Universidade Federal do Rio de Janeiro

Escola Politécnica/COPPE

Express: otimizando o uso do MDArte

Autor:

Gabriel Alcantara Gebara Tavares

Orientador:

Prof. Sérgio Barbosa Villas-Boas, D. Sc.

Examinador:

Prof. Felipe Maia Galvão França, Ph. D.

Examinador:

Rafael Targino dos Santos, M. Sc.

Poli/Coppe

Maio de 2011

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
Escola Politécnica/Coppe
Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária
Rio de Janeiro – RJ CEP 21949-900

DEDICATÓRIA

Dedico este trabalho ao meu avô Roland por ser o grande homem que foi e o exemplo que quero seguir.

AGRADECIMENTO

Agradeço carinhosamente todo o apoio da minha noiva Luana durante os anos de grandes superação e conquistas.

Agradeço também o apoio do povo brasileiro. Depois de mais de cinqüenta anos pagando impostos para educação sem utilizar quaisquer serviços, a oportunidade de cursar uma faculade de ponta como a UFRJ será de extrema importância para meu futuro.

RESUMO

Conforme o hardware evolui, sistemas cada mais complexos são feitos, requerendo práticas mais avançadas de Engenharia de Software para geri-los e mantê-los. Uma importantíssima ferramenta, que surge como uma das tendências futuras nessa área, é o MDA, o processo de desenvolvimento de sistemas apoiado nos diagramas de especificação do sistema. Como veremos, ao mesmo tempo que as vantagens do MDA só se tornam mais claras no desenvolvimento de sistemas de grande porte, um grande custo computacional sobrecarrega diferentes etapas, e é de suma importância para o sucesso dessa arquitetura e a evolução da Engenharia de Software uma forma de otimizar o MDA e reduzir seus custos de adocão.

Palavras-Chave: MDA, desenvolvimento de softwares de grande porte, otimização.

ABSTRACT

As hardware evolves, growing complex systems are created, requiring more advanced Software Engineer knowledge to manage and maintain them. An important tool cited as one of the future trends in this area is MDA, the process of developing systems from system specification diagrams. While its advantages are better explored developing large systems, the computing overhead slows down the many life cycle stages and it’s utterly needed to improve it in order to enhance its usability and increase its spread.

Key words: MDA, development of large software, optimization.

SIGLAS

UFRJ – Universidade Federal do Rio de Janeiro

MDA – *Model Driven Architecture*

PIM – *Plataform-Independent Model*

PSM – *Plataform-Specific Model*

MOF – *MetaObject Facility*

OMG – *Object Management Group*

UML – *Unified Modelling Language*

JDK – *Java Development Kit*

GPL – *GNU General Public License*

SVN – *Subversion*

XML – *Extensible Markup Language*

XMI – XML *Metadata Interface*

SPB – Software Público Brasileiro

LPM – Licença Pública de Marca

CRUD – *Create, Retrieve, Update and Delete*

KB – *Kilobyte*

KBps – *Kilobytes per second*

GB – *Gigabyte*

RAM – *Random Access Memory*

IDE – *Integrated Development Environment*

POJO – *Plain Old Java Object*

Sumário

Lista de Figuras

No index entries found.

Lista de Tabelas

Capítulo 1
Introdução

1.1 – Tema

O trabalho utiliza conhecimentos de Engenharia de Software e de Processamento Distribuído para otimizar o funcionamento do framework MDArte, um *framework* para uso simplificado e extensível de uma plataforma MDA (*Model Driven Architecture*). A MDA basicamente é uma arquitetura para o desenvolvimento de sistemas baseada na geração da implementação a partir de diagramas UML que definirão todo o sistema. O design do sistema é todo feito através destes e devem cobrir todos os requerimentos funcionais do sistema, enquanto que a arquitetura para a qual a implementação está sendo gerada será escolhida de modo a atender aos requisitos não-funcionais, como escalabilidade e performance. Isso concentra o trabalho de construção de software na especificação e na lógica do negócio e não mais tanto na implementação e no uso de um determinado conjunto de tecnologias, diminuindo custos e erros na implementação e numa possível migração para diferentes plataformas e tecnologias.

O MDArte foi desenvolvido pela COPPETEC e traz grandes benefícios para o desenvolvimento de sistemas, em especial os de grande porte, porém, em parte devido pela escala desses sistemas e outra pelo seu funcionamento, há um grande custo de tempo envolvido. Se esse custo é contrabalanceado pela economia de tempo decorre das vantagens do MDA, é ainda extremamente importante estudar formas de reduzi-lo para tornar o framework mais atrativo e diminuir as barreiras para sua adoção.

1.2 – Delimitação

O objetivo do trabalho é criar um utilitário que diminua os tempos de processamento envolvidos na operação do framework, sem, no entanto, alterá-lo. Será desenvolvida uma versão inicial para ser usada internamente na COPPETEC, começando pela equipe de um determinado projeto em desenvolvimento, chamado

daqui em diante de PILOTO, que aos poucos se estenderá a outros projetos e estará disponível para projetos externos no Portal do Software Público Brasileiro, juntamente do MDArte.

Desse modo, a arquitetura do utilitário deverá permitir uma flexibilidade e adaptabilidade grande para que este suceda nos mais variados ambientes, sendo parte do trabalho tentar entender necessidades adicionais e, mesmo não as implementando, deixar espaço para que futuramente o sejam.

Deve haver ainda um trabalho exaustivo de documentação do código e do utilitário, objetivando assim uma fácil manutenção e evolução do utilitário por futuras equipes.

1.3 – Justificativa

É de suma importância para a Engenharia de Software descobrir novas formas de acompanhar o passo acelerado no qual as tecnologias têm envolvido e os sistemas têm se tornado mais complexos. Como a MDA surge como uma das mais fortes respostas a esse desafio, estudar formas de tornar todos seus interessantíssimos conceitos factíveis de serem realizados deve ser encarado como uma grande prioridade para esse campo de conhecimento.

O MDArte é um grande trabalho que vêm sendo feito por diversas mãos e, ano a ano, é melhorado pela equipe dedicada e interessados em suas monografias e dissertações. Hoje, desponta por todo esse esforço, pela sua arquitetura aberta e flexível e pela orientação clara e bem feita como um dos grandes *frameworks* baseados em MDA.

Partindo dessas premissas, podemos ver que a Engenharia de Software dará um passo significativo quando o MDArte for bom o bastante para ser adotado nas mais diversas instituições e empresas. Para isso, 3 grandes pontos devem ser trabalhados:

1) Deve ser completo o suficiente para atender às necessidades de especificação de requerimentos funcionais

2) Seu custo de adoção deve ser baixo o suficiente. Como é um *software* livre e não necessita de *hardware* específico, isso quer dizer que os custos de aprendizado devem ser reduzidos o máximo possível para facilitar sua adoção.

3) O custo envolvido na sua operação deve ser tratado da mesma forma que os custos de aprendizado para que o MDArte seja uma alternativa vantajosa cada vez mais vantajosa, contribuindo para sua adoção mais difundida e a evolução da Engenharia de Software.

O Express, nome dado ao utilitário que será feito, considerando o escopo definido, se aterá aos itens 2 e 3 da lista, enquanto que o primeiro deve ser buscado pela equipe de desenvolvimento do MDArte.

Como já permite o uso para sistemas web, usando Struts, EJB e Java, uma plataforma que atende a grande parte das demandas de sistemas hoje em dia, podemos ver que é crítica para que toda essa nova visão de Engenharia de Software se tornar realidade o desenvolvimento do Express e seu correto e eficaz atendimento às metas traçadas.

1.4 – Objetivos

Tendo em mente as razões que definem o porquê da importância e relevância do corrente trabalho, fica claro que os grandes objetivos do Express devem ser as reduções do custo de adoção e de uso do MDArte.

1.5 – Metodologia

Para que o objetivo seja alcançado, o primeiro passo é definir como de fato o utilitário atenderá seus objetivos. Foi definido que as duas primeiras semanas de agosto de 2009 seriam utilizadas para realizar entrevistas e *brainstorming* de idéias (primeiro item no diagrama mais abaixo), fechando com a definição inicial do funcionamento do Express através da priorização e estruturação das necessidades e idéias levantadas nessas duas semanas.

A grande prioridade seria um sistema de execução paralela dos comandos utilizados durante o ciclo de desenvolvimento (basicamente para realizar as transformações entre modelos UML e código e compilações do código gerado e do alterado) e que serão chamados de macro-processos daqui em diante. Uma vez que cada macro-processo demora entre 2 e 10 minutos (variando de acordo com o tipo das

instruções e do tamanho dos arquivos a serem processados) e essa é a principal fonte de custos associada ao uso do MDArte, torna-se claro que essa priorização é mais que adequada para a idéia supracitada ou para outras. Há assim algumas otimizações relacionadas que também serão adicionadas, como a execução automática de todas as etapas, uma após a outra, com a seguinte exibição de um alerta quando tudo acabar, enquanto que antes o usuário deveria estar atento para acionar próximas etapas, perdendo desde alguns segundos até alguns minutos quando decidia aproveitar o tempo com outra tarefa e se distraía.

Algumas outras idéias que buscavam facilitar o dia-a-dia do usuário, como um repositório de conhecimento e de scripts auxiliares de uso pouco freqüente também serão implementadas, porém com prioridade reduzida.

A interface gráfica da aplicação é uma outra grande preocupação, pois ela é responsável por repassar ao usuário final uma série de valores que são importantes para o MDArte, como modernidade, simplicidade no uso e confiabilidade, e requerimentos-chave foram determinados para que todos sejam atingidos.

A partir de tudo isso, foi fechada uma proposta que imediatamente foi discutida com a direção da empresa (segundo item no diagrama mais a frente). Desta foram definidos:

1) O escopo do projeto, com elaboração de uma primeira versão do manual do usuário. Se tratando de um utilitário com diversas limitações de recursos para seu desenvolvimento, diversas otimizações foram feitas para se obter um ciclo de vida enxuto e, por isso, optou-se a especificar em alto nível os recursos presentes no software e já documentá-los no formato de um manual de usuário para reduzir retrabalho.

2) A equipe envolvida no projeto, constituída de um analista\desenvolvedor e um gerente.

3) Uma janela de tempo para todo o trabalho de desenvolvimento do Express com base na disponibilidade dos recursos humanos envolvidos e que vai de agosto de 2009 até junho de 2010. Como a princípio todos os recursos humanos estarão envolvidos também com o desenvolvimento do projeto PILOTO, essa divisão de prioridades também foi levada em consideração durante a determinação dessa janela e a elaboração do cronograma final.

4) A plataforma tecnológica e ferramentas utilizadas: JDK 6 e Eclipse.

Com isso, já era possível determinar um cronograma para o projeto, que segue os seguintes prazos:

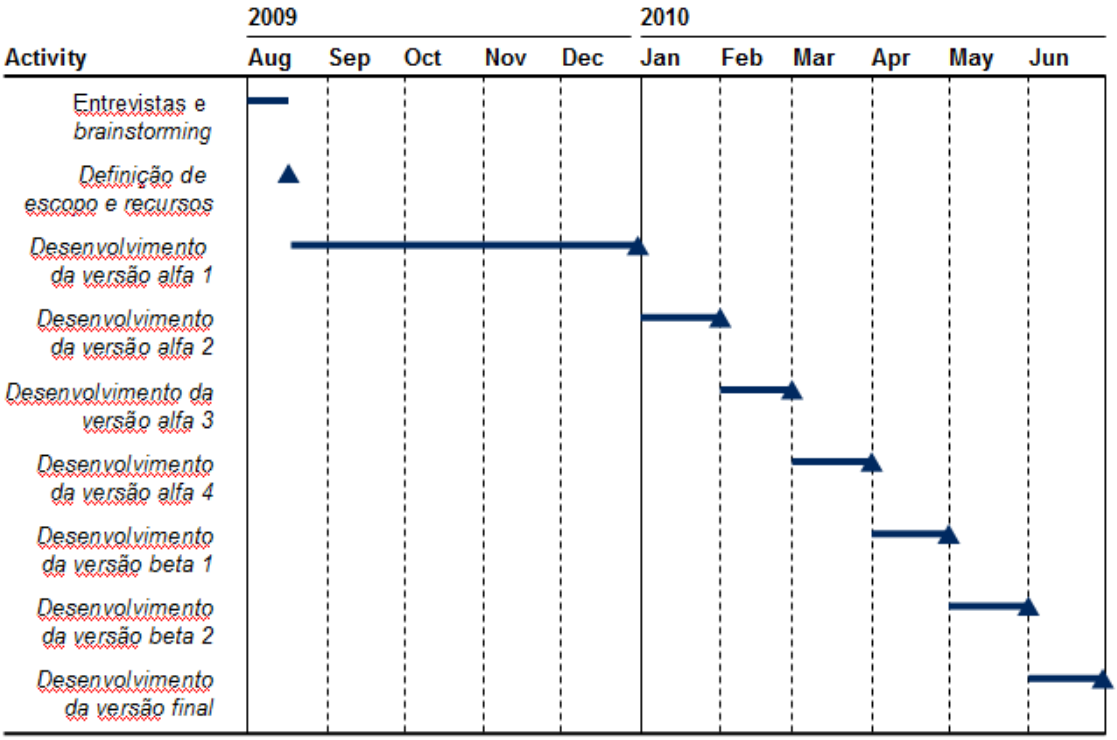


Figura 1.1 – Cronograma Gantt
Fonte: Equipe do projeto

O desenvolvimento foi quebrado em diversas etapas, todas com o mesmo funcionamento. Ao final de cada, é feita uma entrega para um grupo determinado de usuários. A primeira é mais longa, visto que toda as funções básicas, bibliotecas e recursos mínimos para uso devem ser desenvolvidos. Após esta, entra em funcionamento um sistema de relato e acompanhamento de funcionalidades pendentes, *bugs* e sugestões (chamados todos de tarefas na perspectiva do desenvolvedor daqui em diante). O sistema escolhido foi o Mantis (www.mantisbt.org), uma vez que segue a política do *software* aberto, é facilmente extensível e alterável e apresenta a seguinte lista extensa de recursos importantes para o projeto:

- Lienciado sob GPL
- Facilidade na instalação
- Intuitivo e fácil de usar
- Web
- Codificado em PHP
- Suporte a múltiplos projetos
- Suporte a autenticação de usuários via LDAP
- Atualização automática da lista de alterações por versão

- Priorização e alocação a versões das tarefas
- *Dashboard* customizável de tarefas por usuário
- Fácil integração às tecnologias existentes (Apache Server e banco de dados MySQL)
- Executa em Windows ou Linux
- Executa em Internet Explorer, Firefox ou Google Chrome perfeitamente

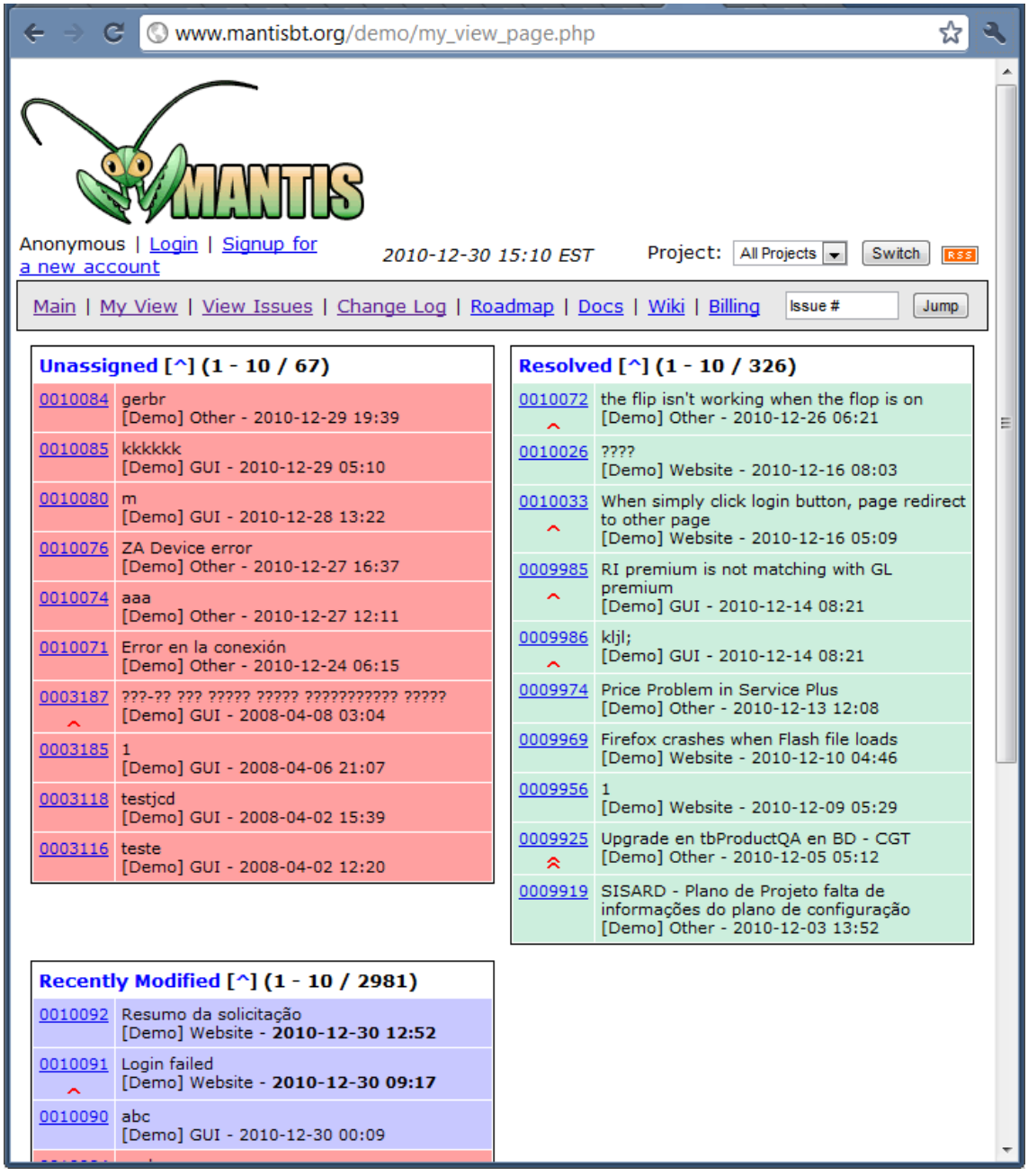


Figura 1.2 – Tela "Minha Visão" do Mantis visualizada no Google Chrome

Fonte: Site do Mantis

No começo de cada versão, é planejado o que deverá ser incluído nesta. *Bugs* críticos, que impeçam o funcionamento correto de funcionalidades fundamentais, são corrigidos imediatamente e uma nova versão intermediária é liberada. Isso é possível através do uso do SVN, um sistema de versionamento de projetos que possibilita que um projeto tenha diversas versões em paralelo. Foi estabelecido que deve sempre haver uma versão em desenvolvimento (chamada de dev) e uma outra que corresponda à última lançada, de modo que possa ser possível corrigir *bugs* críticos sem liberar recursos inacabados ou até mesmo novos defeitos.

Após o primeiro ciclo, quando a maioria das funcionalidades já seriam implementadas, seria possível realizar ciclos mais curtos, de um mês, para facilitar o planejamento e permitir um teste mais rápido dos recursos novos para encurtar o tempo total de desenvolvimento do utilitário.

As primeiras quatro versões são restritas a um grupo pequeno de testadores, pois há risco significante de falhas graves e estes devem ser capazes de entender todos os processos auxiliados pela ferramenta profundamente. A comunicação deve ser especialmente ágil e freqüente. Por isso são chamadas de alfa, seguindo conceito difundido da computação.

Já nos últimos 3 meses, é importante divulgar o software para um número maior de pessoas, liberando-se versões beta abertas seguindo a nomenclatura anterior. Praticamente todos os recursos previstos já devem estar implementados e estáveis e o objetivo é conseguir encontrar erros menos freqüentes que não foram ainda detectados através do teste com massa de usuários maior.

Nesse período, os usuários estarão restritos aos da equipe do projeto PILOTO, de modo a simplificar a variação de necessidades a serem atendidas. Nas fases alfa os usuários serão a equipe do Express e os gerentes e nas fases beta, toda a equipe do projeto. Apesar do framework ser o MDArte, há diversos projetos na COPPETEC cada qual com necessidades específicas diferentes e por isso foi feita essa escolha. Adequação e teste em outros projetos ficarão para uma segunda fase do Express, ainda por definir.

Todas as alterações e recursos nas versões liberadas devem ser refletidos no manual do usuário (versão final no Apêndice 1).

Por fim, será utilizada como IDE o Eclipse, software utilizado pela COPPETEC em seus projetos, inclusive aqueles criados através do MDArte, e muito popular entre os desenvolvedores e a força de trabalho da área.

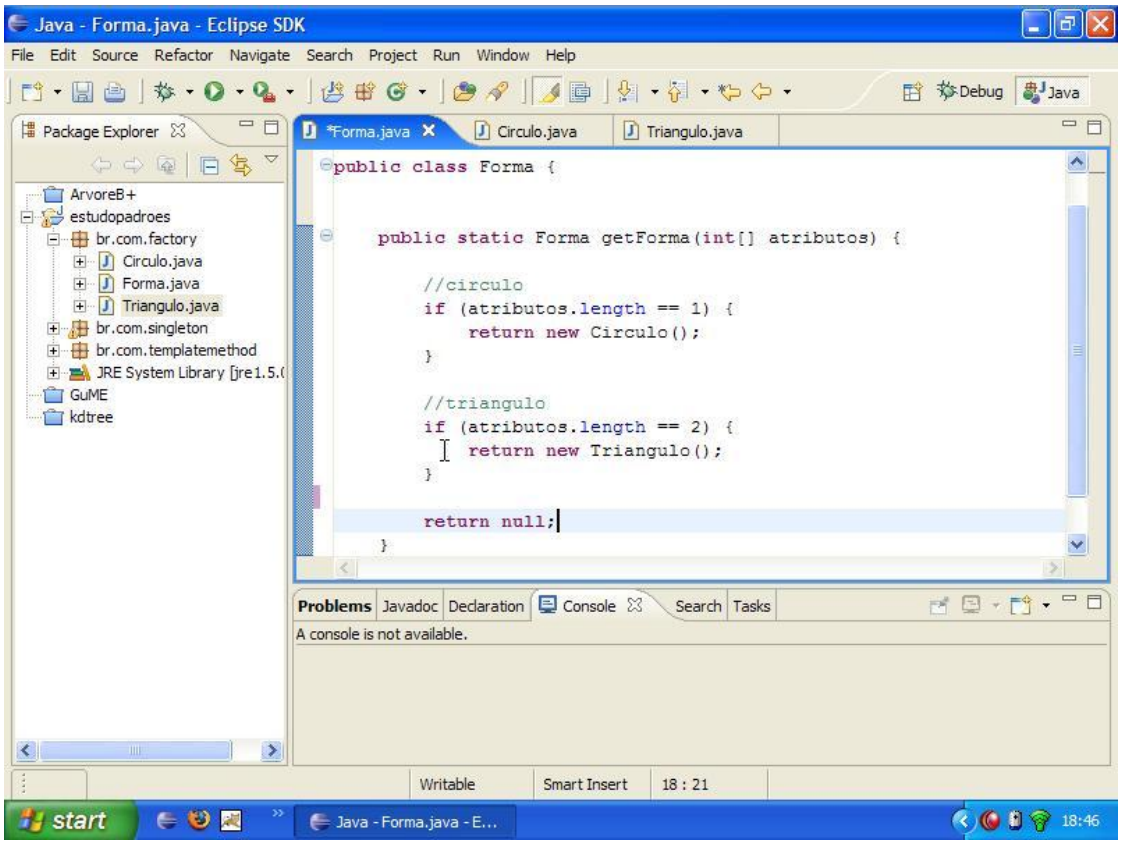


Figura 1.3 – Captura de tela do Eclipse

Fonte: www.devmedia.com.br

1.6 – Descrição

No capítulo 2 será apresentada uma descrição do que é o framework MDArte em mais detalhes e de todos os conceitos e ferramentas nos quais se baseia, como MDA e Maven.

O capítulo 3 apresenta o manual do usuário, mostrando todas suas funcionalidades em detalhes.

O capítulo 4 descreve os aspectos técnicos de suas funcionalidades e como foram implementadas.

Por fim, temos o capítulo 4, no qual há uma explanação sobre as dificuldades, resultados e considerações finais.

Capítulo 2

MDArte e conceitos básicos

2.1 – MDA

2.1.1 – Descrição

A *Model Driven Architecture* (MDA) foi idealizada e definida pelo *Object Management Group* (OMG) com o objetivo de prover engenheiros de software com uma série de padrões para especificar seus sistemas. O termo arquitetura, inclusive, se referencia muito mais à forma com que esses padrões se dispõe do que ao meio em que a solução tecnológica se estruturá. A OMG se baseia não só nas vantagens lógicas do MDA ao defender seu uso, mas também em seu histórico de sucessos:

"OMG has developed some of the industry's best-known and most influential specifications, including CORBA, OMG IDL, IIOP, UML, MOF, XMI, CWM, the OMA, and Domain Facilities in industries such as healthcare, manufacturing, telecommunications, and many others. The Model Driven Architecture (MDA) builds on these successes, providing a comprehensive interoperability framework for defining the interconnected systems of tomorrow."

