HowTo deploy J2EE Application using ATG Dynamo Version 5.0

25.07.2001, mm

Autor	Michael Maretzke
eMail	michael@maretzke.com

Stand	31.07.2001
Status	preliminary finished released
	x intern x extern
Version	1.0
Druckdatum	17.11.03 22:25
Verwendungszweck	Knowledge Management

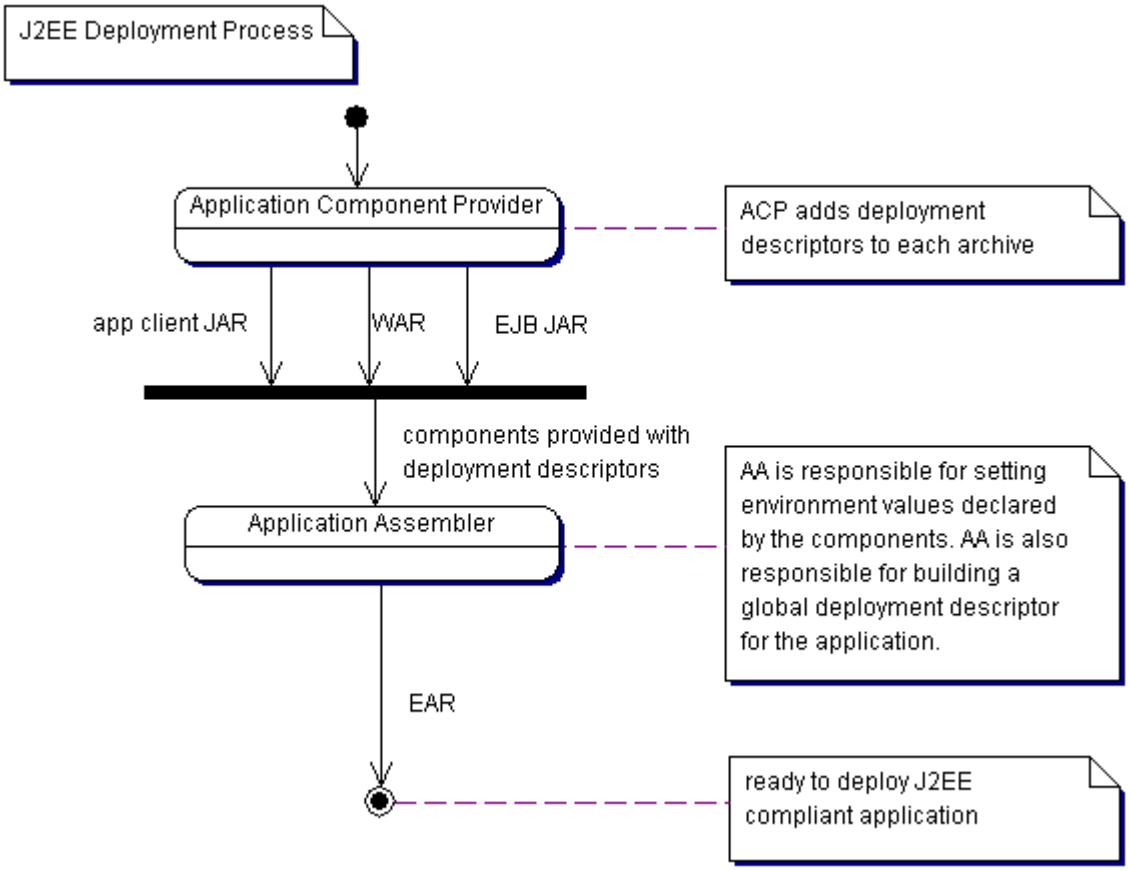
1 Overview of J2EE Deployment

J2EE means applications needs will be coded into one of four acceptable component types. These are Servlets, JavaServer Pages, Tag Libraries and Enterprise JavaBeans. Plugging them together to an application is one of J2EE-deployment tasks.

J2EE introduces very strict application packaging. J2SE doesn't matter about how to deploy applications. This point was strictly changed introducing J2EE. A reliant process and format was defined.

While J2EE strictly rules creating the application archive the deployment process is not described at all. Deployment means here making the application archive run in a specific J2EE container (e.g. ATG Dynamo). Here are many ways a J2EE container deploys the application.

