integrate

A classe *Process* tem como objetivo representar os processos de software envolvidos na integração. Um processo da classe *Process* possui um identificador, um nome e um estado. O estado do processo pode ser: *Not_Started* (o processo ainda não foi iniciado), *Enacting* (o processo está em execução) ou *Finished* (o processo foi concluído).

Atividades são representadas através da classe *Activity*. Cada atividade possui identificador, nome, datas de início e término da execução, além de indicação de estado. O estado é o atributo que armazena a situação de cada atividade e serve de referência para a habilitação das dependências entre atividades. Os possíveis estados de uma atividade são descritos a seguir:

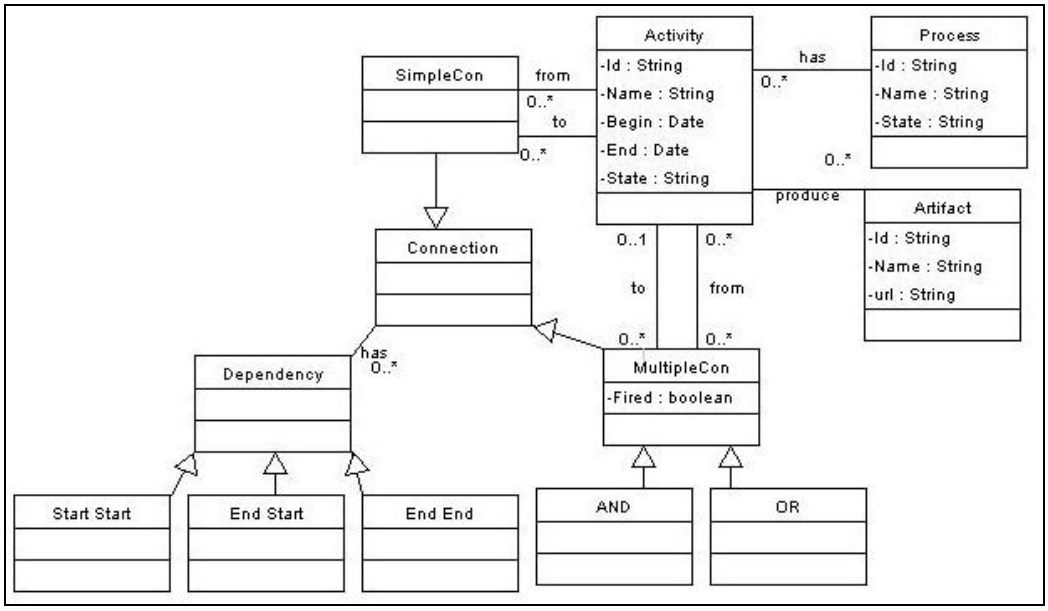

Figura 2 - Meta-modelo APSEE-Integrate apresentado através de um diagrama de classes UML

- *Waiting*: as pré-condições para execução da atividade definidas no modelo de processo no qual ela foi modelada ainda não estão satisfeitas;
- *Ready:* a atividade está pronta para começar;
- *Active:* a atividade está sendo realizada;
- *Finished:* a atividade foi concluída;
- *Canceled:* a atividade foi cancelada antes de iniciar;
- *Failed:* a atividade falhou após o inicio.

Atividades podem produzir artefatos de saída. A cada atividade podem estar associados artefatos, com a indicação da *url* (*unique resource location*) onde estão armazenados. O objetivo da definição de artefatos no APSEE-Integrate é permitir que os artefatos resultantes da execução de uma atividade estejam disponíveis para o PSEE onde ela foi modelada.

As conexões interligam atividades modeladas em diferentes processos, representando o fluxo de controle entre elas. São propostos fluxos de controle simples (com uma atividade origem e uma atividade destino) e múltiplos (envolvendo várias atividades origem e uma atividade destino).

As conexões possuem um tipo de dependência (*end-start, start-start, end-end*) que influencia diretamente na execução de atividades e pode representar diferentes dependências encontradas em processos reais. Conexões múltiplas podem ser combinadas com operadores lógicos (AND, OR). Os tipos de conexão e de dependência são detalhados a seguir:

- Conexão simples *(SimpleCon)*: define o fluxo de controle entre duas atividades (origem e destino). Este tipo de conexão indica que a atividade destino depende da atividade origem. Os tipos de dependência entre atividades podem ser:
  a) *End-Start*: a atividade destino será iniciada quando a origem terminar;
  b) *Start-Start*: a atividade destino será iniciada quando a origem for iniciada;
  c) *End-End*: a atividade destino será finalizada quando a origem terminar.

- Conexão múltipla (*MultipleCon*): possui várias atividades ou conexões múltiplas como origem e uma atividade ou conexão múltipla como destino. A semântica da conexão depende do operador lógico associado (AND – todas, OR – pelo menos uma) e do tipo de dependência utilizado (*end-start, start-start, end-end*). Uma aplicação desta conexão seria que assim que todas as atividades terminarem, então uma atividade destino pode começar (operador lógico AND e tipo de dependência *end-start*). Conexões múltiplas possuem o atributo *Fired*, que indica se a conexão já foi ou não disparada.

