Charon: Uma Máquina de Processos Extensível Baseada em Agentes Inteligentes

Leonardo Gresta Paulino Murta
Márcio de Oliveira Barros
Cláudia Maria Lima Werner
COPPE/UFRJ – Programa de Engenharia de Sistemas e Computação
Universidade Federal do Rio de Janeiro
Caixa Postal 68511 – CEP. 21945-970
Rio de Janeiro – Brasil
{murta, marcio, werner}@cos.ufrj.br

Resumo

Com o aumento da complexidade dos sistemas, aliado à diminuição dos prazos, e conseqüente necessidade de constituir grandes equipes de desenvolvimento, torna-se cada vez mais importante a definição de um processo que sistematize o desenvolvimento de software. Entretanto, além da definição do processo, é necessário fazer um acompanhamento de sua execução, se possível ainda guiando os desenvolvedores dentro de um ambiente de desenvolvimento de software (ADS) que contenha as ferramentas necessárias para a construção de aplicações.

O objetivo deste trabalho é prover uma arquitetura extensível, baseada em agentes inteligentes, que permita modelar, simular, executar e acompanhar processos de desenvolvimento de software no contexto de um ADS. O uso de agentes inteligentes possibilita que a arquitetura evolua de forma simples, através da criação de novos agentes que se comunicam com os demais através de bases de conhecimento sobre processos instanciados, segundo uma ontologia definida. As bases de conhecimento de processos consistem em uma representação Prolog, gerada através da tradução de uma representação gráfica, que usa uma notação estendida do diagrama de atividades da UML.

1 Introdução

O desenvolvimento de software, como toda atividade de engenharia, necessita de um processo que o sistematize, envolvendo atividades, pessoas e ferramentas necessárias para a sua execução, assim como artefatos consumidos ou produzidos pelas atividades. A partir de um processo bem definido, é possível alavancar a qualidade do produto a ser gerado, pois a qualidade do produto é fortemente dependente da qualidade do processo pelo qual ele é construído e mantido. Além do estabelecimento do processo, é necessário fazer um acompanhamento de sua execução, objetivando, entre outras coisas, fornecer informações que permitam a tomada de decisões gerenciais, de acordo com situações detectadas durante sua execução, e controlar o fluxo de trabalho dos engenheiros de software através da automação do processo. O software responsável por automatizar a execução de processos é conhecido como máquina de processos.

Atualmente, existem várias abordagens para a construção de ambientes de desenvolvimento de software (ADS) com ênfase na modelagem e execução de processos, chamados de ambientes centrados em processo, como será apresentado na seção 4. Entretanto, essas abordagens têm deficiências quanto às suas formas de utilização, atuação e extensão. Ainda não existe uma forte preocupação na forma de **utilização** das máquinas de processo. As abordagens existentes permitem que processos sejam modelados, mas restringem o seu uso a um determinado nível de abstração. Seria interessante construir processos através de blocos, ou componentes, e reutilizar esses componentes de processos em outros contextos, facilitando a utilização da máquina de processos como um todo. A **atuação** da maioria das abordagens é reativa, consistindo no tratamento de eventos provenientes dos próprios desenvolvedores. Todavia, a máquina de processos pode agir de forma pró-ativa, prevendo certas situações e atuando sem a interferência do desenvolvedor, fazendo com que a execução siga adiante ou notificando um determinado desenvolvedor que algo deve ser feito num dado momento. Os

requisitos de um ambiente centrado em processo podem variar com o tempo, motivando a **extensão** do ambiente. A facilidade de extensão do ambiente está diretamente relacionada com a forma em que a sua arquitetura foi projetada, facilitando o atendimento de novos requisitos.

A motivação deste trabalho está na possibilidade de uma diminuição nos custos financeiros e do tempo necessário para a inclusão de novos requisitos nesse ambiente.

Dentre as tecnologias atualmente disponíveis, a tecnologia de agentes inteligentes apresenta a estruturação necessária para a construção de sistemas pró-ativos e de fácil extensão, devido às suas características de decomposição, abstração e organização [1]. Agente inteligentes podem ser identificados através de um conjunto de propriedades [2, 3, 4]:

- **Autonomia**: Capacidade do agente de operar sem intervenção externa (do ambiente em que ele está inserido, de humanos ou de outros agentes), tendo controle de suas ações.
- **Interação**: Capacidade do agente de interagir (com o ambiente em que ele está inserido, com humanos e com outros agentes), de forma reativa ou pró-ativa. A interação reativa ocorre através de respostas a eventos externos. A interação pró-ativa ocorre através da busca ao objetivo, mesmo que não haja nenhum evento externo.
- **Adaptação**: Capacidade do agente de modificar seu estado mental (em função do ambiente em que ele está inserido, de ações de usuários ou de outros agentes).
- **Aprendizado**: Capacidade do agente aprender utilizando experiências anteriores quando interage com o ambiente em que ele está inserido.
- **Mobilidade**: Capacidade do agente se transportar do ambiente em que ele está inserido para outro através da rede.
- **Colaboração**: Capacidade do agente cooperar com outros agentes para cumprir o seu objetivo e o objetivo do sistema como um todo.

O objetivo deste trabalho é fornecer um ambiente para a modelagem, instanciação, simulação, execução, monitoramento e evolução de processos de software reutilizáveis que seja pró-ativo e de fácil extensão, utilizando uma arquitetura para a construção de agentes inteligentes e uma ontologia para a comunicação entre os agentes que define o vocabulário necessário para a representação dos processos modelados.

Este trabalho está organizado em 5 seções, além desta primeira de introdução. Na segunda seção, é exibido um conjunto de critérios de avaliação de máquinas de processos, que foram utilizados para guiar a construção da abordagem proposta e posterior comparação com as abordagens existentes na literatura. Na terceira seção, é descrita a abordagem proposta, que consiste na construção de uma máquina de processos de software baseada na tecnologia de agentes inteligentes. Na quarta seção, a abordagem proposta é comparada com algumas outras abordagens para a automação de processos de software existentes na literatura. Finalmente, na quinta seção, são descritas as contribuições e limitações desse trabalho e os trabalhos futuros.

