DiSEN-CI: Um Servidor de Integração Contínua de Modelos

Guilherme da C. Silva[1] **e Elisa H. M. Huzita**[1]

[1]Universidade Estadual de Maringá (UEM), Maringá, PR, Brasil

guinobody@hotmail.com, elisa.h.m.huzita@gmail.com

Abstract. *Many companies choose for Distributed Software Development (DSD), once it brings benefits like cost reduction and productivity increase. Together with the benefits, come also challenges regarding coordination, communication and control. Model Driven Development (MDD) has models as main artifacts. Although it's adoption increases development productivity, to take advantage from it, working with models, adjustments in software development are requireds. This paper presents a Continuous Integration Server (CIS) for models. It aims to mitigate the challenges of DSD and also offer support to MDD.*

Resumo. *Muitas empresas optam pelo Desenvolvimento Distribuído de Software (DDS), por este trazer benefícios como redução de custos e aumento de produtividade. Junto com os benefícios vêm os desafios de coordenação, comunicação e controle. O Desenvolvimento de Software Dirigido por Modelos (DSDM) possui como artefato principal modelos. Sua adoção aumenta a produtividade do desenvolvimento, porém para utilizá-lo são necessárias adaptações no ambiente para poder trabalhar com modelos. Este artigo apresenta um Servidor de Integração Contínua (SIC) para modelos, que tem como objetivo mitigar os desafios de DDS e oferecer apoio ao DSDM.*

1. Introdução

Atualmente, os desenvolvedores de software têm a sua disposição uma série de ferramentas tais como interpretadores, *frameworks* para testes, sistemas de controle de versão, servidores de integração contínua, etc. Algumas destas ferramentas oferecem apoio para trabalhos colaborativos, estando integradas a outras ferramentas, enviando os mais diferentes tipos de informações, trabalhando em uma determinada atividade e, possivelmente, em um Ambiente de Desenvolvimento Distribuído de Software (ADDS).

O Desenvolvimento Distribuído de Software (DDS) traz consigo várias vantagens como: maior competitividade, redução de custos e aumento de produtividade. Juntamente com as vantagens, vêm os desafios, relacionados a: dificuldades de comunicação, controle e coordenação [Lindqvist et al. 2006]. Em um cenário onde existem dois times atuando em um projeto, sendo um time alocado na sede da empresa e o outro geograficamente disperso, observa-se uma maior dificuldade em gerenciar o time disperso [da Silva et al. 2012]. Uma maneira de reduzir tais dificuldades é utilizar ferramentas colaborativas. Estas ferramentas podem realizar trabalhos desgastantes para um desenvolvedor, como integrar o código gerado pelas diversas equipes, montar a aplicação a partir do código e executar diversos tipos de testes na aplicação. Uma ferramenta que realiza as tarefas mencionadas, de forma autônoma, é um Servidor de Integração Contínua (SIC).

SICs possuem a missão de integrar o trabalho realizado de forma paralela, construir (compilar) o sistema e testá-lo. Desta forma, uma tarefa geralmente complexa e que é deixada para o final do processo de desenvolvimento, se torna automática e é realizada diariamente. Com isso, a integração do sistema se torna menos trabalhosa e os erros de programação podem ser detectados antecipadamente, reduzindo, assim, os custos com retrabalho. Logo, não há necessidade do desenvolvedor esperar a finalização da tarefa, podendo continuar seu trabalho enquanto o SIC é executado. Para manipular arquivos de código-fonte já existem vários SICs disponíveis tanto *open source*, como [Hudson 2013], [CruiseControl 2013] e [ApacheContinuum 2013], quanto proprietários, porém não foram encontrados *plugins* que oferecessem suporte a modelos.

Pesquisas sobre Desenvolvimento de Software Dirigido por Modelos (DSDM) têm recebido grande atenção. Nesta abordagem o elemento central é o modelo. Apesar dos esforços de organizações como a *Object Management Group* no sentido de padronizar o DSDM, as ferramentas atuais estão longe de oferecerem um ambiente rico e integrado como o do desenvolvimento dirigido por código-fonte. Isto acontece pelo fato de que dentre as ferramentas que oferecem apoio ao DSDM, muitas delas não estão integradas. Com isso, o desenvolvedor fica incumbido de realizar as tarefas praticamente manualmente.

O objetivo deste artigo é apresentar um SIC para modelos. Com isto, espera-se oferecer apoio em ADDSs e, também, reduzir a falta de integração entre ferramentas de DSDM.

Na Seção 1 foi feita uma contextualização acerca dos temas deste artigo. A Seção 2 reúne alguns conceitos das principais áreas do conhecimento relativas ao SIC proposto. A Seção 3 apresenta o projeto do SIC proposto, denominado *Distributed Software Engineering Environment - Continuous Integration* (DiSEN-CI). A Seção 4 compara o SIC com outros trabalhos semelhantes. Por fim na Seção 5 está a conclusão do trabalho.