Fonte: *Website* da MDA (http://www.omg.org/mda/)

A proliferação de diferentes ambientes de trabalho e de diversos *middlewares*, cada qual com suas características, mas sempre visando à aceleração do processo de desenvolvimento, tornou claro que uma linguagem de especificação única é necessária e deveria ser ao mesmo tempo universal e específica.

Para resolver esse problema, a modelagem do sistema foi dividida em duas, como é melhor explicado adiante, no entanto um mesmo conjunto de símbolos e relacionamentos foi especifado, a *MetaObject Facility* (MOF), que normalmente se apoia sobre *Unified Modeling Language* (UML) por compor este um padrão amplamente difundido e conhecido.

Ao abstrair a plataforma de uma MOF, a modelagem do sistema se distancia da sua implementação, o que lhe confere maior estabilidade e adaptabilidade às mudanças tecnológicas. No entanto, há diversos pormenores dependentes da plataforma e o espaço para que coexistam com o restante da especificação não pode ser ignorados na sua elaboração.

2.1.2 – Funcionamento

Uma aplicação de MDA consiste em um Plataform-Independent Model (PIM) definitivo, um ou mais Plataform-Specific Model (PSM) e uma implementação completa para cada plataforma objetivada. Cada um desses componentes representa um nível:

- PIM: independente de plataformas, abrange em um modelo as regras e os processos do negócio.
- PSM: gerado a partir do PIM, é a representação ainda em modelo da aplicação codificada, sendo, portanto, necessário um para cada plataforma-alvo.
- Implementação: código gerado e arquivos auxiliares, prontos para compilação, referência em outros sistemas e *deployment*.

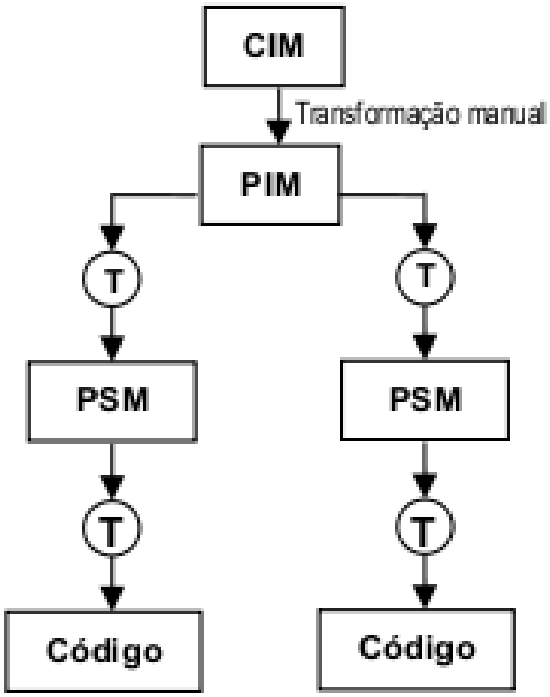


Figura 2.1 – Ilustração do processo supracitado.

Fonte: Wiki oficial do MDArte
(http://www.softwarepublico.gov.br/dotlrn/clubs/mdarte/xowiki/mnEvIntroducaoMDA)

Há diversas ferramentas no mercado que automatizam a passagem de um nível para o outro. Estima-se que elas realizam de 50% a 70% da transformação de PIM para PSM e 100% (ou algo muito próximo disso) da transformação de PSM para código. Vale ressaltar que essa diferença deve-se ao fato dessa indústria ter começado nesse segundo passo muito mais cedo, já que o uso de PSM é bem mais difundido do que o de PIM.

Há diversos conjuntos de transformações entre modelo e código, chamados de cartuchos, pois podem ser plugados na sua engine de transformação e acionados de acordo. Cada cartucho pode entender um determinado trecho da especificação modelada na sua própria maneira, constituindo um forte príncipio de interdependência necessário para um grande potencial de extensibilidade da arquitetura, o que a permite por sua vez ser adotada nos mais diversos cenários possíveis e evitar o máximo de retrabalho na recriação das transformações.

2.1.3 – Vantagens

Ao adotar MDA, uma organização vai garantir que seu sistema estará sempre pronto para lidar com mudanças tecnológicas e de paradigmas, facilitando sua integração com aplicações e recursos em diversas configurações tecnológicas.

Isso tudo é reforçado pela prioridade constante dada pelos responsáveis pelo padrão MDA à sua interoperabilidade através de diferentes ambientes, o que é fundamental para que atenda a seus objetivos.

A atenção fica focada no correto funcionamento do sistema e no melhor modo para atender aos seus requerimentos de forma eficiente e correta, permitindo que os envolvidos se ocupem estritamente com o que é mais crítico para o seu bom funcionamento. Aspectos técnicos, mesmo importantes, são relativamente mais irrelevantes e muitas vezes podem ser tratados de forma semi ou totalmente automatizada.

Usuários da MDA podem então se dedicar aos aspectos mais relacionados ao negócio do seu cliente do que nos meios de viabilizar e integrar a tecnologia necessária para atendê-lo. Esses últimos se tornam responsabilidade dos criadores das

transformações entre modelos e código, que estão encapsuladas nos cartuchos descritos acima.

Há cartuchos para diversas tecnologias, como EJB, Struts etc, e um engenheiro de software pode selecionar um conjunto de cartuchos de acordo com a tecnologia requerida e esse será, no caso ótimo, a única preocupação estritamente técnica que ele terá.

2.2 – MDArte

2.2.1 – O *Framework*

"O MDArte tem como propósito a criação de um novo referencial de software público, com o uso de tecnologias modernas, redução do custo total dos serviços de tecnologia da informação e da dependência de soluções proprietárias. Estes objetivos serão alcançados através do uso, evolução e disponibilização de uma infra-estrutura MDA (Model Driven Architecture) open source. A aplicação de uma abordagem de arquitetura orientada a modelo (MDA) permite dentre outras vantagens a padronização da arquitetura da aplicação, a aplicação e reuso de melhores práticas de programação e a sincronia entre os modelos que documentam o sistema e a implementação. A evolução do framework será direcionada no sentido de se obter uma infra-estrutura MDA voltada para o desenvolvimento de softwares de governo a ser disponibilizada como software público.

O MDArte compreende um conjunto de cartuchos AndroMDA com diversas soluções de projeto e arquitetura incorporadas nos procedimentos de transformação de modelos seguindo a abordagem MDA."

Website oficial do MDArte (http://www.softwarepublico.gov.br/ver-comunidade?community_id=9022831)

2.2.2 – O dia-a-dia de um desenvolvedor MDArte

Enquanto que o MDArte tem claras vantagens e facilita diversos processos, não é uma plataforma simples de operar.

Quando inicia a trabalhar, o desenvolvedor demora de 1 a 2 para conseguir configurar seu ambiente, sendo auxiliado por um passo-a-passo e os demais desenvolvedores.

A cada dia de trabalho, para um projeto do tamanho do PILOTO, precisa de 10 a 20 minutos para atualizar os modelos UML manualmente um a um pelo MagicDraw, depois processá-los com o MDArte e compilá-los pelo *prompt*, atualizar o código Java do projeto (pelo SVN), voltando novamente para o *prompt* para compilá-los. Com o Express, criaremos uma interface simples que facilite o uso do processamento em paralelo que será implementado.

Como não sabe que diagramas estão desatualizados, tem que abrir e tentar atualizar cada um que for usar. Com o Express, terá visibilidade sobre o que está desatualizado.

Como é demorada a compilação dos módulos (cada instrução demora normalmente entre 2 e 5 minutos), o desenvolvedor acaba realizando tarefas pouco relevantes em paralelo em boa parte do tempo de compilação ou se distrai conversando com os colegas ou navegando na Internet. Como não está atento ao *prompt* de compilação, acaba se distraindo e perdendo ainda mais tempo entre o envio de uma insutrção de compilação e outra. O Express permitirá o envio de uma sequencia de instruções de uma só vez e irá para o primeiro plano do sistema quando acabar, alertando o usuário de seu término.

Quando há alguma necessidade extraordinária de compilar algum módulo não usual, caso ninguém lhe avise, só descobre depois de inúmeras e demoradas tentativas frustradas de compilação. Com o Express, planejamos um módulo de avisos.

2.2.3 – AndroMDA

Figura 2.1 – Ilustração do processo supracitado.

Fonte: Wiki oficial do MDArte

(http://www.softwarepublico.gov.br/dotlrn/clubs/mdarte/xowiki/mnEvAndroMDA)

O diagrama complexo acima ilustra como as diferentes partes do *framework* AndroMDA se relacionam. Ao longo do trabalho temos referenciado o MDArte como

sendo um *framework*, mas, de fato, ele é uma extensão do AndroMDA, não constituindo um por si só, apenas um conjunto de cartuchos.

Numa explicação com o nível de detalhe que precisamos para esse trabalho, o diagrama acima nos mostra que a partir do MagicDraw temos os modelos salvos em arquivos XMI e XML, que serão processados através do plugin do AndroMDA para Maven, este último explicado na próxima seção. O plugin, baseado em diversas especificações de linguagem e nos cartuchos adicionais ligados a ele, processa os modelos, gerando o código de acordo com esses cartuchos.

O código é todo gerado numa pasta chamada "target" e que deve se restringir somente a código gerado, pois tudo será sobrescrito a cada novo processamento. O código customizado deve estar numa pasta chamada "src" no mesmo diretório da "target". Como o código é gerado na linguagem Java, que suporta herança, caso o desenvolvedor queira alterar ou implementar algum método abstrato do código gerado, basta criar uma classe na pasta "src" que estenda a classe gerada no "target" e escrever nela seu código.

Isso é muito utilizado quando se quer dar um comportamento não-padrão a uma classe ou então quando precisamos definir métodos do negócio, considerados hoje complexos e específicos demais para serem especificados através de diagramas generalistas. Um exemplo seria a função que efetua uma compra num sistema de ponto-de-venda, que precisa controlar a impressora fiscal e o registro dessa transação no banco, além de seus fluxos alternativos e validações, enquanto que uma parte de um sistema que corresponda a um CRUD (*Create, Retrieve, Update and Delete*) simples poderia ser modelada via diagramas, pois apresentam comportamento-padrão.

A mesclagem entre código gerado e customizado é toda feita através do mecanismo de compilação e de herança do compilador e manter essa separação lógica de arquivos entre essas duas diferentes partes é primordial, uma vez que:

1) Simplifica o processo de sobrescrição do código gerado.
2) Evita erros de sobrescrição do código customizado pelo código gerado.
3) Torna o trabalho do desenvolvedor menos complexo e evita erros humanos.

2.2.4 – Maven

Inicialmente uma ferramenta interna na Apache para realizar o *build* de seus projetos, foi sendo melhorada e extrapolada para o mundo externo.

É a escolha dos desenvolvedores do AndroMDA, que desenvolveram um plugin próprio, uma vez que constitui uma alternativa mais simples e rápida do que métodos convencionais, como scripts e Ant, já que baixa de repositórios as versões mais novas das bibliotecas e outras dependências utilizadas com apenas um comando, sem ser necessário executar o processo manualmente para cada uma.Além disso, tem se tornado um padrão da indústria, o que faz com que cada vez mais pessoas estejam acostumadas a utilizá-lo.

Em conjunto com o plugin do AndroMDA, é possível validar os modelos, processá-los para gerar código, compilar esse último, colocar os arquivos gerados no servidor, baixar as dependências mais novas, entre outras tarefas costumeiras.

Se por um lado é simples e mais eficiente que os métodos antigos, continua sendo longe do ideal, uma vez que se faz necessário enviar diferentes comandos em sequência e cada um, devido ao tamanho dos projetos da COPPETEC, pode vir a demorar alguns minutos e o que ocorre é que os desenvolvedores perdem grande tempo do seu dia nesses processamentos e reprocessamentos. Por exemplo, para trocar o conteúdo de uma mensagem de texto ou, mais simplesmente, corrigir uma acentuação, pode-se levar até 10 minutos no projeto PILOTO. Outro exemplo: a atualização completa do projeto completa demora em torno de 15 a 20 minutos.

Temos algumas formas de tentar resolver o problema, que é a questão-chave a ser atendida pelo Express:

1) Tentar remodelar o AndroMDA e otimizá-lo para o nosso caso.
2) Utilizar computadores mais potentes ou estudar o uso de um super-computador com uma rede rápida.
3) Com os recursos, estrutura e funcionamento que se têm hoje, tentar construir algo por cima disso.

Como a terceira opção é a menos custosa e, não só por isso, a menos eficaz, foi a escolhida e constitui a grande motivação do Express. Mais a respeito será dissertado no próximo capítulo.

2.2.5 – MagicDraw

É a ferramenta de modelagem utilizada pelo AndroMDA. Suporta a importação de linguagens e extensões, o que é de suma importância para a customização de funcionamento inerente ao *framework.*

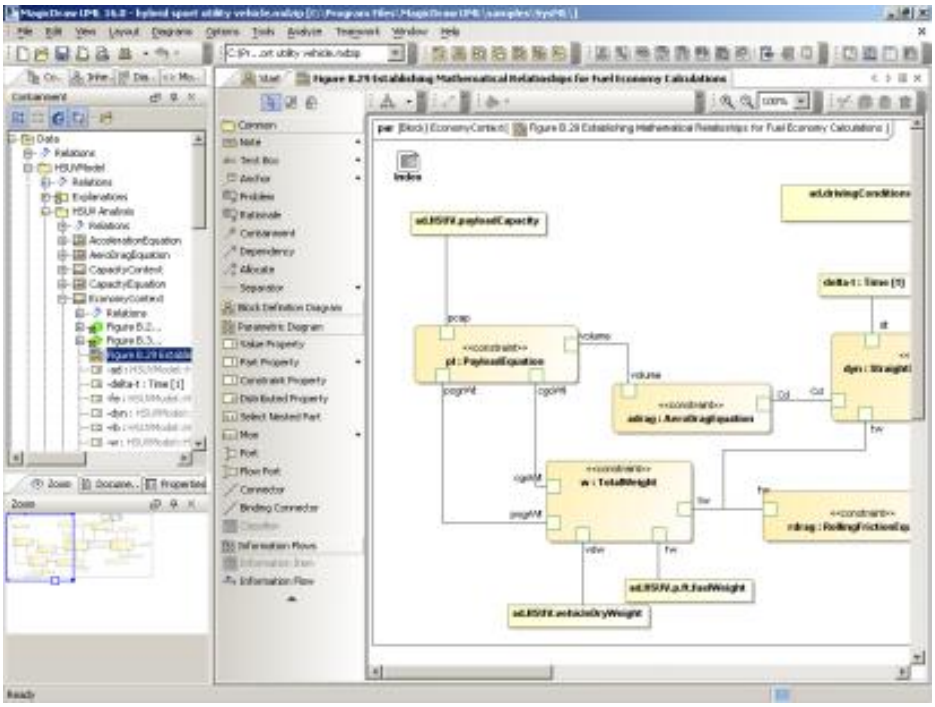


Figura 2.2 – Ilustração do processo supracitado.

Fonte: *Website* oficial do MagicDraw

(http://www.magicdraw.com/screen_shots)

2.2.6 – Portal do Software Público Brasileiro

O MDArte está hospedado no Portal do Software Público Brasileiro (http://www.softwarepublico.gov.br/) uma iniciativa de extrema relevância do governo brasileiro. Em 2005, foi licenciado o primeiro software público brasileiro pelo governo federal e seguindo as prerrogativas legais vigentes, no caso a Lei do Direto Autoral, a Lei do Software e a Resolução Nº58 do Instituto Nacional de Propriedade Intelectual (INPI). Com a rápida adoção e dispersão em diversos setores econômicos e crescente colaboração da sociedade, não agindo somente como usuária, foi criado um portal para organizar os diferentes ewforços e projetos.

“Em função da legislação corrente, sabe-se que o software desenvolvido por instituições de direito público é por natureza um bem público. A união da premissa de

que o software é um bem público com a percepção de que a disponibilização (amparada pela Lei) de um software pelo setor público extrapola o universo do código livre, se estabeleceu a primeira base para o conceito de software público, cujo mote principal é a manifestação do interesse público por determinada solução.

O Brasil, nos últimos cinco anos, tem desenvolvido o conceito do Software Público – SPB e, com a implementação do arcabouço jurídico produzido para este modelo, que garante a proteção legal dos autores sobre o código produzido, estimulou a necessidade de proteger também os agentes envolvidos no que se refere ao processo de uso, distribuição e comercialização da marca associada a este produto. Atualmente existe uma lacuna, que se ressente a comunidade pois, se proteje os fontes mas não o nome e a marca associada ao código, e deixa a descoberto os doadores e a sociedade, que são seus receptores.”

Fonte: http://www.softwarepublico.gov.br/lpm

Para então resolver essa questão, foi criada a LPM, Licença Pública de Marca, que já conta com a adesão de três dos projetos hospedados no portal.

Figura 2.3 – Marca da LPM

Fonte: *Website* oficial da LPM (http://www.softwarepublico.gov.br/lpm)

Todos esses movimentos de proteção legal, profissionalização do software público brasileiro e colaboração forte da comunidade aberta dão mais um respaldo para o MDArte e mais fortes razões para acreditarmos que ainda pode vir a desempenhar um papel modificador na forma como a sociedade brasileira encara a produção de software. Além disso, em poucos anos, o Portal do Software Público Brasileiro já hospeda quase cinqüenta projetos, que contam, por sua vez, com mais de cento e cinqüenta mil

membros. O MDArte têm mais de mil e seiscentos membros cadastrados e alguns prestadores de serviço, pessoas físicas e jurídicas, em quarto das cinco regiões naturais brasileiras.

Capítulo 3

Manual do Usuário

3.1 – Introdução

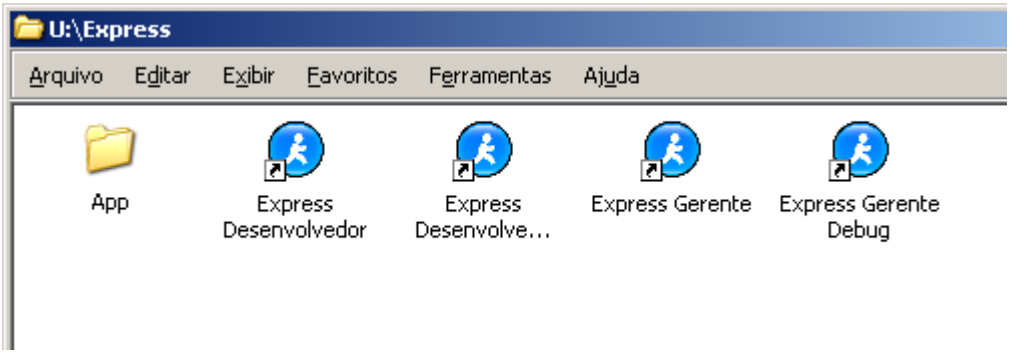
3.1.1 – Objetivo

O Express foi construído para auxiliar todos os envolvidos nos diversos projetos da Coppetec, em especial nas tarefas envolvidas com o novo *framework* MDA. Seja através de utilitários, centralização de informações importantes ou melhorias na performance de processos rotineiros, sua grande meta é economizar o tempo de cada um e nos tornar mais produtivos.

3.1.2 – Instalação

O Express dispensa instalação, rodando diretamente da rede. Em cada grupo de trabalho tem sua pasta própria, portanto, consulte seu gerente sobre qual é no seu caso.

Sabendo o diretório, basta acessar o atalho referente ao seu perfil.



No exemplo, temos dois perfis que acessam o Express: o Desenvolvedor e o Gerente, cada qual com dois atalhos: versão normal e versão com informação de debug.

Dica: copie o atalho para sua Área de Trabalho ou qualquer outro lugar de fácil acesso, já que o Express irá se tornar seu amigo de todas as horas.

Avançado: a instalação do Express na rede pode ser feita através da simples cópia de uma outra instalação para a pasta desejada, estando pronto já para ser rodado.

3.1.3 – Notas Importantes

O usuário deve ter acesso de escrita na pasta onde o MagicDraw está, pois a usa para se comunicar com esse aplicativo.

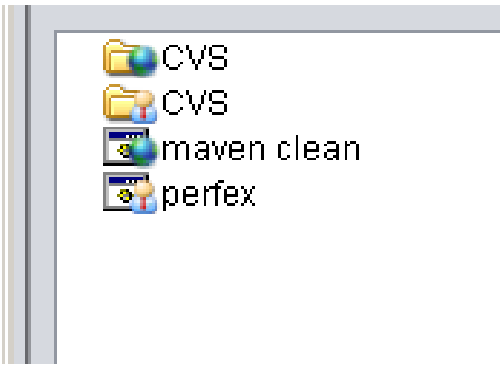
3.1.4 – Configuração

O Express usa uma estrutura simples para sua configuração baseada em diretórios. Há três repositórios no qual busca suas configurações, realizando uma mescla entre eles:

1. O subdiretório ".\Config\Todos\" na raiz da instalação remota. Essa pasta conterá configurações que serão aplicadas para todos os usuários que utilizarem o Express.
2. O subdiretório ".\Config\<nome do perfil>\" na raiz da instalação remota, que será utilizado para todos os usuários do perfil referente.
3. O subdiretório ".\.express\" na raiz da pasta do usuário do sistema operacional, contendo configurações que serão utilizadas somente quando o Express for executado por este.

Dentro de cada um desses repositórios há diversas pastas, cada uma relacionada a um tipo de configuração. Falaremos mais sobre cada uma conforme formos as apresentando no aplicativo.

Muitas vezes, esses dados serão organizados em árvores de arquivos, como esta:



Se o arquivo ou diretório estiver na pasta remota de todos, será exibido um planeta sobre seu ícone. Se estiver na pasta do perfil, um ícone de pessoa. Se estiver no repositório local, um ícone de um *hard drive*.

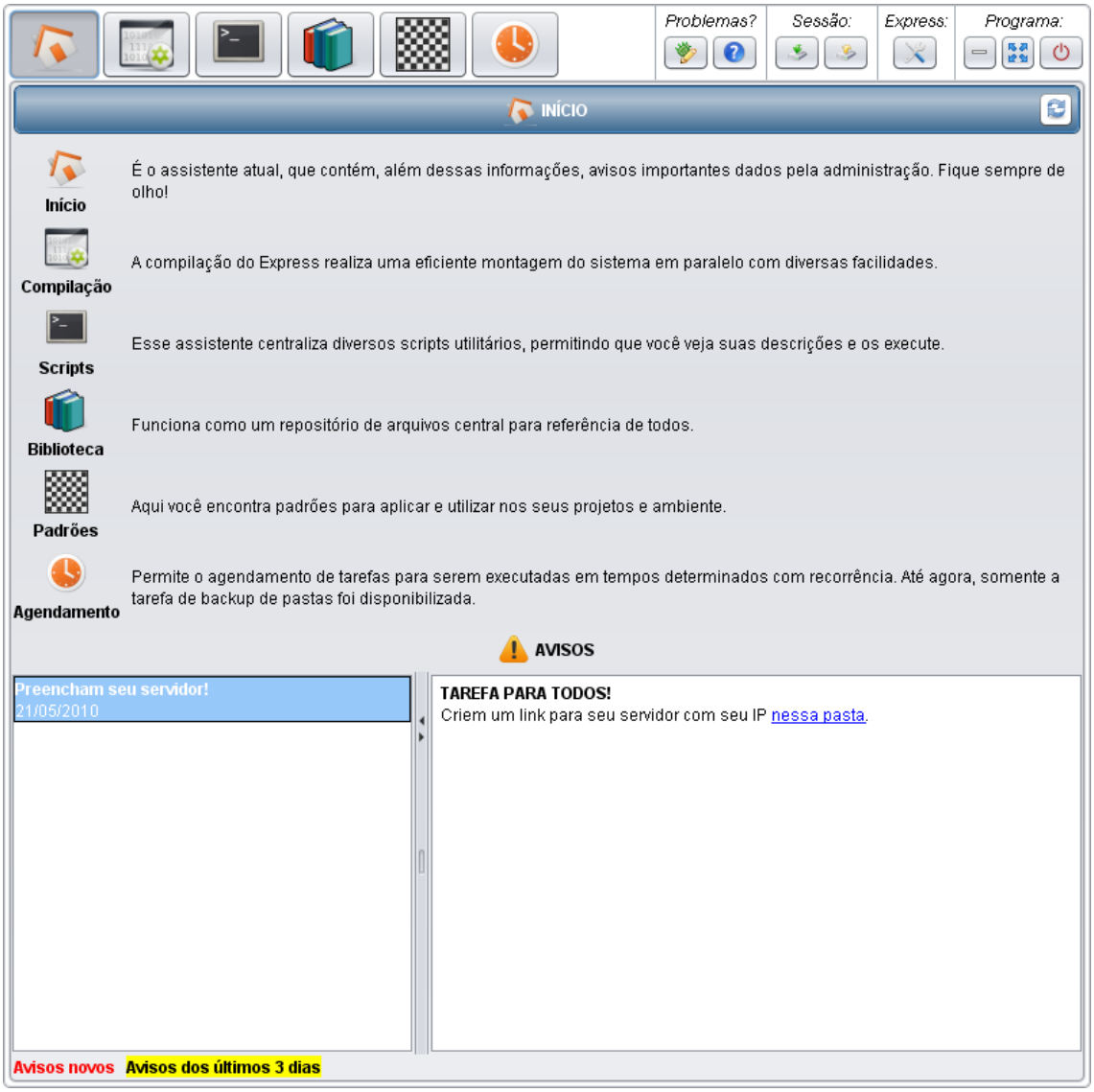
3.1.5 – Sugestões

Para sugerir ou relatar *bugs* acesse o Mantis do Express. Uma forma fácil é através do botão que será apresentado na próxima seção.