2.3. Linguagem de Modelagem

A modelagem da integração entre PSEEs requer formalismos de alto nível para descrever aspectos de coordenação entre atividades. Estes formalismos devem ser convenientes não somente para a representação, mas também para a execução. Com base nisso é proposto um formalismo de modelagem para o APSEE-Integrate que permite representação gráfica das dependências entre atividades com base no meta-modelo mostrado anteriormente.

Nesta seção será apresentada uma descrição informal da notação da linguagem e exemplos de seu uso. Os símbolos visuais da linguagem de modelagem são ilustrados na Figura 3.

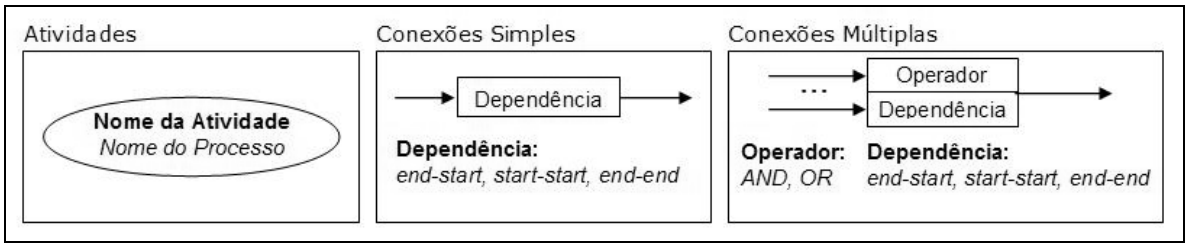

Figura 3 – Símbolos visuais da linguagem de modelagem

A definição de atividades é feita de forma automática, a partir das atividades existentes nos modelos de processo que participam da integração. Não podem ser inseridas novas atividades no APSEE-Integrate; pode-se apenas relacionar, através de conexões, as atividades existentes nos modelos de processo.

As conexões relacionam atividades modeladas em diferentes processos. O APSEE-Integrate, com base nas conexões definidas, inicia ou finaliza a execução de atividades. A conexão simples tem inicio e fim em atividades diferentes, e o nodo que a representa é um retângulo contendo o tipo de dependência escolhido (*end-start, start-start ou end-end*). Uma restrição importante é que a definição de uma conexão simples não pode inserir ciclos nas dependências entre processos, ou seja, não pode haver um fluxo de controle no sentido contrário ao sentido da conexão simples entre as duas atividades envolvidas.

A representação de conexões múltiplas é feita através de um retângulo dividido, onde a parte superior indica o operador lógico associado (AND, OR), enquanto a parte inferior indica o tipo de dependência (*end-start, start-start ou end-end*). Por definição, uma conexão múltipla pode ter vários antecessores e somente um sucessor. Os antecessores e sucessores de conexões múltiplas podem ser atividades ou conexões múltiplas. Portanto, o fluxo de controle entre processos pode ser definido mais detalhadamente através da combinação de várias conexões múltiplas encadeadas.

Um exemplo de definição de conexões entre atividades é ilustrado na Figura 4. Neste exemplo, a *Atividade C*, modelada no *Processo 1*, será iniciada após o término (dependência *end-start*) da *Atividade A*, modelada no *Processo 2*, e da *Atividade B*, definida no *Processo 3* (operador lógico AND), ou após o término da *Atividade E*, definida no *Processo 3* (operador lógico OR). Quando a *Atividade D*, definida no *Processo 2*, for finalizada, a *Atividade C* também será concluída (dependência *end-end*).

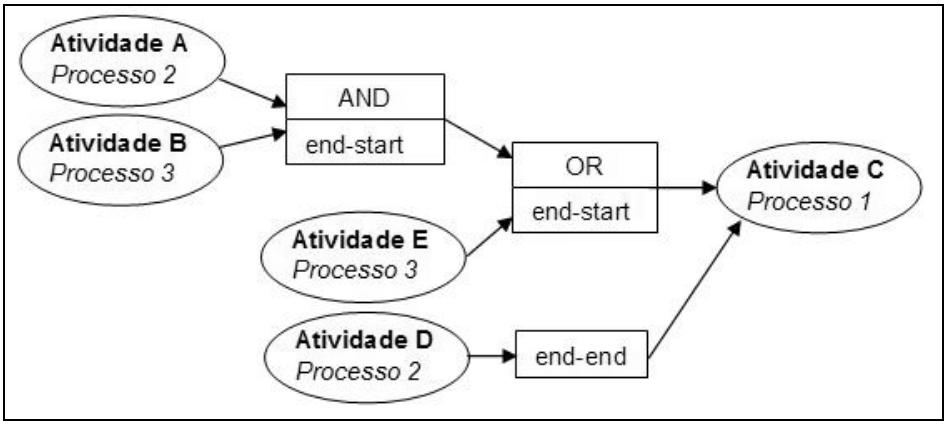

Figura 4 - Exemplo de definição de conexões entre atividades

2.4. Mecanismo de Execução

Durante a execução do modelo de integração, atividades são iniciadas e terminadas, de acordo com as conexões existentes e com o estado das atividades em execução nos diferentes PSEEs. O Mecanismo de Execução interpreta o modelo de integração e gerencia o estado de execução das atividades, com base nos modelos de processo distribuídos. A execução do modelo de integração tem como principais características:

- Permitir a execução de modelos de integração incompletos e modificação dinâmica: o modelo de integração pode continuar sendo modelado enquanto algumas conexões já foram ativadas. A modelagem também pode ocorrer em partes do modelo que já iniciaram. Nesse caso, mudanças dinâmicas são permitidas, desde que não afetem a consistência do modelo de integração. A criação de um meta-modelo unificado e a construção de um mecanismo de execução que interage com as fases de modelagem e execução possibilita essa característica;

- Monitorar as alterações realizadas nos modelos de processo distribuídos, para garantir a integridade das conexões existentes entre os modelos. A exclusão de uma atividade pode implicar na alteração de conexões;

- O mecanismo de execução fornece a semântica da linguagem de modelagem. Essa semântica é definida formalmente através de Gramática de Grafos. A especificação formal da semântica da linguagem de modelagem permite tratar o problema em alto nível de abstração, provê uma base para rastreamento e análise de impacto de alternativas, e define as conseqüências das transições de estado de forma resumida e simplificada. Essa semântica será apresentada na seção 3;

- Permitir visualização da execução através do formalismo gráfico e executável para modelagem de conexões entre processos. O mesmo formalismo é usado para modelar as conexões e acompanhar sua evolução.

A execução verifica continuamente o estado de execução dos modelos de processo, de modo a tratar as alterações no modelo e a evolução do estado de execução. A seguir é ilustrado o comportamento da execução em virtude dessas modificações para atingir os objetivos citados do APSEE-Integrate. Deve-se observar que, para uma compreensão geral da execução, é necessário levar em consideração a semântica das conexões entre atividades.

A Figura 5 apresenta um diagrama de transição para os possíveis estados de uma atividade, sendo que os estados e seus significados foram apresentados na seção 2.2. Os estados *Waiting* e *Canceled* não foram incluídos no diagrama porque eles são determinados apenas pelo modelo de processo no qual a atividade foi modelada. O APSEE-Integrate não define transições com origem ou destino nestes estados. Eles refletem a evolução da execução dos modelos de processo, e são utilizados pelo Mecanismo de Execução para garantir consistência nas conexões entre atividades.

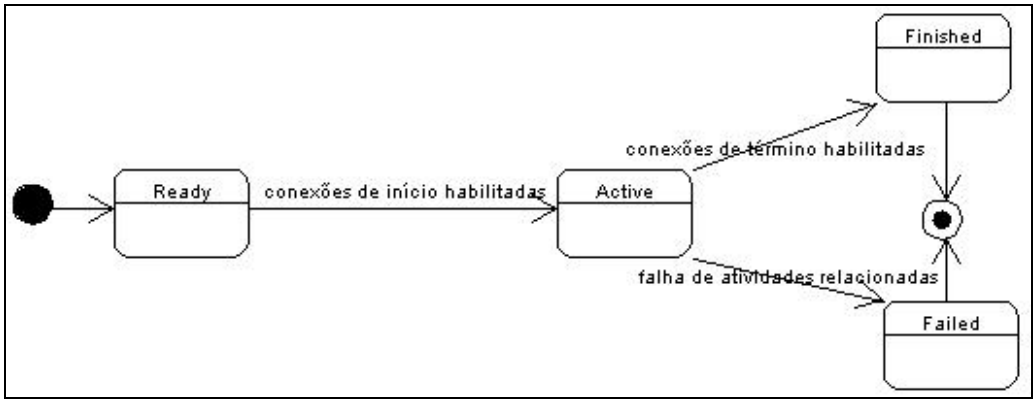

Figura 5 - Transição de estados de uma atividade

O mecanismo de execução verifica as mudanças de estado ocorridas nos modelos de processo, e dispara transições quando necessário. Quando todas as conexões relacionadas ao inicio de uma atividade (dependências *end-start* e *start-start*) forem habilitadas, então o estado de execução da atividade será modificado para *Active*. Já a habilitação das conexões de término (dependência *start-end*) implica na alteração do estado de execução da atividade para *Finished*.

As atividades de um modelo de processo podem falhar durante sua execução. Quando uma atividade falha e existem conexões nas quais ela é origem, as atividades destino irão falhar somente se dependerem do fim da atividade que falhou. Daí, conexões *start-start* não propagam falhas. A falha é propagada para todas as sucessoras que dependem da atividade, inclusive as que ainda não iniciaram. Conexões múltiplas OR só propagam falhas caso todas as atividades origem tenham falhado.

3. SEMÂNTICA FORMAL DO APSEE-INTEGRATE

A seção anterior descreveu informalmente o meta-modelo, a linguagem de modelagem e o mecanismo de execução propostos para o APSEE-Integrate. Esta seção utiliza Gramática de Grafos para apresentar a sintaxe da linguagem de modelagem e a semântica do mecanismo de execução. A definição formal do meta-modelo não será apresentada neste artigo. Os componentes do meta-modelo já foram apresentados na seção 2.2 através de diagrama de classes UML.

A partir das características do problema, foi realizada uma avaliação de métodos mais adequados para especificação. Considerando que o trabalho está inserido no grupo de pesquisa PROSOFT, coordenado pelo Prof. Dr. Daltro José Nunes, que propõe um ambiente de desenvolvimento formal de software de mesmo nome, foi avaliada a definição das propostas para funcionamento integrado ao ambiente.