2 Critérios de avaliação

Após a análise de diversas abordagens para a automação de processos descritas na literatura, foram levantados os seguintes critérios de avaliação de máquinas de processos [5]:

- **Modelagem do processo:** Indica se a abordagem contém mecanismos para que o processo seja modelado. Essa modelagem pode ocorrer de forma gráfica ou através de linguagens próprias para descrição de processos. Quando uma abordagem não atende a esse critério significa que ela tem o seu processo modelado dentro da sua própria especificação, não permitindo a criação de outro processo diferente do inicialmente previsto nem a sua adaptação.
- **Grafo com ciclos ou recursão:** Indica se a abordagem permite que o modelo de processo faça uso de ciclos ou recursão na sua representação. Um ciclo é descrito através de um fluxo para uma atividade anterior no fluxo de execução (*workflow*) e uma recursão é descrita através de uma referência no *workflow* a um processo pai na árvore de sub-processos. Ambos ocasionam a repetição da execução de atividades.
- **Mapeamento automático:** Indica se é necessária a intervenção do desenvolvedor de software após a modelagem gráfica do processo, para permitir o seu mapeamento para uma representação executável. A

maioria das abordagens usa representações diferentes para a modelagem e a execução dos processos. A não automação desse procedimento pode gerar a introdução de erros e tornar a execução do processo muito trabalhosa para a equipe de desenvolvimento.

- **Interação pró-ativa:** Indica se a abordagem toma a iniciativa de interagir com o desenvolvedor, fornecendo soluções para determinadas situações sem que seja explicitamente solicitada. Este critério visa distinguir abordagens que só reagem a ações do desenvolvedor das abordagens que, com uso de Inteligência Artificial, prevêem situações que ainda não foram vislumbradas de forma autônoma.
- **Processos reutilizáveis:** Indica se a abordagem se preocupa com a construção de processos que possam ser reutilizados em outras situações. Essa preocupação pode ser verificada através da implementação de mecanismos que não obrigam o uso de um único nível de abstração para o processo, permitindo que ele seja reutilizado tanto como um processo global quanto como uma atividade específica de outro processo.
- **Extensão flexível:** Permite que a extensão da máquina de processos seja flexível, sendo possível, por exemplo, coletar uma nova métrica sem que seja necessário implementar esta característica dentro do núcleo do sistema. Para que esse critério seja atendido, a abordagem deve prover algum mecanismo que permita uma extensão dos seus recursos de forma desacoplada.
- **Foco em processos de software:** Indica se a abordagem se restringe a automação de processos de software ou se é aplicável a qualquer tipo de processo. Quando uma abordagem não está focada em processos de software, várias características existentes nos mesmos são desprezadas e características desnecessárias para software, mas necessárias para processos de negócio são incorporadas, dificultando o uso da abordagem no contexto de desenvolvimento de software.
- **Protótipo implementado:** Indica se a abordagem apresenta um protótipo ou se a implementação foi postergada para trabalhos futuros. Abordagens com protótipos implementados usualmente são mais aderentes à realidade, pois mostram como as idéias são passíveis de construção. Obviamente que a partir dessa construção seria necessário realizar validações para se obter indícios de que as mesmas trazem benefícios.

Os critérios acima, apesar de não formarem um conjunto completo nem necessariamente suficiente, servem como um guia para a elaboração de uma abordagem que se proponha a ser abrangente e que, se possível, traga novas contribuições em relação às abordagens atualmente disponíveis na literatura, que é o caso da abordagem proposta nesse trabalho.

3 A máquina de processos Charon

Visando a construção de uma máquina de processos [5] que atenda aos critérios citados anteriormente, optamos pela utilização da tecnologia de agentes inteligentes. As características dos agentes permitem que a máquina de processos atenda especificamente aos critérios de pró-atividade e facilidade de extensão. Entretanto, a construção de agentes inteligentes requer uma infra-estrutura para o seu suporte.

A abordagem proposta, descrita pela arquitetura exibida na Figura 1, consiste em permitir que o processo seja modelado graficamente dentro de um ambiente de desenvolvimento de software (ADS), como exibido na Figura 1.a. O ADS utilizado na implementação da máquina de processos Charon foi o Ambiente Odyssey [6], que visa prover uma infra-estrutura de suporte à reutilização baseada em modelos de domínio. A reutilização de software, no Ambiente Odyssey, ocorre através dos processos de Engenharia de Domínio, que tem o objetivo de construir componentes reutilizáveis que solucionem problemas de domínios de conhecimento específicos, e de Engenharia da Aplicação, que tem o objetivo de construir aplicações em um determinado domínio, reutilizando os componentes genéricos já existentes. O Ambiente Odyssey utiliza extensões de diagramas da UML para representar o conhecimento de um domínio, e permite que esses diagramas sejam reutilizados para facilitar a construção de aplicações.

Após a modelagem do processo, é necessário fornecer mecanismos para a sua instanciação, que ocorre através da tradução do modelo gráfico para uma linguagem executável, como exibido na Figura 1.b. A linguagem executável escolhida para esse mapeamento foi o Prolog, por ser uma linguagem declarativa e baseada em inferência. A linguagem Prolog também fornece a separação necessária entre o processo modelado, que é representado por fatos Prolog, e os planos de execução desse processo, que são representados

por regras de inferência Prolog. A estrutura de inferência Prolog facilita a aplicação das regras de execução em fatos de um determinado processo modelado.

Após o mapeamento, toda a informação sobre o processo está descrita na base de conhecimento e pode ser acessada por qualquer entidade que conheça a notação utilizada na sua descrição. Essa notação, que define como os elementos do processo devem ser descritos na base de conhecimento para que seja possível a representação inequívoca do processo modelado, é conhecida como ontologia [7]. Desta forma, neste trabalho o termo "ontologia" será utilizado para expressar a conceitualização utilizada para a representação dos processos de software utilizando Prolog e servirá como a linguagem de comunicação entre os elementos que desejam interagir com a base de conhecimento para proporcionar a execução do processo.