2. Revisão Bibliográfica

Esta Seção contém uma revisão bibliográfica sobre os temas mais relevantes deste trabalho que são: Desenvolvimento Distribuído de Software, Integração Contínua e Desenvolvimento de Software Dirigido por Modelos.

2.1. Desenvolvimento Distribuído de Software

Projetos de software podem adotar algum tipo de distribuição, seja ela entre organizações, locais geográficos ou mesmo de artefatos. Com isso, tanto o gerenciamento quanto o desenvolvimento se tornam mais complexos e surgem vários desafios como a comunicação e a colaboração entre os participantes [Gumm 2005]. Isto tem motivado estudos em buscar soluções para esses desafios.

Segundo [Lindqvist et al. 2006], os desafios no DDS podem ser visualizados através de dois prismas: a distância, podendo esta ser temporal, geográfica ou sócio-cultural; e o processo, o qual é dividido em comunicação, coordenação e controle. Combinando os desafios da distância com as do processo, são obtidos nove possíveis desafios que podem estar presentes em projetos distribuídos.

No estudo de [Huzita et al. 2008], são apresentadas 12 possíveis soluções para a combinação de desafios de [Lindqvist et al. 2006]. Na Tabela 1 pode-se visualizar a

distribuição das soluções de [Huzita et al. 2008]. As soluções são S1, oferecer mecanismos para facilitar a comunicação; S2, desenvolver o modelo de produtos; S3, gerenciar processo/projeto; S4, praticar co-alocação temporária; S5, estabelecer critérios para a constituição de equipes e encorajar o senso de equipe; S6, disponibilizar e compartilhar informações de projeto; S7, lidar com heterogeneidade; S8, distribuir responsabilidades; S9, apoiar a colaboração por meio de *awareness* e *group awareness*; S10, distribuir atividades; S11, definir métricas e; S12, estabelecer sentimento de confiança.

Tabela 1. Soluções para DDS [Huzita et al. 2008]

Processo	Distância Temporal	Distância Geográfica	Distância Sócio-cultural
Comunicação	S6, S1, S9	S1, S4, S2	S6, S4, S5, S12, S9
Coordenação	S6, S3, S7, S9	S7, S4, S12, S10, S8, S9, S2	S4, S5, S12, S8
Controle	S6, S3, S7	S7, S12, S10, S8, S2	S7, S12, S8

2.2. Integração Contínua

No processo de desenvolvimento de software, várias atividades devem ser realizadas até o sistema estar pronto para ser implantado. Como exemplo, podem ser citadas as atividades de compilação, integração de módulos, execução de testes e preparação do instalador. Muitas destas atividades são custosas e propensas a erros por parte dos desenvolvedores. Em organizações ou projetos que adotam o DDS, estas atividades se tornam mais complexas do que no desenvolvimento co-alocado. Visando reduzir o esforço dos desenvolvedores e quantidade de erros dessas atividades repetitivas, a prática da Integração Contínua (IC) vêm sendo adotada na indústria, principalmente em grupos que utilizam as metodologias ágeis [Duvall et al. 2007].

A IC pode ser definida como uma prática de desenvolvimento de software onde os membros dos times integram frequentemente seus trabalhos. Geralmente cada pessoa integra pelo menos uma vez por dia, desta forma gerando múltiplas integrações por dia. Como em cada integração é executada uma construção do sistema, os erros são detectados com antecedência e os desenvolvedores são informados na mesma hora, reduzindo o custo com retrabalho e os possíveis problemas de integração encontrados após um longo ciclo de desenvolvimento [Fowler 2006]. A IC faz parte da Gerência de Configuração de Software, que é um conjunto de atividades de acompanhamento e controle que começa quando o projeto tem início e só termina quando o software é retirado de operação [Pressman 2005].

A Figura 1 apresenta um cenário de como funciona um Servidor de Integração Contínua (SIC) e seus componentes. Primeiramente, (1) os desenvolvedores enviam os artefatos para um Sistema de Controle de Versão (SCV). Em um intervalo de tempo predeterminado, por exemplo, 1 minuto, o SIC verifica se houve alguma alteração no sistema consultando o SCV (2). Caso alguma alteração seja encontrada, então o SIC deve executar um *script* de construção (3), o qual pode abranger: compilação, integração de módulos, execução de testes, geração de métricas e até a implantação do sistema. Por fim, o SIC deve informar o desenvolvedor dos erros encontrados durante a construção ou se o sistema está pronto para ser implantado (4).

