Standardizing J2EE ® industrialization

Version : 1.0
Date : 19/08/2006
Author : ¨Patrick Germain Placidoux

SUMMARY

OBJECTIVES

The objective of this document is to provide a ready to use Arcitectural skeleton to support J2EE Application industrialization.

This standard is not an official one of any kind but offers my 15 years experiences as J2EE Administrator for the major companies of the market.

Note:
The technics and tricks provided in this documentation do not directly concerns Kikonf but must be seen as a general way to work it out in heavy Application hosting environment.
Obviously Kikonf is designed by nature to support this task.

This document point of view:

This document is a technical charte writed and provided by the System department.
The System department is the one whose responsability is to support and maintain available the hosted Applications on the required environments.

1 INTRODUCTION

First let's see what we mean by industrialization

By industriazation we mean:

_ Disponibility: The ability to quickly provide a suitable environment for one incomming Application.
_ Isolation: one incoming Application must not put at risk the existing Architecture or other Applications.
_ Security : A minimum (or highter) security must be implemented at each level to meet the previous condition, the Architecture must allow it.
_ Delegation : The Architecture Administration tasks must be sustainable by non expert **ingeniers.**
_ Automation : This point is deducted from the previous. Once a human operation is often repeated and happens to concerne the need of a **large enougth portion** of the hosted Applications : **it must be Automated** (Do not automate one shot tasks).

"large enougth portion":
As you guessed the design of a hight scaled industrialized Architecture is a matter of "just equilibre".

The game is always to identify what will concerns the main part of the hosted Application and what may concern a few portion.
A figth between what is general and what is particular.

As long as the 90% of the needs of the hosted Application is satisfied, the Architecture must be considerated **as accurate**.

We must never have the dictatorial approach of the 100% objective,
because in real world this never works.

The more you'll try to get close to the 100% the more you'll write particular the more you get you Architecture unstable.

Remember this rules true for ever :
Its demands 10% of the energy to reach the 90% and 90% of the energy to get the 10% left.

For the other 10% write particular for the few Applications that may need them.

The 90% needs must Normalised.

And this is our concern in the following chapters.

The Norm must be propageted to all department interacting with the architecture (Application Managers, Integrators, Developpers, Supervisers)

Note:
In the next chapters what appears in italic are comment and explaination, and not supposed to remain in the final documentation.

2 NAMES SEPARATOR

Ok, this may sound strange but the first think you have to think about is the separator.

Along this Norm we'll have to manipulate a bunch of Names (Application names, Servers names Virtual host names, ...)

And often those individual names will be concatenated together to build another significative names.

Let's run an example:

Environment : ***prod***
Application : ***invoice***

To identify an Aplication Server running for Application "invoice" on the "oat" environment will may call it:
invoice_oat_srv_01
invoiceOatSrv01
or
invoice-oat-srv-01

My arbitrary choice here is separator:"_".

This means that later in this Standard separator:"_" will be forbiden for brick names like: Application or Environments.

Later in this documentation,
a few **fundemental name**s are described (like Environment names, Application names, ...),
afterard fundemental names are used as bricks to build trivial name (like Application names, Servers names, Virtual host names, ...).
What we call the name separator is the glue that melt these bricks.

The names separator is "_".

3 FREE OR FIXED LENGTH FOR NAMES

As we saw, we have to run overs a bunch of names along this document.

Before starting we have to decide either we choose fixed or free name.

Let's run an example:

Here is a list of environments :
prd : Production
oat : Operational Acceptance Testing
uat : User Acceptance Testing
dit : Development Integration Testing
dev : Development

If we choose fixed names, we should consider a length short enought to be concatenated with other names but significative enought to understand what we are talking about, let's say 3, this gives:
prd for Production
oat for Operational Acceptance Testing
uat for User Acceptance Testing
dit for Development Integration Testing
dev for Development

Ok this is significative enougth.
Fixed names is an easy choice when their number is limited.
But let's take a look to the next situation.

Here is a non exhaustive list of Applications :
invoice
incomes
accounts
accounters
contract
sells
bayers

If we choose fixed names, we should consider a length short enought to be concatenated with other names but significative enought to understand what we mean.
But how short enougth, let's say s*hortened to* 4, this gives:

invo
inco
acco
acco

cont
sells
bayr

We have two identical alias "acco" and a not so significative one : "bayr".
Shortening Application names can soon becomes a nightmare:
_ Because the number of incoming Applications is unlimited and unknown.
_ An alias may often preexists for the ...th incoming Application.