3.2 – O Aplicativo

3.2.1 – Interface

Após iniciar o programa, seu logo será exibido no centro da tela enquanto ele carrega. Quando concluído todo esse processo, a tela inicial será mostrada:



Tela inicial do Express

O Express é formado por diversos assistentes, cada um contendo um certo grupo de funcionalidades. Temos na parte superior do aplicativo os botões de seleção de assistente, além de alguns outros.

Parte superior do aplicativo destacando em amarelo os botões de ativação de assistente à esquerda e outros diversos à direita em azul.

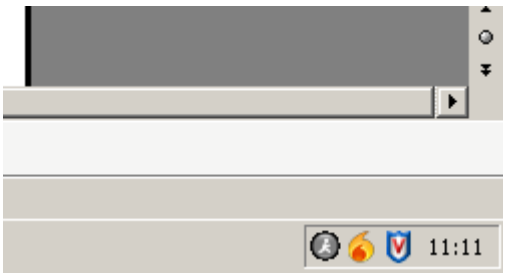
Os assistentes disponíveis são determinados pelo perfil em execução, que por sua vez é obtido a partir do atalho sendo executado (no exemplo anterior, Desenvolvedor ou Gerente).

Os seis botões restantes estão divididos em 4 seções. A primeira, “Problemas?”, contém o botão que abre o Mantis do Express, *website* onde o usuário pode relatar erros ou escrever sugestões, e outro que abrirá o manual.

Em “Sessão:”, temos os botões que salvam e carregam as configurações da sua sessão: a posição dos painéis, o número de instruções recentes, entre outras propriedades.

Em “Express:”, há somente o botão de configuração, que abre um diálogo descrito em seção própria.

Em “Programa” estão os controles da janela do aplicativo: o de minimizar para a área de notificação, o de restaurar\maximizar a janela e o de fechar, que termina o aplicativo.



O ícone cinza na bandeja referente ao Express

Ao clicar com o botão no ícone acima, duas opções serão mostradas: “Abrir” ou “Fechar”:

Opções resultantes de um clique com o botão direito

Ao clicar em abrir, o aplicativo será restaurado e exibido. A opção fechar é idêntica ao botão fechar visto algumas imagens acima, terminando por completo o aplicativo.

Dica: É ao fechar que o Express salva suas configurações. Desse modo, caso ele não seja terminado corretamente, pode ser que algumas alterações sejam perdidas. Caso faça alterações importantes, pressione o botão “Salvar Sessão” descrito acima.

Além disso, o ícone também serve para indicar o status da compilação, como será mais bem explicado adiante.

3.2.2 – Configurações Gerais

Ao clicar no botão de “Configurações” o respectivo diálogo será aberto. Nele há diversas abas, que serão ao longo do documento.

Nesse ponto, somente nos interessa explorar a primeira aba, que contém configurações gerais do aplicativo divididas em dois grupos.

Diálogo de “Configurações”, aba “Geral”

No primeiro podemos configurar os diretórios do *workspace*, JBoss, Maven e MagicDraw, escrevendo seus caminhos nos campos respectivos. Ao pressionar o botão ao lado do campo, o usuário pode buscar um diretório mais facilmente.

Somente ao pressionar o botão aplicar no canto inferior direito que as alterações serão aplicadas.

Dica: o sistema adiciona a barra no fim do caminho automaticamente caso isso não tenha sido feito pelo usuário, portanto, qualquer uma das formas (com ou sem barra no final) é entendida perfeitamente pelo sistema.

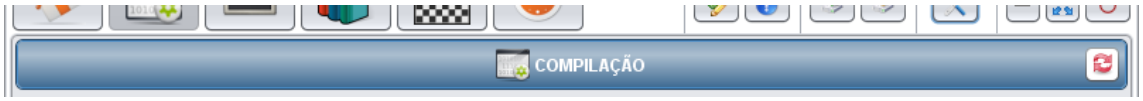
Avançado: ao modificar o caminho do MagicDraw, a pasta plugins na raiz do Express é copiada para esse diretório. Ela contém o plugin do Express para o MD e é necessário reiniciar o MD antes de ele aparecer.

3.2.3 – Assistentes

Ao clicar em um dos botões de assistente, este será ativado e exibido no painel principal da aplicação. O que todos têm em comum é a barra de título:

Barra de título do assistente "Compilação"

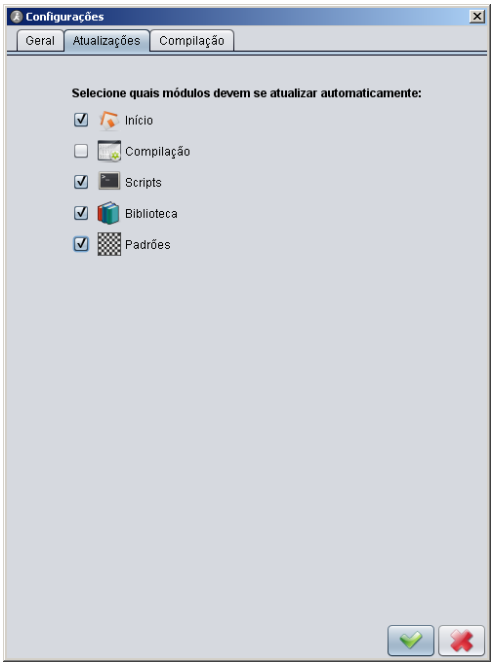
Há um botão no canto direito da barra de títulos que recarrega o assistente com as informações mais novas disponíveis.



Barra de título do assistente "Compilação"

Note agora que para este outro assistente o botão atualizar está em vermelho, pois a atualização desse assistente é somente manual, ou seja, quando é pressionado o botão. Nos outros casos, com botão azul, ao entrar no assistente essa atualização também é feita.

Como a velocidade de carregamento vai depender de diversas condições como o tráfego na rede e a quantidade de arquivos no perfil do usuário, por exemplo, o tipo de atualização pode ser modificado na aba "Atualizações" do diálogo de "Configurações":



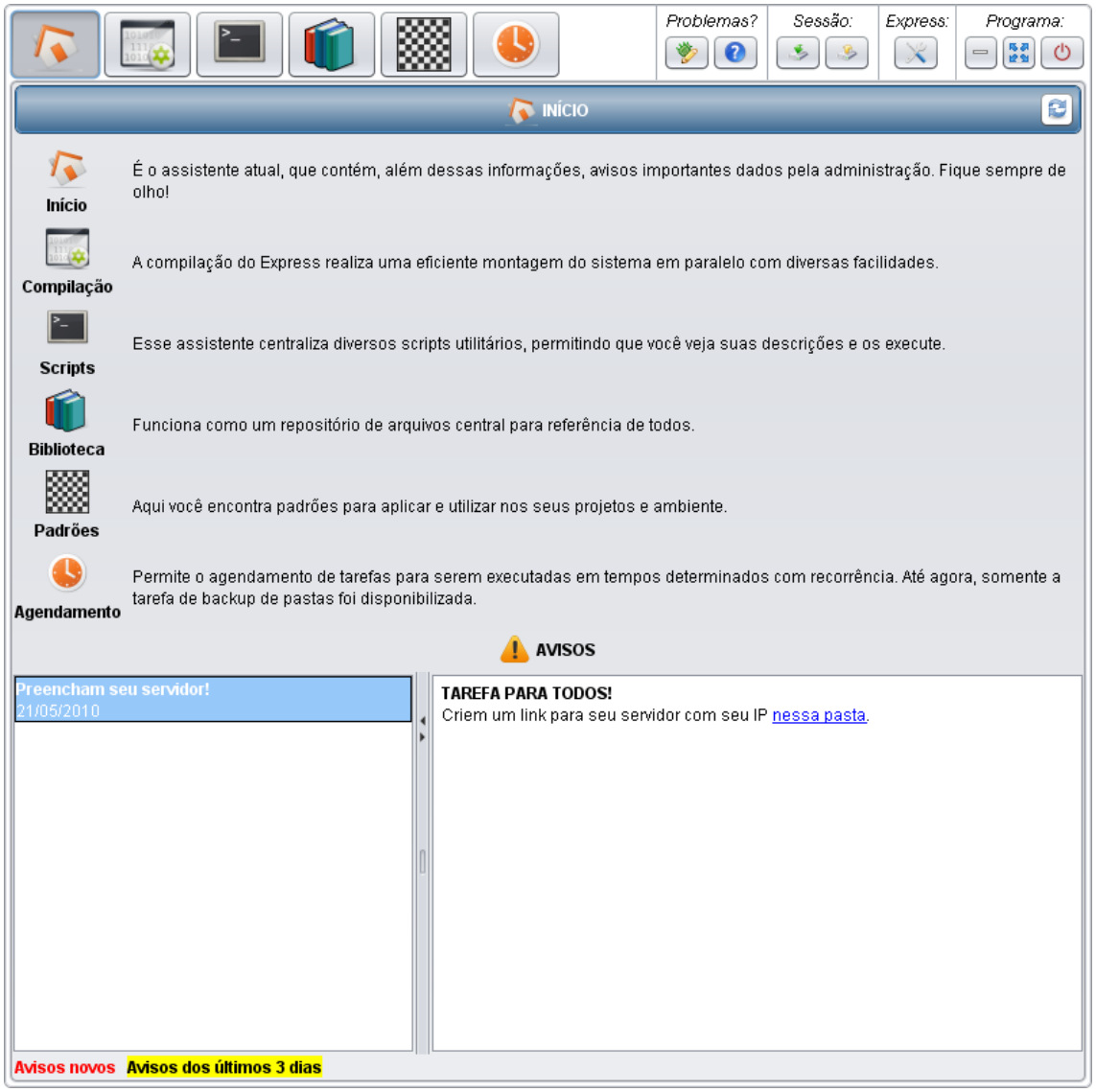
Diálogo de "Configurações", aba "Atualizações"

Basta marcar para ativar a atualização automática e desmarcar para que o assistente seja atualizado somente manualmente.

3.3 – Início

3.3.1 – Descrição

È o assistente mostrado ao iniciar a aplicação.



Assistente "Início"

3.3.2 – Assistentes

Contém uma breve descrição de cada assistente disponível no Express para o perfil executado.

3.3.3 – Avisos

No painel esquerdo, temos uma lista de avisos. No direito, a descrição do aviso selecionado.

Avisos novos serão escritos em vermelho **ASSIM**.

Avisos escritos nos últimos dois dias, terão fundo amarelo **ASSIM**.

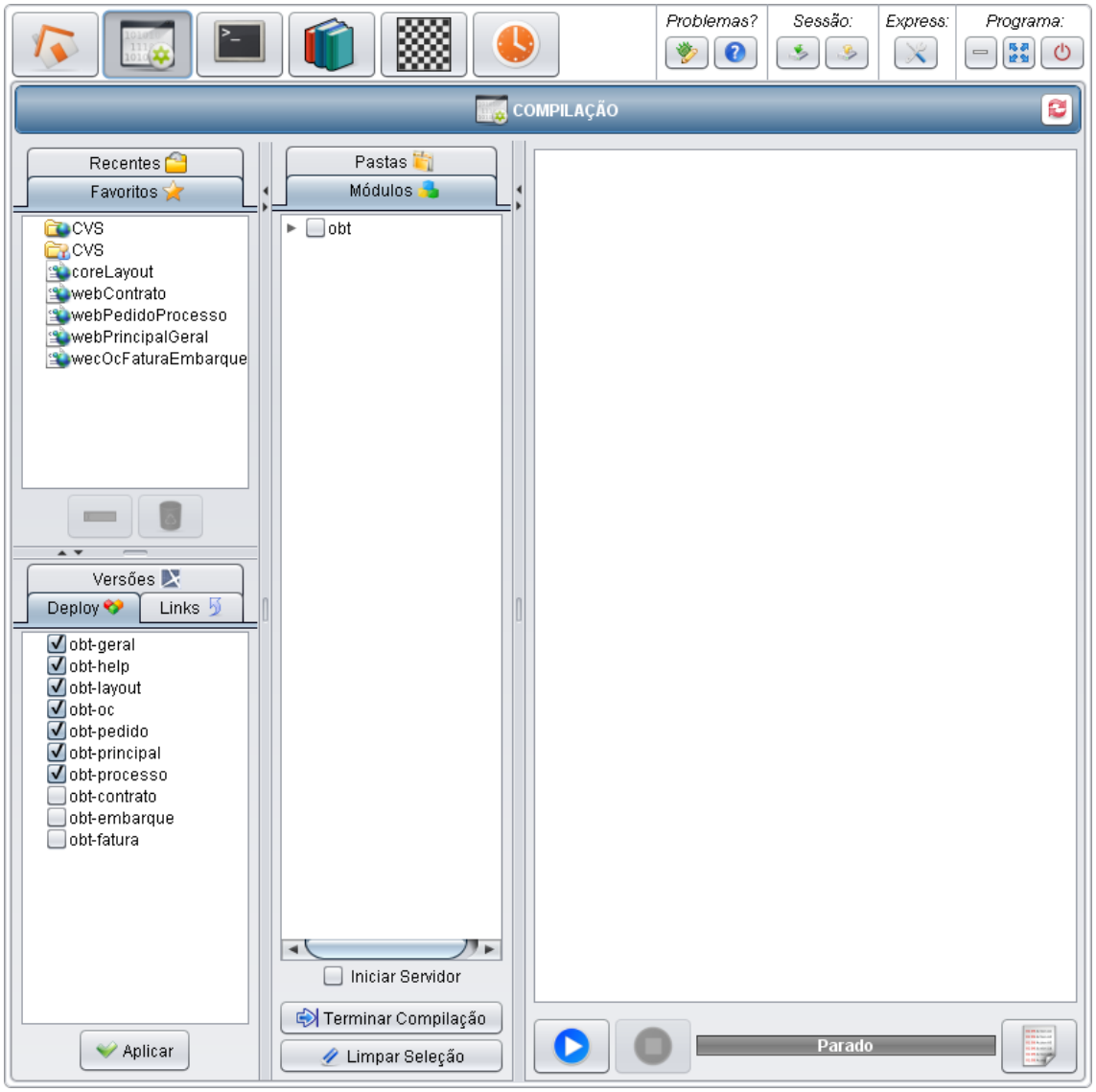
Avisos escritos nos dois últimos dias e que ainda não foram lidos, serão formatados dois modos **ASSIM**.

Avançado: Os alertas serão arquivos no subdiretório “Alertas” nas pastas de configuração. O nome do alerta será o nome do arquivo referente e sua data de escrita será a data de modificação deste. O arquivo pode tanto ter conteúdo textual simples ou HTML.

3.4 – Compilação

3.4.1 – Descrição

Esse assistente é o principal do projeto, aquele no qual a maioria dos usuários passará mais tempo, logo, reúne, além de painéis referentes às suas operações básicas, diversos facilitadores, sendo o assistente mais complexo do Express.



Assistente "Compilação"

Um a um, os painéis desse assistente serão descritos a seguir.

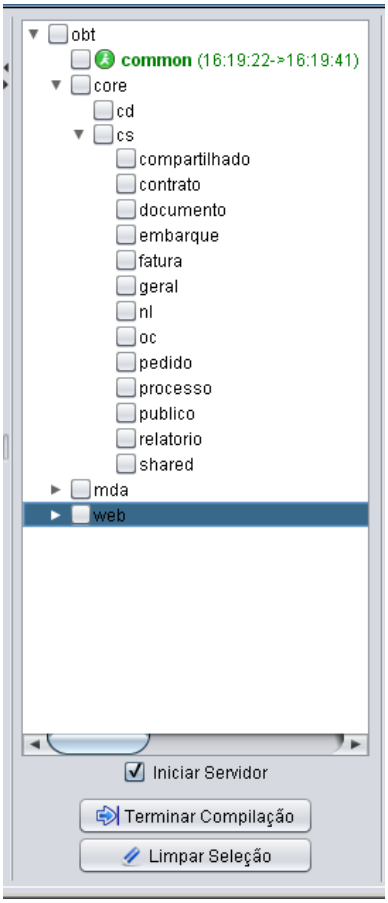
3.4.2 – Comandos

Aqui temos duas árvores, uma em cada aba, com todos os módulos de todos os projetos localizados no seu *workspace*, variando apenas a forma de agrupá-los.

A seleção de um módulo determinará se este deve ser executado. Quando a compilação for iniciada, cada módulo receberá um ícone cinza do Express, indicando que está parado. Conforme a execução for sendo feita, os ícones se tornarão laranjas, determinado que estão sendo executados. Ao terminar, um comando fica ou verde em caso de sucesso, ou vermelho em caso de erro.

O ícone do Express na bandeja do sistema operacional também seguirá esse esquema de cores, indicando o progresso da instrução atual. Quando acabar, o computador apitará e o aplicativo virá para frente do seu monitor.

Caso ocorra algum erro no meio do conjunto de comandos selecionados, um diálogo aparecerá, perguntando se você deseja interromper a compilação ou se deseja prosseguir mesmo assim.



Painel de "Comandos", indicando uma compilação bem-sucedida do common.

Há uma caixa de seleção e dois botões abaixo da árvore. A caixa, nomeada como "Iniciar Servidor", quando marcada indica que o Eclipse deve ser ativado e o JBoss nele iniciado. Isso ocorre assim que as compilações do core terminarem caso haja alguma ou logo no início se não houver.

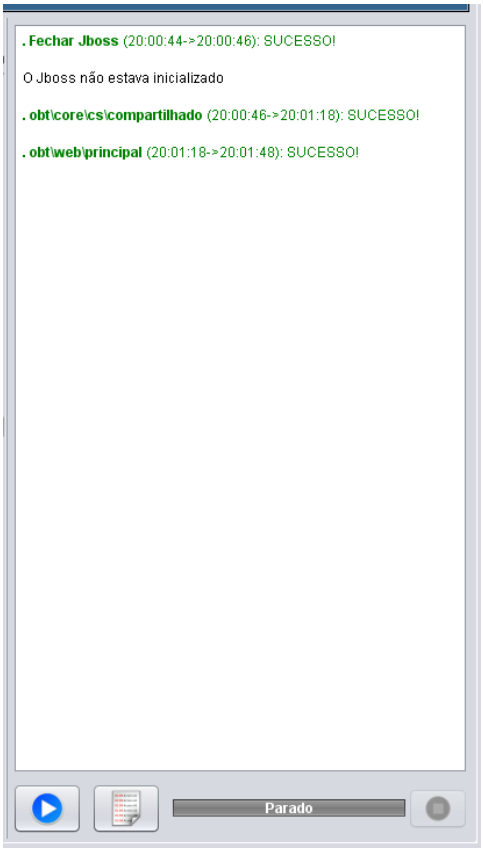
Por sua vez, o primeiro botão, "Terminar Compilação", executa todos os comandos que não foram terminados com sucesso, ou seja, aqueles em cinza ou vermelho. O segundo, "Limpar Seleção", desmarca todos os comandos.

Dica: Quando mudar a assinatura de algum serviço é necessário recompilar o módulo cs\compartilhado, pois nele está codificada a interface entre serviços de diferentes módulos.

Avançado: para agrupar por módulo do sistema, o Express se baseia no nome dos arquivos de modelos xml. No caso de um arquivo como o do obt-emb.xml, que tem como correspondentes os módulos core\embarque e web\embarque, se faz necessário criar um arquivo no diretório dos modelos chamado "obt-emb.xml-nome-verdadeiro.txt" contendo uma linha de texto com o nome que deve ser utilizado para essa correspondência, no caso, "embarque". O Express também remove o prefixo do nome do arquivo de modelos se esse começar com "nome do projeto-".

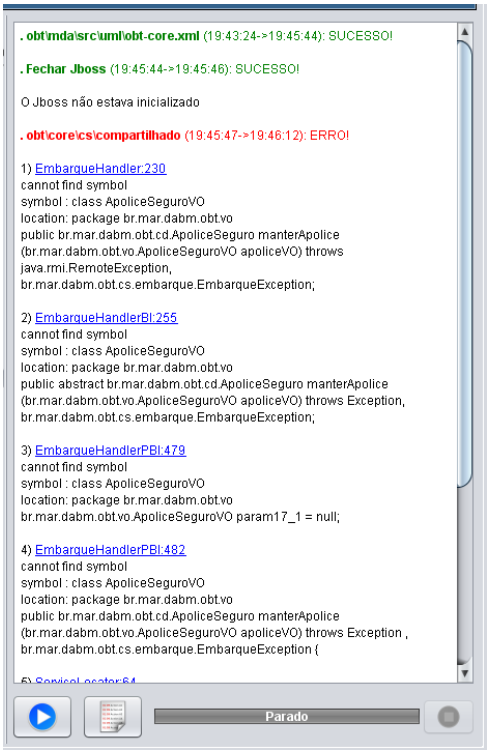
3.4.3 – Log

No painel de *log*, encontramos os botões que controlam a execução dos comandos selecionados ("Iniciar" e "Parar"). Além disso, informações resumidas sobre o estado de cada comando, bem como informações dos erros mais organizadas do que no *log* inicial do maven, também são exibidas. De forma auxiliar, há uma barra de progresso indicando de forma ainda mais sucinta o estado da execução.



Painel de "Log"

Em caso de erro, a informação virá organizada como na imagem a seguir, sendo possível clicar no nome do arquivo onde o erro ocorreu para ser levado até ele. O arquivo abrirá no programa padrão do seu sistema operacional para arquivos Java.



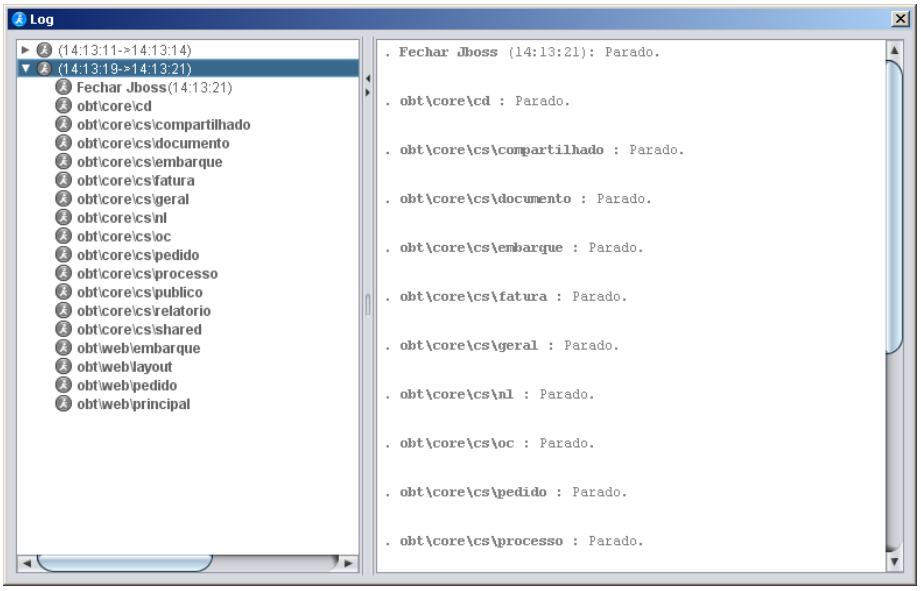
Painel com uma compilação mal-sucedida

Dica: ao selecionar um comando no painel anterior, a informação visível se restringirá somente à deste comando, o que serve especialmente para facilitar a análise de erros. Ao selecionar uma instrução recente no respectivo painel (que ainda será descrito) toda a saída da compilação dos comandos dessa instrução será exibida.

Avançado: no diretório de configuração local, há uma pasta chamada logs onde ficam registros de todas as execuções em pastas nomeadas de acordo com sua data e hora. Dentro de cada pasta há um arquivo de registro para cada comando rodado.

Avançado: ao tentar compilar algum módulo do core um comando para fechar o JBoss será incluído na instrução e executado antes da compilação de algum destes módulos.

