Engenharia de Software : Procedimentos de Empacotamento

This page last changed on Apr 21, 2009 by amadeu.

Esse documento esclarece a metodologia adotada para compilar e instalar o OpenBus (barramento de E&P). Essa metodologia é diferente daquela adotada nos projetos baseados no *framework* CSBase. Na seção requisitos é possível identificar as demandas identificadas que motivaram a definição dessa metodologia própria.

Requisitos

amadeu: lembrar de copiar das minhas anotações do sistema de pacotes e do jira

Ambientes

Nessa seção entende-se por **ambiente** um conjunto de diretórios e variáveis de ambiente que organizam o ciclo de desenvolvimento e liberação de versões. Isolamos em ambientes específicos cada atividade:

1. **Desenvolvimento**
2. **Compilação e testes**
3. **Validação de um pacote para ser entregue ao cliente**

Ao final é apresentada uma visão geral da organização de diretórios motivada pela existência desses **ambientes**.

Ambiente de desenvolvimento

Entende-se por **ambiente de desenvolvimento** o local onde o desenvolvedor mantém os códigos-fonte, realiza modificações, utiliza as funcionalidades do repositório subversion (adiciona, remove, renomeia, move), entre outras. Nesse local **só** deve ser mantido os artefatos de software importantes a estarem no repositório subversion. Por exemplo, produtos das compilações (binários, bibliotecas) **não** devem poluir esse ambiente.

Com isso pretendemos **motivar** o desenvolvedor a utilizar mais os assistentes de compilação e geração de pacotes. Espera-se evitar que o desenvolvedor adote estratégias manuais e propensas a erros, no que tange o empacotamento. Assim será diminuída a chance de erros nas etapas seguintes de testes e validação.

Diretórios padrões:

- **openbus/** — raíz do ambiente
  - **lib/** — código das bibliotecas externas
  - **trunk/** — código do barramento
  - **OB_v1_10_00_2009_03_03/** — código do barramento em versão específica
  - **OB_v1_30_00_2009_03_10/** — código do barramento em versão específica

Ambiente de compilação e testes

Entende-se por **ambiente de compilação e testes** o local que **contém os produtos da compilação** alocados pelo uso do assistente de compilação. Essa separação em um diretório **exclusivo** para os alvos da compilação (ou outros artefatos a serem distribuídos) é um importante passo no empacotamento.

Ao desenvolver um pacote **X**, o programador **deve** criar uma **descrição** para esse pacote. Essa descrição contém tanto as **instruções de compilação** (qual config.mak usar no caso do tecmake, ou quais parâmetros do ./configure precisam existir) quanto a **seleção de arquivos e configurações** que, de fato, constarão no pacote final a ser entregue.

Assim o assistente de compilação procede à compilação e ao final copia os arquivos selecionados para o local do ambiente de compilação e testes. Além disso, o assistente aproveita para armazenar metadados sobre a constituição do pacote que será fundamental nas etapas de geração do pacote final e instalação.

Em particular, os testes mais cotidianos (suíte de testes unitários e demos) podem ser executados desse ambiente de compilação, uma vez que nesse diretório existirão todos os artefatos de software necessários à execução. Contudo é importante observar que **todos** os pacotes que forem compilados terão seus binários, bibliotecas e configurações copiados para esse ambiente. Assim é possível que alguns **problemas não sejam notados** pela co-existência de certos binários e bibliotecas.

Por isso antes de entregar um pacote ao cliente é fundamental validá-lo no ambiente descrito a seguir.

Diretório padrão:

- **openbus/** — raíz do ambiente
  - **install/** — instalação temporária para compilação e testes

Ambiente de validação dos pacotes

Entende-se por **ambiente de validação** o local onde seja possível **instalar** (usando o assistente de instalação) um pacote previamente gerado pelos assistentes. Esse ambiente deve ser o mais próximo possível da configuração do cenário para qual o pacote será entregue. Em detalhe é importante que não haja variáveis de ambientes previamente configuradas ou reuso de diretórios usados para compilação. Espera-se com isso isolar possíveis problemas que só ocorreriam numa instalação sob uma máquina virgem.

Recomenda-se a utilização de um usuário de sistema diferenciado daquele usado pelo desenvolvedor para evitar que o ambiente de desenvolvimento não esteja misturado a esse de validação.