Com o objetivo de possibilitar a execução do processo contido na base de conhecimento, agentes inteligentes devem se conectar à base, alterando-a se necessário, como exibido na Figura 1.c.

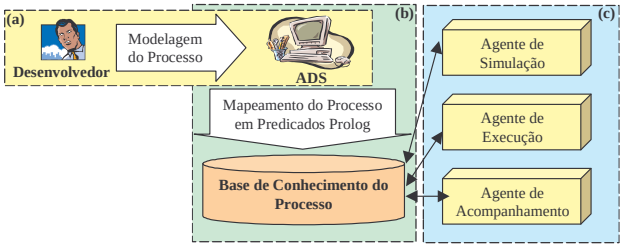

Figura 1: Arquitetura da abordagem Charon.

Os fatos Prolog formam a base de conhecimento do processo, que é o local no qual agentes inteligentes irão interagir. Inicialmente, três agentes básicos são necessários:

- **Agente de Simulação:** Responsável por verificar o processo modelado quanto a sua corretude, permitindo ou não a sua entrada em execução. Este agente será detalhado na seção 3.3;
- **Agente de Execução:** Responsável por verificar o estado da base de conhecimento, procurando por atividades finalizadas ou decisões que foram tomadas, permitindo o andamento do processo. Este agente será detalhado na seção 3.4;
- **Agente de Acompanhamento:** Responsável por interagir com o desenvolvedor, indicando as atividades que estão pendentes e as decisões que devem ser tomadas. Também permite a finalização dessas atividades ou a tomada dessas decisões. Este agente será detalhado na seção 3.4.

Para que a arquitetura seja extensível, optamos pela definição de uma infra-estrutura que provesse um agente genérico com as propriedades desejáveis de agentes inteligentes, que são: autonomia, interação, adaptação, aprendizado, mobilidade e colaboração. Para construir um novo agente basta especializar o agente genérico, definindo os planos e objetivos, e utilizando a ontologia para tornar possível a comunicação do novo agente com os demais agentes existentes através da base de conhecimento.

Neste contexto, portanto, podemos identificar as seguintes atividades, apresentadas esquematicamente na Figura 2: (1) modelagem, (2) instanciação, (3) simulação, (4) execução, (5) monitoramento e (6) evolução do processo. A modelagem do processo consiste na estruturação das atividades necessárias para a sua execução, identificando os recursos existentes para que essas atividades possam ser executadas. A instanciação do processo consiste na seleção de um processo modelado e início da sua simulação. A simulação do processo consiste na verificação da corretude do mesmo, permitindo que a sua execução seja iniciada. A execução do processo permite que os desenvolvedores verifiquem a lista de trabalhos pendentes, executem esses trabalhos e, posteriormente, notifiquem o seu término à máquina de processos, para que seja possível dar andamento a outros trabalhos subseqüentes. O monitoramento do processo consiste em exibir o estado atual da execução do processo para fornecer subsídios aos gerentes para que possam estimar como está o andamento da execução. A evolução do processo permite que modificações no processo modelado sejam incorporadas em suas instâncias em execução.

A base de conhecimento, que contém todas as informações sobre o processo, permite que agentes se conectem a ela, efetuem suas inferências e se desconectem, possibilitando que outros agentes possam se conectar. A arquitetura proposta prevê a execução concomitante de diversas instâncias de processos. Uma instância de processo representa um determinado processo em execução no desenvolvimento de um software. Assim, caso várias aplicações estejam sendo desenvolvidas utilizando o mesmo processo, existirão várias instâncias desse processo em execução, uma para cada aplicação em desenvolvimento. Cada instância de processo em execução implicará na criação de uma base de conhecimento própria. Entretanto, somente um agente poderá estar conectado a uma determinada base de conhecimento em um instante de tempo, para evitar problemas de acesso concorrente às informações. Através da propriedade de mobilidade os agentes poderão se locomover de uma base de conhecimento para outra, se desconectando da base de conhecimento anterior e se conectando na nova base de conhecimento.

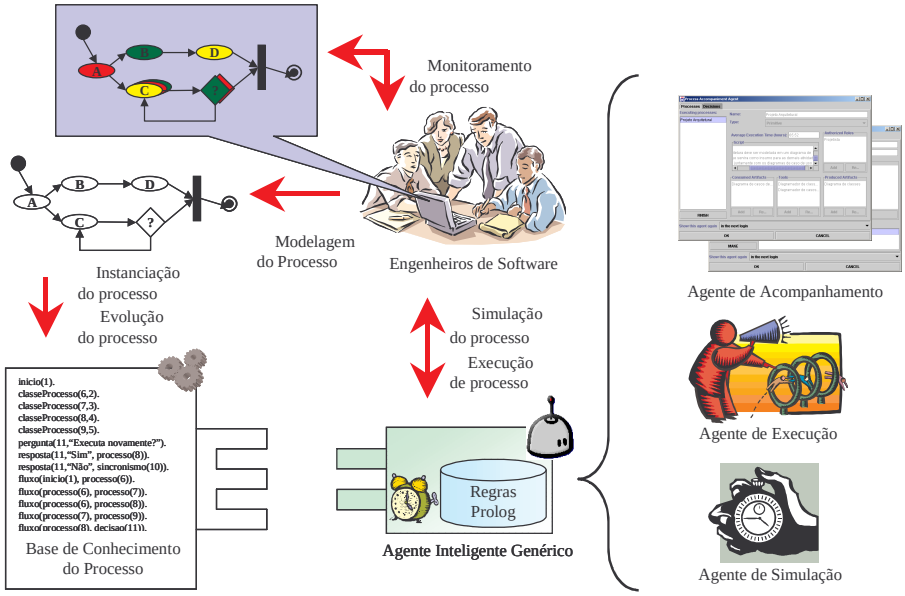

Figura 2: Atividades envolvidas na abordagem Charon.