3.4.4 – Log Estendido

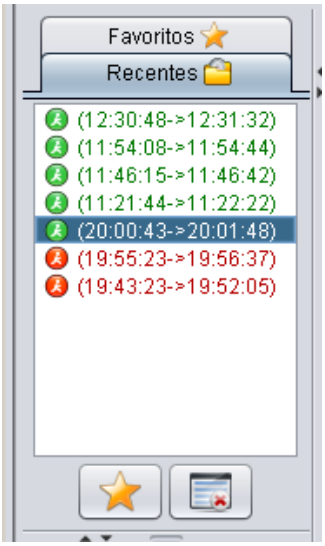


Diálogo de "Log"

O *log* estendido é um diálogo aberto no botão de *log* do painel acima com informação de todas as compilações feitas nessa execução do Express. Ao contrário do anterior, aqui todas as informações geradas pela compilação são repassadas ao usuário, logo, pode ser útil em casos onde for necessária uma melhor análise das etapas realizadas.

Dica: para visualizar o registro de um comando em específico, basta selecioná-lo, como no painel anterior.

3.4.5 – Recentes



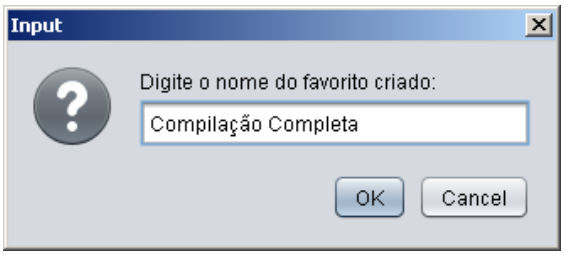
Painel "Recentes"

Esse painel contém as últimas instruções executadas, identificadas pelo seu horário de início e término. O ícone do Express indica qual foi o seu resultado: interrupção pelo usuário, sucesso ou erro, seguindo mesmo esquema de cores apresentado.

Ao selecionar uma instrução, a árvore de comandos será atualizada para indicar o resultado de cada comando através de um ícone ao lado de cada um que foi executado. Aqueles que não foram não o apresentarão.

As informações de *log* resumido e estendido também apresentarão as informações da instrução selecionada.

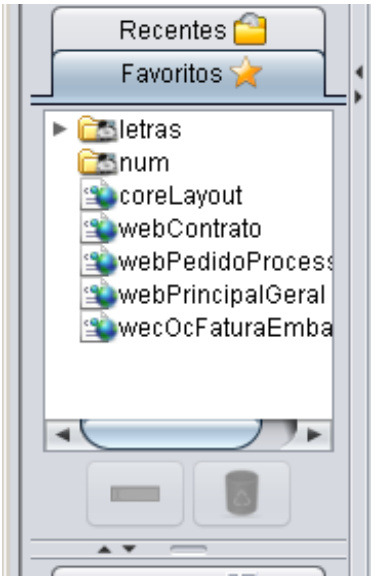
Há dois botões abaixo da lista: "Adicionar Favorito", que salva uma instrução recente como um novo favorito e pergunta qual o nome utilizado para este, e "Limpar Lista", que remove todas as instruções recentes da lista, menos a que estiver em execução (se houver).



Diálogo para nomeação do novo favorito

3.4.6 – Favoritos

Os favoritos são criados a partir das instruções recentes conforme descrito acima. Nesse painel, o usuário pode selecionar um favorito para executá-lo.

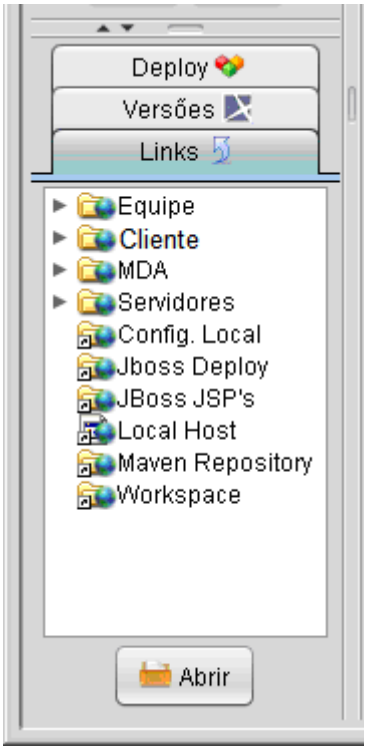


Painel de "Favoritos"

Além disso, há duas opções para manipular os favoritos: "Renomear" e "Excluir", que são operações que dispensam maiores explicações, bastando o usuário selecionar o favorito desejado e pressionar o botão referente.

3.4.7 – Links

Em "*Links*" encontramos diversos atalhos para páginas *web* e pastas importantes na estrutura do ambiente, localizando-se no assistente "Compilação" (o mais utilizado) para seu fácil acesso.



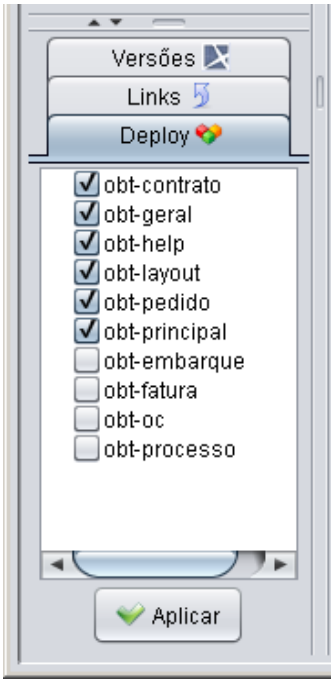
Painel de "Links"

PILOTO: Em "Equipe" temos os atalhos para os servidores de cada um da equipe. Em "Cliente", links hospedados na rede interna do cliente (de onde a equipe travalhava), como o do cardápio do dia e a nossa Wiki. Em "MDA", o link para a página do projeto no portal do Software Público. Em "Servidores", atalhos para os diversos servidores de homologação e produção. Por fim, não agrupados em um diretório específico, atalhos diversos relacionados ao ambiente de trabalho. "Express Config" o leva à pasta local de configuração do Express. "JBoss Deploy" ao diretório de deploy do servidor. "JBoss JSP's" te direciona para o diretório onde uma JSP pode ser modificada *on the fly*. "Local Host" é a página do seu obt local. "Maven Repository", o seu diretório de repositório do Maven. "Workspace", a pasta configurada no Express para o seu workspace.

Avançado: Qualquer atalho pode ser colocado na subpasta "Links" dos diretórios de configuração. Atenção que links dinâmicos (como o workspace e o diretório de deploy do JBoss) são gerenciados internamente pelo Express. Seus arquivos levam o usuário para outros lugares, sendo importante manter seu nome na exata grafia para que sejam corretamente interpretados pelo aplicativo.

3.4.8 – Deploy

Aqui é possível selecionar quais módulos dos projetos no seu ambiente deverão estar no servidor para serem carregados. Selecionar um subconjunto resumido que satisfaça suas necessidades é uma ótima forma de agilizar a inicialização do servidor.

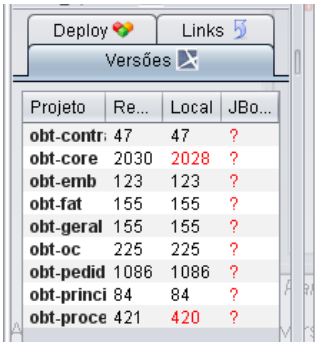


Painel de "Deploy"

Para isso, basta marcar os módulos e apertar o botão "Aplicar".

Dica: é possível mudar esse configuração mesmo depois da inicialização do banco, adicionando módulos extras conforme a necessidade surja.

3.4.9 – Versões



Painel de "Versões"

Aqui você pode ver qual a versão no servidor do *teamwork* na primeira coluna, no seu diretório local de modelos em seguida, e por fim o número do último modelo

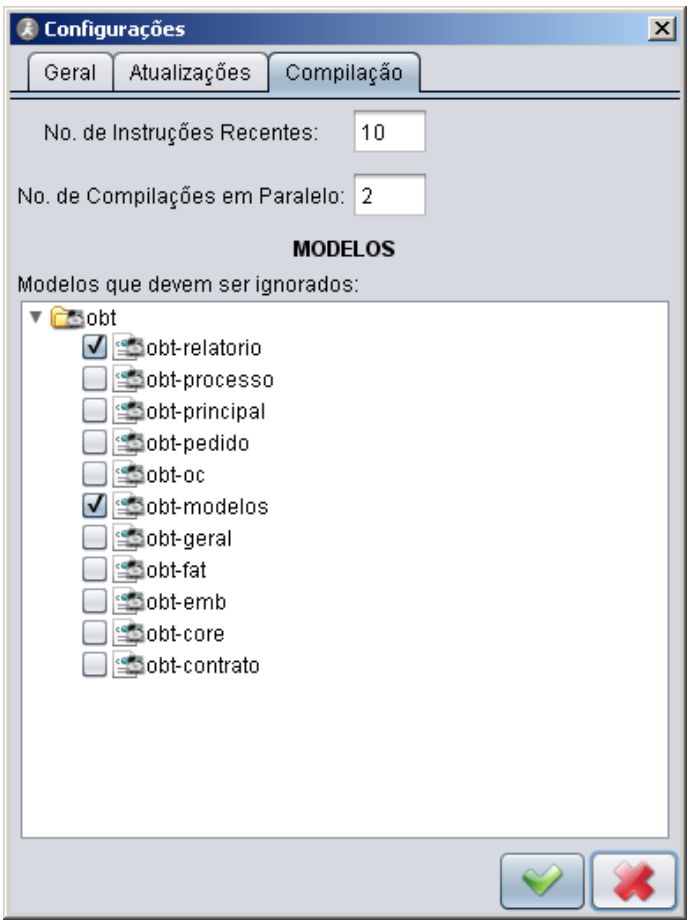
compilado usando o Express. As versões que estiverem defasadas em relação à versão do servidor estarão em vermelho.

Avançado: para desativar o acesso ao *teamwork*, basta adicionar o argumento no_teamwork durante a inicialização do Express. Essa configuração não está na interface porque, sendo única para uma equipe, é recomendado que seja feita no próprio atalho que todos usam para acionar o aplicativo.

3.4.10 – Configurações

A aba de "Compilação" contém a configuração do número máximo de instruções recentes exibidas no painel respectivo, sendo descartadas as mais antigas toda vez que esse número for excedido, e o número de compilações em paralelo.

Dica: recomenda-se que o número de compilações seja o número de *cores* do seu processador. Um número muito grande acaba prejudicando a performance devido ao *overhead* gerado pela necessidade de controle dos diversos processos.



Diálogo de "Configurações", aba de "Compilação"

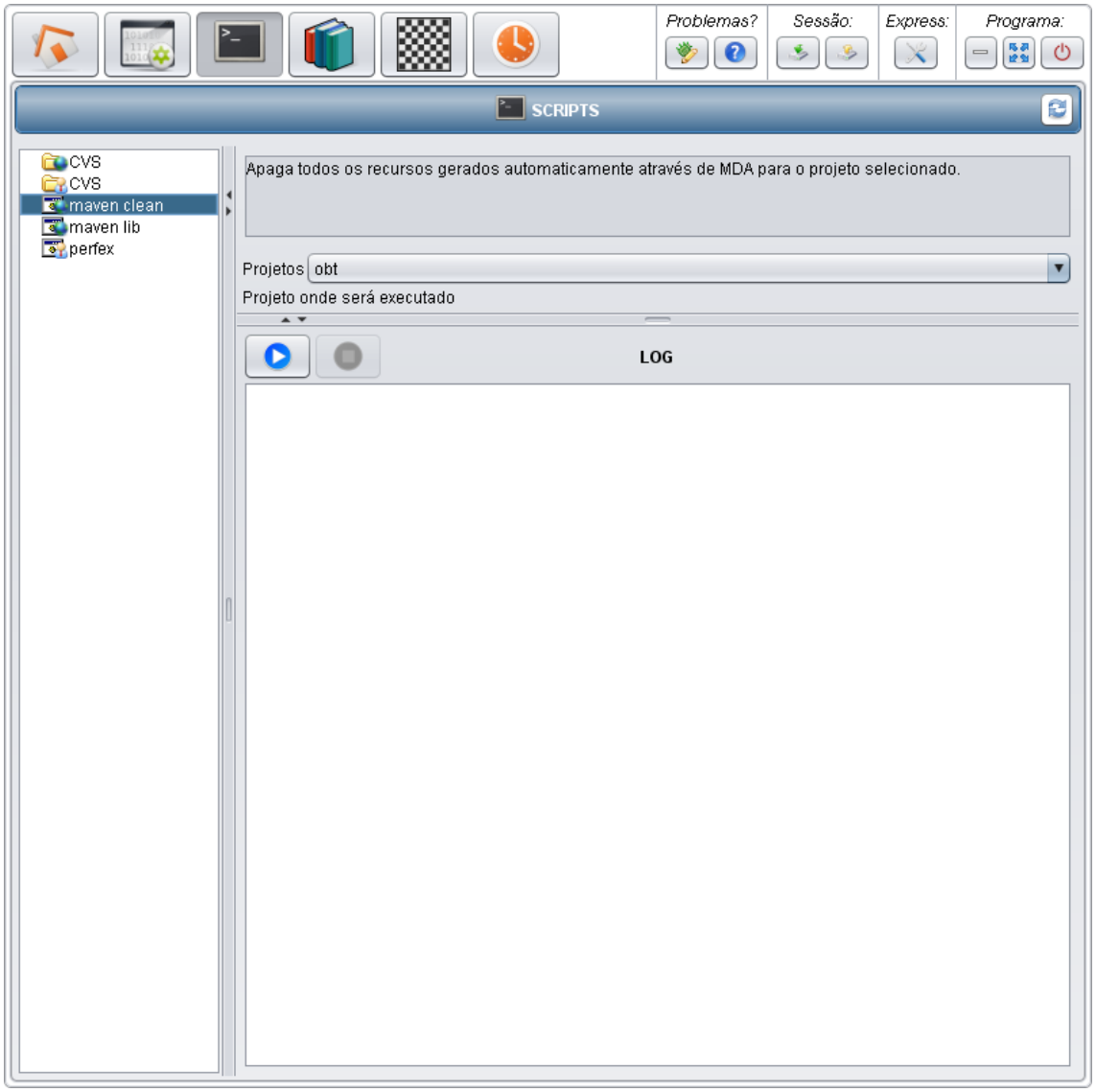
Também temos uma árvore contendo os modelos no *workspace* do usuário para que este selecione aqueles que não devem ser validados e processados, pois têm somente informações auxiliares que não devem gerar código.

PILOTO: o modelo obt-modelos é um desses casos e deve ser selecionado.

3.5 – Scripts

3.5.1 – Descrição

Esse assistente centraliza diversos scripts úteis e permite sua execução.



Assistente "Scripts"

3.5.2 – Scripts

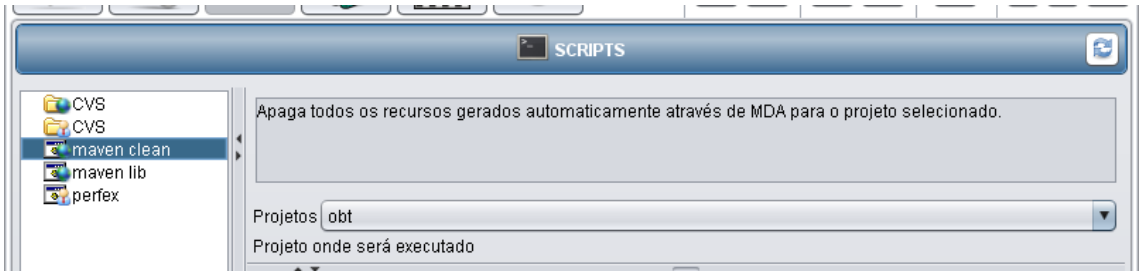
A árvore à esquerda contém os diversos scripts. Ao selecionar um, sua descrição será atualizada e o usuário poderá executá-lo.

Avançado: a árvore é montada a partir do diretório “Scripts” das pastas de configurações.

PILOTO: o gerente pode, ao clicar duas vezes em qualquer item (pasta ou arquivo), abri-lo com a ação padrão do sistema operacional para este. Isso é útil para editá-los.

3.5.3 – Descrição

Esse painel só é exibido quando o usuário selecionar um script com descrição. Veja o exemplo abaixo:



Script com descrição selecionado

Nesse painel, pode-se ler a descrição do script e editar os parâmetros utilizados na sua execução. Note que cada argumento também tem sua própria descrição.

Avançado: a descrição é obtida a partir de um arquivo XML simplificado no mesmo diretório e com o mesmo nome do arquivo de script ao qual está associado. Seu formato é o seguinte: primeiro, vem a descrição do script, que pode ser em HTML. Após, temos as variáveis assim: <VAR NAME = "nome_variavel" DEFAULT = "valor_padrao">, sendo que o atributo DEFAULT pode ser omitido. Caso seja preenchido, o campo de edição referente a essa variável já virá preenchido. Na linha abaixo de cada variável, pode ser colocada uma descrição HTML desta. Veja o exemplo abaixo:

Arquivo XML:

“Imprime na tela os argumentos passados

<VAR NAME = "Primeiro Argumento" DEFAULT = "Valor Default 1">

Esse argumento é muito importante.

<VAR NAME = "Segundo Argumento" DEFAULT = "Valor Default 2">

Pode deixar em branco que ele só serve para alguns casos.

<VAR NAME = "Terceiro Argumento" DEFAULT = "Valor Default 3">

Esse argumento pode causar uma catástrofe! Cuidado!

<VAR NAME = "Quarto Argumento" DEFAULT = "Valor Default 4">

Esse script tem muitos argumentos.”

O nome reservado SPECIAL_PROJECT para uma variável gera uma listagem com todos os projetos disponíveis como valores. Veja o exemplo sobre como utilizá-la:

Arquivo BAT:

“cd\"%1"

maven clean”

Arquivo XML:

“Apaga todos os recursos gerados automaticamente.

<VAR NAME = "SPECIAL_PROJECT">

Projeto onde será executado”

Caso seja necessário durante o script que o usuário forneça informações para prosseguir, através do seguinte comando especial no script será mostrado um diálogo de entrada de dados:

“set /p "variável_que_guarda_a_entrada=[P]Mensagem exibida no diálogo"”

Também há a opção de usarmos uma combobox com valores pré-determinados ao invés de um campo texto básico para receber um argumento. Após a descrição da variável no arquivo XML, você deve incluir a descrição, uma a uma, dos possíveis valores, especificando o valor a ser passado no script (na propriedade NAME) e o valor a ser exibido na interface (na propriedade DESCRIPTION). Veja o exemplo:

Arquivo XML:

“Copia os dados de serviços e perfis associados do banco CNBE em PortalDJ para o banco selecionado nos argumentos.

<VAR NAME = "Esquema" DEFAULT = "">

<VALUE NAME = "E" DESCRIPTION = "E - CNBE">

<VALUE NAME = "W" DESCRIPTION = "W - CNBW">

<VALUE NAME = "T" DESCRIPTION = "T - CNBE/CNBW">

Esquema do banco para o qual os dados serão copiados.”

3.5.4 – Log

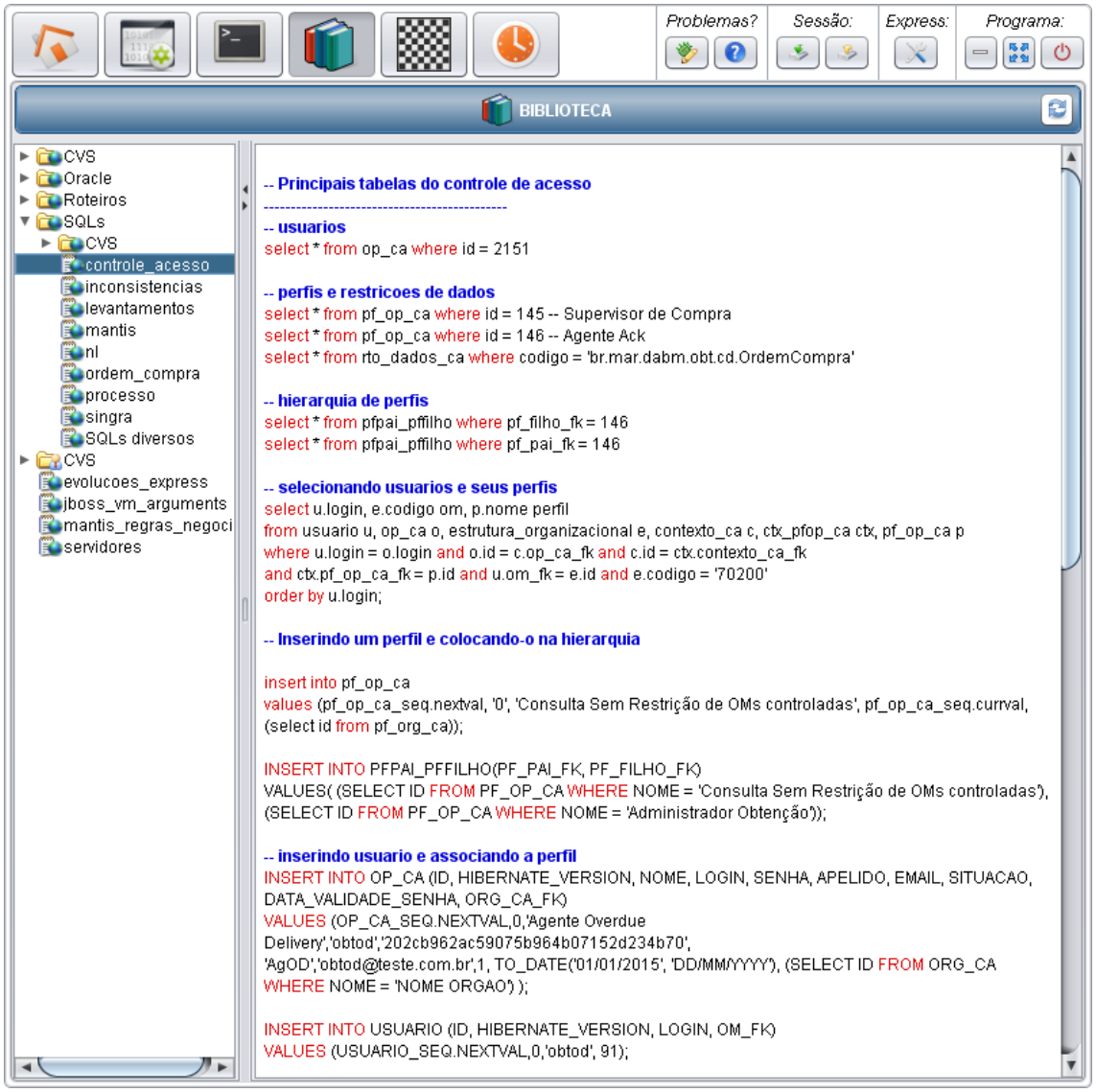
Aqui estão os botões de “Iniciar” e “Parar”, desabilitado a princípio. Ao pressionar o primeiro, o usuário começará a executar o script selecionado com os argumentos passados (se houver) e os dois botões terão seu estado trocado: “Iniciar” se desativa, enquanto que “Parar” fica ativo para o caso de desejar interromper a execução.

A partir daí, todo comando rodado e seu resultado serão escritos na área de texto de log.

3.6 – Biblioteca

3.6.1 – Descrição

A finalidade da “Biblioteca” é funcionar como uma coletânea de arquivos importantes, centralizando-os e facilitando o acesso a eles.



Assistente "Biblioteca" exibindo um arquivo SQL

PILOTO: Aqui você encontrará alguns roteiros passo a passo de certas operações, alguns SQL's importantes e os argumentos a serem passados na execução do JBoss pelo Eclipse.

3.6.2 – Arquivos

No painel esquerdo, há uma árvore de arquivos e ao selecionar um seu conteúdo será exibido no próximo painel.

Avançado: a árvore é montada a partir do diretório "Biblioteca" das pastas de configurações.

3.6.3 – Conteúdo

Mostra o conteúdo do arquivo selecionada da árvore acima.

Dica: caso o arquivo termine com “sql” ou “SQL”, o Express irá formatar seu código, destacando comentários e palavras reservadas.

