Um motor de execução de processos interoganizacionais para ambientes de desenvolvimento de software

Gustavo Yuji Sato[1] [2]

Departamento de informática
Universidade Estadual de Maringá
Maringá, Brasil

Elisa Hatsue Moriya Huzita[3]

Departamento de informática
Universidade Estadual de Maringá
Maringá, Brasil

Gislaine Camila Lapasini Leal[4]

Departamento de Engenharia de Produção
Universidade Estadual de Maringá
Maringá, Brasil

Abstract

Aiming to achieve time and cost reduction as well new markets, software development organizations have adopted global software development and outsourcing. However, it also brings new challenges, such as more effort to coordinate developers, sites and organizations involved in software development. This paper presents a process engine that meet this need. An important contribution of this paper refers to the fact that the approach presented allows coordinating process running among multiple instances of the process engine, each one in different organizations without the necessity to specify the internal process of each participating organization, but only the artifacts exchanged among them.

Keywords: Process Engine, Global software development, Process centered software engineering environment

[1] O autor agradece a CAPES pelo financiamento da pesquisa
[2] Email: gus.sato@gmail.com,
[3] Email: emhuzita@din.uem.br
[4] Email: gclleal2@uem.br

Resumo

Na busca pela redução de tempo e custo, bem como para atingir novos mercados, as organizações vêm adotando novos modelos de desenvolvimento de software, tais como: desenvolvimento global de software e *outsourcing*. Entretanto, também traz novos desafios. Entre eles, estão a necessidade de maior esforço para coordenar desenvolvedores e organizações envolvidas no processo de desenvolvimento de software. Este trabalho apresenta uma *engine* de processos que busca reduzir este esforço. A abordagem apresentada se diferencia por permitir a coordenação de processos entre múltiplas instancias do motor de processos, cada uma em diferentes organizações, sem que seja necessário o compartilhamento de detalhes sobre o processo interno de cada organização, apenas dos produtos trocados entre elas.

Palavras chave: Motor de processos, desenvolvimento de software global, ambiente de desenvolvimento de software centrado em processos.

1. Introdução

Os avanços da tecnologia de telecomunicações possibilitam uma nova forma de desenvolvimento de software, no qual recursos, investimentos, usuários e equipes de desenvolvimento podem estar distribuídos em diferentes locais físicos, configurando, assim, o desenvolvimento global de software. Neste cenário, a distribuição geográfica das equipes pode ser categorizada como: *onshore* (dentro do mesmo país) e *offshore* (em países diferentes). Já no que se refere ao controle organizacional pode-se classificar em: *insourcing* (formada de uma organização e suas subsidiárias) e *outsourcing* (no qual uma outra empresa é contratada para fornecer determinados serviços ou produtos de software) [1].

Segundo *Dibbem et al.* [2] *outsourcing*, não especificamente em sistema de informação, é a utilização de agentes externos da organização para realizar alguma de suas atividade (ex: aquisição de bens e serviços). *Outsourcing* também pode ser entendido como transferir parte da responsabilidade do negócio ou função de um grupo de funcionários para um grupo de não funcionários [3].

Na terceirização de sistemas de informação, diferentes modelos de coordenação interorganizacionais podem ser usados pelo cliente [4]. Os modelos formais são: o controle comportamental e o controle de produto. No primeiro, o cliente define o processo do fornecedor de serviços que, geralmente, é utilizado quando o cliente possui conhecimento sobre a área terceirizada. Já no controle de produto, o cliente não se preocupa como o produto será desenvolvido, mas apenas com a saída do processo e realiza atividades para avaliação da qualidade do produto.

Um ambiente de desenvolvimento de software (ADS) pode ser usado para coordenar o processo colaborativo entre as organizações envolvidas. Um ADS consiste de uma parte variante (processo, ferramentas e dados de objetos) e fixa (gerenciador de objetos e interface gráfica para esse). O motor de execução de processos é responsável por interpretar o modelo de processo, possibilitando o uso de diversos processos em um ambiente de desenvolvimento [5].

ADS centrados em processos auxiliam no desenvolvimento colaborativo de software reduzindo a coordenação necessária para iniciar um projeto; fazendo com que desenvolvedores ganhem experiência utilizando a mesma estrutura de processo, resultando na redução da coordenação necessária para definir pontos de colaboração entre equipes [6].

O objetivo deste artigo é apresentar um motor de execução de processos (*engine*) que permite executar e acompanhar o processo de desenvolvimento de software entre empresas que praticam o desenvolvimento *outsourcing* e, também, efetuar o

controle de produto. Dessa forma, reduz-se o esforço necessário para a coordenação do processo de desenvolvimento colaborativo interorganizacional. Também, visa-se à descentralização da infraestrutura, informações, processos e artefatos utilizados no motor de execução.

Este artigo está organizado em cinco seções, além desta introdutória: A Seção 2 discute alguns conceitos sobre *engine* de processos; a Seção 3 apresenta trabalhos relacionados; a Seção 4 apresenta uma visão geral do funcionamento da *engine*; a Seção 5 detalha o projeto da *engine.* Por fim, na Seção 6, são apresentadas as conclusões e considerações finais.