After the graphical overview the relevant entities for J2EE deployment will be described shortly. Further information about J2EE deployment may be found at [SUN01], [SUN02], [SUN03], [SUN04], [SUN05] and [ATG01].



Graphic 1Overview of J2EE deployment process and involved roles

1.1 Deployment Descriptors

The functional parts of an application are stored in component archives. To make these archives fit together more logical information about how to plug them together is needed. It also declares the environment expected by the

components. This environment must be configured before the application can be run. This higher-level information (e.g. name and description of components) is stored in so called deployment descriptors.

The deployment descriptor is a XML-file packaged with the component's or application's archive file.

1.2 Archive Files

Each of the single archive files has its own archive format. They all use the JAR file format but differ internally.

1.2.1 Web Archives - WAR

The web archive contains classes, content, JSPs and a single deployment descriptor „`web.xml`" for a web application. One way to think of a WAR file is a JAR-ed version of the web site's document root.

`WAR archive structure`

```
<JSP pages and content resources>
WEB-INF/
      web.xml (deployment descriptor)
      classes/
        <classes and resources>
      lib/
        <class and resource .jar files>
```

1.2.2 Enterprise Java Bean Archives – EJB JAR

The EJB JAR contains the classes and deployment descriptor for an EJB application. This archive is basically a JAR file of the classes with a deployment descriptor „`ejb.xml`" added.

`EJB JAR archive structure`

```
<classes and resources>
META-INF/
      ejb.xml (deployment descriptor)
```

1.2.3 Application Client Archives – AppClient JAR

The AppClient JAR contains the classes and deployment descriptor for an application client. The archive is basically a JAR file of the classes, a deployment descriptor and a `MANIFEST.MF` file.

AppClient JAR archive structure

```
<classes and resources>
META-INF/
      application-client.xml (deployment descriptor)
      MANIFEST.MF (names the main class)
```

1.2.4 Enterprise Application Archvies – EAR

An EAR archive file is also basically a JAR archive file. It contains the EJB JAR, WAR and the AppClient JAR file. They are packaged as single units into the EAR file, means they will not be decompressed. They are added as they are. To package the single archives correctly together a deployment descriptor is added.

EAR archive structure

```
<EJB JAR, WAR and AppClient JAR>
META-INF/
      application.xml (deployment descriptor)
```

1.3 Roles involved in the deployment process

Roles may imply that one person or perhaps more persons may implement the role. Roles are more like a view on the work a person does.

1.1.1 Application Component Provider (ACP)

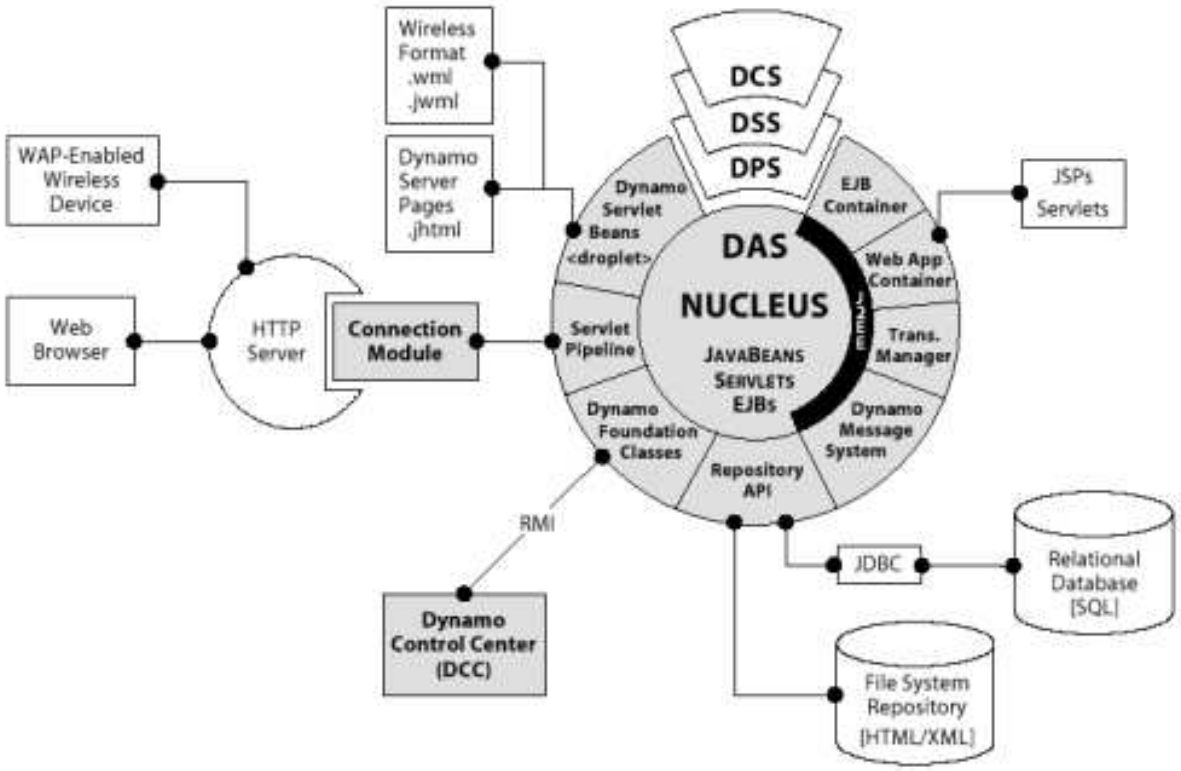
One or more ACP produce archive files in the J2EE context. They produce WARs, EJB JARs, AppClient JARs and provide them all with deployment descriptors. These components are delivered to the AA described next.