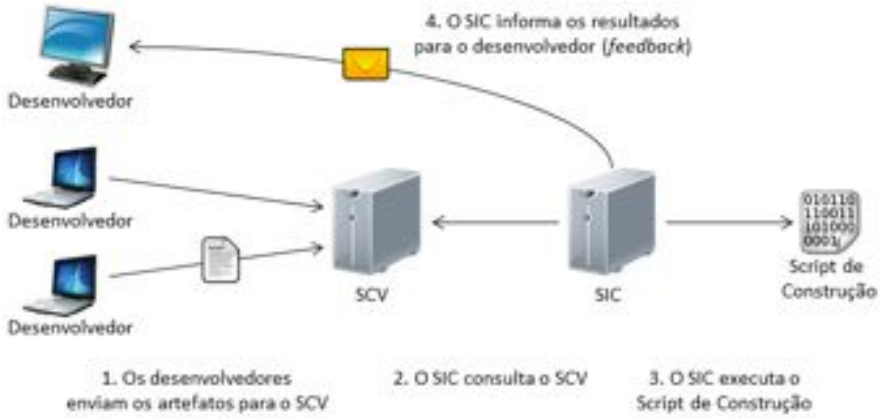

Figura 1. Cenário de um Servidor de Integração Contínua

2.3. Desenvolvimento de Software Dirigido por Modelos

O Desenvolvimento de Software Dirigido por Modelos (DSDM) tem como artefato principal os modelos. Desta forma, o nível de abstração no qual os desenvolvedores trabalham é superior, transformando os próprios modelos em um sistema executável automaticamente. No contexto de DSDM, um modelo consiste em um conjunto de elementos que descreve de forma abstrata uma realidade [Mellor et al. 2004].

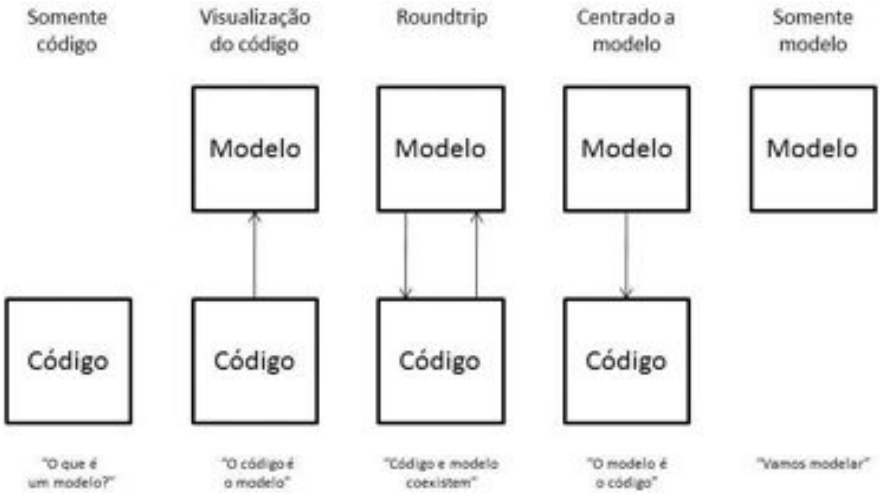

Figura 2. Espectro de abordagens de modelagem [Brown et al. 2005]

Existem várias maneiras dos desenvolvedores trabalharem com modelos. Cada abordagem exige uma forma diferente de sincronização entre os modelos e o código-fonte. Na Figura 2 pode ser observado um espectro contendo essas formas de se trabalhar. Na primeira abordagem simplesmente não existem modelos no processo de desenvolvimento. Já na segunda, que é a mais usual, os modelos são utilizados para se ter uma visão mais abstrata do código-fonte, podendo esta ser por meio de uma engenharia reversa. Na abordagem *Roundtrip*, existe o desenvolvimento tanto no nível de modelos quanto no nível de código-fonte. Isto faz com que seja necessário existir um mecanismo de sincronização entre modelos e código-fonte. No quarto tipo de desenvolvimento, o modelo é o artefato

principal, no qual os desenvolvedores modelarão o sistema e que este será transformado em código-fonte. Por fim, existe o desenvolvimento em que o modelo é transformado diretamente em executável. Neste caso, o modelo também pode ser interpretado ao invés de compilado [Brown et al. 2005].