2. *Engine* de processos

Bandinelli *et al.* [5] e Kammer [11] classificam a execução de processos em relação à sua restrição em dois modelos: a *engine* pode servir apenas como um guia de processos ou restringir completamente o processo às tarefas pré-definidas.

Como um guia de processos (*process guidance*), a *engine* oferece suporte indireto aos desenvolvedores, ou seja, o ambiente oferece informação que ajuda no desenvolvimento de software, tais como: estado do processo, próximas tarefas, pontos de decisões e seus significados.

Já na forma de restrição ao processo (*process enforcement*), além de oferecer informações sobre o processo de desenvolvimento, este modelo controla a iniciativa de desenvolvimento. Esta forma de desenvolvimento pode ser resultado das necessidades políticas organizacionais para que o trabalho não desvie do processo.

Segundo Kammer [11], um ambiente não deve ser totalmente restrito ao processo. Isso se deve ao fato de que no decorrer de um projeto ocorrem exceções do processo, resultando na necessidade de desviar do fluxo do processo para realizar tarefas não previstas em sua modelagem.

Reis [13] e Gary *et al.* [15] classificam as linguagens de modelagem de processos da seguinte forma:

Execução procedimental (ou execução por linguagens de definição de processos) - O processo é descrito passo à passo, com estruturas de controle semelhantes a uma linguagem de programação. Esta deve ser interpretada pela *engine* de execução após a realização da análise léxica, sintática e semântica.

Execução baseada em regras o processo é modelado como um conjunto de regras, semelhantes às regras da linguagem de programação Prolog [16]. Essas regras devem ser interpretadas por uma máquina de inferência.

Execução baseada em regras ECA (Evento-condição-ação) possui um conjunto de restrições associadas a um evento. Quando ocorre um evento no sistema, as restrições são verificadas. Se essas são satisfeitas, então é executada a ação, podendo essa disparar novos eventos.

Execução baseada em Redes de Petri o processo é representado com o formalismo matemático de Redes de Petri. Na modelagem de processos de software, artefatos e recursos são geralmente representados por *tokens* inseridos nos estados. Os arcos expressam dependência entre lugares e transições e podem ser rotulados com pesos.

Execução baseada em redes de tarefas pode ser interpretada como um grafo direcionado em que cada nó do grafo equivale a uma atividade, seus arcos representam o fluxo de controle e dados que trafegam entre as atividades. Um modelo de processos baseado em redes de tarefas consiste em um grafo direcionado que representa a estrutura do processo. Em geral, os nodos representam atividades

enquanto os arcos representam o fluxo de controle e de dados entre eles.

3. Trabalhos relacionados

Moura [7] apresenta um motor de execução de processos para ambientes de desenvolvimento centrados em processos, utilizando a linguagem SPEM para especificação de processos de gerencia de configuração no desenvolvimento baseado em componentes.

O trabalho de Cereja Junior [8] apresenta um motor de execução de processos, cuja arquitetura é baseada em agentes. Estes executam as seguintes funções: criam novas instancias de processos; coordenam a execução das tarefas; selecionam a melhor opção entre as pessoas para desempenhar um determinado papel; notificam pessoas sobre a alocação de novas tarefas e monitoram a execução das tarefas.

Entretanto, não é parte do escopo desses trabalhos a coordenação de processos em execução em duas instâncias de motores de execuções diferentes. Este é um requisito desejável quando é necessário manter a independência na infraestrutura de locais, como no caso de desenvolvimento no modelo *outsourcing*, requisito que o motor de execução apresentado neste trabalho pretende cobrir.

No trabalho de Lima e Reis [9], a coordenação de processos em diferentes instâncias de motores de execução ocorre por meio de contratos eletrônicos. Inicialmente, o processo é definido na instância do cliente, responsável por definir quais atividades do processo serão realizadas por ele e pelo prestador de serviços. Em seguida, as atividades a serem realizadas pelos prestadores de serviços são enviadas para as suas respectivas máquinas de execução de processos, enquanto as atividades do cliente são executadas na instância do cliente.

Diferentemente da abordagem de Lima e Reis [9], o motor aqui proposto, possibilita que o processo a ser executado pelo fornecedor de serviço possa ser definido por ele mesmo, ao invés de ser imposto pelo cliente. Assim, os próprios profissionais e líderes de projeto são responsáveis por definir o processo de desenvolvimento, desobrigando organizações a seguirem padrões de documentação e processo definidos por terceiros. Isto é considerada uma boa prática de melhoria de processo [10].

Em Kammer [11] é apresentada uma arquitetura sobre a qual a infraestrutura para o trabalho colaborativo é formada. Embora esse trabalho seja uma arquitetura e não um motor para execução de processos, apresenta objetivos similares ao trabalho apresentado nesse artigo, no que se refere à descentralização da infraestrutura e definição do processo sendo usado. O processo a ser executado é definido por meio da composição de seus subprocessos, sem que seu controle seja centralizado, ou seja, não há autoridade responsável pela definição completa do processo. Por exemplo, se um processo de desenvolvimento de software é constituído de fases (subprocessos) de análise, desenvolvimento, testes e integração, cada uma dessas é atribuída à equipe responsável. O gerenciamento interno dos subprocessos é de responsabilidade da equipe, que pode adicionar novos participantes, atividades e artefatos, sem que seja necessária uma requisição ao nível hierárquico superior.