1.1.2 Application Assembler (AA)

The AA takes components delivered from one or more ACP and plug them together to get the applications‘ functionality. AA is responsible for creating an overall deployment descriptor describing the whole application, setting needed component attributes and resolving needed references. AA produces another archive file – the EAR, enterprise application archive.

2 Deployment process

To understand what happens when the J2EE application morphs into an ATG Dynamo J2EE application it's important to know how ATG treats J2EE applications. Have a kind look at the graphic below. [ATG02]



Graphic 2 Architecture of ATG Dynamo

J2EE functionality extends core functionality provided by the ATG nucleus component. ATG decided to have a similar view on J2EE applications as they do on their own applications. Therefor ATG encapsulated the functionality of J2EE applications into their own application format. J2EE container functionality is provided by an ATG component called J2EEContainer. Inside this container „lives" J2EE.

This approach means there are some ATG specific extensions which will be described below.

2.1 ATG specific extensions

2.1.1 ATG module extensions

[ATG03], [ATG04] Two files encapsulate the J2EE application and make it look like a ATG application. These files are:

`/config/atg/dynamo/service/j2ee/J2EEContainer.properties`

`/META-INF/MANIFEST.MF`

The first file sets properties for the component `J2EEContainer`. Typically it extends the portfolio of existing applications with the new one. A `J2EEContainer.properties` file may look like this:

```
# Version: $RCSfile: J2EEContainer.properties,v $$Revision: 1.2 $$Date:
2000/06/16 22:12:44 $

applications+=\

      {atg.dynamo.root}/AdditionBean/AdditionBean.dar
```

The second file makes the first file visible to ATG Dynamo. It sets the configuration directory and some requirements. Have a look:

```
Manifest-Version: 1.0

ATG-Config-Path: config/

ATG-Required: DAS

ATG-Patch-Build: 2714

ATG-Patch-Date: 20010619

ATG-Merge-Log: 5.1p2 5.1.1p2

ATG-Patch-Time: 18:52:52

ATG-Patch-Version: 5.1.1p2
```

Think of this directory structure for a deployable J2EE application:

```
ATGAPP
  |- config
  |    |- atg
  |         |- dynamo
  |              |- service
  |                   |- j2ee
  |                        |- J2EEContainer.properties
  |- META-INF
  |    |- MANIFEST.MF
  |- J2EEAPP
       |- WAR
       |- EJB JAR
       |- APP CLIENT JAR
```

The red marked parts of the directory tree are extensions to the J2EE standard. The blue part represents the J2EE compliant application archive.