A organização *Object Management Group* possui uma iniciativa para padronizar este tipo de desenvolvimento. Esta padronização chama-se *Model Driven Architecture* (MDA) [Group 2012]. Visando um elevado grau de interoperabilidade, a MDA se restringe a definir somente a especificação do DSDM, desta forma, deixando livre para a indústria implementar suas próprias soluções [Brown et al. 2005]. Na Figura 3 encontram-se as fases do desenvolvimento de software adotando a abordagem MDA. A diferença com o desenvolvimento tradicional são basicamente os artefatos gerados em cada fase. A princípio é desenvolvido o Modelo Independente de Plataforma (PIM – *Plataform Independent Model*). Este modelo reflete o domínio do problema e é o que possui o nível mais alto de abstração. O próximo artefato é o Modelo Específico de Plataforma (PSM – *Plataform Specific Model*). Ele é o resultado da transformação do PIM para um nível de abstração dependente de plataforma, desta forma refletindo o domínio da solução. A última transformação é o código-fonte, o qual é gerado a partir do PSM [Kleppe et al. 2003].

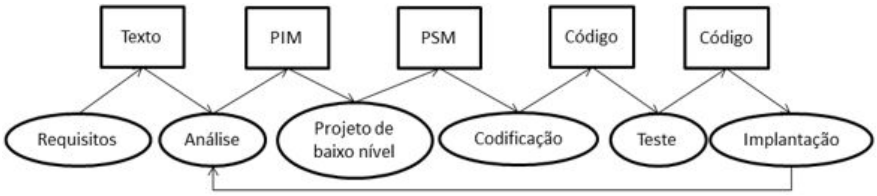

Figura 3. Ciclo de vida do desenvolvimento de software com MDA [Group 2012]

3. DiSEN-CI

O *Distributed Software Engineering Environment*(DiSEN) é um ambiente integrado que visa reduzir os desafios de projetos distribuídos. Ele oferece apoio à cooperação, à formação de equipes virtuais, à persistência e à comunicação [Huzita et al. 2008]. Neste trabalho foi criado o sistema DiSEN - *Continuous Integration* (DiSEN-CI) com o objetivo de somar ao ambiente, o apoio à integração contínua tanto de arquivos de código-fonte quanto à modelos. Esse sistema foi planejado para atender as necessidades do desenvolvimento global, no que diz respeito à comunicação, coordenação e controle. O sistema possui um controle aprimorado para projetos que utilizam o DSDM e, também, possui um controle básico para projetos centrados em código-fonte.

O DiSEN-CI tem papel de integrador de diversas tarefas realizadas pelas ferramentas do ambiente de desenvolvimento. Essas ferramentas seriam tanto para trabalhar com código-fonte quanto para modelos. A seguir as três principais características que o DiSEN-CI possui:

- ser uma ferramenta colaborativa para aumentar o compartilhamento das informações em um ambiente distribuído;
- ser autônoma, não exigindo o acompanhamento de uma pessoa durante a execução do DiSEN-CI, para reduzir o esforço gasto pelos desenvolvedores; e

- oferecer mecanismos que possibilitem a manipulação tanto de código-fonte quanto de modelos.

Pelas características do DiSEN-CI, é possível observar que as soluções S2, desenvolver o modelo de produtos; S6, disponibilizar e compartilhar informações de projeto; e S9, apoiar a colaboração por meio de *awareness* e *group awareness*, da Tabela 1 são atendidas, dessa forma colaborando com a mitigação dos desafios de DDS. Para obter estas características, o DiSEN-CI combinou as áreas de DDS, IC e DSDM. A Figura 4 contém uma visão geral do DiSEN-CI. Este é executado utilizando uma das formas de inicialização. Após iniciar a execução, para cada projeto, é criado um Gerenciador de Execução do Projeto (GEP). O GEP executa o ciclo principal de funcionamento, que é verificar e recuperar os artefatos utilizando o SCV; processar os dados, realizar a leitura, transformação e testes nos modelos; notificar os interessados, enviar mensagens para as equipes distribuídas e; armazenar os resultados, permitindo assim, consultas futuras aos mesmos. As subseções seguintes explicarão em detalhes o funcionamento do DiSEN-CI.

Figura 4. Visão geral do DiSEN-CI

3.1. Gerenciador de Execução do Projeto

O DiSEN-CI poderá ser executado como *stand-alone*, integrado com algum outro SIC ou juntamente com o DiSEN. O DiSEN-CI não depende do DiSEN ou de outro SIC para ser executado. Sendo executado como *stand-alone*, seriam carregadas as configurações globais e configurações de cada projeto por meio de arquivos *eXtensible Markup Language* (XML). Caso o sistema seja executado por um outro SIC, haverá uma integração entre eles para um poder usufruir das funcionalidades do outro. Foram escolhidos os seguintes SICs: Hudson, CruiseControl e Apache Continuum. Todos possuem suporte ao *script* Ant. Por meio deste *script* serão executadas as funcionalidades desejadas do DiSEN-CI. A vantagem de utilizar outro SIC é que o usuário poderá utilizar os *plugins* de outros SIC e também o DiSEN-CI como se fosse um *plugin*. Mesmo com a integração ainda terá a necessidade de configurar os projetos por XML. No último caso, que é ser executado pelo ambiente DiSEN, este terá total acesso às funcionalidades por meio do código-fonte Java. Como as informações de projeto já estarão cadastradas no DiSEN, então a utilização de XMLs será opcional. Além da manipulação do DiSEN-CI pelo DiSEN, ele também poderá ser beneficiar com as informações de contexto, como quem, quando e porque foram alterados os artefatos, vindas do SCV capturadas pelo sistema. As configurações globais existentes são os locais aonde procurar as configurações de projeto e as classes padrão que serão utilizadas. Desta forma, caso um projeto queira utilizar a implementação padrão, não há necessidade de especificar as classes.