4. Visão geral do *Global Software Process Engine* (GSPE)

O motor de execução *Global Software Process Engine* (GSPE) possuirá as seguintes funcionalidades: instanciar e executar processos importados de uma ferramenta

externa; possibilitar a integração com sistemas de controle de versão para a troca de artefatos e permitir maior independência de infraestrutura no desenvolvimento *Outsourcing*.

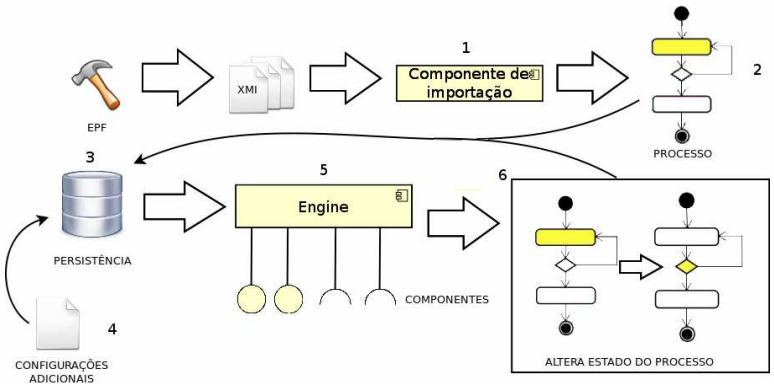
Para instanciar o processo de desenvolvimento de software, este será importado por meio de uma ferramenta externa, EPF (*Eclipse Process Framework*[5]), usada para modelagem de processos. A EPF utiliza o meta-modelo UMA (*Unified Method Architecture*), baseado no meta-modelo SPEM 1.0.

O meta-modelo SPEM, não oferece mecanismos próprios para a especificação de fluxo de tarefas. No lugar disso, possibilita que suas implementações escolham modelos para serem utilizados para essa tarefa, como por exemplo diagrama de estados e seqüência da UML 2.0 e, de forma alternativa, BPDM/BPMN.

Da mesma forma, o meta-modelo UMA não oferece esses recursos, mas a implementação utilizada na ferramenta EPF faz o uso de diagramas de atividades UML 2.0 para documentar o fluxo das atividades no processo. Os modelos UMA e UML utilizados na EPF são desenvolvido sobre o meta-modelo Ecore[6].

O meta-modelo Ecore é baseado no meta-modelo MOF (*Meta Object Facility*) [12], especificamente nos pacotes do Essencial MOF (EMOF). Como o EMOF, o Ecore abstrai o conceito de classes, atributos e operações, possui mapeamento para Java e XMI. Ecore não se trata de uma implementação do MOF, como o JMI[7], e sim um meta-modelo diferente, usado em diversas ferramentas e bibliotecas da IDE Eclipse[8].

A Figura 1 apresenta o modelo conceitual da forma que o motor cria instâncias e executa processos. Os processos exportados no formato XMI, são carregados pelo motor de execução por meio de um componente de importação (1). Este é responsável por transformar os meta-modelos UMA e UML na representação do modelo de processos do motor de execução (2). Este modelo é então persistido no banco de dados (3). No banco de dados são, também, armazenadas configurações adicionais (4) para a execução do projeto que não podem ser obtidos por meio da importação, tais como: participantes do projeto e seus perfis no processo (gerente, desenvolvedor, arquiteto, etc.). A *engine*, ou motor de execução, (5) oferecerá serviços como: iniciar processo, concluir tarefa e suspender processo. Ao receber uma chamada desses componentes, a *engine* deve alterar a representação interna do estado do processo (6) e persistir novamente no banco de dados.



5 http://www.eclipse.org/epf
6 http://www.eclipse.org/emf
7 http://java.sun.com/products/jmi
8 http://eclipse.org

Fig. 1. Modelo conceitual da instanciação e execução de processos

Um sistema de controle de versões (SCV) gerencia quem, porquê, quando e quais modificações ocorreram em um projeto de software. SCVs distribuídos (como: Bazaar[9], Mercurial[10] ou Git[11]), diferentemente de SCVs centralizados (como: Subversion e CVS), trabalham de forma *peer to peer*, sem a necessidade de um servidor centralizado. Assim, sistemas de controle de versões distribuídos possuem uma cópia do histórico de versões em cada *peer* que pode ser sincronizado com os demais *peers* de acordo com consenso.

Em um cenário de colaboração entre duas empresas, é possível definir um *peer* como principal em cada empresa com a versão a ser compartilhada. Dessa forma, elimina-se a dependência de infraestrutura em relação ao controle de versões. Caso haja falha na comunicação de rede entre essas duas organizações a colaboração interna das equipes não será comprometida. Além disso, no fim do projeto, ambas as organizações possuirão o histórico de versão do trabalho que possibilita a extração de dados para a melhoria do processo de desenvolvimento.

Um dos componentes desenvolvidos no motor de execução de processos é a integração com sistemas de controle de versões distribuídos. Por meio deste componente, é possível associar uma submissão de versão no sistema de controle de versões a uma atividade e, assim, finalizar atividades. O sistema de controle de versões escolhido para a implementação inicial é o Mercurial, sendo trabalho futuro, a implementação dos demais.