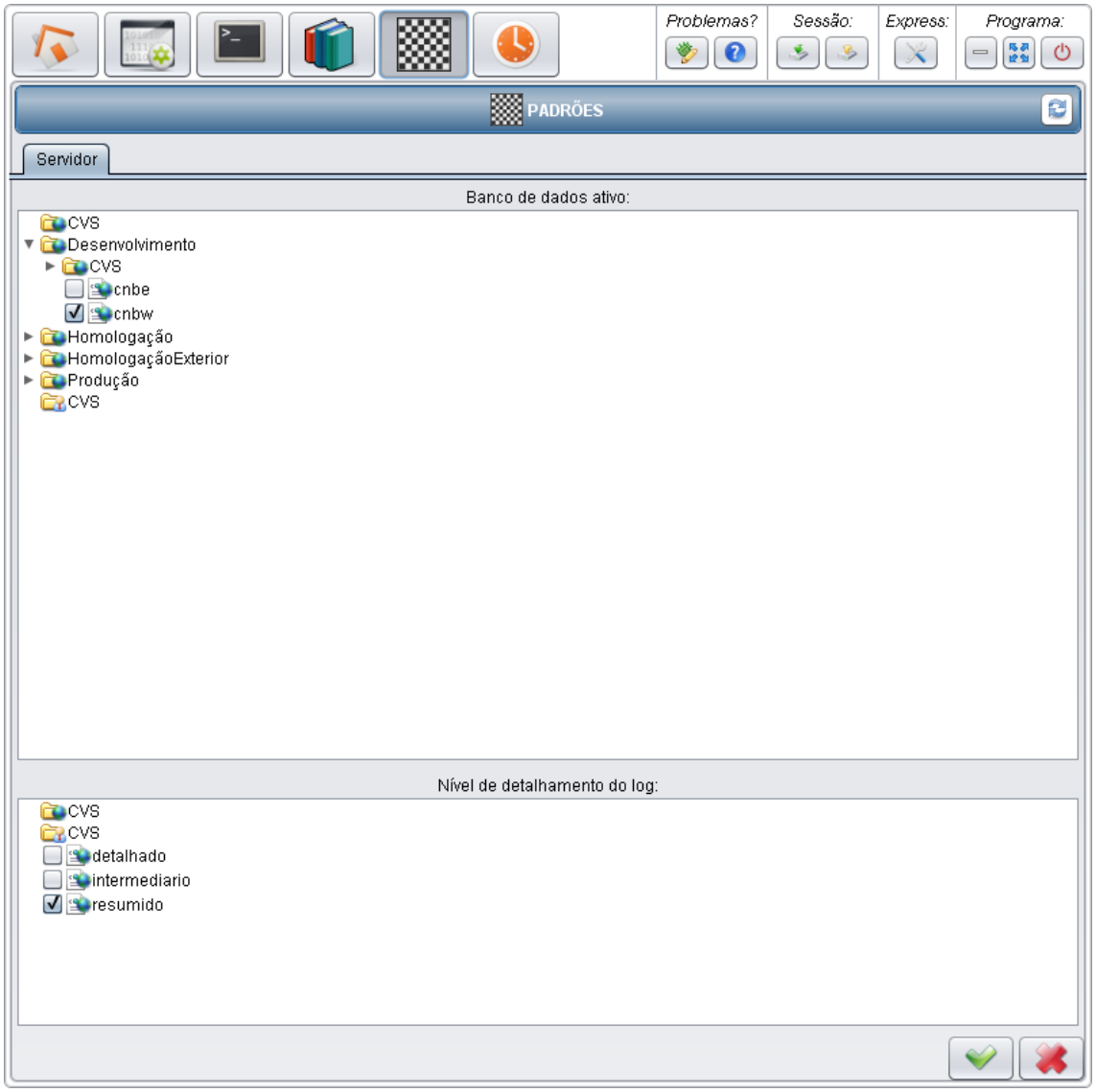
3.7 – Padrões

3.7.1 – Descrição

Esse assistente tem como finalidade fornecer aos usuários um modo fácil de aplicar configurações padronizadas no seu ambiente. Por enquanto, há somente a aba “Servidor”, descrita logo abaixo.

3.7.2 – Servidor

Nessa aba, podemos configurar o banco de dados ativo no servidor JBoss e o nível de detalhamento do log impresso no console onde executa:



Assistente "Padrões", aba "Servidor"

O arquivo selecionado em ambas árvores será copiado para seu diretório respectivo ao pressionar o botão aplicar no canto direito inferior. Caso pressione o botão cancelar, as configurações selecionadas voltarão a seu valor inicial.

Avançado: a primeira árvore será montada a partir do diretório de configurações "Bancos de Dados", e a segunda, a partir do diretório "JBoss Log".

PILOTO: em banco de dados ativo estão todos os bancos do projeto, divididos de acordo com a finalidade de cada servidor e nomeados de acordo com o usuário utilizado.

Em nível de detalhamento do log, temos três configurações: detalhado, intermediário e resumido, que funcionam assim:

- O detalhado é o padrão ao qual todos estão acostumados.

- O resumido provê uma performance muito maior ao evitar excesso de mensagens de log, porém pode ocorrer algum erro que não será registrado, portanto, use com cautela.

- Já o intermediário, como o nome diz, está entre os dois em desempenho e detalhamento do registro.

3.8 – Tarefas

O último dos assistentes é o de tarefas. Criado para agrupar um número de diversos tipos de ações, inicialmente tem somente uma tarefa de *backup*.

O usuário especifica uma pasta ou arquivo da rede e um local de destino. O primeiro será então compactado no segundo. A terceira configuração é uma expressão CRON que determina a freqüência com a qual o backup será feito. CRON é uma linguagem poderosa, simples de usar, uma vez que há muito material e exemplos na Internet, e que permite que o usuário tenha um grau de controle alto na definição de freqüência. Mais instruções e exemplos são encontrados diretamente no aplicativo.

O intuito inicial era ter alguma forma de fazer *backup* dos repositórios remotos, então foi criada essa funcionalidade que pode tanto atender a esse, quanto a muitos outros casos e necessidades.

3.9 – Resolução de Problemas

Caso haja algum problema e seja necessário fechar a aplicação há dois modos. Caso a janela esteja no seu modo normal, vá ao Gerenciador de Tarefas e feche o Express. Caso o Express esteja minimizado para a bandeja e não seja possível restaura-lo, vá ao Gerenciador de Tarefas e feche o processo chamado javaw.exe que ocupe entre 40 e 60MB de memória RAM.

Procure também instruções de como relatar esses erros para a equipe de desenvolvimento do Express.

Capítulo 4

Aspectos técnicos do aplicativo

4.1 – Mesclagem de repositórios

Um dos recursos fundamentais, como descrito no manual, é a mesclagem de repositórios. O código que monta as árvores é bem prolongado e pode ser encontrado nos arquivo FileTree.java e FileTreeNode.java, respectivamente anexos A e B.

A configuração das pastas onde cada tipo de repositório e pasta de assistente fica está em duas enumerações:

```
package express.config;

import java.io.File;

/**
 *
 * Enumera os diversos repositórios de dados que serão
utilizados para obter os recursos da
 * aplicação.
 *
 * @author Gebara
 * @see Resource
 */
public enum Repository {

      SERVER("Config" + File.separator + "Todos" +
File.separator), PROFILE("Config" + File.separator
                  + AppConfig.PROFILE.getDescription() +
File.separator), USER(DataConfig.EXPRESS_HOME);

      private String path;

      private Repository(String path) {
            this.path = new File(path +
File.separator).getAbsolutePath() + File.separator;
      }

      public String getPath() {
            return path;
      }

}
```

```
package express.config;

import java.io.File;

/**
 * Enumera todos os tipos de recursos utilizados pelo Express.
Seus nomes serão o próprio caminho
 * utilizado para buscá-los (dentro dos repositórios
normalmente).
 *
 * @author Gebara
 * @see Repository
 */
public enum Resource {

      ALERTS("Avisos"), DATABASES("Bancos de Dados"),
BOOKMARKS("Favoritos"), LIBRARY("Biblioteca"), LINKS(
                  "Links"), RECENT_INSTRUCTIONS("Instruções
Recentes"), JBOSS_LOG_TYPE("JBoss Log"), SCRIPTS(
                  "Scripts"), TASKS("Tarefas"),
SESSION_FILE("Session.cfg", true);

      private String path;
      private boolean isFile;

      private Resource(String path) {
            this(path, false);
      }

      private Resource(String path, boolean isFile) {
            this.path = path + (isFile ? "" : File.separator);
            this.isFile = isFile;
      }

      public String getPath() {
            return path;
      }

      public boolean isFile() {
            return isFile;
      }

}
```

4.2 – Perfis

O utilitário foi montado de modo a suportar que uma variedade de perfis rode no mesmo aplicativo. Esses perfis são acionados através do atalho, que contém um argumento que determina qual perfil executará.

Cada perfil, DEVELOPER e MANAGER abaixo por exemplo, configura se o usuário que o utilizará poderá editar os arquivos no repositório de todos e de seu perfil e a que assistentes terá acesso. Isso permite que o software seja estendido através da criação de novos assistentes facilmente.

```
/**
 *
 * Enumera os diferentes perfis disponíveis e define os módulos
({@link AssistantController}) aos
 * quais cada um tem acesso.
 *
 * @author Gebara
 *
 */
public enum Profile {
      DEVELOPER("/dev", "Desenvolvedor", false, false) {
            @Override
            public void populateAccessibleModules() {

      accessibleModules.add(HomeAssistantController.class);

      accessibleModules.add(CompilerAssistantController.class);

      accessibleModules.add(ScriptsAssistantController.class);

      accessibleModules.add(LibraryAssistantController.class);

      accessibleModules.add(StandardsAssistantController.class);
            }
      },
      MANAGER("/ger", "Gerente", true, true) {
            @Override
            public void populateAccessibleModules() {

      accessibleModules.add(HomeAssistantController.class);

      accessibleModules.add(CompilerAssistantController.class);

      accessibleModules.add(ScriptsAssistantController.class);

      accessibleModules.add(LibraryAssistantController.class);

      accessibleModules.add(StandardsAssistantController.class);

      accessibleModules.add(TasksAssistantController.class);
            }
      };

      private String argument;
      private String description;
      protected List<Class<? extends AssistantController<?>>>
accessibleModules;
      private boolean canAccessProfile;
      private boolean canAccessAllProfile;
```

```
      Profile(String argument, String description, boolean
canAccessProfile,
                  boolean canAccessAllProfile) {
            this.argument = argument;
            this.description = description;
            this.accessibleModules = new ArrayList<Class<?
extends AssistantController<?>>>();
            populateAccessibleModules();
            this.canAccessProfile = canAccessProfile;
            this.canAccessAllProfile = canAccessAllProfile;
      }

      abstract protected void populateAccessibleModules();

      public String getArgument() {
            return argument;
      }

      public String getDescription() {
            return description;
      }

      public String getArgumentHelp() {
            return getArgument() + "\tExecuta o aplicativo usando
o perfil " + getDescription() + ".";
      }

      public List<Class<? extends AssistantController<?>>>
getAccessibleModules() {
            return accessibleModules;
      }

      public boolean canAccessProfile() {
            return canAccessProfile;
      }

      public boolean canAccessAllProfile() {
            return canAccessAllProfile;
      }

}
```

4.3 – Assistentes

Há duas classes abstratas básicas: a AssistantController e a AssistantPanel. A AssistantController contém todo o funcionamento-padrão básico esperado de um controlador de um dos assistentes do Express, basicamente a criação do seu

AssistantPanel e o método de salvamento, que podem ser sobrescritos para customizar seu funcionamento.

```
package express.control.assistant;

import java.awt.Cursor;

import express.config.SessionConfig;
import express.control.Controller;
import express.control.main.MainController;
import express.gui.assistant.AssistantPanel;
import express.gui.main.MainFrame;

/**
 *
 * Classe com o funcionamento e configuração básicos dos
assistentes, como seu nome, ícone e se deve
 * ser atualizado automaticamente.
 *
 * @author Gebara
 * @see AssistantPanel
 *
 */
public abstract class AssistantController<AP extends
AssistantPanel> extends Controller {

	private static final long serialVersionUID = 1L;

	protected AP panel;

	protected String name;
	protected String iconName;
	protected String description;

	private boolean loaded;

	public AssistantController(String name, String iconName,
String description) {
		this(name, iconName, description, true);
	}

	public AssistantController(String name, String iconName,
String description, boolean autoRefresh) {
		this.name = name;
		this.iconName = iconName;
		this.description = description;
		this.panel = createPanel();
		SessionConfig.instance.setDefaultAutoRefresh(name,
autoRefresh);
	}

	public AP getPanel() {
		return panel;
	}
```

```
      public String getName() {
            return name;
      }

      public String getIconName() {
            return iconName;
      }

      public String getDescription() {
            return description;
      }

      public void refresh(boolean keepSizes) {
            MainFrame mainFrame =
MainController.instance.getMainFrame();
            if (mainFrame != null) {
                  mainFrame.setCursor(new
Cursor(Cursor.WAIT_CURSOR));
            }

            loaded = true;
            panel.refresh(keepSizes);
            panel.validate();

            if (mainFrame != null) {
                  mainFrame.setCursor(new
Cursor(Cursor.DEFAULT_CURSOR));
            }
      }

      public void save() {}

      protected abstract AP createPanel();

      public boolean isLoaded() {
            return loaded;
      }
}
```

4.4 – Processamento em paralelo

O assistente de compilação mais utilizado da aplicação e com as funções mais críticas e impactantes. Nele que se encontra o coração da aplicação e grande motivação: o processamento dos conjuntos de instruções envolvidas no ciclo de desenvolvimento do sistema, sejam gerações de código, compilações ou instalações no servidor, chamados daqui em diante de macro-processos para que não haja ambigüidade com nenhum outro termo da computação.

Para que o objetivo fosse atingido, foi desenvolvido um mecanismo de distribuição dos macro-processos a partir de uma *thread* alimentadora para diversas *threads* executoras. Assim que uma executora termina seu trabalho (um macro-processo), pede um novo para a alimentadora.

Esse mecanismo obedece a dois conjuntos de regras basicamente e que serão explanados nas próximas seções:

1) Regras de escalonamento de macro-processos com objetivo de garantir o tempo ótimo de execução do conjunto de macro-processos.

2) Regras de interdependência entre os módulos (as menores partes que podem ser alvos de macro-processos da aplicação).

4.4.1 – Escalonamento de macro-processos

Para garantir o menor tempo total utilizando as diversas execuções em paralelo, foi desenvolvido um algoritmo que distribui os macro-processos em ordem decrescente de tamanho para cada fila assim que estiver vazia.

A partir do seguinte raciocínio dedutivo, conclui-se que esta é a forma ótima de se garantir o menor tempo total, o que se traduz no mínimo tempo máximo de uma fila de execução:

1) Consideremos que haja n filas de execução. Caso haja m processos, sendo m <= n, se distribuirmos um processo para cada fila, teremos a distribuição ideal, com tempo máximo P equivalente à duração do macro-processo mais demorado.

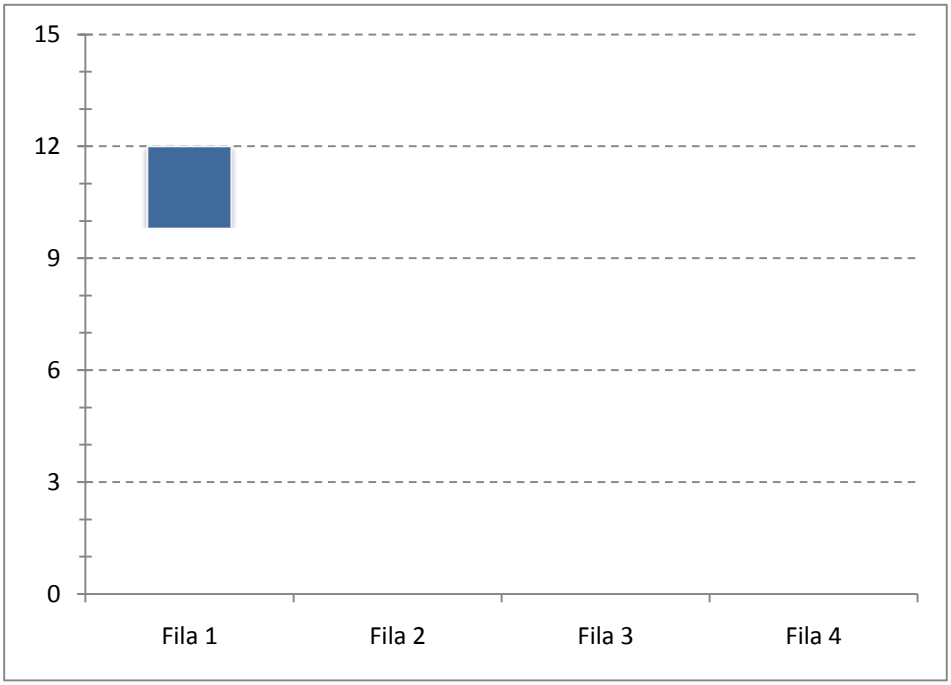


Figura 2.3 – Exemplo com n filas e P = 8

2) Conforme adicionarmos mais macro-processos, quando m > n e enquanto que, para cada fila, os novos processos distribuídos somados com os finalizados não ultrapassarem o maior tempo P, continuaremos tendo a distribuição ótima.

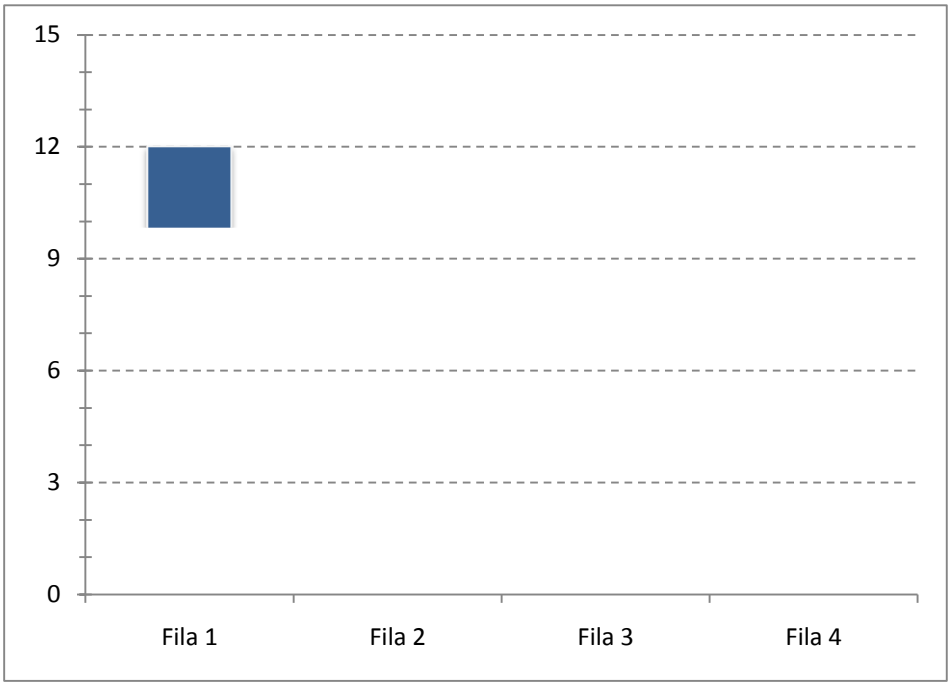


Figura 2.3 – Exemplo com n filas e P = 8

3) Quando formos adicionar um macro-processo que faça o tempo de uma fila ultrapassar P, continuaremos tendo o tempo ótimo, uma vez que teremos um novo tempo máximo mínimo para as filas, já que:

Essa fila será a teve, antes da adição do último macro-processo, o menor tempo de execução entre todas, sendo a fila ideal para receber uma instrução extra e manter nossa função-objetivo ótima;

O processo a ser adicionado será o menor de todos, o que, juntamente com o ponto anterior, garante que a adição à função-objetivo, antes P, continuará sendo mínima para o caso, uma vez que ela é o resultado da soma do tempo de execução corrente da fila mais o tempo do processo a ser adicionado e esses serão mínimos.

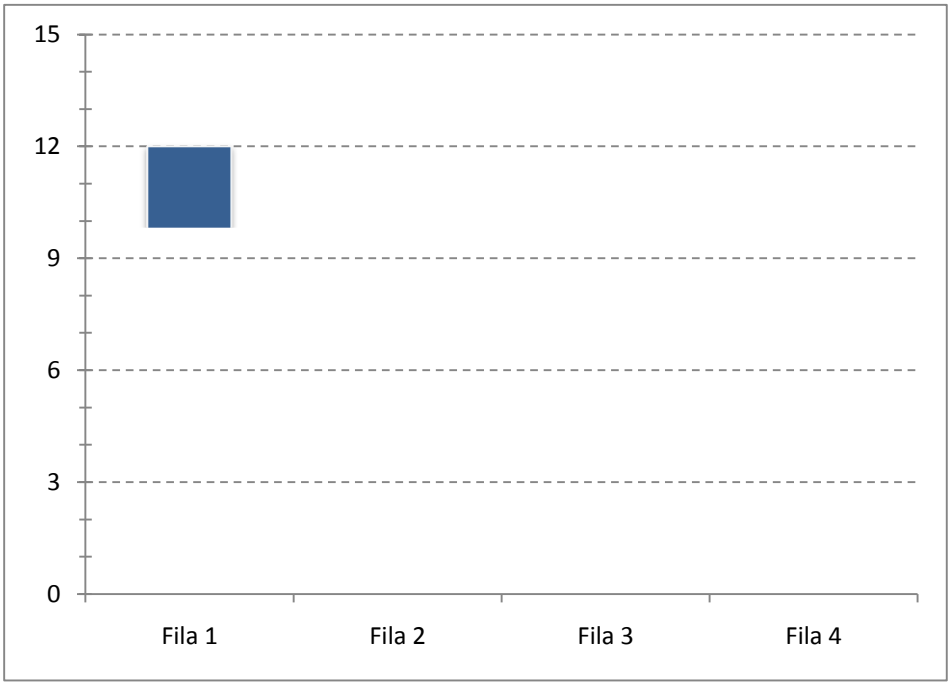


Figura 2.3 – Exemplo com n filas e P = 8

4) O raciocínio levantado em 3 pode ser usado para as sucessivas adições de macro-processos e nos mostra que essa é a forma ideal de distribuição para o nosso caso.

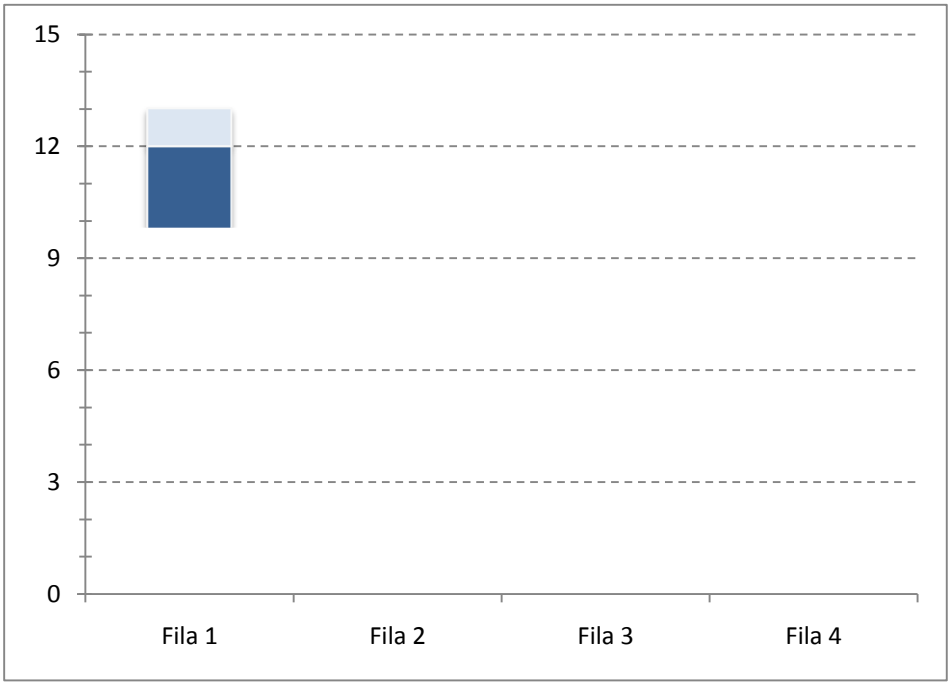


Figura 2.3 – Exemplo com n filas e P = 8

Para determinar de antemão quais são os processos maiores, foram realizados alguns testes e empiricamente se determinou que havia uma relação entre o tamanho do arquivo XMI e o tempo que demorava para se gerar o código, como podemos ver na seguinte tabela:

Módulo	Tempo	Segundos	KB	KBps
core	01:16	76	9289	122
proc	01:23	83	1867	22
oc	00:34	34	608	18
geral	00:54	54	1125	21
princ	00:31	31	655	21
fat	00:24	24	448	19
pedido	02:40	160	3473	22

Tabela 2.1 – Médias para a geração de código dos diversos arquivos XMI

A coluna KBps mostra a velocidade do processamento, muito próxima a uma velocidade da 20 KBps. A única exceção é o core, uma vez que é um módulo diferente dos demais. Enquanto que os outros determinam o comportamento da aplicação (o fluxo entre as telas e as chamadas de métodos a partir de eventos da interface gráfica), o core define as classes de domínio e de serviço.

