ODTUG Technical Journal - Fusion Column - Q2

Smart Application Upgrading - High application availability through zero planned down-time

Among the many requirements laid down by the business with regard to IT, including ultimate flexibility, omni-channel, ultra short time to market and Scrooge-level cost efficiency, is top level availability one of the most critical ones. In a 24/7 economy, even seconds of down-time may cause loss of customers. And with continued globalization of both the market and the [cloud based] infrastructure, even the peaks in the 24/7 timeline are fairly evenly distributed - leaving no room for quick and unnoticed down-time. Up-time is what is required from the IT department.

Another key requirements from the business is an almost continuous stream of small (or even large) changes to the functionality and look & feel of business applications. Agility is the name of the game, and a rapid succession of changes with extremely short time to market the consequence.

This installment of the Fusion Column will look at this challenge that seems an impossible one at first: changing almost all the time and not losing up-time while rolling out those changes. Surely that is a catch 21 - or is it? How can Fusion Middleware - where needed in joint effort with the Database - help us achieve application upgrades without perceived application downtime?

How in short can we fight that most illogical of concepts: *planned downtime*.

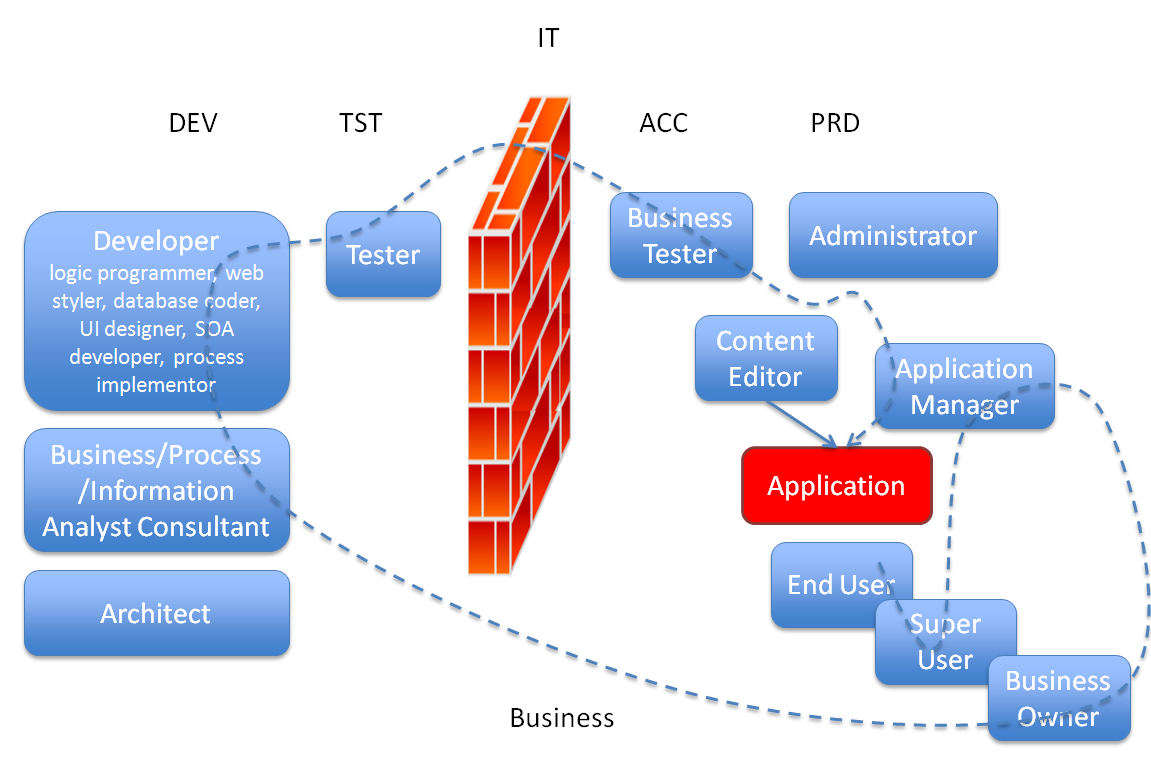
This article will touch upon the multi-version capabilities of BPM and SOA Suite, the production redeployment feature of WebLogic Server for Web Services and Web and ADF applications, the use of Oracle Database Edition Based Redefinition for database object level version management. First of all it will dive into run time configurability to even preempt the need for application redeployment.