A Figura 2 apresenta um cenário que ilustra a interação entre duas instancias do motor de execução necessária para quando se quer ou vai terceirizar uma atividade do processo. Por meio de um serviço, é possível marcar uma tarefa a ser terceirizada pela empresa A para a empresa B, previamente configurada no sistema (1). Uma requisição de terceirização é enviada da *engine* da empresa A para *engine* da empresa B (2) por meio de um protocolo de comunicação, para a implementação foi adotado o RMI. Em seguida, a empresa B deve aceitar ou rejeitar o pedido (3). Ao aceitar, a empresa B deve customizar um processo para a atividade terceirizada a ser desenvolvida e associar o processo ao pedido (4). Este processo passa a ser executado pela *engine* da empresa B como os demais processos. Dessa forma, os processos internos da organização não são compartilhados com a outra, uma vez que apenas acordos e artefatos são trocados entre elas, realizando dessa forma a integração de processos via controle de produto.

Por meio de um componente de notificação (5), a *engine* notificará os participantes e, ao concluir a atividade terceirizada, os desenvolvedores submeterão as modificações no sistema de controle de versões distribuído da empresa B (6). O componente de integração com o sistema de controle de versões verificará quando ocorre a submissão de uma versão relacionada a uma atividade e marcará esta como finalizada (7). Ao completar as atividades, os desenvolvedores (empresa B) também deverão submeter às modificações ao sistema de controle de versões da empresa A (8) que marcará a atividade como finalizada por meio do componente de integração (9 e 10) com o sistema de controle de versão.

[9] http://bazaar.canonical.com
[10] http://mercurial.selenic.com
[11] http://git-scm.com

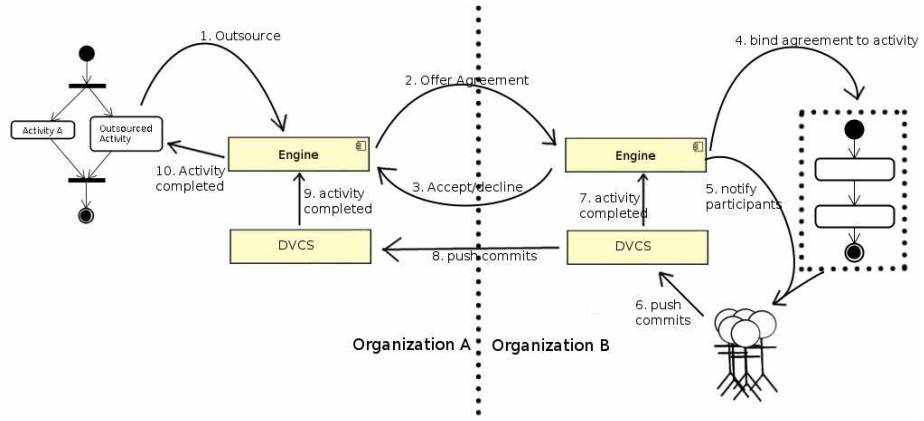


Fig. 2. Diagrama conceitual da realização de outsourcing

5. Detalhamento do GSPE

Esta seção irá detalhar o projeto da engine. Para facilitar o seu entendimento, ela foi subdividida em quatro subseções, cada uma detalhando partes do diagrama de pacotes (Figura 3). A seção 5.1, explica o pacote core e seus subpacotes: *model* e *dao*; a seção 5.2 detalha o pacote *process-import*, utilizado para instanciar o processo a partir dos meta-modelos UMA e UML; a seção 5.3 descreve o pacote *engine,* usado para interpretar e alterar o estado do processo; a seção 5.4 discute sobre a integração com sistemas de controle de versões e a seção 5.5 explica os serviços oferecidos para outras instancias da *engine* no pacote outsource-services.

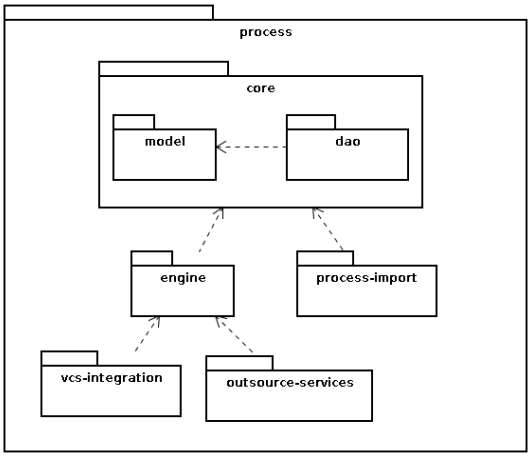


Fig. 3. Diagrama de pacotes da *engine*

5.1. O Pacote core

O pacote core, é constituído de dois subpacotes: *dao* e *model*. O pacote *dao* contém as classes usadas para a persistência em banco de dados, utilizando o padrão J2EE[12] Data Access Object. O pacote *model* possui classes que constituem o modelo de processo, isto é, as classes que representam o processo e seu estado.

O modelo definido para a representação de processos do motor de execução pode ser classificado como uma rede de tarefas, dentro da classificação de formalismos de execução definido por [13]. Essa representação foi escolhida motivada pelos modelos UMA e SPEM que são redes de tarefas, a utilização de um modelo diferente inviabilizaria o desenvolvimento de um componente que permitisse a importação desses.