A evolução do ambiente PROSOFT está intimamente ligada com o objetivo de construir ferramentas que apóiem o uso de métodos formais no ciclo de vida do software. Além disso, o ambiente é baseado em objetos e distribuído, o que facilita a integração de ferramentas que necessitam desse recurso. No PROSOFT, as especificações formais constituem uma base sólida que guia a implementação de protótipos ou sistemas de software completos. Nesse ambiente as ferramentas são construídas utilizando um paradigma algébrico, e há uma correspondência entre os componentes especificados algebricamente e a sua implementação. Assim, o formalismo algébrico mostra-se adequado para especificar os tipos de dados e operações básicas do meta-modelo aqui proposto.

Para a especificação da linguagem de modelagem visual e do mecanismo de execução foi escolhida a abordagem de Gramática de Grafos. A escolha foi inspirada no trabalho proposto por Lima Reis [12], onde se utiliza esse formalismo para especificar a sintaxe e a semântica da linguagem visual e do mecanismo de execução do APSEE. O fato de Gramáticas de Grafos serem formais e intuitivas ao mesmo tempo, e de poderem tratar com simplicidade aspectos de concorrência e distribuição de sistemas influenciou na sua escolha para complementar a especificação da semântica da execução de processos.

Gramáticas de grafos baseiam-se no processo de transformação que um grafo pode sofrer em função de um conjunto de regras previamente definidas. Um sistema é especificado em termos de estados, que são modelados por grafos, e de mudanças de estados, modeladas por regras ou derivações. A aplicação de uma regra a um grafo G é chamada passo de derivação, e isso só é possível se existe uma ocorrência do lado esquerdo (L) da regra no grafo atual G. O lado direito (R) da regra define o grafo resultante da aplicação desta regra. A interpretação de uma regra r:L → R é feita da seguinte forma:

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

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

3.1. Sintaxe da Linguagem de Modelagem

Para desenvolver uma linguagem visual é necessária inicialmente uma especificação desta linguagem. Devido ao caráter multidimensional de linguagens visuais, suas especificações textuais são normalmente difíceis de escrever e entender. Por isso, especificações visuais de linguagens visuais têm sido cada vez mais utilizadas. Uma técnica recente para especificar linguagens visuais foi proposta por Bardohl em [1] e [2]. Nesta abordagem, chamada GENGED, uma linguagem visual é especificada por um alfabeto e uma gramática. De acordo com Bardohl, o uso de gramáticas de grafos provê um formalismo mais natural e visual para a especificação de linguagens visuais.

Segundo Bardohl em [2], existem dois níveis de descrição de linguagens visuais: o nível de sintaxe abstrata, que descreve o significado lógico das sentenças e o nível de sintaxe concreta, que é usado para descrever o layout das sentenças. A combinação dos dois níveis é chamada Sintaxe Visual. A sintaxe abstrata geralmente corresponde a um grafo tipo onde os vértices são os elementos do alfabeto da linguagem e os arcos denotam o relacionamento entre eles. Para complementar a sintaxe visual, devem ser definidas regras para especificar como são construídas as sentenças visuais e, quando necessário, sua semântica.

A especificação da linguagem de modelagem do APSEE-Integrate foi inspirada na abordagem GENGED [2]. O grafo-tipo da linguagem de modelagem forma um diagrama que conecta vários componentes da linguagem. A Figura 6 mostra o diagrama construído para definir a sintaxe da linguagem Os símbolos do visuais do grafo-tipo são os mesmos da linguagem de modelagem, apresentados na seção 2.3.

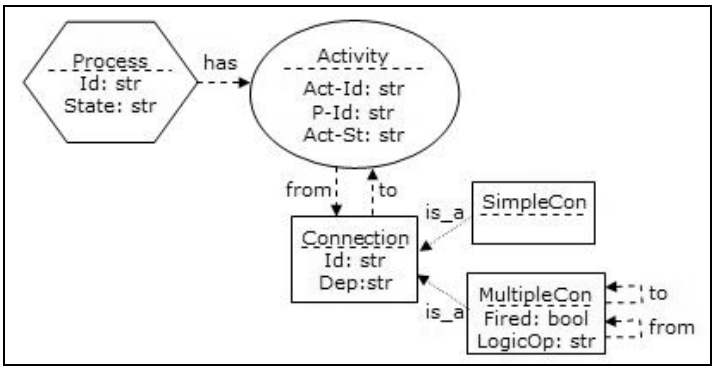

Figura 6- Grafo-tipo da linguagem de modelagem

Cada símbolo está associado a um nome e um conjunto de atributos (separados por uma linha do nome do símbolo). Atributos são elementos tipados usados para representar o estado do processo. As setas do diagrama representam relacionamentos, e correspondem aos arcos do grafo.

No grafo-tipo, um processo de software é composto por atividades. Os arcos *from* e *to* descrevem a origem e o destino das conexões. O relacionamento *is_a* pode ser compreendido como herança, similar ao uso de herança em linguagens de programação orientadas a objetos. O relacionamento *is-a* simplifica bastante o grafo-tipo e as regras de transformação correspondentes, já que atributos e relacionamentos são herdados aos subtipos. Conexões possuem o atributo *dep*, que representa o tipo de dependência (*end-start*, *start-start*, *end-end*). Conexões múltiplas possuem ainda indicação do operador lógico (AND ou OR), através do atributo *LogicOp*.