Para contornar essa exceção, toda vez que um módulo tiver o nome de core, seu tamanho é dividido por seis de modo a manter a proporcionalidade observada para o tempo de processamento de cada módulo.

A metodologia utilizada para essa medição seguiu duas regras básicas:

1) Os tempos foram obtidos num mesmo Core 2 Duo com 3GB de RAM, sem tarefas ou aplicativos rodando no fundo e representam a média de três execuções consecutivas.

2) Antes de cada uma dessas seqüências executar, foi ainda rodada uma execução prévia para garantir que não haveria qualquer tipo de cache influenciando na média final e no estudo e, como não foi observada diferença significativa (diferente da variação de tempo na seqüência), esta execução prévia foi adicionada no cálculo da média final.

A mesma ordenação de tempo dos módulos em relação à demora para a geração de código foi observada na compilação e instalação do servidor e, logo, a mesma é aplicada nessas outras etapas.

Como há uma correspondência entre os nomes dos módulos nas quais devem ser realizadas essas diferentes etapas de processamento, após simples processamento é possível reaplicar a ordenação obtida para garantir o tempo ótimo.

4.4.2 – Interdependências entre os módulos

A aplicação do MDArte se divide seguindo uma organização pré-determinada de agrupamento dos módulos que a compõe. Os módulos são as menores partes da aplicação que podem ser alvo de macro-processos, ou seja, chamaremos de módulo as menores partes do sistema que podem ser ou processadas para gerar código, ou serem compiladas para gerar *bytecode*, ou instalada no servidor.

O primeiro dos três tipos corresponde aos arquivos XMI gerados pelo MagicDraw. Já o segundo e o terceiro correspondem a divisões da aplicação geradas a

partir da divisão feito pelos arquivos XMI. Se, por exemplo, divido minha aplicação de controle de recursos humanos de uma empresa em um arquivo XMI contendo todos os diagramas relativos ao cadastro das pessoas e em outro contendo todas as funções administrativas e de folha de pagamento do software, essa divisão será replicada no código gerado a partir desses arquivos, agrupados em dois grandes grupos de acordo com o que foi colocado em cada XMI.

Essa divisão é importante para granulizar mais as partes do sistema, tornando seu processamento menos custoso, uma vez que uma das restrições primordiais do AndroMDA é que não é possível processar o que foi alterado somente, mas o módulo todo, e desse modo diminuímos o *overhead* de processamento do que não foi alterado, pois o todo é menor.

O código é então gerado e distribuído utilizando a seguinte organização de diretórios dentro da pasta-raiz do projeto:

1) Pasta “common” contendo classes compartilhadas por diversas partes do sistema. Não é gerado código nessa pasta.
2) Pasta “core” com todas as classes de domínio (no subdiretório “cd”) e de serviço (no subdiretório “cs”). As primeiras correspondem às entidades do sistema e estão todas na mesma pasta, uma vez que são compartilhadas por diferentes partes do sistema. Já as últimas correspondem a lógica de negócio do sistema e são divididas em mais um nível de acordo com o módulo ao qual pertencem. Toda essa pasta é gerada a partir do arquivo XMI “core”.
3) Pasta “mda” com todos os arquivos XMI. Nenhum código é gerado aqui.
4) Pasta “web” com toda a parte da interface gráfica e suas chamadas às regras de negócio. Se divide em mais um nível de acordo com o módulo do sistema ao qual a classe pertence.

Para facilitar a compreensão, segue um exemplo de projeto:

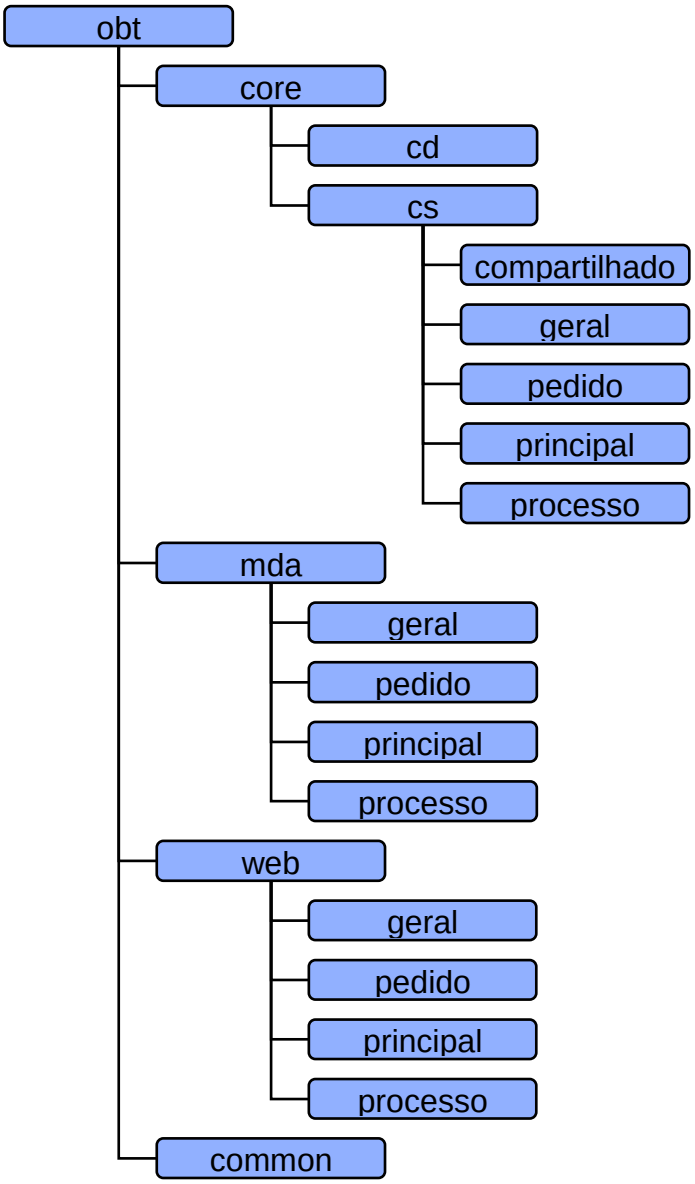


Figura 2.1 – Exemplo de organização em módulos de um projeto

Observa-se ainda no exemplo algumas divisões não citadas, como o módulo "core\cs\shared" e o "web\layout". Como todos são partes menores da aplicação, com tempos de processamento significativamente menores que todos os demais, são simplesmente colocados no fim da fila priorização de execução sem prejuízo para o algoritmo de otimização.

A *thread* alimentadora deve distribuir os macro-processos seguindo a ordem:

1) Arquivos XMI
2) Common
3) Core
4) Web

Dentro dos grupos, os macro-processos podem ser rodados em paralelo, devendo um grupo acabar totalmente antes de outro ser iniciado. Caso seja feita a compilação do core, o Jboss (servidor utilizado pelo MDArte) deve ser desligado, uma vez que essa é uma restrição da plataforma, pois em outro caso as alterações não serão refletidas no sistema. Uma caixa de seleção no programa define se esse deve ser iniciado, o que é feito assim que o core acaba de ser compilado para diminuir o tempo total, uma vez que não precisamos esperar o resto dos macro-processos terminar já que não há restrição alguma para os outros grupos.

4.5 – Processamento da descrição de scripts

O processamento do arquivo XML de descrição é feito na classe ScriptsDescriptionPanel, no método initComponents:

```
private void initComponents() {
           lblsNames = new ArrayList<JLabel>();
           editors = new ArrayList<JComponent>();
           lblsDescs = new ArrayList<JLabel>();

           if (!xmlFile.exists()) {
                txaDescription = null;
           }
           else {
                try {
                     List<String> lines =
FileUtilities.readFile(xmlFile);
                     String desc = "";
                     Variable var = null;
                     vars = new ArrayList<Variable>();
                     boolean varMode = false;// indica se já
estamos no modo de variáveis, ou seja, acabou a
                     // descrição0
                     for (String string : lines) {
                          if (string.startsWith("<VAR")) {
                               String[] parts =
string.split("\"");
                               if (var != null) {
                                    vars.add(var);
                               }
                               var = new Variable();
                               var.name = parts[1];
                               if (parts.length >= 5) {
                                    var.defaultValue =
parts[3];
                               }
                               varMode = true;
                          }
```

```
                              else if (string.startsWith("<VALUE"))
{
                                    if (var != null) {
                                          String[] parts =
string.split("\"");
                                          var.values.add(parts[1]);

      var.valueDescriptions.add(parts[3]);
                                    }
                              }
                              else if (varMode) {
                                    if (var != null) {
                                          var.description += string;
                                    }
                              }
                              else {
                                    desc += string +
DataConfig.LINE_END;
                              }
                        }
                        if (var != null) {
                              vars.add(var);
                        }

                        txaDescription = new JTextArea(desc);

      GUIUtilities.formaTextAreaAsMultiLineLabel(txaDescription,
true);

                        for (Variable variable : vars) {
                              if
(variable.name.equals(SPECIAL_PROJECT)) {
                                    lblsNames.add(new
JLabel("Projetos"));
                                    DefaultMutableTreeNode dirs =
MainController.instance.getMavenDirs();
                                    for (int i = 0; i <
dirs.getChildCount(); i++) {
                                          File dir = (File)
((DefaultMutableTreeNode) dirs.getChildAt(i))

      .getUserObject();

      variable.values.add(dir.getAbsolutePath());

      variable.valueDescriptions.add(dir.getName());
                                    }

                                    if (variable.values.size() > 0)
{
                                          JComboBox cbb = new
JComboBox(variable.valueDescriptions.toArray());
                                          editors.add(cbb);
                                          if
(variable.defaultValue.length() == 0) {
```

```
      variable.defaultValue = "0";
                                        }
                                        try {

      cbb.setSelectedItem(Integer.parseInt(variable.defaultValue
));
                                        }
                                        catch
(NumberFormatException e) {

      cbb.setSelectedItem(0);
                                        }
                                  }
                                  else {
                                        editors.add(new
JTextField(variable.defaultValue));
                                  }
                            }
                            else {
                                  lblsNames.add(new
JLabel(variable.name));
                                  if (variable.values.size() > 0)
{
                                        JComboBox cbb = new
JComboBox(variable.valueDescriptions.toArray());
                                        editors.add(cbb);
                                        if
(variable.defaultValue.length() == 0) {

      variable.defaultValue = "0";
                                        }
                                        try {

      cbb.setSelectedItem(Integer.parseInt(variable.defaultValue
));
                                        }
                                        catch
(NumberFormatException e) {

      cbb.setSelectedItem(0);
                                        }
                                  }
                                  else {
                                        editors.add(new
JTextField(variable.defaultValue));
                                  }
                            }

                            lblsDescs.add(new
JLabel(variable.description));
                      }
                }
                catch (Exception e) {
                      e.printStackTrace();
                }
```

```
            }
      }
```

Em ScriptThread é feito o processamento do output do script e, caso detecte [P], um marcador que indica uma pergunta, exibe um diálogo com a mensagem e um campo para que texto seja escrito e inputado no console.

```
if (s != null && (s.contains("[P]"))) {
      int i0 = s.lastIndexOf("[P]") + 3;
      String result =
JOptionPane.showInputDialog(s.substring(i0));
      if (result != null) {
            BufferedOutputStream bufferout = new
BufferedOutputStream(child.getOutputStream());
            PrintWriter commandInput = new PrintWriter((new
OutputStreamWriter(bufferout)), true);
            commandInput.println(result);
            commandInput.close();
      }
}
```

4.6 – Configurações

As configurações, como disposição da tela, tamanho e posição dos divisores de painéis, são todas salvas através de um POJO (uma classe Java que basicamente contém atributos e seus métodos de acesso) que é persistido através do mecanismo padrão de escrita e leitura de objetos em arquivos do Java. Esses são os métodos utilizados para as duas operações, extraídos da classe MainController:

```
public SessionConfig loadSessionConfig() {
            try {
                  File sessionFile = new
File(DataConfig.getPath(Resource.SESSION_FILE,
Repository.USER));
                  if (sessionFile.exists()) {
                        ObjectInputStream ois = new
ObjectInputStream(new FileInputStream(sessionFile));
                        return (SessionConfig) ois.readObject();
                  }
            }
            catch (Exception e) {
                  e.printStackTrace();
            }

            noConfig = true;
```

```
            return new SessionConfig();
      }

public void saveSessionConfig(SessionConfig sessionConfig) {
            // executa o refresh, quando diversas propriedades
sao atualizadas
            sessionConfig.lastUsage = new Date();

            sessionConfig.mainWindowBounds =
mainFrame.getBounds();

            sessionConfig.logWindowBounds =
LogController.instance.getLogDialog().getBounds();
            sessionConfig.logMavenSplit =
LogController.instance.getLogDialog().getMavenSplit();

            if (CompilerAssistantController.instance.isLoaded())
{
                  sessionConfig.instructionsSplit1 =
CompilerAssistantController.instance.getPanel()
                              .getHorizontalSplit();
                  sessionConfig.instructionsSplit2 =
CompilerAssistantController.instance.getPanel()
                              .getHorizontalSplit2();
                  sessionConfig.instructionsSplit3 =
CompilerAssistantController.instance.getPanel()
                              .getHorizontalSplit3();

                  sessionConfig.instructionsActiveTab =
CompilerAssistantController.instance.getPanel()
                              .getActiveTab();
                  sessionConfig.instructionsActiveTab2 =
CompilerAssistantController.instance.getPanel()
                              .getActiveTab2();
                  sessionConfig.instructionsActiveTab3 =
CompilerAssistantController.instance.getPanel()
                              .getActiveTab3();
            }

            if (LibraryAssistantController.instance.isLoaded()) {
                  sessionConfig.libraryHorizontalSplit =
LibraryAssistantController.instance.getPanel()
                              .getHorizontalSplit();
            }

            if (ScriptsAssistantController.instance.isLoaded()) {
                  sessionConfig.scriptsHorizontalSplit =
ScriptsAssistantController.instance.getPanel()
                              .getHorizontalSplit();
                  sessionConfig.scriptsVerticalSplit =
ScriptsAssistantController.instance.getPanel()
                              .getVerticalSplit();
            }

            if (HomeAssistantController.instance.isLoaded()) {
```

```
                sessionConfig.homeHorizontalSplit =
HomeAssistantController.instance.getPanel()
                            .getHorizontalSplit();
          }
          if (TasksAssistantController.instance != null &&
TasksAssistantController.instance.isLoaded()) {
                sessionConfig.tasksHorizontalSplit =
TasksAssistantController.instance.getPanel()
                            .getHorizontalSplit();
          }

          try {
                File configFile = new
File(DataConfig.getPath(Resource.SESSION_FILE,
Repository.USER));
                if (!configFile.exists()) {
                      configFile.createNewFile();
                }
                ObjectOutputStream oos = new
ObjectOutputStream(new FileOutputStream(configFile));
                oos.writeObject(sessionConfig);
                oos.close();
          }
          catch (Exception e) {
                e.printStackTrace();
          }
     }
```

4.7 – Novos recursos do Java 6

Alguns novos recursos do Java 6 foram utilizados. Vale ressaltar a translucência e formato irregular da janela principal e o uso de um ícone na barra de notificações do sistema.

Para customizar a janela principal, é necessário chamar o método setOpaque com o argumento falso nos painéis e customizar seu método de pintura. A classe SpecialFrame, no Anexo E, descreve todo o comportamento para substituir o padrão de uma janela (redimensionar, minimizar, fechar, mover, entre outros). Segue o trecho da repintura da janela:

```
private int arc = 10;

     @Override
     public void paint(Graphics g) {
          Graphics2D g2d = (Graphics2D) g;
          g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);

          Color color = SessionConfig.instance.bgColor;
```

```
            Color brighter = color.brighter();
            Color brighter2 = brighter.brighter();
            LinearGradientPaint gradient = new
LinearGradientPaint(new Point(0, 0), new Point(0,
                        getHeight()), new float[] { 0, 0.03f, 0.5f
}, new Color[] {
                        new Color(brighter.getRed(),
brighter.getGreen(), brighter.getBlue(), 25),
                        new Color(brighter2.getRed(),
brighter2.getGreen(), brighter2.getBlue(), 200),
                        new Color(brighter.getRed(),
brighter.getGreen(), brighter.getBlue(), 100) });
            g2d.setPaint(gradient);
            g.fillRoundRect(2, 2, getWidth() - 4, getHeight() -
4, arc - 4, arc - 4);
            gradient = new LinearGradientPaint(new Point(0, 0),
new Point(getWidth(), 300), new float[] {
                        0.56f, 0.60f, 0.65f, 0.68f, 0.70f, 0.73f,
0.80f, 0.88f }, new Color[] {
                        new Color(brighter.getRed(),
brighter.getGreen(), brighter.getBlue(), 0),
                        new Color(brighter2.getRed(),
brighter2.getGreen(), brighter2.getBlue(), 120),
                        new Color(brighter2.getRed(),
brighter2.getGreen(), brighter2.getBlue(), 180),
                        new Color(brighter.getRed(),
brighter.getGreen(), brighter.getBlue(), 0),
                        new Color(brighter.getRed(),
brighter.getGreen(), brighter.getBlue(), 0),
                        new Color(brighter2.getRed(),
brighter2.getGreen(), brighter2.getBlue(), 150),
                        new Color(brighter2.getRed(),
brighter2.getGreen(), brighter2.getBlue(), 200),
                        new Color(brighter.getRed(),
brighter.getGreen(), brighter.getBlue(), 0) });
            g2d.setPaint(gradient);
            g.fillRoundRect(2, 2, getWidth() - 4, 100, arc - 4,
arc - 4);
            g.setColor(color.darker());
            g.drawRoundRect(0, 0, getWidth() - 1, getHeight() -
1, arc, arc);
            g.setColor(color);
            g.drawRoundRect(0 + 1, 0 + 1, getWidth() - 3,
getHeight() - 3, arc - 2, arc - 2);

            paintComponents(g);
      }
```

O código para o ícone está na classe MainFrame (que estende um JFrame) e segue:

```
public boolean setTray(MavenExecutionStatus status) {
            ImageIcon icon = status.getIcon();
            this.setIconImage(icon.getImage());
            if (trayIcon == null) {
```

```
                SystemTray tray = SystemTray.getSystemTray();
                PopupMenu trayPopup = new PopupMenu("Express");
                MenuItem mitOpen = new MenuItem("Abrir");
                mitOpen.addActionListener(new
OpenActionListener());
                trayPopup.add(mitOpen);
                trayPopup.addSeparator();
                MenuItem mitClose = new MenuItem("Fechar");
                mitClose.addActionListener(new
CloseActionListener());
                trayPopup.add(mitClose);
                trayIcon = new TrayIcon(icon.getImage(),
AppConfig.TITLE + ": " + status.toString(),
                            trayPopup);
                trayIcon.setImageAutoSize(true);

                trayIcon.addActionListener(new
OpenActionListener());
                try {
                      tray.add(trayIcon);
                }
                catch (AWTException e) {
                      return false;
                }
          }
          else {
                trayIcon.setImage(icon.getImage());
                trayIcon.setToolTip(AppConfig.TITLE + ": " +
status.toString());
          }

          return true;
     }

     private class OpenActionListener implements ActionListener
{

          public void actionPerformed(ActionEvent e) {
                setVisible(true);
                setState(Frame.NORMAL);
                toFront();
          }
     }

     private class CloseActionListener implements
ActionListener {

          public void actionPerformed(ActionEvent e) {
                MainController.instance.finish();
          }
     }
     }
```

Capítulo 5

Conclusão

5.1 – Desafios encontrados

A primeira grande dificuldade foi o trabalho de convencimento de que esse era um projeto importante e que traria uma economia muito grande, surpassando seu custo de desenvolvimento. Foi muito interessante o aprendizado nesse caso, uma vez que servirá como exemplo para quaisquer iniciativas futuras.

O Java, apesar de ser uma linguagem com a qual a equipe tinha um alto grau de proficiência, foi talvez a maior das dificuldades técnicas. Por ser multiplataforma, acaba não podendo entrar tão fundo em questões e comportamento específicos de cada plataforma. Diversos scripts, tanto em sua versão Windows, quanto Linux, tiveram que ser desenvolvidos, como:

- Setar uma variável de ambiente
- Matar um processo
- Verificar processos em execução
- Mover arquivo (o do Java era extremamente lento)
- Rodar o JBoss Eclipse (troca o aplicativo atual para o Eclipse e comanda o pressionamento da tecla de atalho para tal)

Um problema fulcral foi o gerenciamento das *threads* em paralelo. Ao terminar uma *thread* no Java, o processo continuava em aberto, por mais que o terminássemos antes de fechá-la. A solução foi utilizar um script nativo para executar essa operação. Por ser um problema silencioso, de detecção extremamente difícil, só foi descoberto depois da liberação das primeiras versões e totalmente por acaso num teste de desempenho.

A infra-estrutura da rede interna do cliente foi outro grande problema encontrado e diversas otimizações foram feitas para corrigir problemas que, ao se desenvolver o aplicativo localmente, não se detectava, como a busca dos ícones dos arquivos para o montagem das FileTrees, que era extremamente demorada em arquivos

na rede. Como o desempenho da rede já era um problema conhecido, estávamos atentos a isso durante as etapas de testes.

5.2 – Discussão dos resultados

Foram feitos alguns testes de desempenho, o mais importante deles o da execução em paralelo comparada com a execução seqüencial. Foi utilizada a mesma metodologia descrita na seção 4.4.1. O resultado foi:

FS1
01:23
01:16
00:34
00:54
00:31
00:24
02:40
07:42

FP1	FP2	FP3
03:03	00:48	03:52
01:26	01:16	
	00:33	
	00:46	
04:29	03:23	03:52

Economia	42%

Tabela 2.1 – Avaliação de desempenho da execução paralela

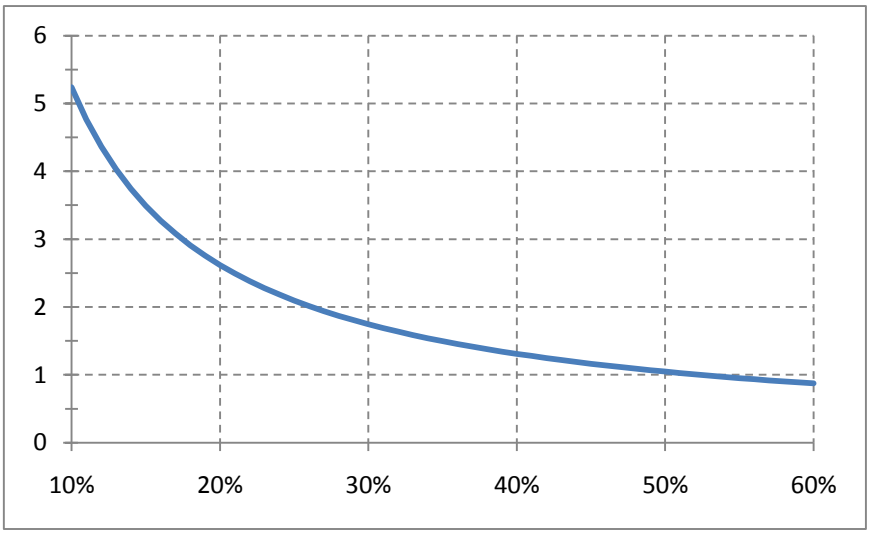
Temos uma fila de macro-processos executados sequencialmente chamada de FS1. O tempo médio final para a compilação dos módulos do projeto foi de 07:42. Por outr olado, ao executarmos os macro-processos em 3 filas em paralelo (FP1, FP2 e FP3) o tempo máximo médio de execução das 3 foi de 04:29, uma economia de 42%.

Mostrou-se então que a execução em paralelo, apesar de ser mais demorada se considerarmos cada macro-processo, uma vez que há menos capacidade de processamento a ele dedicada, no todo é significativamente mais rápida.

Outra conclusão é que o melhor número de macro-processos era de 2, igual ao número de núcleos do processador, e seria muito importante repetir o teste em uma máquina com quatro núcleos. Para justamente se adaptar aos diferentes processadores e

máquinas, que a quantidade de *threads* rodando em paralelo pode ser configurada pela interface gráfica do aplicativo.

Para analisarmos o resultado do ponto de vista financeiro, foram levantados com a Coppetec alguns outros dados. Havia cerca de 25 desenvolvedores, com salários de R$ 1200,00 por mês (aqui nivelamos por baixo pelo salário do estagiário, pois há quem ganhe mais), e, considerando que a equipe de desenvolvimento trabalhou por 11 meses, 50% do seu tempo, seu custo total foi de R$ 6.600,00. Com essas informações, basta saber qual porcentagem do tempo de trabalho em média cada desenvolvedor gasta com o processo de compilação. Uma vez que isso varia de acordo com a tarefa, com o nível de habilidade e com a atenção a detalhes que o funcionário apresenta no dia, esse número é extremamente difícil de ser estimado com a precisão necessária. Como alternativa, foi feito o gráfico abaixo, indicando em quantos meses o utilitário se paga de acordo com essa média.