Diretório padrão:

- **openbus/** — raíz do ambiente
  - **testing/** — nova pasta de destino para o instalador

Visão geral da organização de diretórios

Considerando todos os ambientes descritos acima alocados num mesmo diretório principal teremos a seguinte visão da organização de diretórios:

- **openbus/** — raíz
  - **lib/** — código das bibliotecas externas
  - **trunk/** — código do barramento
  - **OB_v1_01_00_2009_03_03/** — código do barramento em versão específica
  - **OB_v1_03_00_2009_03_10/** — código do barramento em versão específica
  - **install/** — instalação temporária para compilação e testes
  - **install-OB_v1_01/** — instalação temporária em versão específica para compilação e testes
  - **install-OB_v1_03/** — instalação temporária em versão específica para compilação e testes
  - **testing/** — nova pasta de destino para o instalador
  - **packs/** — local onde são armazenados os pacotes obtidos do repositório e os novos pacotes gerados
    - **metadata/** — metadados para a ferramenta de geração de pacotes

Árvore básica em uma instalação do OpenBus

Em cada árvore de instalação do OpenBus[1], a seguinte organização de diretórios reflete o conteúdo dos pacotes entregue ao cliente. Esses diretórios são o conteúdo dos ambientes de compilação e validação. Abaixo lista-se a constituição de cada subdiretório:

- **/** — raiz do middleware OpenBus para onde deve-se apontar a variável de ambiente OPENBUS_HOME
- **/bin** — diretório com binários externos aos serviços básicos do OpenBus, mas que devem ser redistribuídos
- **/core/** — diretório com a implementação do OpenBus e seus utilitários
  - **bin/** — contém scripts para lançamento dos binários multi-plataforma
    - **$TEC_UNAME/** — binários multi-plataforma (atualmente apenas o servicelauncher)
  - **services/** — implementação dos serviços básicos
    - **accesscontrol/** — controle de acesso
    - **registry/** — registro
    - **session/** — sessão
  - **utilities/** — utilitários para cada linguagem
    - **lua/** — fundamentais para execução dos serviços básicos
    - **cppoil/** — diretório da biblioteca de desenvolvimento em C++ com uso do OiL
    - **orbix/** — diretório da biblioteca de desenvolvimento em C++ com uso do Orbix
    - **mico/** — diretório da biblioteca de desenvolvimento em C++ com uso do MICO
    - **java/** — diretório da biblioteca de desenvolvimento em Java com uso do JacORB
- **/data/** — diretório com dados e configurações essenciais ao funcionamento dos serviços básicos
  - **conf/** — arquivos de configuração dos serviços básicos
    - **config** — arquivo com definição das variáveis de ambiente relativas à OPENBUS_HOME
  - **certificates/** — certificados e chaves privadas que identificam os serviços básicos
  - **offers/** — contém as ofertas de serviços publicados no barramento (*diretório gerado durante execução*)
  - **credentials/** — credenciais salvas pelo serviço de acesso (*diretório gerado durante execução*)
- **/idlpath/** — diretório com IDL CORBA
- **/libpath/** — diretório com bibliotecas multi-plataforma, sub-diretórios seguem **TEC_UNAME** de cada plataforma
- **/incpath/** — diretório com arquivos de cabeçalho para todas as bibliotecas dependentes
  - **openssl-0.9.9/** — dentro do **incpath** temos sub-diretórios separados para cada biblioteca que dependemos

Outros diretórios importantes para os softwares dos quais o OpenBus depende (como OpenSSL) podem ser adicionados diretamente na raiz. Um exemplo é o diretório **/openssl** que contém arquivos de configuração para a execução do binário do openssl.

[1] *Nesse exemplo consideramos todo o middleware instalado incluindo suas dependências, configurações, dados de logs, certificados, arquivos de cabeçalho etc. É possível que um determinado pacote entregue ao cliente só contenha parte dessa árvore de diretório.*

Assistentes automatizados

No repositório subversion do projeto OpenBus temos um conjunto de assistentes (scripts lua) que auxiliam a compilação, geração dos pacote e instalação. Os códigos-fonte desses assistentes estão localizados na pasta **trunk/tools** no repositório subversion e são eles, respectivamente: **lua/tools/compile.lua**, **lua/tools/makepack.lua** e **lua/tools/installer.lua**. O código **lua/tools/console.lua** serve como um *frontend* para a execução desses assistentes. Todos os assistentes (e o *frontend*) aceitam a opção de linha de comando **--help** que indica brevemente como usá-los.

Para facilitar o bootstrap é importante usar os assistentes em sua forma pré-compilada, principalmente, ao fazer a implantação no cliente. Assim evitamos a dependência da máquina virtual do Lua.

No JIRA temos os seguintes issues que tratam dessas implementações: OPENBUS-64, OPENBUS-95 e OPENBUS-135.

A seguir, é explicado a localização dos pacotes necessários ao procedimento de compilação do barramento. Na seção Procedimento de Compilação explica-se quais pacotes baixar e onde extrair, bem

como todos os passos da compilação. Na seção Procedimento de Instalação explica-se como baixar o instalador e proceder aos passos da instalação.

Repositório de pacotes

Os assistentes consideram o uso de um repositório de pacotes [2] para obter os códigos-fonte das dependências e prosseguir às etapas de compilação. Por enquanto, apenas o assistente de compilação usa o repositório.