For this specific case of the Application names, I prefer to adopt the politic of free name under a limit of lenth.

This way makes it often possible to keep the Dev department project names.

3.1 THE SPECIFIC CASE OF SHORT SYSTEM NAMES

Because a lot of system names are restrained by design to be short :
_ user names are (<=8).
_ file system names for instance ar not.

Our mechanism to use the Application name to build short system names is put at risk.

For short system name we will always:

- use fixed names for fundemental names.
- use equivalence table for free fundemental names.
- use no separators.

4 ENVIRONMENTS

The supported environmenets are :
Now we publish the exhaustive list of the supported environmenets by our IS in fixed lenth: 3:

Syntax for ENV

<ENV>
ENV: a 3 characters string representing the environment, defined by System department.

Exhaustive list:
prd : Production
oat : Operational Acceptance Testing
uat : User Acceptance Testing
dit : Development Integration Testing
dev : Development

4.1 OPERATING SYSTEM

The Operating System may of any kind (Windows ®, AIX ®, Linux ®, Macs ®...), the proceeding remains the same,
even if for convenience we use the Unixes like path notation.

4.1.1 *Environement directories*

To each environement names is associated a physical directory on the machine (or O.S. Partition).

Lets says that these directories are store at / for convenience (It could also be c:\my\other\base\path or anything). So in the following "/" should be replaced by your real base path for the environment directories.

Path for ENV_HOME:

/<ENV>
ENV: a 3 characters string representing the environment.

Exhaustive list:
/prd : Production
/oat : Operational Acceptance Testing
/uat : User Acceptance Testing
/dit : Development Integration Testing
/dev : Development

5 APPLICATION

5.1 APPLICATION NAME

This chapter focuses on the Application and the resources Allocated by the structure to host it.

To preserve our ability to concatenate Application names to build other names we must fixe an arbitrary limit for their length.

The length of an Application name must not exceed 10.
Application names must not include "_" *(See the Separator chapter)*

Syntax for APP_NAME:

<APP_NAME> : a string representing the Application name often close to the Dev Department name for the project.

The sample Application name used in the next examples is : **invoice**.

5.2 APPLICATION IMPLEMENTATION NAME

The implementation name for an Application is its name declined for a particular environment.

Syntax for APP_IMP_NAME:

<APP_NAME>_<ENV>
APP_NAME : a string representing the Application name often close to the Dev department name for the project.
ENV: a 3 characters string representing the environment, defined by System department.

Example:
invoice_dev : invoice implementation's name on dev environment.
invoice_uat : invoice implementation's name on uat environment.
invoice_prod : invoice implementation's name on prod environment.

In the internal Work Flow talking about the Application should always be in this term.
Talking about "**invoice_oat**" ok every one understands that we are actually talking about the vesion of the invoice project currently on operational testing.

5.3 APPLICATION SYSTEM CODE

The System code length is fixed to 3.
Each Application receive an unique System Code.
The System Code is arbitrary designed by the System department.

Eg: invoice<=> aux (Alphabetic combinaison allows: 3 puis 26 + 26 possibilities, that's enougth)
This code has strictly no significative sementic and an equivalence table must be maintain.

Syntax for APP_SYS_CODE:

<SYSTEM_CODE>
SYSTEM_CODE : arbitrary 3 characters string.

5.4 OPERATING SYSTEM

The Operating System may of any kind (Windows, AIX, Linux, ...), the proceeding remains the same, even if for convenience we use the Unixes like path notation.

5.4.1 Application User name

A user must be defined for the Application.
All process related to this Application should run under this user.
This facilitates processes Work Load Management and supervision.

The user name should reflect the Application's name (eg: invoice) and the environment for wich this Application is currently implemented (eg: uat).
But because the Application'name is free and the System User names are often limited to 8,
we have a situation here.

The best solution I've found is to abitrary attribuate a short system code to match the Application name. See the chapters Names separator and Application System Code.

The System user name is:
_ the user under wich all process related to the Application, are run for a particular environment.
_ the owner of the directory allocated to this Application.

Syntax for APP_USER:

<APP_SYS_CODE><ENV><INDEX>
SYSTEM_CODE : an arbitrary 3 characters string designed by System department for a given Application.
ENV: a 3 charater string representing the environment names.
INDEX: a two digit index.

Example:
auxint01

Index are helpful when in some situations you may have two run multiple processes for the same Application but on distinct users.