***Design Time at Run Time - change on the fly without roll-out and down time***

The initial assumption that an application rollout is required for publishing new functionality into the run time environment, may itself be too hasty. Depending on the definition of *rollout* and the impact a rollout has, we may find that many of the changes desired by the business do not in fact require developer wrought software changes that subsequently need to be deployed. By baking configurability into applications - and providing facilities to edit the functionality and look & feel at run time - we hand the business the option to apply changes in live applications, similar to the way content editors will apply changes to portals and web sites.

Design time at run time makes making changes in the look and feel and behavior of the application very similar to the process of content editing that is well accepted in the run time environment. For example: UCM (Universal Content Manager, now aka WebCenter Content) - an important part of the web presence of organizations, both on the intranet as well as on the internet, can be determined through static web sites and content items maintained through UCM; content editors may create new versions of content items or replace existing items and the sheer act of publishing these changes has an immediate effect on the live applications as perceived by the end users. One moment, the 'old' versions of pieces of text, documents, images and page definitions are retrieved and from the next moment on, the new definitions are available. No deployment with any downtime is required, just a flip of the switch - typically as part of a well defined edit-and-publication workflow.

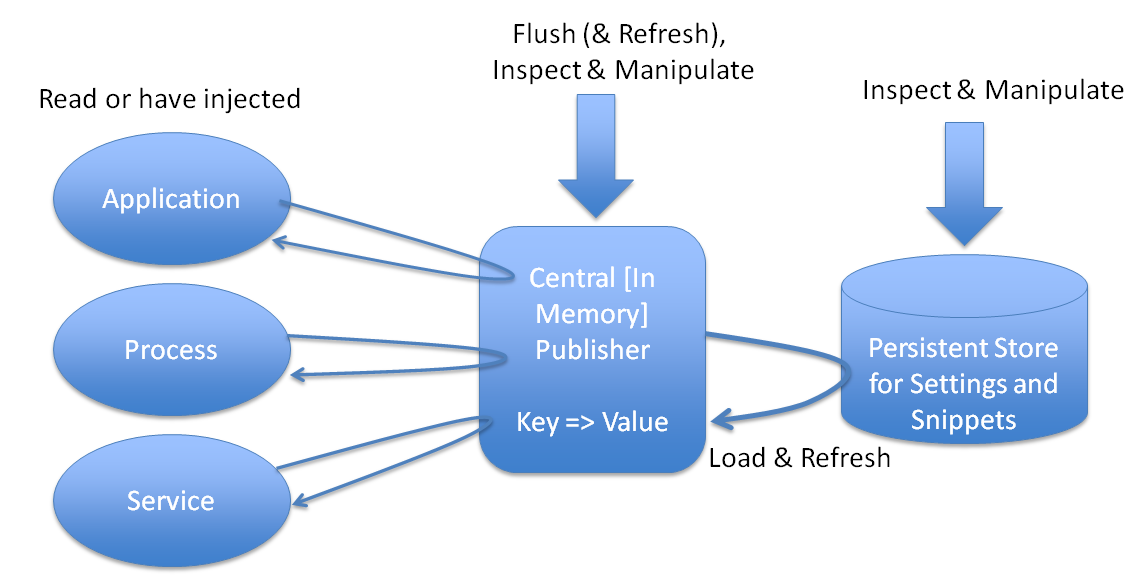
The next illustration compares the traditional, lengthy process of requesting, specifying, designing, implementing, testing and rolling out changes in application functionality with the editing of content elements at run time.



Run time configurability of applications can be custom developed, using various approaches. It requires your analysts, architects, designers and developers to work together in determining which aspects of the application(s) are the candidates for run time manipulation. Some things are more desirable than others, and some things are more expensive than others. You will need to strike a balance.