Cada projeto registrado no sistema será independente, sendo executado em sua própria *thread*. Não haverá troca de informações direta entre projetos, embora seja possível que eles utilizem o mesmo banco de dados para armazenar suas informações. Cada projeto terá somente um GEP, que será responsável pela coordenação dos quatro módulos principais do DiSEN-CI, que são: SCV, processador, notificador e armazenador. Estes módulos é que vão realizar as atividades do DiSEN-CI e coincidem com o fluxo de execução padrão do GEP: verificar alterações no projeto, processar os dados, notificar os interessados e armazenar as informações. Cada um dos módulos serão registrados nas configurações do projeto e terão pontos de extensão que permitirá o desenvolvedor alterar e inserir novos comportamentos no sistema. Juntamente com cada elemento estarão seus parâmetros obrigatórios e opcionais e o nome de sua classe concreta. Caberá ao GEP instanciar as respectivas classes, podendo elas estarem dentro do DiSEN-CI ou em arquivos *Java ARchive* (JAR) externos. O carregamento de JARs externos será feito dinamicamente, permitindo desta forma uma execução contínua, não sendo necessário interromper o trabalho das equipes distribuídas para configurar projetos. Embora não seja aconselhado, o GEP poderá ser especializado caso o desenvolvedor queira alterar o fluxo padrão de execução.

3.2. Módulo: SCV

É altamente recomendado que o ambiente possua um SCV. Por meio deste, é possível utilizar um repositório de artefatos que armazenará os arquivos de projeto. O DiSEN-CI trabalha com repositórios tanto convencionais quanto específicos para modelos, sendo necessário apenas implementar a interface que integra com o SCV. Pelo desenvolvimento ser distribuído, é comum utilizar um repositório distribuído. O DiSEN-CI possuirá uma integração com os SCVs: Mercurial, Subversion e Git.

O integrador do SCV terá o papel de verificar se houve alterações no projeto e caso haja fazer um download das alterações juntamente com as informações de contexto (quem alterou, quando alterou, o que alterou e porque alterou). Caso o SCV também dê suporte, poderá ser escolhido qual versão fazer download. A preferência será para o Mercurial por ser distribuído e por já ser utilizado no DiSEN. Mesmo sendo comum hoje a utilização de um SCV, o DiSEN-CI oferecerá suporte a projetos sem SCV, apesar de não ser adequado no DDS. Neste caso, o sistema seria configurado para ficar verificando a pasta do projeto de tempos em tempos procurando por alterações.

3.3. Módulo: Processador

Após verificar se houveram alterações no projeto e fazer download delas, o GEP irá chamar os processadores. Cada projeto poderá ter vários processadores e eles serão registrados nas configurações do projeto. A ordem de chamada também deverá ser definida nas configurações, havendo possibilidade de execução tanto em série quanto em paralelo. Um processador realiza basicamente duas coisas: recuperar dados relevantes para ele e processar estes dados. Como exemplo de processadores podem ser citados: construção de projeto, testes automatizados, coleta de métricas e empacotamento (criar um executável). Na Figura 5 é apresentada a estrutura padrão de um processador. Ele primeiro carrega os dados, por meio de leitores e transformadores, para então, processar estes dados, podendo utilizar ferramentas externas.

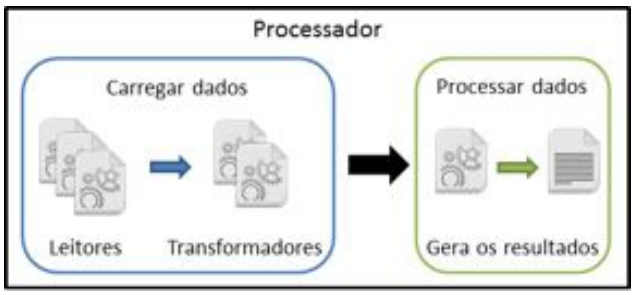

Figura 5. Estrutura interna do processador padrão

Havendo a necessidade de criar processadores para tarefas em que a estrutura padrão não é adequada, o desenvolvedor terá a liberdade de criar seu processador desde o início. A estrutura padrão do processador contém pontos de extensão, deixando assim o desenvolvedor livre para especializar apenas uma parte de seu comportamento, reutilizando o restante. A maneira como são recuperados os dados geralmente é específica de cada processador, entretanto esta tarefa poderá ser delegada para outras classes que poderão ser nativas do DiSEN-CI, diminuindo o esforço na implementação dos processadores. O mesmo vale para a transformação dos dados. A única parte que o desenvolvedor é obrigado a implementar é o processamento real dos dados. Na primeira versão do sistema existirá um processador nativo para executar testes automatizados.