Após a definição do grafo-tipo, é necessário declarar como criar sentenças válidas para a linguagem proposta. Na abordagem GENGED [2] é proposta a criação de uma gramática visual para gerar as sentenças da linguagem e um conjunto de regras de comportamento para definir as transformações que modificam o sistema e os estados do modelo. A gramática visual da linguagem do APSEE-Integrate é responsável por gerar instâncias de modelos de integração entre processos e também garantir sua consistência.

Foram descritas regras para a gramática que gera sentenças da linguagem. Em muitos casos, foram usadas condições negativas de aplicação (NAC – *Negative Application Conditions*) para descrever condições que devem ser negativas para que a regra seja habilitada. Nesse caso as partes cortadas no lado esquerdo da regra formam as condições negativas e não devem estar presentes na instância do grafo para que a regra seja aplicada.

A Figura 7 mostra um exemplo de regra para inserir uma conexão simples entre duas atividades. Na figura, o lado esquerdo da regra contém uma situação onde existem duas atividades, *act_id1* e *act_id2*, pertencentes a processos distintos (*p_id1* e *p_id2*), e acima da figura é mostrada uma chamada que corresponde à solicitação do usuário. A regra da figura é disparada se o usuário solicita a inclusão de uma conexão simples, com uma determinada dependência, entre duas atividades existentes, *act_id1* e *act_id2*. Para prevenir a inclusão de ciclos - que causariam *deadlock* – uma NAC define que a atividade destino não pode ter um fluxo de controle direto ou indireto para a atividade origem, enquanto uma NAC adicional assume que não existe nenhuma conexão de controle entre a origem e o destino. Como resultado, se a regra for aplicada, é acrescentada uma conexão simples entre as atividades. Vale notar que esta regra se aplica caso a atividade origem não tenha sido cancelada ou falhada (não se devem definir conexões entre atividades nesses estados). A atividade destino deve estar no estado *waiting* ou *ready*, ou seja, não pode ter sido iniciada. Esta condição é necessária porque caso a

atividade origem ainda não tenha sido iniciada, se a atividade destino estiver executando (estado *active*), a dependência da conexão terá sido violada.

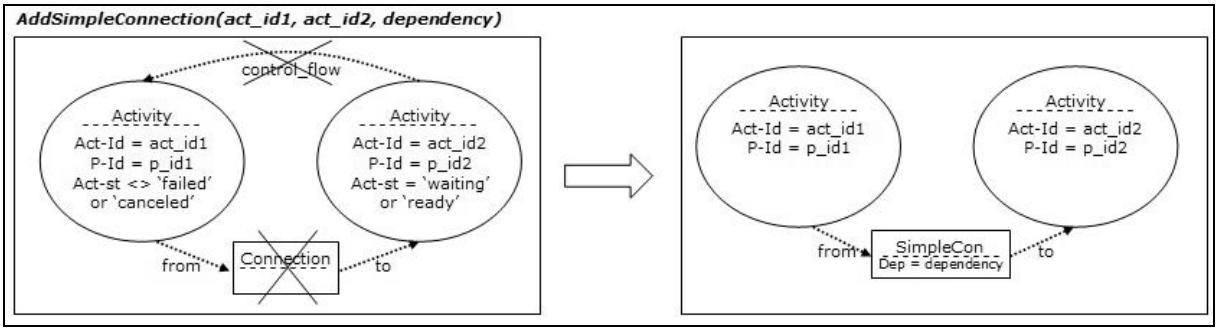

Figura 7 - Regra para inclusão de uma conexão simples entre atividades

A Figura 8 ilustra uma regra para definir o destino de uma conexão múltipla. A regra indica que, dadas uma atividade e uma conexão múltipla, a atividade deve ser definida como destino da conexão. Para isto, é necessário que a atividade não possua conexão de destino pré-existente com a conexão; não exista fluxo de controle da atividade para a conexão (para evitar ciclos) e nenhuma atividade ou conexão tenha sido definida anteriormente como destino da conexão. Além disso, o estado da atividade é determinado do lado esquerdo da regra.