The application needs to be complemented with mechanisms that can read meta-data at run time and process said meta-data into manipulation of the application behavior. Two examples of design guidelines that would have to be implemented:

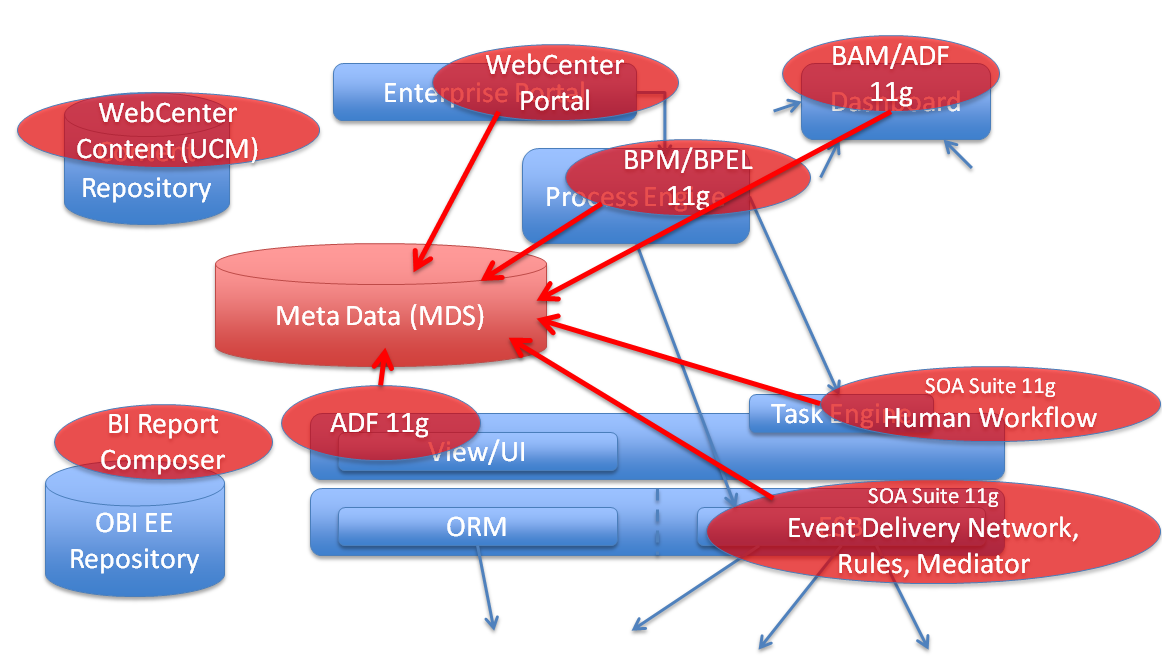
* do not hard code any value - for example boilerplate text, calculation factor or decision component, image url
* always read the values of 'constants' from a central 'manager' (that itself reads the values from a store that can easily be manipulated); this can be a singleton Java object or a database package; the next illustration describes this guideline in a generic fashion.