Nas demais sub-seções serão detalhados os elementos contidos na arquitetura, que dão suporte para a modelagem, instanciação, simulação, execução, monitoramento e evolução do processo, e um exemplo de utilização da máquina de processos.

3.1 Modelagem gráfica de processos

Processos podem ser representados através de linguagens de modelagem de processos ou meta-modelos de processos [8]. Neste trabalho, optamos por representar processos através de um meta-modelo simples, que contém os elementos comuns dos meta-modelos utilizados pelas abordagens existentes na literatura.

Para que esse meta-modelo possa ser utilizado, decidimos adotar uma notação gráfica para diagramar os seus elementos. Dentre as diversas notações gráficas existentes na literatura [9, 10, 11], optamos por utilizar uma extensão do diagrama de atividades da UML [11]. A extensão desse diagrama tem sido foco de pesquisa por vários grupos [12, 13], pois além de ser simples, possibilita a definição de processos de software utilizando uma notação semelhante a que é utilizada no próprio desenvolvimento de software durante a atividade de análise de requisitos.

O meta-modelo de processos utilizado tem como vantagem a sua simplicidade. A simplicidade traz como benefício um isolamento dos detalhes inerentes ao meta-modelo que poderiam dificultar a avaliação das contribuições e limitações reais da abordagem proposta, cujo foco está na máquina de execução do processo. Esse meta-modelo tem como elemento central o processo. Existem dois tipos de processos: processos primitivos e processos compostos. Os **processos primitivos** representam as atividades atômicas que podem

ser executadas dentro do ambiente. Esse tipo de processo descreve o roteiro que deve ser seguido para a sua execução, as ferramentas que devem ser utilizadas, os papéis que agrupam desenvolvedores aptos para a execução e os artefatos consumidos e produzidos pela atividade. Os **processos compostos** representam atividades complexas, que contêm um *workflow* descrevendo como as suas sub-atividades devem ser executadas segundo a notação estendida do diagrama de atividades da UML. Esta notação contém os seguintes elementos:

- **Início ( ):** Elemento onde se dará o início da execução do *workflow* do processo composto;
- **Atividade ( ):** Elemento principal do *workflow*, que pode ser um processo primitivo ou composto, possibilitando a decomposição recursiva de um processo;
- **Decisão ( ):** Elemento que permite a escolha, dentre um conjunto de opções, de quais fluxos serão ativados para prosseguir a execução do *workflow* do processo composto;
- **Sincronismo ( ):** Elemento que permite sincronizar um conjunto de fluxos de processos. O sincronismo pode ser utilizado tanto para definir que diversos fluxos de saída devem ser ativados em paralelo, quanto para definir que o próximo elemento só será executado após a ativação de todos os fluxos de chegada;
- **Término ( ):** Elemento onde se dará o término da execução do *workflow* do processo composto;
- **Fluxo ( ):** Ligação entre dois elementos (não fluxos) do *workflow* representando a ordem de precedência necessária para sua execução.

A extensão do diagrama de atividades da UML visa diminuir as restrições impostas, como, por exemplo, a impossibilidade de associar mais de um fluxo de saída a um elemento. Uma característica adicionada a esse meta-modelo para possibilitar a reutilização dos processos é permitir que a hierarquia de sub-processos não tenha limitação de níveis. Desta forma, um processo pode estar em diferentes *workflows*, podendo ser tratado tanto como um processo global quanto como uma atividade específica. Para que isso fosse possível, foi necessário permitir a construção de repetição[1] através de recursão dentro da hierarquia de processos.

3.2 Instanciação de processos

A instanciação do processo acontece em três diferentes momentos do seu ciclo de vida. Inicialmente, ocorre a instanciação dos processos através do mapeamento das suas informações modeladas graficamente para Prolog e da seleção de quais desenvolvedores do ambiente irão exercer quais papéis. O mapeamento do processo gráfico em predicados Prolog consiste em criar um identificador numérico único para cada elemento modelado e utilizar esse identificador como chave na construção dos predicados Prolog. Para que posteriormente seja possível a obtenção do mapeamento inverso, é necessário manter uma tabela de mapeamento que informe qual predicado Prolog representa qual elemento gráfico, e vice-versa.

Cada processo modelado pode aparecer em diversos *workflows* de outros processos, tornando necessário diferenciar a definição de um processo do seu desenho em um *workflow*. Para permitir essa diferenciação, chamaremos de “classe de processo” a definição de um processo, e de “processo” o processo desenhado em um *workflow*. Desta forma, as propriedades de um processo, como nome, roteiro, ferramentas associadas, *workflow* associado, etc. serão definidas na classe do processo, pois todos os processos desenhados que pertencerem a essa classe compartilharão essas propriedades.

Durante a execução do processo, cada vez que um sub-processo deve ser executado, ocorre uma instanciação dinâmica, que permite identificar unicamente a instância criada dentro do ambiente, possibilitando que existam várias instâncias em execução pertencentes a um mesmo processo. Isso é necessário para permitir a construção de recursões e fluxos de retorno nos *workflows*. Caso a instanciação fosse completa no momento do mapeamento, não seria possível determinar quando o fluxo de retorno ou a recursão param, pois essa informação está atrelada a decisões que serão tomadas pelos desenvolvedores durante a própria execução.

---

[1] A repetição de um processo pode ser modelada tanto com um fluxo de retorno dentro do *workflow* que o processo está desenhado quanto com uma referência recursiva para o próprio processo dono do grafo, ou para um outro ascendente na hierarquia de processos, o que atende ao critério “Grafo com ciclos ou recursão” levantado na seção 2.

O terceiro momento de instanciação ocorre quando o modelo gráfico do processo é evoluído e se deseja que essa evolução seja refletida em uma determinada instância em execução, como será discutido na seção 3.6.