Fica claro que o *software* se paga entre 5 e 1 mês aproximadamente, se considerarmos que a média fica entre 10% e 50% do tempo gasto em compilação (essa faixa foi confirmada com gerentes e colegas de equipe). O projeto PILOTO, que utiliza o utilitário, demorou 2 anos para entrar em produção. A partir desse número, vemos que, indendependentemente de se pagar em 1 ou 5 meses, foi extremamente compensante o desenvolvimento do utiliário, ainda mais se incluirmos em vista os diversos outros projetos no qual pode ser utilizado e os benefícios que traz ao MDArte, facilitando grandemente sua adoção e difusão.

5.3 – Considerações Finais

Foi extremamente satisfatório o resultado final do projeto, já que cumpriu seus objetivos e foi fechado com um grau alto de qualidade, apesar das dificuldades encontradas no caminho.

Os próximos passos são a expansão do utilitário para outros projetos da Coppetec e sua integração como ferramenta oficial ao MDArte, num caminho crescente de atendimento a um número maior de variedades de projetos.

Bibliografia

- MDA website: http://www.omg.org/mda/
- Schmidt, Douglas C., Model-Driven Engineering: http://www.cs.wustl.edu/~schmidt/PDF/GEI.pdf
- Poole, John D., Model-Driven Architecture: Vision, Standards And Emerging Technologies, (2001): http://www.omg.org/mda/mda_files/Model-Driven_Architecture.pdf
- Agile Model-Driven Development: http://www.agilemodeling.com/essays/amdd.htm
- Model-Driven Engineering: http://www.theenterprisearchitect.eu/archive/2008/03/14/model_driven_engineering

Anexo A – FileTree.java

```
package express.lib.gui.filetree;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.Desktop;
import java.awt.Graphics;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.io.File;
import java.io.FileFilter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import javax.swing.Icon;
import javax.swing.ImageIcon;
import javax.swing.JCheckBox;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTree;
import javax.swing.filechooser.FileSystemView;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeCellRenderer;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.TreePath;

import express.config.AppConfig;
import express.config.DataConfig;
import express.config.ImageSize;
import express.config.Resource;
import express.lib.model.WebOrHdFile;
import express.lib.util.FileUtilities;
import express.lib.util.GUIUtilities;

/**
 *
 * Árvore de arquivos padrão do sistema com diversas
customizações através de parâmetros. Utiliza em
 * seus nós a classe {@link FileTreeNode}, mostrando se o
arquivo ao qual o nó se refere se localiza
 * num repositório local ou remoto.
 *
 * @author Gebara
 *
 */
public class FileTree extends JTree {

      private static final long serialVersionUID = 1L;
```

```
    protected static FileSystemView fsv =
FileSystemView.getFileSystemView();

    /**
     * Indica se a extensão dos arquivos na árvore deve ser
ocultada.
     */
    private boolean hideExtensions;

    /**
     * Indica se deve haver uma checkbox para marcação de
arquivos
     */
    private boolean fileMarking;

    /**
     * Indica se arquivos, desde que dentro das permissões do
usuário, podem ser abertos para edição
     * com dois cliques.
     */
    private boolean openFile = false;

    /**
     * Indica se diretórios, desde que dentro das permissões
do usuário, podem ser abertos para
     * edição com dois cliques.
     */
    private boolean openDirectory = true;

    /**
     * Indica se o conteúdo do arquivo deve ser utilizado para
verificar as marcações, e não só seu
     * nome.
     */
    private boolean verifyContentsOnSelect = true;

    private List<WebOrHdFile> selectedFiles = new
ArrayList<WebOrHdFile>();

    private SelectionMouseListener selectionListener;

    private boolean showIcons = true;

    private TreeMarkMode markMode = TreeMarkMode.SINGLE_NODE;

    /**
     * Extensão para ser escondida.
     */
    private String extensionToHide;

    /**
     * @param allowedExtension
     *            Extensão utilizada para filtrar os arquivos
que serão visíveis
     * @param hideExtensions
```

```
       *           Se deve ocultar a extensão dos arquivos na
árvore
       * @param roots
       *           Diretórios cujo conteúdo será exibido na
árvore
       */
      public FileTree(String allowedExtension, boolean
hideExtensions, Resource resource) {
            super(new FileTreeNode(allowedExtension, resource,
true));
            format(hideExtensions);
      }

      /**
       * @param filter
       *           Filtro que será utilizada para selecionar
quais arquivos serão visíveis
       * @param hideExtensions
       *           Se deve ocultar a extensão dos arquivos na
árvore
       * @param roots
       *           Diretórios cujo conteúdo será exibido na
árvore
       */
      public FileTree(FileFilter filter, boolean hideExtensions,
String... roots) {
            super(new FileTreeNode(filter, roots, true));
            format(hideExtensions);
      }

      /**
       * @param hideExtensions
       *           Se deve ocultar a extensão dos arquivos na
árvore
       * @param root
       *           Diretórios cujo conteúdo será exibido na
árvore
       */
      public FileTree(boolean hideExtensions,
DefaultMutableTreeNode root) {
            super(new FileTreeNode(root));
            format(hideExtensions);
      }

      private void format(boolean hideExtensions) {
            // 1. Inicia algumas variáveis e configurações
            this.hideExtensions = hideExtensions;
            selectionListener = new SelectionMouseListener(this);
            setCellRenderer(new FileTreeCellRenderer());
            setRootVisible(false);

            // 2. Ordena seu conteúdo
            sort((FileTreeNode) getModel().getRoot());

            // 3. Atualiza a árvore
            ((DefaultTreeModel) getModel()).reload();
```

```
          // 4. Adiciona a abertura de arquivos e pastas com 2
cliques de acordo com a configuração da
          // FileTree
          addMouseListener(new MouseListener() {

                public void mouseReleased(MouseEvent e) {}

                public void mousePressed(MouseEvent e) {}

                public void mouseExited(MouseEvent e) {}

                public void mouseEntered(MouseEvent e) {}

                public void mouseClicked(MouseEvent e) {
                      if (e.getButton() == MouseEvent.BUTTON1 &&
e.getClickCount() == 2) {
                            TreePath path = getSelectionPath();
                            if (path != null) {
                                  WebOrHdFile file =
((FileTreeNode) path.getLastPathComponent()).getUserObject();

                                  if (file.isFile() && openFile
|| openDirectory) {
                                        if (file.isProfile() &&
AppConfig.PROFILE.canAccessProfile() || file.isAll()
                                                    &&
AppConfig.PROFILE.canAccessAllProfile() || !file.isServerSide())
{
                                              try {
                                                    if
(FileUtilities.endsWithExtension(file,
DataConfig.SCRIPT_EXTENSION)) {

      Desktop.getDesktop().edit(file);
                                                    }
                                                    else {

      Desktop.getDesktop().open(file);
                                                    }
                                              }
                                              catch (IOException
e1) {

      e1.printStackTrace();
                                              }
                                        }
                                  }
                            }
                      }
                }
          });
      }

      /**
       * Ordena a árvore colocando diretórios antes de arquivos.
```

```
	 *
	 * @param node
	 */
	private void sort(FileTreeNode node) {
		List<FileTreeNode> dirs = new
ArrayList<FileTreeNode>();
		for (int i = 0; i < node.getChildCount();) {
			FileTreeNode child = (FileTreeNode)
node.getChildAt(i);
			sort(child);
			if (child.getUserObject().isDirectory()) {
				dirs.add(child);
				node.remove(i);
			}
			else {
				i++;
			}
		}
		for (int i = 0; i < dirs.size(); i++) {
			node.insert(dirs.get(i), i);
		}
	}

	public void changeMark(FileTreeNode node) {
		setMarked(node, !node.isMarked());
	}

	public void setMarked(FileTreeNode node, boolean checked)
{
		node.setMarked(checked);
		if (node.isMarked()) {
			if (markMode == TreeMarkMode.SINGLE_NODE) {
				while (selectedFiles.size() > 0) {
					unmarkFile(selectedFiles.get(0));
				}
			}
			selectedFiles.add(node.getUserObject());
			if (node.isLeaf()) {
				node = node.getParent();
			}
			expandPath(new TreePath(node.getPath()));
		}
		else {
			selectedFiles.remove(node.getUserObject());
		}
	}

	public void markFile(File file) {
		FileTreeNode root = (FileTreeNode)
getModel().getRoot();
		_selectFile(file, root, true);
	}

	public void unmarkFile(File file) {
		FileTreeNode root = (FileTreeNode)
getModel().getRoot();
```

```
            _selectFile(file, root, false);
      }

      /**
       * Função auxiliar da selectFile(file).
       *
       * @param file
       * @param node
       * @param selected
       */
      private boolean _selectFile(File file, FileTreeNode node,
boolean selected) {
            if (node.getUserObject() != null
                        && (verifyContentsOnSelect &&
FileUtilities.compareFiles(node.getUserObject(), file) ||
!verifyContentsOnSelect
                                    &&
node.getUserObject().getName().equals(file.getName()))) {
                  setMarked(node, selected);
                  return true;
            }
            else {
                  for (int i = 0; i < node.getChildCount(); i++) {
                        if (_selectFile(file, (FileTreeNode)
node.getChildAt(i), selected)) {
                              break;
                        }
                  }
            }
            return false;
      }

      protected static Map<String, Icon> iconCache = new
HashMap<String, Icon>();

      private static Map<File, String> rootNameCache = new
HashMap<File, String>();

      private class FileTreeCellRenderer extends
DefaultTreeCellRenderer {

            private static final long serialVersionUID = 1L;

            @Override
            public Component getTreeCellRendererComponent(JTree
tree, Object value, boolean sel,
                        boolean expanded, boolean leaf, int row,
boolean hasFocus) {
                  FileTreeNode ftn = (FileTreeNode) value;
                  File file = ftn.getUserObject();
                  String fileName = "";
                  String extension = "";
                  if (file != null) {
                        if (ftn.isFileSystemRoot) {
                              fileName = rootNameCache.get(file);
                              if (fileName == null) {
```

```
                                    fileName =
fsv.getSystemDisplayName(file);
                                    rootNameCache.put(file,
fileName);
                              }
                        }
                        else {
                              fileName = file.getName();
                        }
                        int lastDot = fileName.lastIndexOf('.');
                        if (lastDot > 0) {
                              extension =
fileName.substring(lastDot + 1);
                        }
                        if (hideExtensions) {
                              if (extensionToHide != null) {
                                    fileName =
fileName.substring(0, fileName.length() -
extensionToHide.length());
                              }
                              else {
                                    if (lastDot > 0) {
                                          fileName =
fileName.substring(0, lastDot);
                                    }
                              }
                        }
                  }
                  JLabel tempLbl = (JLabel)
super.getTreeCellRendererComponent(tree, fileName, sel,
                              expanded, leaf, row, hasFocus);
                  WebOrHddLabel lbl = new WebOrHddLabel();
                  lbl.setText(tempLbl.getText());
                  lbl.setIcon(tempLbl.getIcon());

                  Component comp = null;
                  if (showIcons) {
                        comp = setIcon(lbl, ftn, extension);
                  }
                  else {
                        lbl.setIcon(null);
                  }
                  JPanel pnl = new JPanel();
                  pnl.setLayout(new BorderLayout());
                  if (fileMarking && file != null &&
file.isFile()) {
                        pnl.add(new JCheckBox("", ftn.isMarked()),
BorderLayout.WEST);
                  }
                  if (comp != null) {
                        pnl.add(comp);
                  }
                  else {
                        pnl.add(lbl);
                  }
                  pnl.setOpaque(false);
```

```
                if (selected) {
                      lbl.setForeground(Color.white);
                }
                return pnl;
          }

          public WebOrHddLabel setIcon(WebOrHddLabel label,
FileTreeNode node, String extension) {
                File file = node.getUserObject();
                if (file != null) {
                      Icon icon = iconCache.get(extension);
                      if (icon == null) {
                            if (file.exists()) {
                                  icon = fsv.getSystemIcon(file);
                            }
                            iconCache.put(extension, icon);
                      }
                      label.setIcon(icon);
                }
                return label;
          }
     }

     private class WebOrHDTreeCellRenderer extends
FileTreeCellRenderer {

          private static final long serialVersionUID = 1L;

          private final ImageIcon hdIcon =
GUIUtilities.loadIcon("hdd", ImageSize.EXTRA_SMALL);
          private final ImageIcon profileIcon =
GUIUtilities.loadIcon("profile", ImageSize.EXTRA_SMALL);
          private final ImageIcon webIcon =
GUIUtilities.loadIcon("web", ImageSize.EXTRA_SMALL);

          @Override
          public WebOrHddLabel setIcon(WebOrHddLabel label,
FileTreeNode node, String fileName) {
                super.setIcon(label, node, fileName);
                if (node.isServerSide()) {
                      if (node.isProfile()) {
                            label.setLocationIcon(profileIcon);
                      }
                      else {
                            label.setLocationIcon(webIcon);
                      }
                }
                else {
                      label.setLocationIcon(hdIcon);
                }
                return label;
          }
     }

     private class WebOrHddLabel extends JLabel {
          private static final long serialVersionUID = 1L;
```

```
            ImageIcon locationIcon;

            public void setLocationIcon(ImageIcon locationIcon) {
                  this.locationIcon = locationIcon;
            }

            @Override
            public void paint(Graphics g) {
                  super.paint(g);
                  if (locationIcon != null) {
                        locationIcon.paintIcon(this, g, 8, 4);
                  }
            }
      }

      public void createCellRenderer() {
            setCellRenderer(new WebOrHDTreeCellRenderer());
      }

      public void setFileMarking(boolean fileMarking) {
            this.fileMarking = fileMarking;
            if (fileMarking) {
                  addMouseListener(selectionListener);
            }
            else {
                  removeMouseListener(selectionListener);
            }
      }

      public void setShowIcons(boolean showIcons) {
            this.showIcons = showIcons;
      }

      public void setExtensionToHide(String extensionToHide) {
            this.extensionToHide = extensionToHide;
      }

      public void setVerifyContentsOnSelect(boolean
verifyContentsOnSelect) {
            this.verifyContentsOnSelect = verifyContentsOnSelect;
      }

      public void setOpenDirectory(boolean openDirectory) {
            this.openDirectory = openDirectory;
      }

      public void setOpenFile(boolean openFile) {
            this.openFile = openFile;
      }

      public TreeMarkMode getCheckMode() {
            return markMode;
      }

      public void setCheckMode(TreeMarkMode checkMode) {
            this.markMode = checkMode;
```

```
	}

	public WebOrHdFile getSelectedFile() {
		if (selectedFiles.size() > 0) {
			return selectedFiles.get(0);
		}
		else {
			return null;
		}
	}

	public List<WebOrHdFile> getSelectedFiles() {
		return selectedFiles;
	}

}
```

Anexo B – FileTreeNode.java

```
package express.lib.gui.filetree;

import java.io.File;
import java.io.FileFilter;

import javax.swing.tree.DefaultMutableTreeNode;

import express.config.DataConfig;
import express.config.Resource;
import express.lib.model.WebOrHdFile;

/**
 *
 * Nó utilizado numa {@link FileTree}.
 *
 * @author Gebara
 *
 */
public class FileTreeNode extends DefaultMutableTreeNode {
	private static final long serialVersionUID = 1L;

	/**
	 * Indication whether this node corresponds to a file
system root.
	 */
	boolean isFileSystemRoot;

	private boolean marked;

	private boolean serverSide;

	private boolean profile;

	public FileTreeNode(WebOrHdFile file, boolean server,
boolean profile) {
		this(null, file, server, profile);
	}

	public FileTreeNode(FileFilter filter, WebOrHdFile file,
boolean server, boolean profile) {
		super(file);
		setServerSide(server);
		setProfile(profile);
		if (file.isDirectory()) {
			// setParent(parent);
			File[] files = file.listFiles(filter);
			if (files != null) {
				for (File temp : files) {
					add(new FileTreeNode(new
WebOrHdFile(temp), server, profile));
				}
			}
```

```
            }
      }

      public FileTreeNode(DefaultMutableTreeNode node) {
            if (node.getUserObject() instanceof File) {
                  node.setUserObject(new WebOrHdFile((File)
node.getUserObject()));
            }
            setUserObject(node.getUserObject());
            for (int i = 0; i < node.getChildCount(); i++) {
                  DefaultMutableTreeNode child =
(DefaultMutableTreeNode) node.getChildAt(i);
                  add(new FileTreeNode(child));
            }
      }

      /**
       *
       * @param remove1stLevelChildren
       *            Indica se os filhos diretos da raiz devem ser
removidos. Exemplos de uso estão no
       *            sistema.
       */
      public FileTreeNode(final String allowedExtension,
Resource resource,
                  boolean remove1stLevelChildren) {
            this(new FileFilter() {
                  public boolean accept(File pathname) {
                        return pathname.isDirectory() ||
pathname.getName().endsWith(allowedExtension);
                  }
            }, resource, remove1stLevelChildren);
      }

      public FileTreeNode(FileFilter filter, Resource resource,
boolean remove1stLevelChildren) {
            boolean[] server = { true, true, false };
            boolean[] profile = { false, true, false };
            int i = 0;
            for (String child : DataConfig.getAllPaths(resource))
{
                  if (remove1stLevelChildren) {
                        File[] temp = new
File(child).listFiles(filter);
                        if (temp != null) {
                              for (File file : temp) {
                                    add(new FileTreeNode(filter,
new WebOrHdFile(file), server[i], profile[i]));
                              }
                        }
                  }
                  else {
                        add(new FileTreeNode(new
WebOrHdFile(child), server[i], profile[i]));
                  }
                  i++;
```

```
            }
      }

      public FileTreeNode(FileFilter filter, String[] children,
boolean remove1stLevelChildren) {
            int i = 0;
            for (String child : children) {
                  if (remove1stLevelChildren) {
                        File[] temp = new
File(child).listFiles(filter);
                        if (temp != null) {
                              for (File file : temp) {
                                    add(new FileTreeNode(filter,
new WebOrHdFile(file), false, false));
                              }
                        }
                  }
                  else {
                        add(new FileTreeNode(new
WebOrHdFile(child), false, false));
                  }
                  i++;
            }
      }

      void changeMark() {
            marked = !marked;
      }

      public boolean isMarked() {
            return marked;
      }

      public void setMarked(boolean marked) {
            this.marked = marked;
      }

      @Override
      public WebOrHdFile getUserObject() {
            return (WebOrHdFile) super.getUserObject();
      }

      @Override
      public FileTreeNode getParent() {
            return (FileTreeNode) super.getParent();
      }

      public void setServerSide(boolean serverSide) {
            this.serverSide = serverSide;
            getUserObject().setServerSide(serverSide);
      }

      public boolean isServerSide() {
            return serverSide;
      }
```

```
	public void setProfile(boolean profile) {
		this.profile = profile;
		getUserObject().setProfile(profile);
	}

	public boolean isProfile() {
		return profile;
	}

}
```

Anexo C – MavenController.java

```
package express.control.assistant.compiler;

import java.awt.Toolkit;
import java.util.ArrayList;
import java.util.List;

import javax.swing.JOptionPane;

import express.config.MavenExecutionStatus;
import express.config.SessionConfig;
import express.control.main.MainController;
import express.model.Instruction;
import express.model.MavenCommand;
import express.model.ParallelCommandGroup;
import express.model.SequentialCommandGroup;
import express.model.ShutDownJbossCommand;
import express.model.StartJbossCommand;

/**
 *
 * Chamado pelo {@link CompilerAssistantController} para
comandar a execução em paralelo da
 * {@link Instruction} passada. Fá-lo através de diversos {@link
MavenThread}
 *
 * @author Gebara
 *
 */
public class MavenController extends Thread {
	private List<MavenCommand> commands;

	private SequentialCommandGroup commandsGroup;

	private boolean success;

	private List<MavenThread> mavenThreads;

	private int mavenThreadsIndex;

	private boolean cancelled;

	private Instruction instruction;

	// private File logsDir;

	private boolean busy;

	public MavenController(Instruction instruction) {
		try {
			// Evita que duas execucoes sejam feitas no
mesmo segundo
```

```
                // gerando problema com a nomenclatura das
pastas de log
                Thread.sleep(1000);
           }
           catch (InterruptedException e) {}
           this.instruction = instruction;
           this.commands = instruction.getCommands();

           // Detecta quantos comandos existem para cada status
           MavenExecutionStatus currentStatus = null;
           for (MavenCommand command : commands) {
                MavenExecutionStatus status =
command.getStatus();
                if (status != currentStatus) {
                     currentStatus = status;
                     currentStatus.setCurrent(0);
                     currentStatus.setTotal(0);
                }
                currentStatus.setTotal(currentStatus.getTotal()
+ 1);
           }

           // Processa a lista de comandos, agrupando aqueles
que podem ser executados em paralelo
           List<MavenCommand> mdaCommands = new
ArrayList<MavenCommand>();
           MavenCommand commonCommand = null;
           MavenCommand cdCommand = null;
           MavenCommand compartilhadoCommand = null;
           List<MavenCommand> csCommands = new
ArrayList<MavenCommand>();
           List<MavenCommand> webCommands = new
ArrayList<MavenCommand>();
           for (MavenCommand command : commands) {
                switch (command.getStatus()) {
                     case MDA:
                          mdaCommands.add(command);
                          break;
                     case COMPILING:
                          if
(command.getFile().getName().equals("common")) {
                               commonCommand = command;
                          }
                          else if
(command.getFile().getName().equals("cd")) {
                               cdCommand = command;
                          }
                          else if
(command.getFile().getParentFile().getName().equals("web")) {
                               webCommands.add(command);
                          }
                          else if
(command.getFile().getName().equals("compartilhado")) {
                               compartilhadoCommand = command;
                          }
```

```
                              else if
(command.getFile().getParentFile().getName().equals("cs")) {
                                    csCommands.add(command);
                              }
                              break;
                        }
                  }

                  // Ordena as listas de comandos

      MainController.instance.orderCommandList(mdaCommands);
                  MainController.instance.orderCommandList(csCommands);

      MainController.instance.orderCommandList(webCommands);

                  // Monta o grupo de processamento paralelo
                  boolean commandJBossEnd = false;
                  commandsGroup = new SequentialCommandGroup();
                  if (mdaCommands.size() > 0) {
                        commandsGroup.add(new
ParallelCommandGroup(mdaCommands));
                  }
                  if (commonCommand != null) {
                        commandsGroup.add(new
ParallelCommandGroup(commonCommand));
                  }

                  if (cdCommand != null) {
                        ShutDownJbossCommand sdjc = new
ShutDownJbossCommand(instruction);
                        commands.add(commands.indexOf(cdCommand), sdjc);
                        commandsGroup.add(new
ParallelCommandGroup(sdjc));
                        commandsGroup.add(new
ParallelCommandGroup(cdCommand));
                        commandJBossEnd = true;
                  }
                  if (compartilhadoCommand != null) {
                        if (!commandJBossEnd) {
                              ShutDownJbossCommand sdjc = new
ShutDownJbossCommand(instruction);

      commands.add(commands.indexOf(compartilhadoCommand),
sdjc);
                              commandsGroup.add(new
ParallelCommandGroup(sdjc));
                              commandJBossEnd = true;
                        }
                        commandsGroup.add(new
ParallelCommandGroup(compartilhadoCommand));
                  }
                  if (csCommands.size() > 0) {
                        if (!commandJBossEnd) {
                              ShutDownJbossCommand sdjc = new
ShutDownJbossCommand(instruction);
```

```
      commands.add(commands.indexOf(csCommands.get(0)), sdjc);
                        commandsGroup.add(new
ParallelCommandGroup(sdjc));
                        commandJBossEnd = true;
                  }
                  commandsGroup.add(new
ParallelCommandGroup(csCommands));
            }
            if (instruction.isStartServerSelected()) {
                  StartJbossCommand sjc = new
StartJbossCommand(instruction);
                  // se só tiver comandos MDA, adiciona o comando
de iniciar servidor no início
                  if (commandsGroup.size() == 1 &&
mdaCommands.size() > 0) {
                        commands.add(0, sjc);
                        commandsGroup.add(0, new
ParallelCommandGroup(sjc));
                  }
                  else {
                        commands.add(commands.size() -
webCommands.size(), sjc);
                        commandsGroup.add(new
ParallelCommandGroup(sjc));
                  }
            }
            if (webCommands.size() > 0) {
                  commandsGroup.add(new
ParallelCommandGroup(webCommands));
            }

            // Cria o diretorio destinado para os logs
            // logsDir = new File(DataConfig.LOGS_DIR
            // + DateFormat.getDateTimeInstance().format(new
Date()).replace('/', '-').replace(':',
            // '-') + File.separator);
            // if (!logsDir.exists()) {
            // logsDir.mkdirs();
            // }
      }

      @Override
      public void run() {
            success = true;

            try {
                  if (commands.size() == 0) {
                        finished();
                        return;
                  }

                  // Inicia as threads
                  mavenThreads = new ArrayList<MavenThread>();
                  populateMavenThreads();
                  mavenThreadsIndex = 0;
```

```
            }
            catch (Exception e) {
                  e.printStackTrace();
            }

            return;
      }

      private void populateMavenThreads() {
            int threads =
Math.min(SessionConfig.instance.maximumParallelGroups,
commandsGroup.get(0)
                        .size());
            while (mavenThreads.size() < threads) {
                  MavenThread thread = new MavenThread(this,
mavenThreadsIndex++);
                  thread.start();
                  mavenThreads.add(thread);
                  try {
                        // Evita alguns problemas de sincronismo
                        Thread.sleep(100);
                  }
                  catch (InterruptedException e) {}
            }
      }

      public MavenCommand getNextCommand(MavenThread
mavenThread) {
            while (busy) {

            }
            ;
            busy = true;

            MavenCommand result = null;
            try {
                  result = _getNextCommand(mavenThread);
            }
            catch (Exception e) {
                  e.printStackTrace();
            }
            finally {
                  busy = false;
            }

            return result;
      }

      public MavenCommand _getNextCommand(MavenThread
mavenThread) throws Exception {
            // Verifica o sucesso do ultimo comando
            if (!mavenThread.getSuccess()) {
                  success = false;
                  int notInterrupted = 0;
                  for (MavenThread thread : mavenThreads) {
                        if (thread.isAlive()) {
```

```
                              notInterrupted++;
                        }
                  }
                  // Se nao for o ultimo comando, exibe uma
pergunta sobre a continuidade do processo
                  if (notInterrupted > 1
                              || (commandsGroup.size() > 0 &&
!(commandsGroup.get(0).size() == 0 && commandsGroup
                                          .size() == 1))) {
                        getUserAttention(1);

      JOptionPane.showMessageDialog(MainController.instance.getM
ainFrame(),
                                    "Ocorreu um erro.", "Erro",
JOptionPane.ERROR_MESSAGE);
                  }
            }
            if (cancelled) {
                  return null;
            }

            // Busca o proximo comando
            ParallelCommandGroup parGroup;
            try {
                  parGroup = commandsGroup.get(0);
            }
            catch (IndexOutOfBoundsException e) {
                  return null;
            }
            // Se todas as instrucoes do grupo em paralelo atual
ja tiverem sido dadas, cancela as threads
            // atuais e comeca a executar o proximo quando todas
estiverem canceladas, indicando que todas
            // terminaram
            if (parGroup.size() == 0) {
                  mavenThread.interrupt();
                  mavenThreads.remove(mavenThread);
                  boolean allInterrupted = true;
                  // Isso foi feito pois, misteriosamente, algumas
threads nao eram removidas da lista e
                  // continuavam nela mesmo depois de
interrompidas
                  for (MavenThread thread : mavenThreads) {
                        if (thread.isAlive()) {
                              allInterrupted = false;
                              break;
                        }
                  }
                  if (allInterrupted) {// o grupo paralelo acabou
suas execucoes
                        if (commandsGroup.size() > 0) {
                              commandsGroup.remove(0);
                        }
                        if (commandsGroup.size() > 0) {// se ainda
houver instrucoes
                              parGroup = commandsGroup.get(0);
```

```
                              populateMavenThreads();
                        }
                        else {
                              // Termina se todas as instrucoes
tiverem sido executadas
                              finished();
                        }
                  }
                  return null;
            }

            MavenCommand command = parGroup.get(0);
            parGroup.remove(0);

      CompilerAssistantController.instance.setStatus(command.get
Status());

            return command;
      }

      public void cancel() {
            if (busy) {

      JOptionPane.showMessageDialog(MainController.instance.getM
ainFrame(),
                              "Tente novamente, por favor.");
                  return;
            }


      CompilerAssistantController.instance.setStopped(true);

            for (MavenThread thread : mavenThreads) {
                  thread.interrupt();
            }
            cancelled = true;
            finished();
      }

      private void finished() {
            instruction.setEndTime();
            if (!success) {

      CompilerAssistantController.instance.setStatus(MavenExecut
ionStatus.ERROR);
            }
            else if (cancelled) {

      CompilerAssistantController.instance.setStatus(MavenExecut
ionStatus.STOPPED);
            }
            else {
                  MavenExecutionStatus.SUCCESS.setTotal(1);
                  MavenExecutionStatus.SUCCESS.setCurrent(1);
```

```
      CompilerAssistantController.instance.setStatus(MavenExecut
ionStatus.SUCCESS);
            }

            if (!cancelled) {
                  getUserAttention(3);
                  this.interrupt();
            }
      }

      private void getUserAttention(int beeps) {
            MainController.instance.bringToFront();
            try {
                  for (int i = 0; i < beeps; i++) {
                        Toolkit.getDefaultToolkit().beep();
                        if (i != beeps - 1) {
                              Thread.sleep(300);
                        }
                  }
            }
            catch (InterruptedException e1) {}
      }

      public Instruction getInstruction() {
            return instruction;
      }

      public void setStatus(MavenExecutionStatus status) {
            instruction.setStatus(status);
      }
}
```