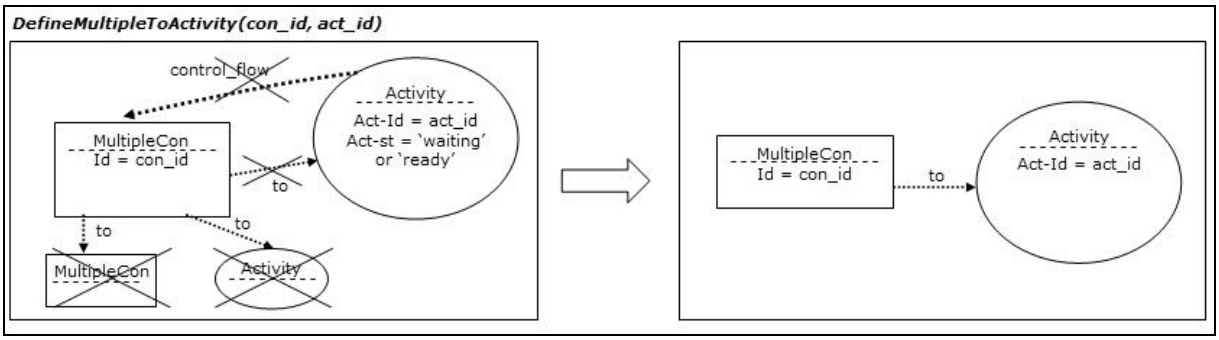

Figura 8 - Regra para definir uma atividade como destino de uma conexão múltipla

Na Figura 9 é apresentada uma regra para definir uma conexão múltipla (*FromCon_id*) como destino de outra (*con_id*). Os pré-requisitos para aplicação da regra são a não-existência de relação de destino entre as conexões e a não-existência de fluxo de controle da conexão origem para a conexão destino. A regra também indica que a conexão destino não pode ter sido disparada (*fired = false*).

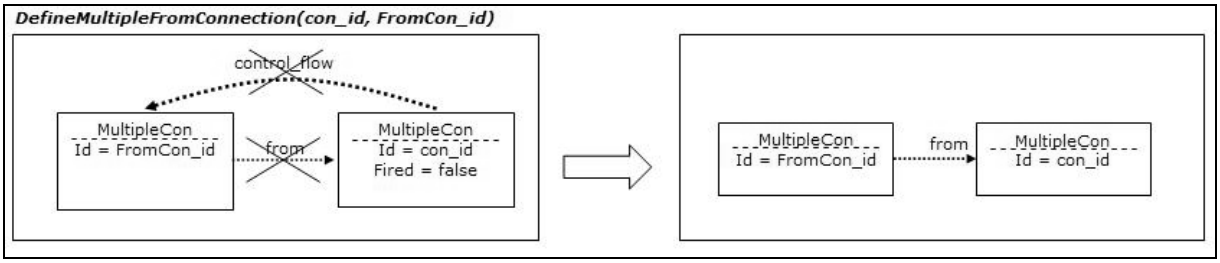

Figura 9 - Regra para definir uma conexão múltipla como origem de outra conexão múltipla

Além de regras relativas à definição de conexões, foram estabelecidas regras para testar a existência de fluxos de controle entre atividades e conexões múltiplas. O conjunto de regras definidas garante a não-inclusão de deadlocks, mantendo o modelo de integração em um estado consistente.

3.2. Semântica de Execução

Nesta seção serão apresentadas regras desenvolvidas para dar semântica à execução dos modelos de integração entre processos definidos segundo a linguagem de modelagem do APSEE-Integrate. Foram criadas regras para definir o comportamento do modelo de integração durante a execução.

A Figura 10 apresenta uma regra que define como é feita a transição de estado de uma atividade de *ready* para *active*. Para que a transição ocorra, a atividade não pode depender com conexão simples *start-start* de uma atividade que ainda não começou; não pode depender com conexão simples *end-start* de uma atividade que ainda não terminou; não pode ser destino de nenhuma conexão múltipla ainda não disparada. Além disso, é necessário que a atividade seja destino de uma conexão simples *start-start* que tenha como origem uma atividade que já começou.