2.1.2 ATG specific J2EE extensions

Beside the two files mentioned above there's one J2EE specific extension with ATG Dynamo. It's the file:

`/J2EEAPP/META-INF/DYNAMO-INF/dynamoJ2EESpecifier.xml`

This configuration file is needed to supply ATG Dynamo or better said the deploytool Darina with all needed configuration settings to resolve all references inside the deployment descriptors. The file structure is similar to the `application.xml` file. For deeper information about this file see [ATG04].

As an example for this file here's one containing specific settings for a session bean and an entity bean with container managed persistance.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE dynamo-j2ee-specifier SYSTEM
"http://www.atg.com/j2ee/dtds/dynamoJ2EESpecifier/dynamoJ2EESpecifier_1.0.dtd">

<dynamo-j2ee-specifier>
  <application-name>
    reverser
  </application-name>
  <war>
    <module-uri>
      web-app
    </module-uri>
  </war>
  <ejb-jar>
    <module-uri>
      ejbs
    </module-uri>
    <enterprise-beans>
      <session>
        <ejb-name>
          Reverser
        </ejb-name>
      </session>

      <entity>
        <ejb-name>
          StorageBean
        </ejb-name>
        <cmp-mapping>
          <repository-name>
            dynamo:/atg/dynamo/service/jdbc/SQLRepository
          </repository-name>
          <repository-view-name>
            Storage
          </repository-view-name>
          <read-only>
            False
          </read-only>

          <field-mapping>
            <field-name>
              itemid
            </field-name>
            <property-name>
              itemid
            </property-name>
          </field-mapping>
          <field-mapping>
            <field-name>
              item
            </field-name>
            <property-name>
              item
            </property-name>
          </field-mapping>
          <field-mapping>
            <field-name>
              description
            </field-name>
            <property-name>
              description
            </property-name>
          </field-mapping>
          <field-mapping>
            <field-name>
              location
            </field-name>
            <property-name>
```

```
              location
            </property-name>
          </field-mapping>
        </cmp-mapping>

        <finder-method>
          <method-name>
            findByPrimaryKey
          </method-name>
          <finder-query>
            <![CDATA[ emp_itemid = ?0 ]]>
          </finder-query>
        </finder-method>

        <finder-method>
          <method-name>
            findAll
          </method-name>
          <finder-query>
            ALL
          </finder-query>
        </finder-method>
      </entity>
    </enterprise-beans>
  </ejb-jar>
</dynamo-j2ee-specifier>
```

2.2 ATG's deployment tool Darina

An J2EE compliant application needs to be adapted to the final production J2EE container. The adaptions are e.g.:

- checking the deployment descriptor
- checking the directory structure
- compiling the sources
- adding stub and skeleton features to the beans
- make specific adaptions
- ...

These tasks will be done with Darina. Have a look at the following sections about how to run Darina.

2.3 Deploying an unpacked application

The deployment tool Darina is capable of building a ATG compliant J2EE application from scratch. This means if there's no other tool in the development process producing EAR files (e.g. JBuilder, TogetherSoft ControlCenter, ... create EAR files) Darina does this for the developer. This approach is interesting if there's nothing more than a text editor and a telnet client available for development (can't imagine a company developing this way software).

2.3.1 a possible directory structure

Below is a possible directory structure. It's a structure ATG uses in it's demo application. It prooved its concept and therefor let's look at it:

```
ATG_J2EE_Application
  |- j2ee_application.dar
  |- META-INF
  |    |- MANIFEST.MF
  |- config
  |    |- atg
  |         |- dynamo
  |              |- service
  |                   |- j2ee
  |                   |    |- J2EEContainer.properties
  |                   |- jdbc
  |                        |- ...
  |- j2ee-apps
       |- META-INF
       |    |- application.xml
       |    |- DYNAMO-INF
       |         |- dynamoJ2EESpecifier.xml
       |- appclient
       |    |- META-INF
       |    |    |- application-client.xml
       |    |    |- MANIFEST.MF
       |    |- (packaged sources)
       |- ejbs
       |    |- META-INF
       |    |    |- ejb-jar.xml
       |    |- (packaged sources of ejbs)
       |- web-app
            |- (web resources as JSP's, HTML's, etc.)
            |- WEB-INF
                 |- web.xml
                 |- classes
                 |    |- (packaged sources of servlets & tags)
                 |- taglibs
                 |    |- (TLD files)
                 |- lib
                      |- (libraries, resources, etc.)
```

Red parts are ATG specific, blue parts are J2EE. Underlined elements are deployment descriptors.

2.3.2 running ATG's deployment tool Darina

Start Darina's deploy process by typing this command in the application's root directory (here in the directory `ATG_J2EE_Application`):