A Figura 4 apresenta o modelo usado pelo motor para representar os processos em execução. As classes do modelo são:

ProcessElement abstrai os conceitos de Tarefa e Atividade.

ProcessActivity representa uma atividade que pode ser subdividida em outras atividades, as quais podem ser encontradas no grafo referenciado por Start, nos nós que são instâncias de *ProcessElementNode*

Task representa uma tarefa, que diferentemente de atividade, não pode ser subdividida e possui ferramentas, papéis e artefatos diretamente associados.

Role é um conjunto de pessoas que desempenham o mesmo papel no desenvolvimento de uma atividade. Essa classe está relacionada com **User**, usada para identificação e autenticação no sistema.

Start, **End**, **Fork**, **Join** e **Decision** correspondem a elementos em um diagrama de atividades UML, respectivamente: nó inicial, final, bifurcação e decisão. Todos esses são abstração de **Node**. **Edge** é uma aresta, ligando dois nós no diagrama.

ProcessElementNode corresponde a um nó representando uma atividade ou tarefa.

VisitableNode é a interface usada para definir elementos visitados por meio do padrão de projeto Visitor [14], utilizado para a execução do processo pelo motor de execução.

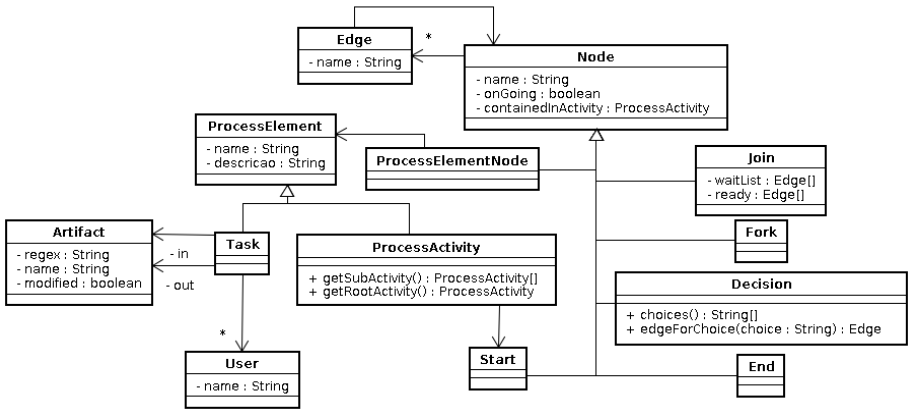


Fig. 4. Diagrama de classes do pacote *engine*

[12] http://java.sun.com/blueprints/corej2eepatterns/Patterns/

5.2. O Pacote process-import

O pacote process-import contém classes que possibilitam que sejam instanciados processos a partir do meta-modelos UMA e UML, utilizados pelo *Elicpse Process Framework*. Os meta-modelos UMA e UML não são usados diretamente no motor de execução de processos por serem complexos e possuírem diversas classes não inerentes à execução do processo.

Tabela 1. Mapeamento de UML/UMA para proposto para o modelo proposto

Modelo Proposto	**Modelo UML/UMA**
ProcessElement	Definido por uma subclasse
Task	UMA::TaskDescriptor
ProcessActivity	UMA::Activity
Tool	UMA::ToolMentor
Artifact	UMA::WorkProductDescriptor
Role	UMA::RoleDescriptor
User	Inserido manualmente
Start	UML::InitialNode
Decision	UML::DecisionNode
End	UML::ActivityFinalNode
Join	UML::JoinNode
Fork	UML::ForkNode
ProcessNode	UML::ActivityNode
Edge	UML::ActivityEdge
Node	Definido por uma subclasse
VisitableNode	Definido por uma subclasse

Portanto, a importação do meta-modelo UMA é feita por meio do *parse* do arquivo XMI e mapeando classes do meta-modelo para o modelo do motor de execução de acordo com a Tabela 1. A maior parte desse modelo é instanciada importando dados dos meta-modelos UMA e UML. Entretanto, alguns atributos necessários à execução do processo não podem ser obtidos desses modelos, sendo necessário a entrada manual dos dados, como por exemplo, a associação entre User e Role. Em trabalhos futuros, serão desenvolvidos componentes de importação para outros meta-modelos, como o SPEM.

A interface Importer (Figura 5) especifica o componente de importação UMA. ImporterImpl possui a implementação do modelo de importação, que é responsável por instanciar o modelo de processos baseado no meta-modelo UMA (Tabela 1). XMIManager é responsável pelo parse dos arquivos XMI, utilizando a biblioteca EMF. ImportDatabase funciona como o padrão decorator [14], apenas adiciona o comportamento de persistência do modelo de processos importado por ImporterImpl.