3.3 Simulação de processos

A simulação dos processos ocorre através do Agente de Simulação, que tem sua execução reativa iniciada assim que o mapeamento do processo é finalizado. Este tipo de simulação consiste na execução simulada do processo raiz e dos seus sub-processos, visando determinar o tempo médio de execução de cada processo composto, utilizando como entrada os tempos médios de simulação dos processos primitivos e das decisões e as probabilidades de seleção dos fluxos da saída das decisões, que são informados pelo desenvolvedor durante a modelagem gráfica do processo. Como a simulação executa o processo, podem ser encontrados erros de modelagem durante esse procedimento. Os seguintes erros podem ser detectados:

- *Workflow* de processo sem nenhum caminho entre algum nó de início e algum nó de término;
- Soma das probabilidades de seleção de fluxos saindo de decisão diferente de 100%;
- Possibilidade de repetição infinita, tanto através de fluxo de retorno quanto através de recursão.

Para que o agente possa terminar a sua execução mesmo que o processo não esteja correto, seu mecanismo pró-ativo monitora a simulação, verificando se ela está executando por um tempo muito grande e notificando ao desenvolvedor que pode existir um erro na modelagem do processo.

3.4 Execução de processos

A execução dos processos é possibilitada pela interação entre os agentes de execução e de acompanhamento com os desenvolvedores de software.

O agente de execução é o principal agente da máquina de processos. Ele conhece o que é necessário para controlar a execução de um processo, repassando o fluxo da execução para os elementos corretos quando um determinado elemento termina a sua execução. Esse agente atua tanto de forma reativa, cadastrando-se como escutador (elemento *listener*, ou *observer* do padrão *Observer*[2]) do agente de acompanhamento e verificando se algo novo foi inserido na base de conhecimento do processo, quanto de forma pró-ativa, verificando se alguma base está pronta para entrar em execução e dando início a essa execução. A atuação pró-ativa do agente consiste em verificar se alguma base de conhecimento já passou com sucesso pelo agente de simulação. Caso positivo, é criada a primeira instância desse processo. Já a atuação reativa do agente consiste em tratar o evento de desconexão do agente de acompanhamento de uma base de conhecimento. O tratamento desse evento é composto pela busca de alterações na base que sofreu a desconexão. Essas alterações podem ser ou um processo primitivo que foi finalizado, ou uma decisão que foi tomada. Em ambos os casos, o agente de execução deve verificar quais são os próximos elementos que entrarão em execução e dar prosseguimento a essas execuções. Após todo o tratamento de evento, é verificado se a instância inicial do processo raiz chegou ao seu término de execução, o que motivaria a finalização do processo principal.

O agente de acompanhamento tem como objetivo fazer a ponte de comunicação entre a máquina de processo e os desenvolvedores, informando aos desenvolvedores quais elementos estão pendentes e adicionando à base de conhecimento as ações tomadas pelos desenvolvedores. Esse agente atua de forma reativa, através de um menu do ambiente que o desenvolvedor pode acessar para verificar quais são as suas pendências. Quando o desenvolvedor acessa o agente, é possível configurar um perfil que indica como aquele desenvolvedor deseja que o agente se comporte. Dentre as opções, é possível pedir para o agente aparecer sempre que o desenvolvedor entrar no ambiente de desenvolvimento que está sendo guiado por algum processo, de tempos em tempos, ou nunca. Se o perfil do desenvolvedor indicar que o agente não deve aparecer nunca, o agente passa a ter um comportamento puramente reativo para o desenvolvedor em questão. A execução pró-ativa do agente consiste em verificar qual é o desenvolvedor atual do ambiente e acessar o seu perfil, vendo como esse desenvolvedor deseja que o agente se comporte. Caso, segundo o perfil, esteja na

[2] O padrão *Observer* define uma estrutura onde elementos podem se cadastrar como interessados em receber notificação quando algo ocorrer com um determinado elemento. Esse padrão também é conhecido como *Publisher-Subscriber* [25].

hora do agente interagir com o desenvolvedor, isso acontecerá de forma análoga ao que aconteceria se o desenvolvedor selecionasse o menu de ativação do agente. Durante a interação, o agente fornece a lista de processos atribuídos ao desenvolvedor que estão atualmente em execução. Caso o desenvolvedor deseje, o agente fornece informações detalhadas sobre como executar um determinado processo da lista. O agente também fornece a lista de decisões atribuídas ao desenvolvedor que estão pendentes de resposta. Caso o desenvolvedor deseje, o agente lista a pergunta e todas as respostas possíveis para a decisão que deve ser tomada.

3.5 Monitoramento de processos em execução

Após a instanciação do processo, a sua execução é iniciada, tornando necessário o uso de uma notação que permita visualizar o seu estado atual de execução. Essa notação deve ser capaz de expressar, de forma clara, quantas vezes cada processo de um *workflow* foi executado, como foi o desempenho dessas execuções em relação à previsão obtida pela simulação do processo e quais processos estão em execução no momento. Essas informações são úteis para os gerentes do projeto detectarem gargalos no processo e tomarem as devidas providências, entre elas: redimensionar a equipe, realocar os desenvolvedores em outros papéis ou otimizar o próprio processo minimizando os gargalos.

Para atender a esses requisitos, optamos por utilizar uma notação que se sobrepõe à notação estendida do diagrama de atividades da UML. Essa notação utiliza o diagrama construído na etapa de modelagem do processo, modificando o desenho dos elementos para passar a semântica desejada.

Para representar as várias versões de execução de um processo, utilizamos a sobreposição dos desenhos do processo adicionada a uma convenção de cores que indica como foi a execução, onde verde (tom de cinza intermediário, caso impresso em preto e branco) representa uma execução dentro do tempo simulado, e vermelho (tom de cinza escuro, caso impresso em preto e branco) representa uma execução com tempo pior do que o tempo simulado. Para representar os processos que estão em execução no momento, utilizamos a cor amarela (tom de cinza claro, caso impresso em preto e branco). A Figura 3 exibe um processo em execução sendo monitorado.