```
$DYNAMO_HOME/ATG_J2EE_Application#> $DYNAMO_HOME/bin/runDarina ./j2ee-apps -o
j2ee_application.dar -build -overwrite-dar
```

After successfully running Darina a runnable ATG application `j2ee_application.dar` was built. For more information about Darina see [ATG05].

2.4 Deploying an application in EAR format

A J2EE compliant EAR packed application is a much more easier process to deploy than the way above. But here are two processes of deployment involved.

First the J2EE EAR is created using whatever tool available. Afterwards the adaption for the specific J2EE container have to be made (here: ATG Dynamo). This makes the incremental development process a time wasting one.

2.4.1 running ATG's deployment tool Darina

After copying the EAR application into it's application directory in $DYNAMO_HOME directory some things have to be done:

- run Darina to get a raw `dynamoJ2EESpecifier.xml`
- create `/META-INF/MANIFEST.MF`
- create `/config/atg/dynamo/service/...` files to configure ATG components
- run Darina again to get a runnable ATG application

We will operate on this example directory structure:

```
ATG_J2EE_Application
  |- j2ee_application.dar
  |- META-INF
  |    |- MANIFEST.MF
  |- config
  |    |- atg
  |         |- dynamo
  |              |- service
  |                   |- j2ee
  |                   |    |- J2EEContainer.properties
  |                   |- jdbc
  |                        |- ...
  |- j2ee_application.ear
```

Red parts are ATG specific, blue parts are J2EE.

First step will be done running following command:

```
$DYNAMO_HOME/ATG_J2EE_Application#> $DYNAMO_HOME/bin/runDarina
j2ee_application.ear
```

Darina creates its staging directory named as the EAR-file (here: `j2ee_application`). The structure of the directory is similar to the one mentioned in 2.3.1 a possible directory structure.

After this step Darina has created a `dynamoJ2EESpecifier.xml` file. Edit this one to meet your needs.

Now it's time to create `/META-INF/MANIFEST.MF` and the ATG specific configuration files in `/config/atg/dynamo/service/...`. Have a kind look at 2.1 ATG specific extensions.

After having the application completely configured it's time to run Darina last time.