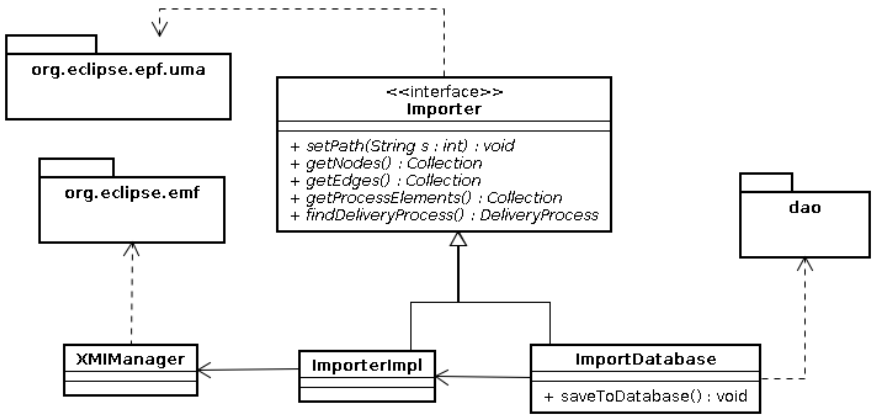


Fig. 5. Diagrama de classes do pacote process-import

5.3. O Pacote *engine*

O Pacote engine possui classes que alteram o estado do processo após receber a requisição em algum de seus serviços e, então, invoca os componentes apropriados. A classe *Engine* é responsável por alterar para o próximo estado de acordo com a modelagem do processo. Isso é especificado pela interface NodeVisitor (Figura 6), baseada no padrão de projeto *Visitor* [14] . Ao visitar uma instancia de Fork o método *visit* deve marcar todas as atividades dos nós ligados a eles a serem realizadas; caso seja Decision, o processo segue o fluxo de atividades escolhida pelo gerente.

Estas alterações de estados são disparadas por outros componentes do ambiente responsáveis pela percepção de eventos tais como: monitoramento de sistema de controle de versões, testes automatizados de software e ferramentas de decisões colaborativas. Essas ferramentas devem usar os serviços oferecidos pela classe EngineServices como interfaces providas para componentes. Esta interface recebe decisões gerenciais relacionadas à notificação de tarefa concluída, por exemplo. Outra interface provida é EngineOutsourceServices, exposta para outras instancias do motor de execução de uma outra organização externa.

Componentes podem ser associados ao motor por meio de um arquivo de configuração e são instanciados pela classe ComponentFactory. A referência dos componentes instanciados é mantida pela classe EngineComponentManager. Toda vez que o motor altera o estado de um processo as instâncias desses componentes são executadas por sua interface.

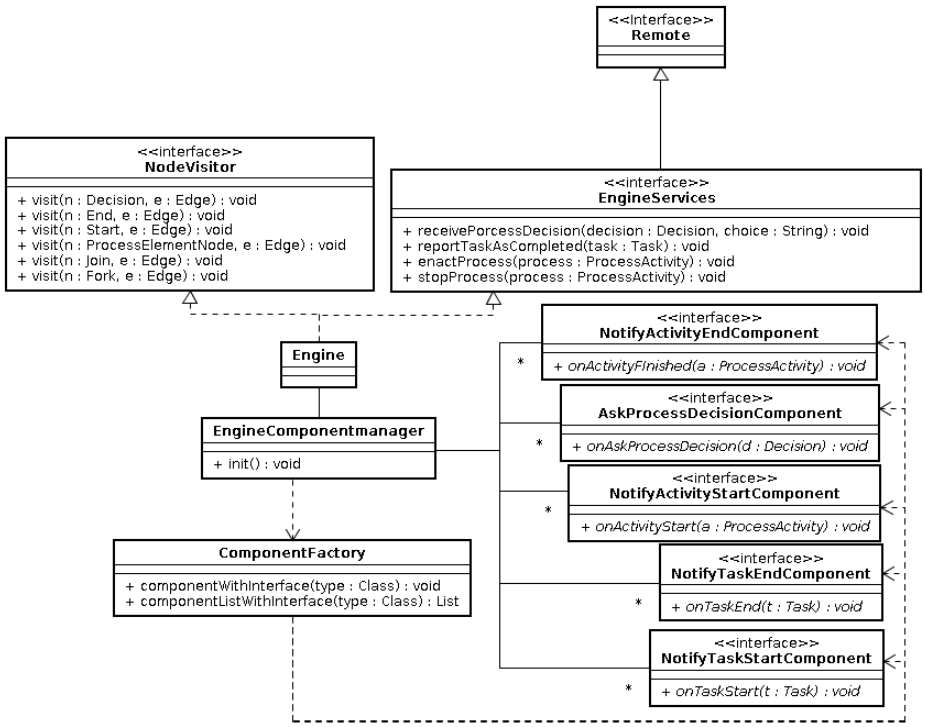


Fig. 6. Diagrama de classes da *engine*

5.4. O Pacote vcs-integration

Este pacote oferecerá scripts, componentes e serviços necessários para realizar a integração com sistemas de controles de versões. A implementação foi realizada com o sistema de controle de versões distribuído Mercurial. Entretanto, a mesma arquitetura pode ser usada para outros sistemas de versões distribuídos (como Git e Bazzar), uma vez que eles oferecem as mesmas funcionalidades básicas necessárias para realizar a integração.

Sistemas de controle de versões distribuídos se diferem de sistemas de controle de versões centralizados. Nos sistemas de versões centralizados, o histórico de modificações é armazenado em um único servidor central e novas versões podem ser submetidas por clientes. Os clientes possuem apenas o espaço de trabalho (workspace), com uma imagem de uma versão com a adição das alterações do usuário.

As operações comuns do sistema de controle de versões centralizados são:

- *checkout*: O cliente obtém uma cópia do repositório, iniciando o *workspace*.
- *commit*: O cliente envia as alterações realizadas no *workspace* para o servidor central.
- *update* O cliente atualiza *workspace* com alterações enviadas por outros clientes.