5.4.2 Application directrory

In order to apply the splitted boxes separation rule, each Application must have its own directory. Ideally this directory should be an individual File-System or partition that can be mounted/unmounted.
All the resources needed by the Application must be stored or redirected under this directory.

The Application directory is where the whole Application resources are stored.
The owner of this directory and its subsquents files is the Application user.
This directory also is the home directory for this user.
This directory is a subdirectory of the currrent environment.
This allows multiple environments to co-exist on the same machine (or O.S. partition).

Path for APP_HOME:

/ENV_HOME/<APP_NAME
ENV_HOME : the environment home directory.

Samples:
/uat/invoice
/oat/invoice
/prod/invoice

The Application home directory structure:

delivery : The directory where Application Managers store the Application package ready to be proceed by the administrative deployment procedure.
binaries : Is where the Application package is expanded by the specific vendor's J2EE Application Server on the deployment procedure.
sharedlibs : a place to store shared librairies, common to all Application server instances.
This directory is explicited into the Application Servers JVM ® args as -DextDir.
security : this directory is available to store eventual security keys need to operate this Applciation.

The implementation of one Application on one environment may support more than one Application Servers (the Application could be verticaly clustered for example).
The following describes the Application server directory structure.

The Application Server instance directory path:

APP_HOME/<APP_SRV_NAME>
APP_HOME: the Application home directory.
APP_SRV_NAME: See the chapter J2EE Application Server Archetypes.

Samples:
/uat/invoice/invoice_uat_srv_01
/oat/invoice/invoice_oat_srv_01
/oat/invoice/invoice_oat_srv_02

/prod/invoice/invoice_prod_srv_01
/prod/invoice/invoice_prod_srv_02

The Application Server instance directory structure:

properties : here are stored the specific propertiy files for this Application Server instance.
This directory is explicited into the Application Server JVM args as -DextDir.

temp : The public Application temporay directory. The Application code may use it.
Note:
This directory is explicited into the Application Server JVM args as:
-Djava.io.temp=path
-Dtemp.dir=path
-Djava.io.tmpdir=path

The Application code can retreive it using thi:
String tmpDir=System.getProperty('java.io.tmpdir');

logs : here are redirected all the vendor specific J2EE Application Server logs for one Application Server instance.

dump : The core dumps for the JVM of this Application Server instance are redirected here.

coretemp : here are redirected all the vendor specific J2EE Application Server, scratches files for one Application Server instance (.class, perf temp files, etc...).

tranlog : here are redirected the vendor specific J2EE Application Server, transaction logs for one Application Server instance (.class, perf temp files, etc...).

Often Application servers instance are placed behind WebServer instance charged to served static files because they are more efficient for that. This also need to be strutured.

The WebServer directory path:

APP_HOME/<WEB_SRV_NAME>
APP_HOME: the Appication home directory.
WEB_SRV_NAME: See the chapter Web Server Archetypes.

Sample:
/uat/invoice/invoice_uat_wbs_01
/oat/invoice/invoice_oat_wbs_01

The WebServer Server directory structure:

conf : configuration for this web server instance take place de here.
html : this is where are deployed the static web package delivered by the Application Managers.
logs : WebServer logs and vendor specific Application Sever web plugin logs are redirected here.

6 J2EE APPLICATION SERVER ARCHETYPES NAMES

By J2EE Application Server Archetype we the names under wich the vendor specific Application Server knows and store a typical resource into is repository.
This Application Sevrer may be WebSphere Application Server ®, Weblogic Application Server ®, Apache Tomcat ® or Jboss ® the rules remain the sames.

The basic rule for the syntax is that an Archetype names must contain the Application implementation name and an acronym for its type.

With the environement included into the implementation name a Cell or domain may support more than one environement.

The general syntax for the J2EE Application Server Archetype names is

Syntax:
<APP_IMP_NAME>_<TYPE>
APP_IMP_NAME: The Application implementation name.
TYPE: a short string alias for the type.

Here a non exhaustive sample list of these archetypes :

Application
This refers to the name under wich the Application has been deployed into the Applciation Server.

Syntax:
<APP_IMP_NAME>

Sample:
invoice_oat
invoice_oat

Application server

Syntax:
<APP_IMP_NAME>_srv_<INDEX>
INDEX: a 2 digits value.
This allows the support of more than one Application server for one Application implementation (in case of cluster for example).

Sample:
invoice_oat_srv_01
invoice_oat_srv_02

Cluster

Syntax:
<APP_IMP_NAME>_cls_<INDEX>
INDEX: a 2 digits value.
This allows the support of more than one cluster for one Application implementation.