```
$DYNAMO_HOME/ATG_J2EE_Application#> $DYNAMO_HOME/bin/runDarina
j2ee_application –o j2ee_application.dar –build –overwrite-dar
```

Now Darina created the ATG runnable application called `j2ee_application.dar`.

2.5 Starting ATG Dynamo with deployed application

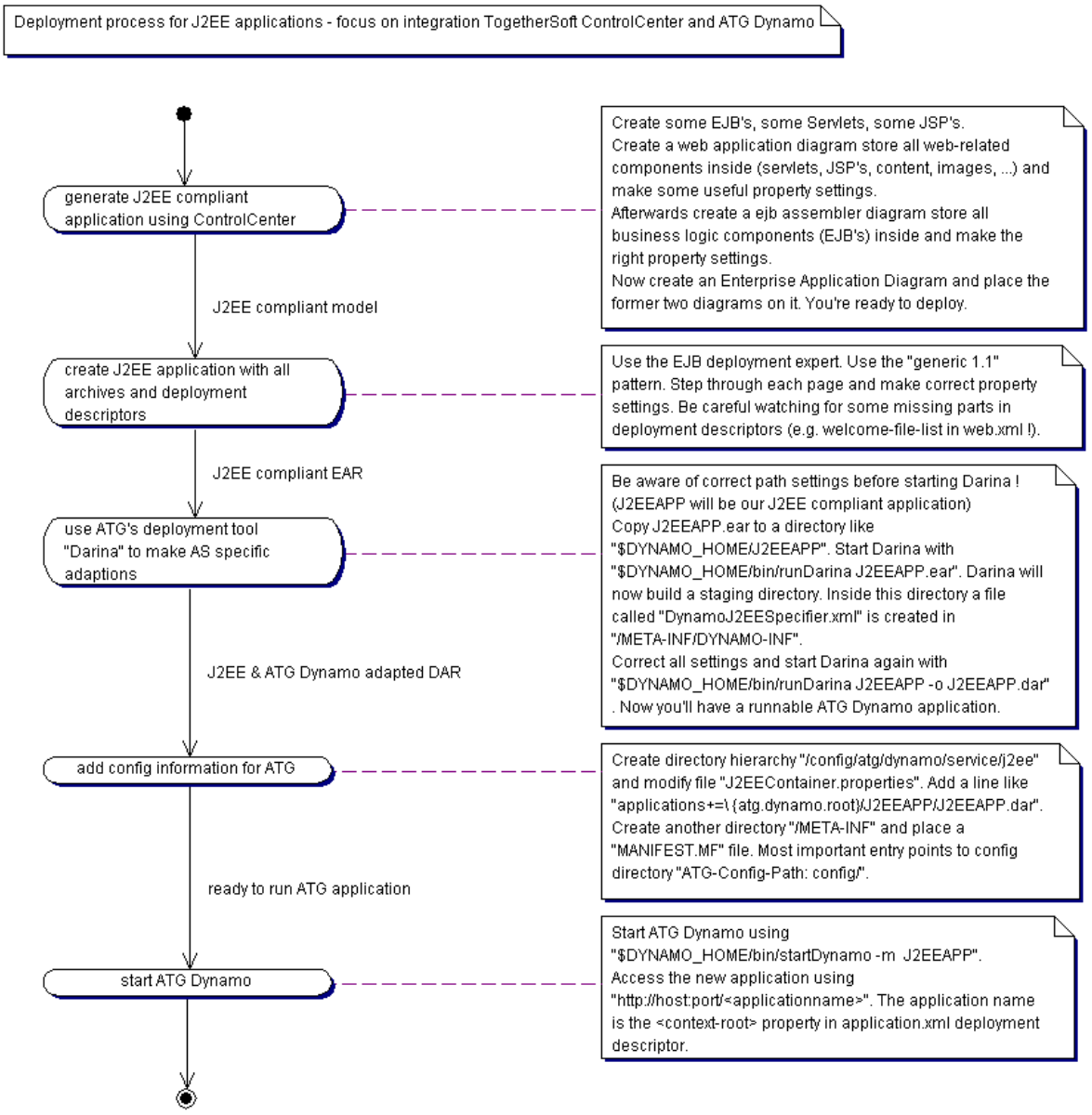
Starting Dynamo with the new application is rather easy. Switch to $DYNAMO_HOME/bin and enter following command

```
$DYNAMO_HOME/bin#> ./startDynamo –m ATG_J2EE_Application
```

The parameter after the –m option points to the directory which contains the Dynamo Archive file (DAR) and the ATG specific directories for configuration purposes.

Watch the logs after having typed this command above. If there are any errors your configuration is somewhere incorrect. A lot of fun searching !

1.2. Summary of ATGs J2EE deployment process using TogetherSoft ControlCenter



Graphic 3 Deployment process in detail - focus on integration between ControlCenter and ATG Dynamo

3 References

[SUN01] http://java.sun.com/j2ee/download.html#platformspec, J2EE specification from SUN

[SUN02] http://java.sun.com/products/ejb/docs.html, specification for ejbs

[SUN03] http://java.sun.com/products/servlet/download.html, specification for servlets

[SUN04] http://java.sun.com/products/jsp/download.html, specification for jsps

[SUN05] http://developer.java.sun.com/developer/technicalArticles/J2EE/behindscenes/BehindtheScenesII.PDF, good explanation of deployment process, very detailed

[ATG01] ATG Dynamo 5, Programmers Guide, Version 5.1.1, p. 1159 ff, packaging and deployment

[ATG02] ATG Dynamo 5, Programmers Guide, Version 5.1.1, p. 1181 ff, Dynamo Modules

[ATG03] ATG Dynamo 5, Overview, Version 5.1.1, p. 3 ff, Overview of ATG Dynamo Architecture

[ATG04] ATG Dynamo 5, Programmers Guide, Version 5.1.1, p. 1212 ff, Creating `dynamoJ2EESpecifier.xml`

[ATG05] ATG Dynamo 5, Programmers Guide, Version 5.1.1, p. 1235 ff, runDarina options