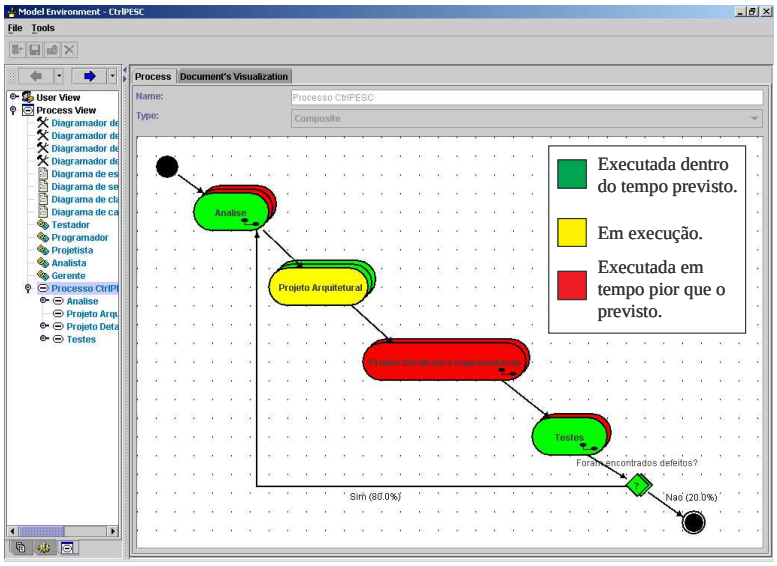

Figura 3: Monitoramento de processo em execução.

3.6 Evolução de processos

A arquitetura proposta também permite que os processos em execução evoluam com o tempo. Quando o processo é colocado em execução, acredita-se que a sua modelagem foi feita tentando refletir as reais necessidades do projeto que será guiado pelo processo. Entretanto, tanto a modelagem pode ter sido falha

quanto os requisitos do projeto podem ter mudado, o que pode implicar na modificação do processo em execução. Uma outra motivação para evoluir um processo é a adoção de políticas que visem a sua otimização.

A evolução do processo consiste em uma reinstanciação, onde os fatos sobre a execução em andamento são mantidos e o processo modelado é mapeado novamente, gerando uma nova representação na base de conhecimento compatível com os fatos anteriores sobre a execução.

3.7 Utilização da máquina de processos Charon

A utilização da máquina de processos Charon pode ser descrita através de um processo simples, composto pelas seguintes atividades:

1. Inicialmente, é necessário modelar o processo graficamente;
2. A partir de um processo modelado, é possível iniciar a construção de um domínio ou de uma aplicação através da instanciação do processo;
    a. A instanciação inicia pela seleção do processo raiz;
    b. Após a seleção, deve-se definir quais papéis serão exercidos por quais desenvolvedores;
3. Com o processo instanciado, o Agente de Simulação tenta executar a simulação e, a partir do resultado, informa se é possível dar início a execução real do processo;
4. O Agente de Execução coloca o processo raiz em execução, permitindo que a engenharia de domínio ou de aplicação possa ser iniciada;
5. Quando o desenvolvedor entrar no ambiente de desenvolvimento do domínio ou da aplicação, o Agente de Acompanhamento irá informar quais atividades estão pendentes e quais decisões devem ser tomadas;
6. Sempre que alguma atividade pendente for finalizada ou alguma decisão for tomada, o Agente de Execução será acionado para dar continuidade à execução do processo;
7. Durante a execução:
    a. O ambiente de monitoramento do processo poderá ser acessado para verificar quais atividades já foram executadas, quantas vezes foram executadas e como foi a execução em relação à previsão gerada pelo Agente de Simulação;
    b. Um menu poderá ser acionado para verificar as atividades e decisões pendentes;
    c. Um menu poderá ser acionado para retroceder o processo para um determinado instante de tempo;
    d. Um menu poderá ser acionado para reinstanciar o processo, refletindo alterações no seu modelo;
8. Quando o processo raiz for finalizado, o Agente de Acompanhamento informará que o processo de desenvolvimento terminou. Entretanto será possível reiniciá-lo através de um retrocesso, perdendo as informações do que foi feito, ou de uma reinstanciação, mantendo os dados anteriores sobre o processo.

4 Comparação com outras abordagens

As abordagens para automação de processos de software descritas na literatura podem ser divididas em três tipos [8]. As abordagens **baseadas em máquinas de estado ou redes de Petri**, como, por exemplo, SPADE [14], Memphis [15] e ProSoft [16], enfatizam a modelagem gráfica do processo, entretanto dificultam a modelagem de dependências específicas entre atividades [8], pois usualmente a semântica dos seus modelos é pobre. A execução do processo normalmente ocorre no próprio *workflow*, o que aumenta o acoplamento entre os dados do processo e da sua execução. As abordagens **baseadas em agentes, regras ou scripts**, como, por exemplo, HyperCode [17] e CAGIS [18], fornecem a flexibilidade necessária para a construção de qualquer tipo de *workflow*, entretanto dificultam o projeto do mesmo, pois a ausência de um ambiente gráfico para modelagem torna o seu entendimento complexo. Em outros casos, a ferramenta contém o *workflow* embutido no seu código, não permitindo a criação ou modificação do processo. As abordagens **híbridas**, que combinam notação gráfica com regras, como, por exemplo, EPOS [19], Merlin [20], ProNet/ProSim [21], Dynamite [13], APEL [22] e Mokassin [23], fornecem um ambiente propício para a modelagem e um mecanismo que permite

a transformação do *workflow* modelado em regras de evento-condição-ação, que são mais facilmente tratadas por computadores no momento da execução do *workflow*.

A Tabela 1 [5] classifica as abordagens de acordo com os critérios apresentados na seção 2. Os campos preenchidos com ✓ indicam que os critérios em questão são cobertos totalmente pela abordagem. Os campos preenchidos com ✗ indicam que os critérios em questão não são atendidos pela abordagem. Os campos preenchidos com ? indicam que não foi possível inferir se o critério é atendido pela abordagem através da documentação disponível. Ao analisar a Tabela 1 é possível constatar que nenhuma das abordagens que têm protótipo implementado, com exceção da Charon, atende aos requisitos de "Interação pró-ativa", "Processos reutilizáveis" e "Extensão flexível" concomitantemente.