Sample:
invoice_oat_cls_01
invoice_oat_cls_02

WebServer

Syntax:
<APP_IMP_NAME>_wbs_<INDEX>
INDEX: a 2 digits value.
This allows the support of more than one Web server for one Application implementation.
Note: Instead of "wbs", TYPE could also be "apa" or "ihs".

Sample:
invoice_oat_wbs_01
invoice_oat_wbs_02

Virtual Host

Syntax:
<APP_IMP_NAME>_vhost

Sample:
invoice_oat_vhost
invoice_oat_vhost

6.1 J2EE RESOURCES

As you know many Application Server softwares allow to declare resources as many levels : cell, domain level, node, server level, ...
In our split boxes individualization view, the bes is to use the scope Applciation server. This way a wrong resource declaration wont never impact another JVM.

The general syntax is the same than above, except that for some resources we may add a string close as possible to the effective target of the resouces.

The general syntax for J2EE resources names is

Syntax:
<APP_IMP_NAME>_<TYPE>_<RESOURCE_NAME>
APP_IMP_NAME: The Application implementation name.
TYPE: a short string alias for the type.
RESOURCE_NAME: name of the target resource.

Jdbc provider

Syntax:
<APP_IMP_NAME>_jdbc_<RESOURCE_NAME>

Sample:
invoice_oat_jdbc_mydb

Data source

Syntax:
<APP_IMP_NAME>_ds_<RESOURCE_NAME>

Sample:
invoice_oat_ds_myds

JMS Queue Connection Factory

Syntax:
<APP_IMP_NAME>_qcf_<RESOURCE_NAME>

Sample:

invoice_oat_qcf_myqmanager

Queue

Syntax:
<APP_IMP_NAME>_queue_<RESOURCE_NAME>

Sample:
invoice_oat_queue_myqueue

Resource Adaptor

Syntax:
<APP_IMP_NAME>_ra_<RESOURCE_NAME>

Sample:
invoice_oat_ra_myra

Cache instance

Syntax:
<APP_IMP_NAME>_cache_<RESOURCE_NAME>

Sample:
invoice_oat_cache_mycache

Mail session

Syntax:
<APP_IMP_NAME>_mail_<RESOURCE_NAME>

Sample:
invoice_oat_mail_mymail

Url

Syntax:
<APP_IMP_NAME>_url_<RESOURCE_NAME>

Sample:
invoice_oat_url_myurl

7 PACKAGING

What we mean by package is the format under wich the Application is delivered to be deployed on the Application server.

A package may be an EAR or a WAR.

As you know Application Server software allows you to deploy more than one package into one JVM. You may decide to allow more than one package for one Applciation on the same JVM :
e.g. : invoice_oat_1.ear and invoice_oat_2.ear
or
you may decide to allow more than one Applciation on the same JVM :
e.g. : invoice_oat_1.ear + invoice_oat_2.ear and income_oat.ear

But please let me tell you this is a bad idea, you will save a lot of time spent into trying to resolve : class loader conflict, trying to manage reciproc compliant version of common libs, working combinaison of versions, and trying to figure out wich Application is really causing a critical issue.

Plus in our split boxes politic we do not admint that an Application could put at risk another. This is not the case sharing the same JVM.

From my point of view the best solution is to :

- *definitly choose one format ear or war.*
- *isolate one Application per JVM,*
- *definitly allow one uniq package (war or ear) per Application.*

Let's say we choose the ear format for the following.

7.1 PACKAGING FOR THE APPLICATION SERVER

Only one unique package is allowed to delivery for an Application implementation.
The allowed format is ear.

Syntax:
<APP_IMP_NAME>.ear
APP_IMP_NAME: the Applciation implementation name.

Sample:
invoice_oat.ear

7.2 PACKAGING SHARED LIBRAIRIES

Only one unique librairie package is allowed to delivery for an Application implementation.
The allowed format is zip.

Syntax:
<APP_IMP_NAME>_lib.zip
APP_IMP_NAME: the Applciation implementation name.

Sample:
invoice_oat_lib.zip

7.3 PACKAGING FOR THE WEBSERVER

Only one unique librairie package is allowed to delivery for an Application implementation.
The allowed format is zip.

Syntax:
<APP_IMP_NAME>_web.zip
APP_IMP_NAME: the Applciation implementation name.