Após o processador ser iniciado, ele realizará a leitura dos dados. O leitor a ser utilizado deverá ser definido nas configurações do projeto. O leitor terá acesso a todos os arquivos do projeto e, por padrão, ele começará a busca na raiz do projeto. Pelas configurações é possível definir uma outra pasta para ser o início da busca. O leitor padrão do DiSEN-CI utilizará uma busca em profundidade para encontrar os arquivos e filtrará os resultados utilizando expressões regulares no nome e/ou no conteúdo dos arquivos. Caso existam vários leitores dentro de um mesmo processador, então o resultado será a união dos resultados individuais. O leitor padrão para modelos do DiSEN-CI filtra arquivos no formato *XML Metadata Interchange* (XMI). Atualmente, o DiSEN-CI oferece suporte ao diagrama de classes da UML e à Object Constraint Language.

Uma vez encontrados os arquivos, existirá a possibilidade de aplicar diversas transformações neles. O transformador de cada processador deve ser especificado nas configurações do projeto e seu papel principal será transformar o arquivo para o formato que o processador consiga utilizar. Além de preparar o arquivo, o transformador também pode acrescentar informações nos dados coletados como se fosse um pré-processamento. Desta forma os arquivos que forem inválidos já poderão ser descartados nesta etapa. As transformações não são limitadas em 1 para 1, podendo fundir vários arquivos em uma única estrutura de dados e também separando um arquivo em vários. O DiSEN-CI não irá alterar os arquivos do projeto nesta etapa do processamento, então todas as transformações ficarão somente em memória. O sistema terá nativamente um transformador para chamadas em cadeia de transformadores, que poderá ser utilizado quando for necessário realizar várias transformações seguidas. Para modelos, será nesta etapa que eles poderão ser transformados em código-fonte.

Tanto os leitores quanto os transformadores do DiSEN-CI poderão ser reutiliza-

dos em vários processadores. Para reduzir a necessidade de adaptação dessas classes, o desenvolvedor poderá utilizar as classes existentes e criar apenas um novo transformador registrando-o como último transformador a ser utilizado pelo processador. Desta forma, antes dos dados irem para o processamento real, eles poderão sofrer as alterações desejadas. Uma outra alternativa é, invés de criar um transformador, realizar a transformação diretamente no processamento. Isto pode ocorrer no caso de transformações que dificilmente serão reutilizadas.

Com os dados já carregados pelo processador, inicia-se o processamento, de fato, dos dados. Cada processador terá sua própria lógica. O DiSEN-CI terá, em sua primeira versão, dois processadores nativos para realizar testes automatizados em código-fonte e em modelos da *Unified Modeling Language* (UML). A seguir a descrição de cada um.

- para os testes unitários em código-fonte Java: será utilizado um leitor de arquivos em Java e nenhum transformador. No processamento: será utilizado o *framework* JUnit para rodar os casos de testes;
- para os testes em modelos: será utilizado um leitor de arquivos XMI para ler o modelo, um leitor de arquivos XML para ler o casos de testes com uma notação específica do DiSEN-CI e um transformador para converter o XMI e o XML para a notação de uma ferramenta de teste de modelos. Com os dados já preparados será rodada a ferramenta de teste de modelos que poderá ser externa ao DiSEN-CI. A ferramenta padrão será o Alloy Analyzer [Dinh-Trong et al. 2006], por ser uma ferramenta madura para verificação de modelos, que possui vasta documentação e é desenvolvida em Java.

Tanto o processador de testes para código-fonte quanto para modelos terão pontos de extensão para o desenvolvedor alterar qual ferramenta deseja utilizar para realizar os testes e para alterar o leitor de casos de testes. Todas essas alterações deverão estar especificadas nas configurações do projeto. Após o processamento, a ferramenta deverá enviar os resultados encontrados para o GEP, que, por sua vez, irá repassá-los para os armazenadores e notificadores registrados.