Tabela 1: Quadro comparativo entre as abordagens descritas nessa seção.

Critérios	Máquinas de estado ou redes de Petri			Agentes, regras ou scripts		Híbridas (Gráfica + Regras)						
	SPADE	Memphis	ProSoft	HyperCode	CAGIS	EPOS	Merlin	ProNet / ProSim	Dynamite	APEL	Mokassin	Charon
Modelagem do processo	✓	✓	✓	✗	?	✓	✓	✓	✓	✓	✓	✓
Grafo com cíclico e recursão	✓	✗	?	✓	✗	?	?	✓	✓	✓	✓	✓
Mapeamento automático	✗	✓	✓	✓	?	✓	✓	✗	✗	✓	✓	✓
Interação pró-ativa	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✓	✓
Processos reutilizáveis	✗	✓	?	✗	✗	✗	✗	✗	✓	✗	✓	✓
Extensão flexível	✓	✗	✗	✗	✓	✗	✓	✗	✗	✓	?	✓
Foco em processos de software	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✗	✓
Protótipo implementado	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✗	✓

Além das abordagens descritas nesta seção existem várias outras, de igual importância, tal como ADELE, ALF, APPL/A, DesignNet, Entity, FunSoft, HFSP, Marvel, MVP-L e Oikos. Comparações envolvendo algumas das abordagens descritas nesta seção com as demais abordagens podem ser encontradas em *surveys* existentes na literatura [24].

5 Conclusões

As abordagens tradicionais para a automação de processos de software agem de forma reativa, respondendo a eventos gerados pelos desenvolvedores. Essas abordagens geralmente não têm preocupações referentes à reutilização de processos e à extensão da própria máquina de processos, permitindo que novos requisitos sejam atendidos com facilidade.

Para suprir essa necessidade, especificamos e construímos uma máquina de processos, que fornece as seguintes características:

- Modelagem gráfica de processos, que permite a utilização de ciclos ou recursões, seguindo uma extensão do diagrama de atividades da UML como notação;
- Reutilização através da possibilidade da aplicação de um mesmo processo em diferentes contextos e níveis de abstração;
- Mapeamento automático do processo da notação gráfica para uma base de conhecimento Prolog, segundo uma ontologia definida;
- Simulação do processo modelado antes do início da execução, permitindo a detecção de erros sintáticos de modelagem e indicando possíveis caminhos para a correção;
- Mecanismo para a execução do processo, descrito em regras de inferência Prolog;
- Acompanhamento do processo através de *worklists* que descrevem quais atividades ou decisões estão pendentes para um determinado desenvolvedor;

- Monitoramento da execução do processo através de um diagrama estendido do diagrama de atividades da UML, que utiliza cores para denotar o estado da execução e sobreposição para denotar o número de execuções anteriores dos processos;
- Evolução de processos via um mecanismo de reinstanciação que permite que modificações em processos em execução sejam refletidas sem que seja necessário reiniciar o processo;
- Suporte a pró-atividade e extensão da máquina de processos através da arquitetura baseada em uma infra-estrutura para a construção de agentes inteligentes;
- Implementação da máquina de processos dentro do Ambiente Odyssey, utilizando a linguagem Java combinada com a máquina de inferência Prolog JIP, permitindo a instanciação de processos para a engenharia de domínio e engenharia de aplicação através da seleção do processo raiz e da associação entre os papéis e os desenvolvedores;

Entretanto, algumas limitações puderam ser detectadas nesse trabalho, que motivam a elaboração de trabalhos futuros, possibilitando a evolução da abordagem proposta. No tocante à reutilização de processos, pode ser criado um novo tipo de processo, chamado processo abstrato, que consiste em permitir que a decisão de qual processo utilizar em um determinado contexto possa ser postergada para o momento da instanciação do processo raiz. Um processo abstrato é um elemento que não pode entrar em execução, mas pode ser substituído por um processo primitivo ou composto no momento da instanciação.

Outra melhoria no suporte a reutilização seria a criação de processos dependentes de domínio, que consiste em fornecer um ambiente para a modelagem de processos dentro de um domínio de aplicação, onde processos seriam gerados levando em conta as características funcionais do domínio. No momento da criação de uma aplicação dentro desse domínio, ocorreria a junção entre os processos genéricos com os processos dependentes de domínio, permitindo a construção do processo raiz da aplicação. Atualmente, é possível criar processos dependentes de domínio no mesmo local onde são criados os processos genéricos, o que quebra o encapsulamento do domínio em questão.

A versão atual da abordagem não interfere na execução das ferramentas, pois atua somente como um guia que sugere qual ferramenta deve ser executada em um determinado momento. Entretanto, seria interessante controlar o estado do ambiente através da informação de quais ferramentas estão sendo executadas por quais desenvolvedores. Esse controle seria útil para permitir o monitoramento do que cada desenvolvedor fez em cada ferramenta e restringir, caso desejado, o uso de uma ferramenta em um determinado período de tempo.

Todas as informações referentes à execução do processo são armazenadas na base de conhecimento do processo, entretanto, a única externalização dessas informações ocorre através do diagrama que permite o monitoramento da execução do processo. Seria interessante a construção de um agente para a especificação de relatórios gerenciais, onde o próprio gerente utilizasse uma linguagem declarativa para descrever que tipo de informações gostaria nesse relatório. Caso essa linguagem declarativa fosse o próprio Prolog, a consulta seria processada pela máquina de inferência do ambiente e enviada a um *parser*, que seria responsável pela formatação final do relatório. Desta forma, seria possível obter uma visão micro da execução do processo, em contrapartida à visão macro existente através do diagrama de monitoramento da execução do processo.

Agradecimentos

Os autores gostariam de agradecer ao CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) pelo investimento financeiro neste trabalho e aos demais integrantes do Projeto Odyssey pelo apoio durante o seu desenvolvimento.