Sample:
invoice_oat_web.zip

8 PROPERTIES

General rules for bindings:

The binding mechanisisms refers to the linking of J2EE ressources used into the Application packaged (EAR/WAR file) to a real existing ressource on the Application server.

The creation and the availability of ressource in the Application server is the System Department responsability.

The use of these ressources is the Dev department responsability.

Do restrain developpers to externalize their ressources to the deployment descriptor
or to use **Java ® 6 Annotations**.
Do restrain developpers to use a generic way like **java:comp/env** to bind ressources.
Do requires them to use Application Servers vendors specific binding extension descriptors (like ibm-ext...binding for WebSphere Application Server).

Note :
The previsous 2 first point are J2EE best practises.

Eventually block the ability of some Application Server to create new ressources from scratch from Application package.

Definitly avoid:
_ to make binding by yourself on the console on deployement.
_ to try to automate binding by scripting.

Because of the amount of the Application being deployed, the number of available J2EE ressources, scripting binding in a lost black hole.
And you'll finally lost in writing a specific install foreach Application.

The installation script must be simple as possible.

Rule 1:
Defines a standardized jndi-contract document.
The Manager of the incoming Applicatin explicitly list the Name/characteristics of the whole resouves he needs for the Application and their JNDI names.

Rule 2:
System creates thos ressources in the Application dedicated server(s) and the System contract and responsability stop here.

Rule 3:
Dev team run the specifics vendors extension on packaging their files (EAR, WAR) in one click (there IDE are powerful enougth for that) or to use Java 6 Annotations.

9 WEB SERVER ARCHETYPES

The webserver configuration file is stored at : ***APP_HOME/<WEB_SRV_NAME>/conf***
see the chapter: Application directrory.
e.g.: /uat/invoice/invoice_uat_wbs_01/ invoice_uat_wbs_01.conf

Here we name the WebServer configuration file because this name appear in the process name.

Syntax:
<APP_IMP_NAME>_wbs_<INDEX>.conf
APP_IMP_NAME: the Applciation implementation name.
INDEX: a 2 digtits index.

Sample:
invoice_uat_wbs_01.conf

10 VENDOR SPECIFIC APPLICATION SERVERS

In this chapter will try tor cover different vendor specific Application Server.

Obviously we describe here, how are we intend to name some of those sotware archetypes to meet our industrialization objective, ***this not cover their many possibilites.***

10.1 WEBSPHERE ARCHETYPES

Cell

Syntax:
<HOSTNAME><ENV>Cell<INDEX>
HOSTNAME: The current hostname of the machine/partition on wich the binary is installed.
ENV: the environement name allocated to this Cell.
INDEX: a 2 digit index.

Sample:
myhostUatCell01

Deployment Manager

Syntax:
<HOSTNAME><ENV>Dmgr<INDEX>
HOSTNAME: The current hostname of the machine/partition on wich the binary is installed.
ENV: the environement name allocated to this Cell.
INDEX: a 2 digit index.

Sample:
myhostUatDmgr01

Deployment Manager Profil

Syntax:
<HOSTNAME><ENV>DmgrProf<INDEX>
HOSTNAME: The current hostname of the machine/partition on wich the binary is installed.
ENV: the environement name allocated to this Cell.
INDEX: a 2 digit index.

Sample:
myhostUatDmgrProf01

Node

Syntax:
<HOSTNAME><ENV>Node<INDEX>
HOSTNAME: The current hostname of the machine/partition on wich the binary is installed.
ENV: the environement name allocated to this Cell.
INDEX: a 2 digit index.

Sample:
myhostUatNode01

Node Profil

Syntax:
<HOSTNAME><ENV>NodeProf<INDEX>
<hostname><environment><stream>Node<index>
HOSTNAME: The current hostname of the machine/partition on wich the binary is installed.
ENV: the environement name allocated to this Cell.
INDEX: a 2 digit index.

Sample:
myhostUatNodeProf01

11 TRADEMARKS:

"Linux“ is a trademark registred to Linus Torvalds.
"AIX" and "WebSphere Application Server" are registred trademarks of International Business Machines Corporation.
"Windows" is a registred trademark of Microsoft Corporation.
"Macintosh" and "Mac" are registered trademarks of Apple Inc.
"JBoss Application Server" is a registered trademark of Red Hat and its affiliates in the U.S. and other countries.
"Oracle WebLogic Server", "JVM", "J2EE" and "Java“ are registred trademarks of Oracle and/or its affiliates.
"Apache Tomcat" is a registred trademarks of the Apache Software Foundation.

Other names may be trademarks of their respective owners.