3.4. Módulo: Notificador

A notificação dos interessados aumentará a colaboração entre as equipes e participantes. Cada participante poderá escolher receber ou não mensagens de cada notificador registrado no projeto. Cada notificador poderá ter vários meios de comunicação como *Short Message Service*, e-mail, chat e/ou o *framework* DiSEN.

O recebimento da notificação poderá ser limitado por equipes ou global. No modo limitado por equipe, as notificações são enviadas somente para os membros da mesma equipe e no modo global os participantes recebem notificações de todos. O gerente local poderá receber além das mensagens, um relatório diário ou por algum período de tempo predeterminado sobre a sua própria equipe e/ou global. O gerente global poderá receber as mensagens e um relatório global resumido por equipe.

3.5. Módulo: Armazenador

No final do ciclo de atividades do GEP está o armazenador de resultados. Este possibilitará visualizar os dados históricos do projeto, que poderão conter as informações

de contexto retiradas do SCV, o resultado dos processadores e as notificações enviadas. As formas de armazenamento padrão serão arquivos textos ou banco de dados (MySQL ou PostgreSQL). Os dados armazenados poderão ser, futuramente, minerados utilizando gestão do conhecimento para transformá-los em informações úteis ao gerente local e global. É, também, no armazenador que se encontram as possíveis integrações com outras ferramentas que estiverem interessadas nos dados gerados pelo sistema, como por exemplo o DiSEN.

Todos os pontos de extensão oferecidos pelo DiSEN-CI serão disponibilizados no formato de *Service Provider Interface* (SPI) utilizando classes abstratas. Todas as classes criadas pelo desenvolvedor que forem colocadas nas configurações do projeto deverão ser especializações concretas das classes que fazem parte da SPI. As classes criadas pelo desenvolvedor ficarão em um JAR separado do DiSEN-CI e serão carregadas dinamicamente, desta forma não havendo necessidade de alterar o código-fonte do DiSEN-CI.

A linguagem de programação escolhida foi a Java, devido ao DiSEN também estar em Java. A estrutura básica e as classes da SPI já estão finalizadas. Várias implementações como o integrador com o SCV, leitores, transformadores e o processador de casos de testes para modelos UML também já foram finalizadas.

4. Trabalhos Relacionados

Foi realizada uma pesquisa na literatura afim de verificar os estudos realizados nas interseções das três áreas DDS, DSDM e IC. Nesta Seção serão comentados os estudos encontrados.

No trabalho de [Blanc et al. 2005], foi criada uma especificação baseada em modelos para os fornecedores de ferramentas implementarem, sendo estes fornecedores qualquer um que desenvolva uma ferramenta de manipulação de modelos. Essa implementação seria um adaptador para conectar a ferramenta ao Model Bus, que é um integrador de ferramentas proposto pelos autores. O Model Bus é orientado a serviços e sua função é invocar, em cadeia, as funcionalidades oferecidas pelas ferramentas que implementam os adaptadores. Como exemplo seria um desenvolvedor chamar em cadeia as funcionalidades: carregar um modelo, realizar uma transformação de modelo para modelo e realizar uma transformação de modelo para texto. O Model Bus se encarregaria de transportar a saída de cada ferramenta para a entrada da outra. Apesar de permitir a interoperabilidade entre ferramentas, ele não possui as características do DDS e da IC. O trabalho é realizado localmente, não existindo a colaboração com a equipe e nem a integração com os trabalhos dos demais desenvolvedores. Também não há integração com o SCV e nenhuma outra maneira que possa captar as informações de contexto.

No trabalho de [Choi et al. 2007] foi criado um *framework* de testes chamado *Unified Test Environment* (UTE) com o objetivo de integrar várias ferramentas de testes de modelos. O UTE é capaz de: gerar casos de testes, realizar um mapeamento entre código-fonte e modelos, depurar modelos e processar XMI. De todas estas funcionalidades, a única que o DiSEN-CI não terá será a depuração de modelos, uma vez que ele não será executado localmente e sua execução não será acompanhada por uma pessoa. Semelhante ao Model Bus, o UTE é uma ferramenta individual e não possui as características do DDS e da IC.

5. Conclusão

Os desafios do DDS podem ser minimizados adotando as soluções propostas no trabalho de [Huzita et al. 2008]. Com base nas soluções propostas, o DiSEN-CI foi elaborado contemplando os itens S2, desenvolver o modelo de produtos; S6, disponibilizar e compartilhar informações de projeto; e S9, apoiar a colaboração por meio de *awareness* e *group awareness*. Devido ao DiSEN-CI ser um SIC, ele é capaz de integrar diversas ferramentas e executar, de forma autônoma, tarefas em um servidor. Sua capacidade de integração auxilia na redução da falta de integração das ferramentas de DSDM. Vale ressaltar que não foram encontrados *plugins* para os SICs existentes que possuíssem o foco na manipulação de modelos.