No sistema de controle de versões distribuído, não existe um servidor central. Entretanto, geralmente, é eleito um nó para fazer papel central e sincronizar o trabalho. Todos os nós armazenam o histórico de modificações localmente, além de

possuir o *workspace*. As operações do sistema de controle de versões são:

- *Commit*: Envia modificações do *workspace* para o histórico local.
- *Push*: Envia as modificações do histórico local para o histórico de outro nó.
- *Pull*: Recebe o histórico de modificações de outro nó para o histórico local.
- *Update*: Atualiza o *workspace* para uma versão do histórico local.

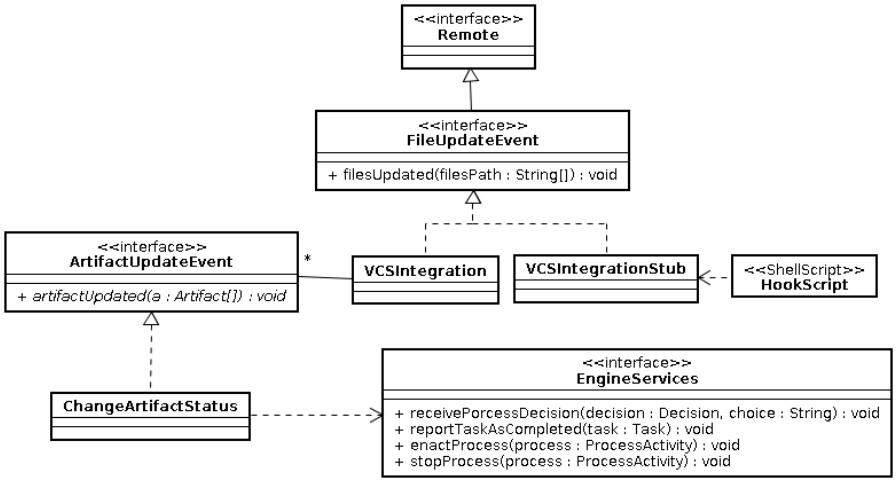


Fig. 7. Diagrama de classes do pacote vcs-integration

A integração com o sistema de controle de versões oferecerá as seguintes funcionalidades:

- Uma tarefa é marcada como concluída quando os artefatos definidos como seus produtos são submetidos ao sistema de controle de versão.
- Um artefato pode representar um ou mais arquivos, estes são definidos por meio de uma expressão regular no modelo, como atributo da classe Artifact.

A integração com o sistema de controle de versões será feita da seguinte forma: diversos sistemas de controle de versões possuem mecanismos chamados hook scripts. Esses scripts são executados quando determinados eventos ocorrem no sistema, como push, pull, commit, update. HookScript e VCSIntegrationStub estarão no nó do sistema de controle de versões. O script executará o componente VCSIntegrationStub passando como parâmetro uma lista com o caminho dos arquivos modificados. VCSIntegrationStub executará a interface remota de FileUpdateEvent por RMI. A implementação de FileUpdateEvent e VCSIntegration se encontram no mesmo nó da *engine* de processos e procuram arquivos que contém a expressão regular de artefatos relacionado à tarefas em andamento. Quando um arquivo é encontrado, o estado do artefato é marcado como modificado (pela classe ChangeArtifactStatus). Quando todos os artefatos de uma tarefa estão marcados como modificados, essa tarefa é marcada como concluída.

5.5. O pacote outsource-services

Este pacote oferece serviços para a comunicação entre múltiplas instâncias da *engine* de execução, cada uma em execução em uma organização. Os serviços oferecidos para realizar esta comunicação é especificada pela interface OusourcingServices

(Figura 8). O método offerAgreement é invocado para oferecer um acordo à uma outra *engine*, este método invoca receiveAgreement. Este acordo deve ser aceito ou rejeitado pelos métodos acceptAgreement ou declineAgreement, que por sua vez, invocam receiveAgreementAnswer da primeira *engine*.

A interface AgreementBind é usada para ligar um acordo aceito à um processo. Quando isso ocorre, o processo é executado normalmente, com o diferencial que em seu fim é realizada uma operação de *push* entre os sistemas de controle de versões para enviar as alterações realizadas.

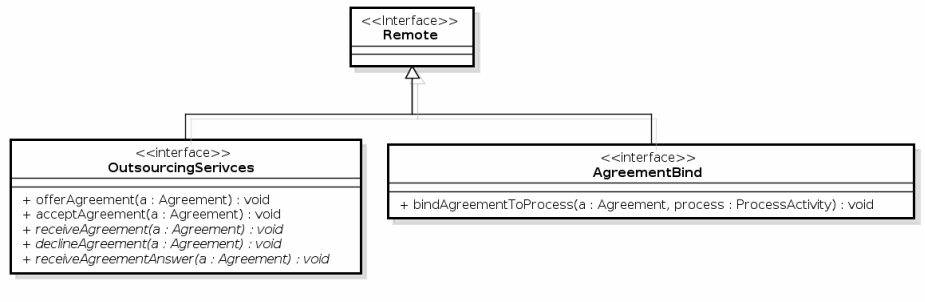


Fig. 8. Serviços oferecidos pela *engine* para realização de outsourcing

Para oferecer estes serviços, foi necessário adicionar novas informações ao modelo (Figura 8). Agreement representa um acordo entre duas organizações terceirizando uma atividade do processo, resultando na produção de artefatos. O acordo pode estar nos estados: oferecido, recebido (aguardando resposta), aceito ou rejeitado. A organização pode realizar papel de cliente ou provedor de serviço.