A url do repositório (temporariamente) é: http://www.tecgraf.puc-rio.br/~openbus/repository/. Nesse se encontram os seguintes arquivos compactados:

- Assistentes pré-compilados:
  - Obter a versão específica de plataforma em: http://www.tecgraf.puc-rio.br/~openbus/repository/bootstraptools/
- Bibliotecas Lua:
  - lualibs.tar.gz — contém diretórios com todas as bibliotecas Lua que dependemos, são elas:
    - **latt**, **lposix**, **lua5.1.2**, **lualdap-1.0.1**, **luasocket2**, **luuid**, **oil04**, **tolua5.1**
- Bibliotecas não-Lua:
  - openssl-0.9.9.tar.gz — OpenSSL versão de desenvolvimento 0.9.9
  - openldap-2.4.11.tar.gz — OpenLDAP versão estável, também depende de:
  - cyrus-sasl2-2.1.22.dfsg1.tar.gz — SASL2 versão estável mais atual
  - db-4.6.21.tar.gz — Berkeley DB versão estável mais atual
  - e2fsprogs-1.40.8.tar.gz — Provê a biblioteca UUID, que é a única desse fonte que compilamos e instalamos
  - cxxtest.tar.gz — Suíte de testes para C++

O download a partir do repositório só acontece caso um pacote ainda não tenha sido obtido e nem tenha sido extraído ainda.

[2] *Atualmente só precisa ser um servidor http comum*

Bootstrap dos assistentes e configurações iniciais

Como os assistentes são scripts *pure-Lua*, em certas situações (na implantação no cliente, por exemplo) é interessante não dependermos do Lua instalado na máquina. Para permitir esse uso é feita a pré-compilação dos assistentes tornando-os binários independentes. Para proceder às tarefas de compilação, geração e instalação é **obrigatório** o uso dos assistentes.

Passo-a-passo de instalação dos assistentes:

1. Certifique-se que as variáveis de ambiente do Tecmake foram configuradas, os assistentes dependem das variáveis TEC_UNAME e TEC_SYSNAME
2. Descarregue o pacote dos assistentes para sua plataforma a partir repositório
3. Extraia o pacote manualmente na pasta **<home usuario>/openbus/trunk/**
   a. É possível usar outro diretório, para isso é **obrigatório** criar um arquivo de configuração para os assistentes contendo ao menos:

      ```
      DEPLOYDIR = "/caminho/completo/de_onde/deseja/usar/assistentes"
      ```

   b. Os assistentes devem, nesse caso, serem executados informando a opção **config**:

      ```
      tools config=/caminho/completo/assistentes ...
      ```

4. É recomendado atualizar sua variável **PATH** com **<diretorio extracao>/tools/bin/$TEC_UNAME** para mais facilmente conseguir usar o comando **tools**

Pronto. Tendo instalado os assistentes, recomendamos usar a opção --help caso seja seu primeiro contato com eles.

```
$ tools --help
$ tools compile --help
$ tools makepack --help
$ tools installer --help
```

Um exemplo rápido de uso é dado a seguir [3]:

1. compilação do próprio assistente (é preciso ter as dependências de compilação instaladas):

   ```
   tools compile select="lualibs lua5.1 lua5.1-bin loop tools"
   ```
2. geração do pacote do próprio assistente para sua plataforma atual:

   ```
   tools makepack profile=bootstraptools
   ```
3. instalação do pacote final com os assistentes pré-compilados:

   ```
   tools installer package=<home usuario>/openbus/packs/openbus-<versao>-bootstraptools-
   <plataforma>.tar.gz
   ```

[3] *as strings com* ***<variavel>*** *devem ser substituídas pelo valor que fizer sentido no cenário de uso do usuário*

O que fazer quando não houver versão pré-compilada dos assistente para sua plataforma?

A motivação principal para ter os assistentes pré-compilados é não depender do Lua-5.1 instalado. Contudo **caso o Lua5.1 esteja instalado** é possível usar os assistentes bastando entrar na pasta **openbus/trunk/tools/lua/tools** e usando diretamente os scripts Lua:

```
$ lua5.1 console.lua --help
$ lua5.1 console.lua compile --help
$ lua5.1 console.lua makepack --help
$ lua5.1 console.lua installer --help
```

Procedimento de Compilação

Procedimento de Geração de Pacotes

Procedimento de Instalação

Testes

Como verificar se a instalação dos serviços básicos teve total sucesso?

1. Entre na pasta do **OPENBUS_HOME/core/bin**
2. Execute o Serviço de Controle de Acesso:

   ```
   ./run_access_control_server.sh
   ```
3. Caso algo saia errado e o serviço não seja levantado, entre em contato comigo: **amadeu@tecgraf.puc-rio.br**

Caso tenha instalado um pacote contendo os testes unitários dos serviços básicos, é possível executar testes de cobertura:

1. Entre na pasta dos assistentes
2. Defina a variável OPENBUS_HOME para o diretório da instalação dos serviços básicos
3. Defina a variável OPENBUS_HOME_DEVEL para o diretório que contém a seguinte estrutura de diretórios: **core/test/lua**
4. Execute o comando:

   ```
   ./run_all_tests.sh
   ```