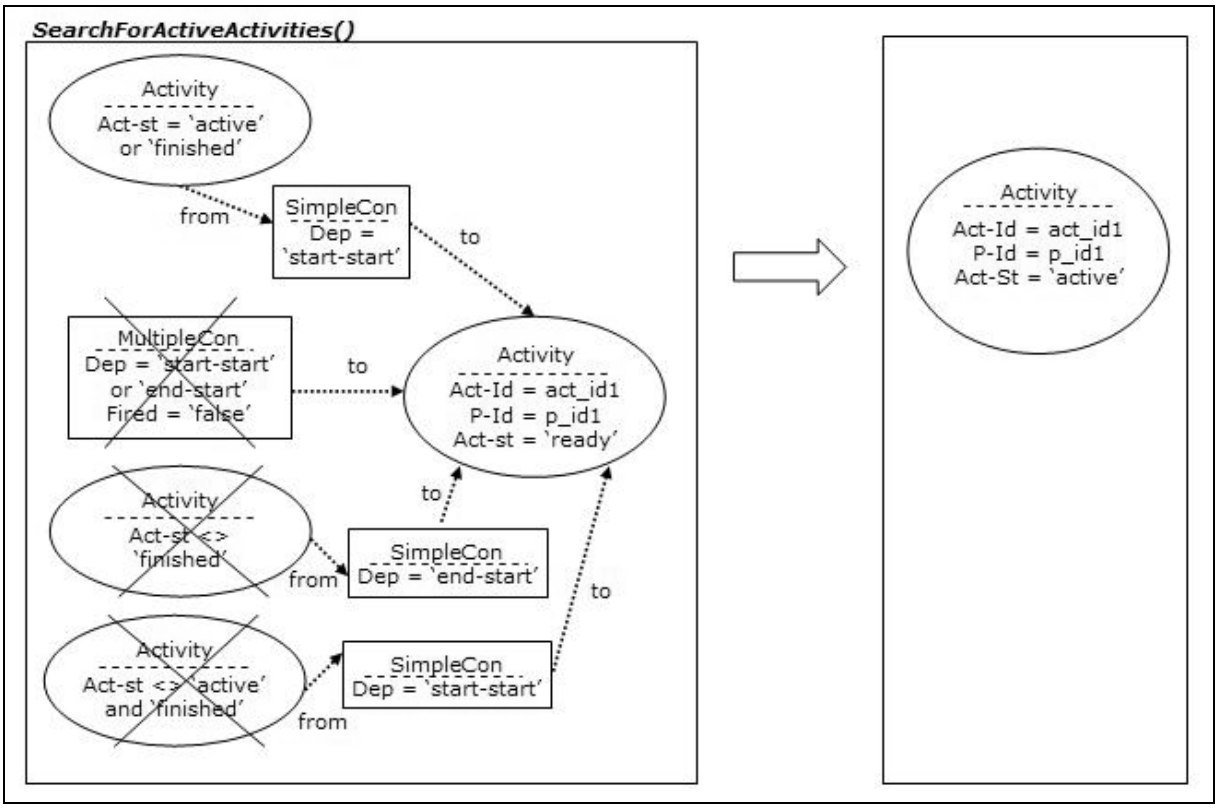

Figura 10 - Regra para transição de estado de uma atividade

Além das regras que procuram atividades prontas para executar, foram definidas regras para buscar atividades que podem ser finalizadas; para propagar falhas entre atividades; para tratar ativação de conexões múltiplas (definem quando o atributo *fired* será igual a *true*). O conjunto de regras obtido define a semântica da execução de modelos de integração entre PSEEs no APSEE-Integrate.

4. TRABALHOS RELACIONADOS

Dentro da comunidade de Processos de Software, a área de pesquisa em integração de processos de software descentralizados tem-se tornado um tópico bastante abordado [13]. Soluções têm sido propostas para tratar a interoperabilidade nos níveis de modelagem e execução de modelos de processo. Esta seção apresenta alguns dos principais trabalhos encontrados na literatura, comparando-os com a abordagem do APSEE-Integrate.

Process Landscaping [10] é um método que descreve o relacionamento entre modelos de processo distribuídos. O foco maior está na fase de definição de requisitos, e o objetivo é reduzir a complexidade dos modelos de processo. Esta abordagem preocupa-se com o nível de modelagem, não abrangendo a execução, e faz a suposição implícita de que existe um único sistema por trás de todos os modelos de processo.

Provence e Endeavors são exemplos de soluções que apresentam heterogeneidade no nível de execução, sem prover interoperabilidade no nível de modelagem [3]. Provence é um ambiente heterogêneo com o objetivo de prover suporte não-intrusivo para a execução de projetos. A interoperabilidade obtida é *ad hoc* e em baixo nível de abstração, sem definição de modelo de interoperabilidade. Endeavors é um sistema de *workflow*/processos distribuído, que suporta execução heterogênea de *handlers* de baixo nível, possivelmente escritos em diferentes linguagens de programação.

A interoperabilidade entre PSEEs homogêneos é suportada por ferramentas como Oz [4], PROSYT [6] e Serendipity-II [11]. A essência do modelo de interoperabilidade do Oz reside em dois mecanismos para

abstração da definição e da execução interprocessos: *treaties* – que permitem a definição de sub-processos compartilhados; e *summits* – que provêem a execução dos *treaties* definidos. O PROSYT integra comunicação baseada em eventos com suporte a mobilidade de código. E o Serendipity-II suporta modelagem distribuída e cooperativa de processos de software. Modelos de processo são replicados entre os usuários, e diferentes versões podem ser executadas.

Ambientes como APEL [8], PIE [7] e CAGIS [14] suportam interoperabilidade de PSEEs heterogêneos. APEL e PIE têm foco na integração de ferramentas distintas, não necessariamente PSEEs. Ambos baseiam-se na idéia de um modelo de processo comum, que define a integração entre as diferentes ferramentas. O modelo de processo comum é executado, e coordena a execução das demais ferramentas. O CAGIS interliga ferramentas de *workflow*, utilizando agentes de software para realizar atividades que envolvam mais de um *workflow*.

O modelo apresentado neste artigo suporta modelagem de processos de software distribuídos e execução descentralizada envolvendo diferentes PSEEs. Assim como no APEL e no PIE, o modelo proposto controla a integração entre PSEEs. Mas tal controle é realizado de forma transparente, e cada PSEE executa de forma autônoma. Enquanto o foco do CAGIS é suportar atividades que envolvam mais de uma equipe para sua execução, o modelo apresentado neste artigo visa suportar a integração entre atividades inter-relacionadas executadas de forma independente pelas equipes.

5. CONCLUSÕES

PSEEs devem prover infra-estrutura para processos que envolvem equipes dispersas geograficamente, possuindo práticas de desenvolvimento distintas, usando diferentes ferramentas e linguagens de programação. PSEEs devem ser capazes de interagir, permitindo interoperabilidade entre modelos de processo, nos níveis de modelagem (integração entre modelos de processo especificados através de diferentes formalismos) e de execução (integração na execução realizada por diferentes PSEEs) [8].

O artigo propõe uma abordagem para integração de processos, modelados e executados em diferentes formalismos. O modelo proposto, o APSEE-Integrate, permite a definição de dependências entre atividades, de modo que a execução de modelos de processo seja integrada.