Anexo D – MavenThread.java

```
package express.control.assistant.compiler;

import java.awt.Desktop;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;

import express.config.CommandState;
import express.config.DataConfig;
import express.lib.util.BatThread;
import express.model.Instruction;
import express.model.MavenCommand;
import express.model.ShutDownJbossCommand;
import express.model.StartJbossCommand;

/**
 *
 * Controla a execução de uma {@link Instruction} a partir do
{@link CompilerAssistantController}.
 * Também processa o conteúdo dos buffers de saída e erro e
repassa ao
 * {@link CompilerAssistantController} para exibição ao usuário.
 * <p>
 * São chamados diretamente pelo {@link MavenController}, que
executa a {@link Instruction} em
 * paralelo através de diversas instâncias da classe atual.
 *
 * @author Gebara
 * @see MavenCommand
 *
 */
public class MavenThread extends Thread {

      private static final long SLEEP_TIME = 1024;
      private static final int BUFFER_SIZE = 1024;
      private MavenController mavenController;
      private int index;

      // private File log;

      private boolean success = true;

      private InputStreamReader es;
      private InputStreamReader is;
      private Process child;
      private List<String> ids;

      public MavenThread(MavenController mavenController, int
index) {
```

```
            this.mavenController = mavenController;
            this.index = index;
      }

      @Override
      public void run() {
            boolean alreadyAlerted = false;
            String batName =
DataConfig.MAVEN_TEMP_FILE_WITHOUT_EXTENSION + index
                        + DataConfig.SCRIPT_EXTENSION;
            File temp = new File(batName);

            ids = new ArrayList<String>();

            MavenCommand command;
            // Pede um comando ao controle
            while ((command =
mavenController.getNextCommand(this)) != null) {
                  success = true;
                  try {
                        if
(CompilerAssistantController.instance.isStopped()) {
                              break;
                        }
                        else {

      CompilerAssistantController.instance.setStatus(command.get
Status());
                        }

                        command.setState(CommandState.EXECUTING);

                        if (command instanceof StartJbossCommand)
{
                              try {
                                    Desktop.getDesktop().open(new
File(DataConfig.VBS_ECLIPSE_RUN));
                              }
                              catch (IOException e) {
                                    e.printStackTrace();
                              }
                        }
                        else {

                              // Escreve o arquivo de script
                              FileWriter fw = new FileWriter(temp);
                              for (String scriptLine :
command.getScript()) {
                                    // if
(scriptLine.contains("maven")) {
                                    // log = new
File(mavenController.getLogsDir().getAbsolutePath() +
File.separator
                                    // +
command.getName().replace('\\', '-') + ".txt");
                                    // }
```

```
                                    // if
(scriptLine.contains("shutdown")) {
                                    // log = new
File(mavenController.getLogsDir().getAbsolutePath() +
File.separator
                                    // + "shutdown.txt");
                                    // }

      fw.append(scriptLine).append(DataConfig.LINE_END);
                               }
                               fw.close();

                               // Executa o arquivo de script
                               if (command instanceof
ShutDownJbossCommand) {
                                    while (!(success =
runCommand(command, temp.getAbsolutePath()))) {
                                         String msg = "JBoss
iniciando... Tentando novamente em 3 segundos.";

      command.addOutputLine(msg);

      command.addExtendedOutput("<br><br>" + msg + "<br>");

      LogController.instance.refresh(command);
                                         try {
                                              Thread.sleep(3000);
                                         }
                                         catch
(InterruptedException e) {
                                              e.printStackTrace();
                                         }
                                    }
                                    while (!((ShutDownJbossCommand)
command).isFinished()) {
                                         if (!alreadyAlerted) {
                                              String msg =
"Aguardando o término do Jboss...";

      command.addOutputLine(msg);

      command.addExtendedOutput("<br><br>" + msg + "<br>");
                                              alreadyAlerted =
true;
                                         }

      LogController.instance.refresh(command);
                                         try {
                                              Thread.sleep(3000);
                                         }
                                         catch
(InterruptedException e) {
                                              e.printStackTrace();
                                         }
                                    }
                               }
```

```
                              else {
                                    success = runCommand(command,
temp.getAbsolutePath());
                              }

                              if
(CompilerAssistantController.instance.isStopped()) {
                                    for (String id : ids) {
                                          BatThread bat = new
BatThread(new File(DataConfig.BAT_KILL_TASK),
                                                      new String[] {
id });
                                          bat.run();
                                    }
                              }
                              if (child != null) {
                                    child.destroy();
                              }
                              if (es != null) {
                                    es.close();
                              }
                              if (is != null) {
                                    is.close();
                              }
                        }

                        // Verifica o resultado e atualiza seus
status de acordo
                        if
(CompilerAssistantController.instance.isStopped()) {
                              success = true;

      command.setState(CommandState.STOPPED);
                        }
                        else if (success) {

      command.setState(CommandState.SUCCESS);
                        }
                        else {
                              command.setState(CommandState.ERROR);
                        }
                  }
                  catch (IOException e) {
                        e.printStackTrace();
                  }
            }
      }

      /**
       *
       * @param mavenCommand
       * @param script
       * @return Verdadeiro se tiver sido interrompido
       */
      private boolean runCommand(MavenCommand mavenCommand,
String command) throws IOException {
```

```
            Runtime rt = Runtime.getRuntime();
            String[] callAndArgs = { command };

            List<String> preIDs = new ArrayList<String>();
            // Verifica quais as IDs de processo java ativas
antes de iniciar
            BatThread bat = new BatThread(new
File(DataConfig.BAT_LIST_TASKS), null);
            bat.run();
            String[] taskLines = bat.getOutput().split("\n");
            for (int i = 5; i < taskLines.length; i++) {
                  preIDs.add(taskLines[i].split("[\\s]+")[1]);
            }

            // Executa o comando
            child = rt.exec(callAndArgs);
            es = new InputStreamReader(child.getErrorStream());
            is = new InputStreamReader(child.getInputStream());
            char[] buf = new char[BUFFER_SIZE];
            int read;
            StringBuffer errorBuffer = new StringBuffer();

            // Verifica quais as IDs de processo java ativas
depois de iniciar
            bat = new BatThread(new
File(DataConfig.BAT_LIST_TASKS), null);
            bat.run();
            taskLines = bat.getOutput().split("\n");
            for (int i = 5; i < taskLines.length; i++) {
                  ids.add(taskLines[i].split("[\\s]+")[1]);
            }
            ids.removeAll(preIDs);

            while (true) {
                  if
(CompilerAssistantController.instance.isStopped()) {
                        return true;
                  }
                  // Aguarda para tentar a leitura
                  try {
                        Thread.sleep(SLEEP_TIME);
                  }
                  catch (InterruptedException e1) {
                        return true;
                  }

                  while (is.ready() && (read = is.read(buf)) > 0)
{
                        if
(CompilerAssistantController.instance.isStopped()) {
                              return true;
                        }
                        String s = new String(buf, 0, read);
                        mavenCommand.addExtendedOutput(s);
```

```
                        if ((mavenCommand instanceof
ShutDownJbossCommand)
                                    &&
parseLogForShutDownSuccess(mavenCommand.getExtendedOutput())) {
                              return true;
                        }
                  }

                  // Executa a leitura assim que (e se) a stream
estiver pronta
                  while (es.ready() && (read = es.read(buf)) > 0)
{
                        if
(CompilerAssistantController.instance.isStopped()) {
                              return true;
                        }
                        String s = new String(buf, 0, read);
                        errorBuffer.append(s);
                        mavenCommand.addExtendedOutput(s);

                        // Marcador de erro para quando o Jboss
estiver iniciando e quando não tiver
                        // inicializado e se tentar fechá-lo
                        if (s.contains("Exception in thread
\"main\" javax.management.RuntimeMBeanException")) {
                              return false;
                        }
                        else if (s.contains("Exception in thread
\"main\"")
                                    || s

      .contains("javax.naming.CommunicationException: Could not
obtain connection to any of these urls:")) {
                              mavenCommand.addOutputLine("O Jboss
não estava inicializado");
                              return true;
                        }

                        // Marcado de erro para problemas no
modelo
                        if
(s.contains("org.andromda.core.ModelValidationException:")) {
                              List<String> temp =
parseLogForModelErrors(mavenCommand);
                              for (String stringTemp : temp) {

      mavenCommand.addOutputLine(stringTemp);
                              }
                              return false;
                        }
                        // Marcador de erro para compilações de
código
                        if (s.contains("BUILD FAILED")) {
                              String[] errors =
errorBuffer.toString().split("\\^\\r\\n");
                              int indice = 1;
```

```
                              for (String error : errors) {
                                    if (error.contains("BUILD
FAILED")) {
                                          break;
                                    }
                                    error = error.replaceAll("\\>",
"");
                                    error = error.replaceAll("<",
"");
                                    int doisPontos0 =
error.indexOf(':', 4);
                                    int doisPontos1 =
error.indexOf(':', doisPontos0 + 1);
                                    int ultimaBarraEndereco =
error.substring(0, doisPontos0).lastIndexOf(
                                                File.separator);
                                    int ultimoPontoEndereco =
error.substring(0, doisPontos0).lastIndexOf('.');

      mavenCommand.addOutputLine(String.format("%d) <a
href=\"%s\">%s:%s</a><br>%s",
                                                indice++,
error.substring(0, doisPontos0), error.substring(

      ultimaBarraEndereco + 1, ultimoPontoEndereco),
error.substring(

      doisPontos0 + 1, doisPontos1), error.substring(doisPontos1
+ 1)));
                              }
                              // Repetiremos a execução se o
problema for cópia
                              // de um recurso utilizado por outro
comando
                              if
(errorBuffer.toString().contains("Failed to copy")
                                                ||
errorBuffer.toString().contains("Unable to delete file")) {
                                    mavenCommand
                                                .addOutputLine("Há
recursos ocupados. Tentando de novo em 3 segundos...");
                                    try {
                                          Thread.sleep(3000);
                                    }
                                    catch (InterruptedException e)
{}

      mavenCommand.addExtendedOutput("<br><br>");
                                    return runCommand(mavenCommand,
command);
                              }
                              return false;
                        }
                  }
                  try {
                        // Checa se já terminou
```

```
                        child.exitValue();
                        // Se sim, termina o loop
                        return true;
                  }
                  catch (IllegalThreadStateException e) {
                        // Indica que a thread ainda está rodando,
logo, continua
                  }
            }
      }

      private List<String> parseLogForModelErrors(MavenCommand
command) {
            // if (log == null) {
            // return null;
            // }
            List<String> lines =
command.getExtendedOutputLines();
            List<String> errorLines = new ArrayList<String>();
            boolean first = true;
            for (String line : lines) {
                  String[] l = line.split("<br>");
                  for (String string : l) {
                        if (string.startsWith("ERROR [AndroMDA]"))
{
                              if (first) {
                                    first = false;
                              }
                              else {

      errorLines.add(string.substring("ERROR
[AndroMDA]".length()).replace("[Data::",
                                                "").replace("]:",
"<br>").trim());
                              }
                        }
                  }
            }
            return errorLines;
      }

      private boolean parseLogForShutDownSuccess(String log) {
            if (log == null) {
                  return false;
            }
            return log.contains("Shutdown message has been posted
to the server.");
      }

      public boolean getSuccess() {
            return success;
      }
}
```

Anexo E – SpecialFrame.java

```
package express.lib.gui;

import java.awt.Cursor;
import java.awt.Dimension;
import java.awt.Frame;
import java.awt.Insets;
import java.awt.Rectangle;
import java.awt.event.MouseEvent;

import javax.swing.JFrame;
import javax.swing.event.MouseInputAdapter;

import com.sun.awt.AWTUtilities;
import com.sun.awt.AWTUtilities.Translucency;

import express.control.main.MainController;
import express.gui.main.MainFrame;

/**
 *
 * Frame que contém algumas das funções do {@link MainFrame}.
 *
 * @author Gebara
 *
 */
public class SpecialFrame extends JFrame {
	private static final long serialVersionUID = 1L;

	private MyMouseInputListener mouseInputListener = new
MyMouseInputListener();

	private Insets resizableArea = new Insets(5, 5, 5, 5);

	public SpecialFrame() {
		setUndecorated(true);
		if
(AWTUtilities.isTranslucencySupported(Translucency.PERPIXEL_TRAN
SPARENT)
				&&
AWTUtilities.isTranslucencySupported(Translucency.PERPIXEL_TRANS
LUCENT)
				&&
AWTUtilities.isTranslucencySupported(Translucency.TRANSLUCENT))
{
			AWTUtilities.setWindowOpaque(this, false);
		}
		getRootPane().addMouseListener(mouseInputListener);

	getRootPane().addMouseMotionListener(mouseInputListener);
	}

	private static final int NORTH = 1;
```

```
	private static final int SOUTH = 2;
	private static final int WEST = 3;
	private static final int EAST = 6;

	private static Resizer[] resizers = new Resizer[] { new
MoveResizer(), new NorthResizer(),
			new SouthResizer(), new WestResizer(), new
NorthWestResizer(), new SouthWestResizer(),
			new EastResizer(), new NorthEastResizer(), new
SouthEastResizer() };

	protected Resizer getResizer(int x, int y) {
		int width = getRootPane().getWidth();
		int height = getRootPane().getHeight();
		Insets insets = resizableArea;

		int index = 0;
		if (y <= insets.top * 2) {
			index += NORTH;
		}
		if (y >= height - insets.bottom * 2) {
			index += SOUTH;
		}
		if (x <= insets.left * 2) {
			index += WEST;
		}
		if (x >= width - insets.right * 2) {
			index += EAST;
		}

		if (index >= resizers.length) {
			index = 0;
		}
		return resizers[index];
	}

	private class MyMouseInputListener extends
MouseInputAdapter {

		private Resizer resizer = null;
		private int x;
		private int y;

		@Override
		public void mouseEntered(MouseEvent ev) {
			if (!isResizable()) {
				return;
			}
			getRootPane().setCursor(getResizer(ev.getX(),
ev.getY()).getCursor());
		}

		@Override
		public void mouseMoved(MouseEvent ev) {
			if (!isResizable()) {
				return;
```

```
                }
                getRootPane().setCursor(getResizer(ev.getX(),
ev.getY()).getCursor());
            }

            @Override
            public void mousePressed(MouseEvent ev) {
                if (!isResizable()) {
                    return;
                }
                x = ev.getX();
                y = ev.getY();

                resizer = getResizer(x, y);
            }

            @Override
            public void mouseDragged(MouseEvent ev) {
                if (!isResizable()) {
                    return;
                }
                if (resizer == null) {
                    return;
                }
                else {

                    int deltaX = ev.getX() - x;
                    int deltaY = ev.getY() - y;

                    int oldX = getX();
                    int oldY = getY();

                    resizer.resize(SpecialFrame.this, deltaX,
deltaY);

                    x = ev.getX() - (getX() - oldX);
                    y = ev.getY() - (getY() - oldY);
                }
            }

            @Override
            public void mouseReleased(MouseEvent ev) {
                if (!isResizable()) {
                    return;
                }
                resizer = null;
            }

            @Override
            public void mouseExited(MouseEvent ev) {
                if (!isResizable()) {
                    return;
                }
                if (resizer == null) {

      getRootPane().setCursor(Cursor.getDefaultCursor());
```

```
                }
            }
        }

        private static class Resizer {
            public Cursor getCursor() {
                return Cursor.getDefaultCursor();
            }

            public void resize(JFrame frame, int deltaX, int
deltaY) {}
        }

        private static class MoveResizer extends Resizer {
            @Override
            public Cursor getCursor() {
                // return Cursor.getDefaultCursor();
                return
Cursor.getPredefinedCursor(Cursor.MOVE_CURSOR);
            }

            @Override
            public void resize(JFrame frame, int deltaX, int
deltaY) {
                frame.setLocation(frame.getX() + deltaX,
frame.getY() + deltaY);
            }
        }

        private static class NorthResizer extends Resizer {
            @Override
            public Cursor getCursor() {
                return
Cursor.getPredefinedCursor(Cursor.S_RESIZE_CURSOR);
            }

            @Override
            public void resize(JFrame frame, int deltaX, int
deltaY) {
                frame.setBounds(frame.getX(), frame.getY() +
deltaY, frame.getWidth(), frame.getHeight()
                            - deltaY);
            }
        }

        private static class SouthResizer extends Resizer {
            @Override
            public Cursor getCursor() {
                return
Cursor.getPredefinedCursor(Cursor.S_RESIZE_CURSOR);
            }

            @Override
            public void resize(JFrame frame, int deltaX, int
deltaY) {
```

```
                frame.setBounds(frame.getX(), frame.getY(),
frame.getWidth(), frame.getHeight() + deltaY);
            }
      }

      private static class WestResizer extends Resizer {
            @Override
            public Cursor getCursor() {
                  return
Cursor.getPredefinedCursor(Cursor.W_RESIZE_CURSOR);
            }

            @Override
            public void resize(JFrame frame, int deltaX, int
deltaY) {
                  frame.setBounds(frame.getX() + deltaX,
frame.getY(), frame.getWidth() - deltaX, frame
                              .getHeight());
            }
      }

      private static class NorthWestResizer extends Resizer {
            @Override
            public Cursor getCursor() {
                  return
Cursor.getPredefinedCursor(Cursor.NW_RESIZE_CURSOR);
            }

            @Override
            public void resize(JFrame frame, int deltaX, int
deltaY) {
                  frame.setBounds(frame.getX() + deltaX,
frame.getY() + deltaY, frame.getWidth() - deltaX,
                              frame.getHeight() - deltaY);
            }
      }

      private static class SouthWestResizer extends Resizer {
            @Override
            public Cursor getCursor() {
                  return
Cursor.getPredefinedCursor(Cursor.SW_RESIZE_CURSOR);
            }

            @Override
            public void resize(JFrame frame, int deltaX, int
deltaY) {
                  frame.setBounds(frame.getX() + deltaX,
frame.getY(), frame.getWidth() - deltaX, frame
                              .getHeight()
                              + deltaY);
            }
      }

      private static class EastResizer extends Resizer {
            @Override
```

```
            public Cursor getCursor() {
                  return
Cursor.getPredefinedCursor(Cursor.E_RESIZE_CURSOR);
            }

            @Override
            public void resize(JFrame frame, int deltaX, int
deltaY) {
                  frame.setBounds(frame.getX(), frame.getY(),
frame.getWidth() + deltaX, frame.getHeight());
            }
      }

      private static class NorthEastResizer extends Resizer {
            @Override
            public Cursor getCursor() {
                  return
Cursor.getPredefinedCursor(Cursor.NE_RESIZE_CURSOR);
            }

            @Override
            public void resize(JFrame frame, int deltaX, int
deltaY) {
                  frame.setBounds(frame.getX(), frame.getY() +
deltaY, frame.getWidth() + deltaX, frame
                              .getHeight()
                              - deltaY);
            }
      }

      private static class SouthEastResizer extends Resizer {
            @Override
            public Cursor getCursor() {
                  return
Cursor.getPredefinedCursor(Cursor.SE_RESIZE_CURSOR);
            }

            @Override
            public void resize(JFrame frame, int deltaX, int
deltaY) {
                  frame.setBounds(frame.getX(), frame.getY(),
frame.getWidth() + deltaX, frame.getHeight()
                              + deltaY);
            }
      }

      private int x = 0;
      private int y = 0;
      private int w = 500;
      private int h = 500;

      public void maximize() {
            Rectangle r = getBounds();
            x = r.x;
            y = r.y;
            w = r.width;
```

```
            h = r.height;
            setExtendedState(Frame.MAXIMIZED_BOTH);

      MainController.instance.getMainFrame().setResizable(false)
;
            Dimension size =
java.awt.Toolkit.getDefaultToolkit().getScreenSize();
            MainController.instance.getMainFrame().setBounds(0,
0, size.width, size.height);
      }

      public void restore() {

      MainController.instance.getMainFrame().setExtendedState(Fr
ame.NORMAL);

      MainController.instance.getMainFrame().setResizable(true);
            MainController.instance.getMainFrame().setBounds(x,
y, w, h);
      }

}
```