O DiSEN-CI beneficiará os projetos que adotam DDS disponibilizando informações sobre o projeto em tempo real e notificando interessados sobre os resultados dos testes. Estas funcionalidades promoverão *group awareness* com os integrantes dispersos geograficamente, aumentando a percepção e a colaboração das equipes distribuídas.

Outra área beneficiada será a do DSDM, uma vez que o sistema poderá integrar diversas ferramentas de manipulação de modelos e executá-las de forma autônoma. Leitores e transformadores de XMI estarão presentes nativamente facilitando a manipulação de diagramas UML. Um processador extensível para testes de modelos estará incluso na primeira versão do DiSEN-CI. Apesar do foco estar no apoio ao DSDM, o DiSEN-CI também será compatível com o desenvolvimento centrado à código-fonte.

Após o término da implementação será realiza a validação do DiSEN-CI utilizando Engenharia de Software Experimental e para posteriormente publicar os resultados. Também será redigido um manual do usuário para o DiSEN-CI e disponibilizado o sistema como *open source* sob a licença *Massachusetts Institute of Technology*. O intuito é incentivar mais pessoas a utilizarem e contribuírem com o desenvolvimento do DiSEN-CI. Como trabalhos futuros à implementação, serão especificados e implementados novos processadores para a geração de código-fonte e a integração de novas ferramentas de testes de modelos. Também poderá ser integrado ao sistema um componente de gestão de conhecimento.

Agradecimentos

Agradeço à CAPES pela concessão de bolsa de estudo.

Referências

ApacheContinuum (2013). Apache continuum continuous integration and build server. Technical report, http://continuum.apache.org/.

Blanc, X., Gervais, M.-P., and Sriplakich, P. (2005). Model bus: Towards the interoperability of modelling tools. In Aßmann, U., Aksit, M., and Rensink, A., editors, *Model Driven Architecture*, volume 3599 of *Lecture Notes in Computer Science*, pages 17–32. Springer Berlin Heidelberg.

Brown, A., Conallen, J., and Tropeano, D. (2005). Introduction: Models, modeling, and model-driven architecture (mda). In Beydeda, S., Book, M., and Gruhn, V., editors, *Model-Driven Software Development*, pages 1–16. Springer Berlin Heidelberg.

Choi, J., Islam, S., and Shankar, R. (2007). Unified test environment-integrated platform for bridging the modeling, testing and code development flow. In *Systems Conference, 2007 1st Annual IEEE*, pages 1 –7.

CruiseControl (2013). Cruisecontrol continuous integration. Technical report, http://cruisecontrol.sourceforge.net/.

da Silva, A. L. M., Cavalheiro, L. T. A., Roman, N. T., and Chaim, M. L. (2012). Implementação de metodologia de desenvolvimento Ágil em projetos com time alocado e não alocado. In *VIII Simpósio Brasileiro de Sistemas de Informação (SBSI 2012)*, pages 327 –336.

Dinh-Trong, T., Ghosh, S., and France, R. (2006). A systematic approach to generate inputs to test uml design models. In *Software Reliability Engineering, 2006. ISSRE '06. 17th International Symposium on*, pages 95–104.

Duvall, P. M., Matyas, S., and Glover, A. (2007). *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley.

Fowler, M. (2006). Continuous integration. Technical report, http://martinfowler.com/articles/continuousIntegration.html.

Group, O. M. (2012). Model driven archtecture. Technical report, http://www.omg.org/mda/.

Gumm, D. C. (2005). The phenomenon of distribution in software development projects: A taxonomy proposal. In *Proc. European Mediterranean Conf. Information Systems - EMCIS 05*, Cairo.

Hudson (2013). Hudson continuous integration. Technical report, http://hudson-ci.org/.

Huzita, E. H. M., da Silva, C. A., Wiese, I. S., Tait, T. F. C., Quinaia, M., and Schiavoni, F. L. (2008). Um conjunto de soluções para apoiar o desenvolvimento distribuído de software. In *II Workshop de Desenvolvimento Distribuído de Software*, pages 101–110, Campinas, Brasil.

Kleppe, A., Warmer, J., and Bast, W. (2003). *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley.

Lindqvist, E., Lundell, B., and Lings, B. (2006). Distributed development in an intranational, intra-organisational context: An experience report. In *Proc. of the 2006 International Workshop on Global Software Development for the Practitioner - GSD06*, pages 80–86, Shangai.

Mellor, S. J., Scott, K., Uhl, A., and Weise, D. (2004). *MDA Distilled: Principles of Model-Driven Architecture*. Addison-Wesley.

Pressman, R. S. (2005). *Software Engineering: A practitioners approach*. McGraw-HIll, sixth edition.