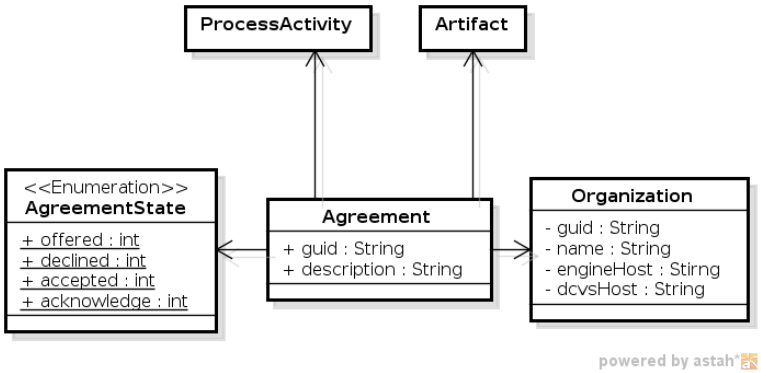


Fig. 9. Adição de Agreement ao modelo

6. Considerações finais

Este artigo apresentou um motor de execução de processos para suporte ao desenvolvimento de software considerando características do desenvolvimento no modelo *outsourcing*. A principal característica que o diferencia dos demais trabalhos encontrados na literatura é de possibilitar a coordenação de processos em múltiplas de suas instancias sem a definição do processo da organização parceira.

Como contribuição, destaca-se a redução de dependência entre a infraestrutura e processo entre organizações. Como cada organização utiliza uma instancia do motor

de execução, ambas as organizações possuirão, ao fim da parceria, dados históricos sobre o projeto e processo, possibilitando a melhoria contínua deste. Além disso, oferecerá independência em relação ao processo interno que cada organização segue, uma vez que sua interação é definida apenas pelos artefatos trocados entre elas, não sendo necessário definir o processo como um todo. Encontra-se em preparação a avaliação do motor utilizando os procedimentos de engenharia de software experimental, cujos resultados deverão ser divulgados posteriormente.

Referências

[1] Robinson, M. and Kalakota, R., Offshore Outsourcing: Business Models, ROI and Best Practices, Atlanta: Mivar Press, 2004.

[2] Dibbern, J., Goles, T., Rudy, H., and Bandula, J., "Information systems outsourcing: a survey and analysis of the literature" SIGMIS Database, Vol. 35, pp. 6–102, November 2004

[3] Zhu, Z., Hsu, K., and Lillie, J. "Outsourcing- a strategic move: the process and the ingredients for success" Management Decision, Vol. 39 Iss: 5, pp.373 - 378, 2001.

[4] Kirsch, L. J., Sambamurthy, V., Ko, D.G., and Purvis, R. L. "Controlling information systems development projects: The view from the client" Management Science, Vol. 48, p.484-498, 2002.

[5] Bandinelli, S.C., Fuggetta, A., and Ghezzi, C. "Software Process Model Evolution in the SPADE Environment". IEEE Transactions on Software Engineering, vol. 19, pp. 1128-1144, 1993

[6] Whitehead, J. "Collaboration in Software Engineering: A Roadmap" 2007 Future of Software Engineering. IEEE Computer Society. pp. 214-225, 2007.

[7] Murta, L.G.P. "Gerência de Configuração no desenvolvimento baseado em componentes" phd thesis, COPE/UFRJ, Rio de Janeiro, 2006.

[8] Cereja Junior, M.G. "Um serviço de coordenação de processos integrados ao ambiente de engenharia de software e-webproject" Ms thesis, INPE - Instituto Nacional de Pesquisas espaciais, São José dos Campos, 2006.

[9] Lima, A. M. and Reis, R. Q. "Uma Proposta de Ferramenta para Execução Descentralizada de Processos de Software" VI Simpósio Brasileiro de Qualidade de Software, 2007.

[10] Diaz, M. e Sligo, J. "How Software Process Improvement Helped Motorola" IEEE Software, Vol. 15, p. 75-81, 1997.

[11] Kammer, P. J. "A distributed architectural approach to supporting work practice" Phd thesis, University of California, Irvine, 2004.

[12] Object Management Group. "Meta Object Facility Core Specification version 2.0" formal/2006-01-01 http://www.omg.org/spec/MOF/2.0/.

[13] Reis, C. A. L. "Uma abordagem para flexível para execução de processos de software evolutivos". Phd thesis, Universidade Federal do Rio Grande do Sul, 2003.

[14] Johnson, R., Gamma, E., Helm, R. and Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

[15] Gary, K., Lindquist, T., Koehnemann, H., and Dernia J. "Component-based Software Process Support" Proceedings of the 13th IEEE international conference on Automated software engineering , 1998.

[16] Peuschel, B. and Schäfer, W. "Concepts and implementation of a rule-based process engine" Proceedings of the 14th international conference on Software engineering p. 262-279, 1992.