Foram apresentados o meta-modelo de integração de processos, a linguagem de modelagem e o mecanismo de execução. Além da descrição informal, a sintaxe formal da linguagem e a semântica formal do mecanismo de execução foram apresentadas, utilizando gramática de grafos.

Neste artigo, inicialmente foi apresentada uma visão geral do APSEE-Integrate. Em seguida foram apresentados informalmente o meta-modelo, a linguagem de modelagem e o mecanismo de execução. A semântica formal da linguagem de modelagem e do mecanismo de execução foi discutida. A abordagem para integração de PSEEs apresentada traz como principais contribuições:

- A semântica formal do mecanismo de execução através do uso do paradigma PROSOFT-Algébrico combinado com Gramática de Grafos. A especificação formal permite trabalhar em alto nível de abstração, provendo uma base para rastreamento, verificação formal e permite definir as conseqüências das transições de estado de forma resumida e unificada;

- A importância dada aos requisitos resultantes do envolvimento humano no processo, aumentando a flexibilidade do sistema resultante e suportando processos criativos e incerteza. Esta característica se reflete na capacidade de executar modelos de integração incompletos, permitir execução de processos colaborativos (com dependência *start-start*), apoiar a modificação dinâmica do modelo de integração, além de fornecer um formalismo gráfico e executável que permite a modelagem e acompanhamento das dependências entre processos;

- A interoperabilidade nos níveis de modelagem e execução entre PSEEs. É possível definir conexões entre atividades modeladas em diferentes formalismos, e executando em diferentes PSEEs, através da tradução desses formalismos para o formalismo apresentado no meta-modelo do APSEE-Integrate.

O ambiente APSEE possui uma arquitetura ampla e em constante evolução. Estão sendo construídas várias ferramentas de gerência de processos para serem integradas na arquitetura. Trabalhos do mesmo grupo estão

desenvolvendo propostas para adaptação de processos de software e utilização do APSEE para processos de educação a distância, dentre outros trabalhos.

A solução proposta pelo APSEE-Integrate é um passo para flexibilizar e aumentar o nível de automação fornecido por PSEEs, podendo ser útil para incrementar a qualidade dos produtos e diminuir o seu tempo de desenvolvimento. O APSEE-Integrate permite que diferentes tecnologias de suporte a processos de software sejam integradas, aproveitando as vantagens inerentes a cada PSEE.

AGRADECIMENTOS

Ao CNPq (Conselho Nacional de Pesquisa) pelo apoio financeiro a este trabalho.

REFERÊNCIAS

[1] BARDOHL, R. et al. “Appliccation of Graph Transformation to Visual Languages”. In: Ehrig, H.; Engels, G.; Kreowski, H-J.; Rozenberg, G. (Eds.) *Handbook of Graph Grammars and Computing by Graph Transformation: Applications, Languages and Tools*. Volume 2. World Scientific, Singapore, 1999.

[2] BARDOHL, R. “GenGED - Visual Definition of Visual Languages based on Algebraic Graph Transformation”. PhD Thesis. Technische Universität Berlin. Kovac Verlag, Hamburg, 2000.

[3] BARTHELMESS, P. "Collaboration and coordination in process-centered software development environments: a review of the literature," *Information and Software Technology*, vol. 45, nº 13, pp. 911-928, 2003.

[4] BEN-SHAUL, I.; KAISER, G. "A paradigm for descentralized process modelling and its realization in the Oz environment," in *Proc. 1996 16th International Conference on Software Engineering*, pp. 179-188.

[5] CUGOLA, G.; GHEZZI, C. “Software Processes: a Retrospective and a Path to the Future”. In: *Fifth International Conference on Software Process*, Lisle. Proceedings... Jun 1998.

[6] CUGOLA, G.; GHEZZI, C. "Design and implementation of PROSYT: a distributed process support system," in *Proc. 1999 IEEE 8th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pp. 32-40.

[7] CUGOLA, G. et al. "Support for software federations: the PIE platform", in *Proc. 2000 European Workshop on Software Process Technology*.

[8] ESTUBLIER, J.; AMIOUR, M.; DAMI, D. "Building a federation of process support systems," in *Proc. 1999 Siplan, Sogmod, Sigsoft WACC Work, Activity, Coordination and Cooperation Conf.*, pp. 22-26.

[9] GRUHN, V. "Process-centered software engineering environments: a brief history and future chalenges," *Anals of Software Engineering*, vol. 14, pp. 363-382, 2002.

[10] GRUHN, V.; WELLEN, U. "Process landscaping: modelling distributed processes and proving properties of distributed distributed process models," *Lecture Notes in Computer Science: Unifying Petry Nets – Advance in Petry Nets*, vol. 2128, 2001.

[11] GRUNDY, J. C. et al. "A decentralized architecture for process modelling and and enactment, " *IEEE Internet Computing*, pp. 53-62, 1998.

[12] REIS, C. A. L. "Uma abordagem flexível para execução de processos de software," Tese de doutorado, Instituto de Informática, Universidade Federal do Rio Grande do Sul, 2003.

[13] WANG, A. "Support for mobile software processes in CAGIS" in *Proc. 2000 European Workshop on Software Process Technology*.

[14] WANG, A. "A process centred environment for cooperative software engineering, " in *Proc. 2002 SEKE*.