Referências Bibliográficas

[1] JENNINGS, N. R., 2000, "On agent-based software engineering", *Artificial Intelligence*, v. 177, n. 2, pp. 277-296.

[2] FRANKLIN, S., GRAESSER, A., 1996, "Is it an Agent, or Just a Program? - A Taxonomy for Autonomous Agents". In: *Third International Workshop on Agent Theories, Architectures and Languages*, pp. 21-35, Budapest, Hungary.

[3] GARCIA, A. F., SILVA, V. T., LUCENA, C. J. P., et al., 2001, "An Aspect-Based Approach for Developing Multi-Agent Object-Oriented Systems". In: *XV Simpósio Brasileiro de Engenharia de Software*, pp. 177-192, Rio de Janeiro, Brasil, October.

[4] WOOLDRIDGE, M. J., JENNINGS, N. R., 1995, "Intelligent agents: Theory and practice", *The Knowledge Engineering Review*, v. 10, n. 2, pp. 115-152.

[5] MURTA, L. G. P., 2002, *Charon: Uma Máquina de Processos Extensível Baseada em Agentes Inteligentes*, Dissertação de M.Sc., COPPE, UFRJ, Rio de Janeiro, Brasil .

[6] WERNER, C. M. L., BRAGA, R. M. M., MATTOSO, M. L. Q., et al., 2000, "Infra-estrutura Odyssey: estágio atual". In: *XIV Simpósio Brasileiro de Engenharia de Software, Seção de Ferramentas*, pp. 366-369, João Pessoa, Brasil.

[7] GRUBER, T. R., 2002, "What is an Ontology?". In: http://www-ksl.stanford.edu/kst/what-is-an-ontology.html, Accessed in 07/01/2002.

[8] JOERIS, G., HERZOG, O., 1998, *Towards Object-Oriented Modeling and Enacting of Processes*. Center for Computer Technologies, University of Bremen.

[9] WFMC, 1999, "Interface 1: Process Definition Interchange". In: http://www.wfmc.org/standards/docs/TC-1016-P_v11_IF1_Process_definition_Interchange.pdf, Accessed in 06/12/2001.

[10] MAYER, R. J., MENZEL, C. P., PAINTER, M. K., et al., 1995, *Information Integration for Concurrent Engineering (IICE) IDEF3 Process Description Capture Method Report*. Knowledge Based Systems, Inc.

[11] OMG, 2001, "OMG Unified Modeling Language Specification version 1.3". In: http://www.omg.org/cgi-bin/doc?formal/01-03-01.pdf, Accessed in 06/12/2001.

[12] OKTABA, H., ESQUIVEL, C., 2001, "Process Diagrams: An Extension of UML Activity Diagrams for the Modeling of Software Processes". In: *IV Workshop Iberoamericano de Engenharia de Requisitos e Ambientes de Software (IDEAS 2001)*, pp. 31-40, Heredia, Costa Rica.

[13] HEIMANN, P., JOERIS, G., KRAPP, C., et al., 1996, "DYNAMITE: Dynamic Task Nets for Software Process Management". In: *18th International Conference on Software Engineering*, pp. 331-341, Berlin, Germany.

[14] BANDINELLI, S., FUGGETTA, A., GHEZZI, C., et al., 1994, *SPADE: An Environment for Software Process Analysis, Design and Enactment*. Research Studies Press.

[15] VASCONCELOS, F. M., WERNER, C. M. L., 1998, "Organizing the Software Development Process Knowledge: An Approach Based on Patterns", *International Journal of Software Engineering and Knowledge Engineering*, v. 8, n. 4, pp. 461-482.

[16] REIS, R. Q., REIS, C. A. N. D. J., 1999, "Ambiente de Desenvolvimento de Software PROSOFT: Evolução e Estágio Atual". In: *Semana de Informática (SEMINF'99)*, Universidade Federal do Pará.

[17] PERRY, D. E., PORTER, A., VOTTA, L. G., et al., 1996, "Evaluating workflow and process automation in wide-area software development". In: *European Workshop on Software Process Technology*, pp. 188-193, Berlin, Germany.

[18] WANG, A. I., HANSSEN, A. A., NYMOEN, B. S., 2001, "Design Principles For A Mobile, Multi-Agent Architecture For Cooperative Software Engineering". In: http://citeseer.nj.nec.com/342416.html, Accessed in 20/11/2001.

[19] JACCHERI, L., LARSEN, J., CONRADI, R., 1992, "Software Process modeling and Evolution in EPOS". In: *4th International Conference on Software Engineering and Knowledge Engineering (SEKE'92)*, pp. 574-581, Capri, Italy.

[20] JUNKERMANN, G., PEUSCHEL, B., SCHÄFER, W., et al., 1994, "Merlin: Supporting Cooperation in Software Development through a Knowledge-based Environment ". In Nuseibeh, B., Finkelstein, A., and Kramer, J., Taunton, Inglaterra, John Wiley and Sons.

[21] CHRISTIE, A., 1995, *Software Process Automation: The Technology and Its Adoption*, Berlin, Springer-Verlag Publishing.

[22] DAMI, S., ESTUBLIER, J., AMIOUR, M., 1998, "APEL: A graphical yet executable formalism for process modelling", *Automated Software Engineering: An International Journal*, v. 5, n. 1, pp. 61-96.

[23] JOERIS, G., 2001, "Mokassin". In: http://www.informatik.uni-bremen.de/grp/mokassin, Accessed in 18/11/2001.

[24] ARMENISE, P., BANDINELLI, S., GHEZZI, C., et al., 1993, "A Survey and Assessment of Software Process Representation Formalisms", *International Journal of Software Engineering and Knowledge Engineering*, v. 3, n. 3, pp. 401-426.

[25] GAMMA, E., HELM, R., JOHNSON, R., et al., 1994, *Design Patterns: Elements of Reusable Object-Oriented Software*, Massachusetts, Addison Wesley.