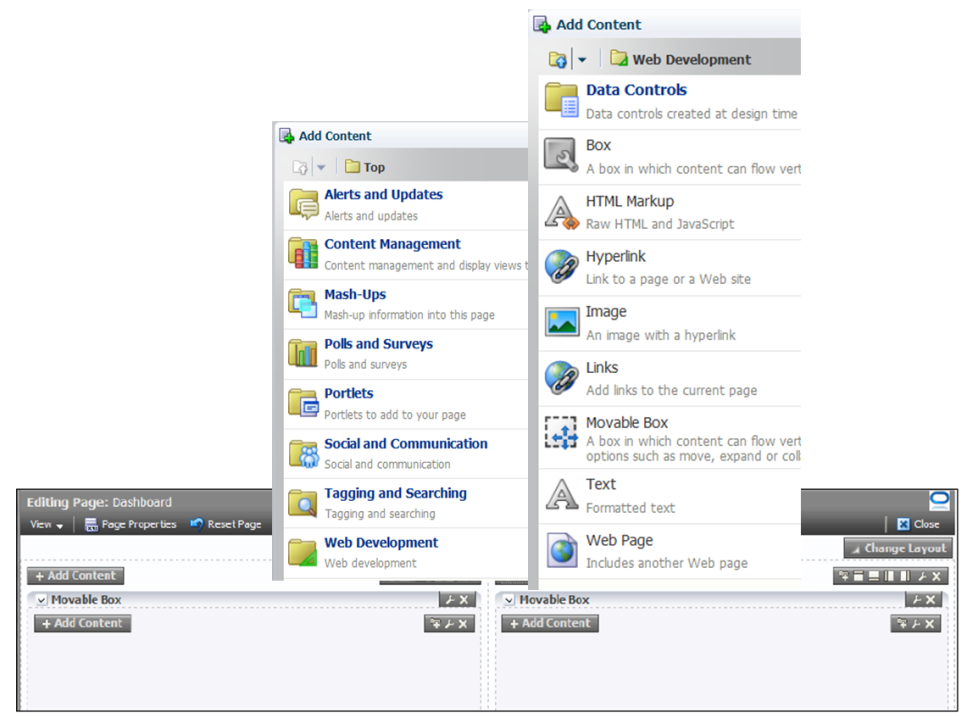
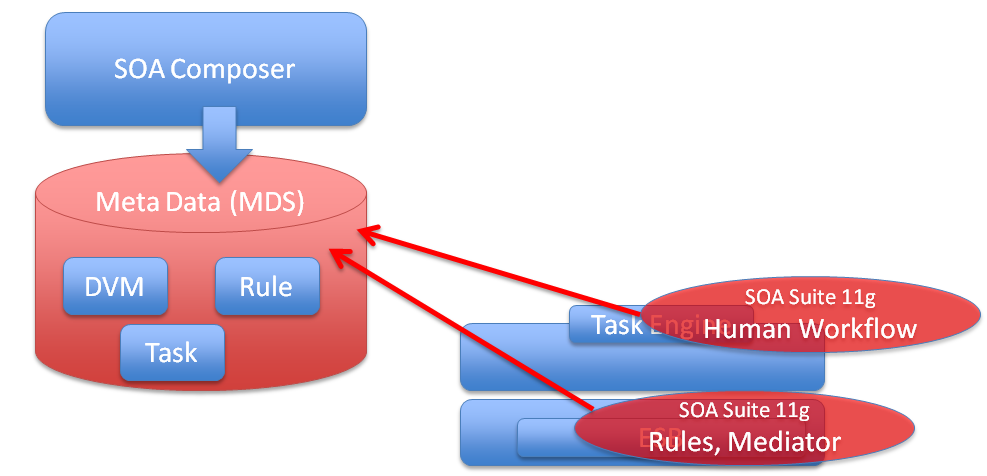
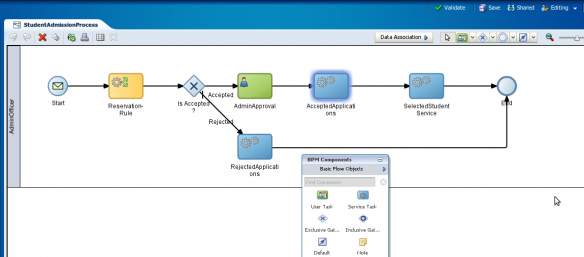
Additionally, the application requires a mechanism to edit the meta-data, to specify additional changes to the way the application appears and behaves. Ideally, such a meta-data editing mechanism is WYSIWYG, fully integrated into the application and available for users with proper authorizations.

For applications created with Oracle Fusion Middleware, we can leverage its "Design Time at Run Time" infrastructure, that strives to provide this same run time style editing for application behavior and look & feel as well. The installment of this Fusion Column in Q2-2011 discussed at length the Design Time at Run Time infrastructure in Fusion Middleware, the baked-in configurability.

FMW's Design Time at Run Time exists at various levels - all based on meta-data, stored in the run-time MDS repository (similar to the UCM Content Server).

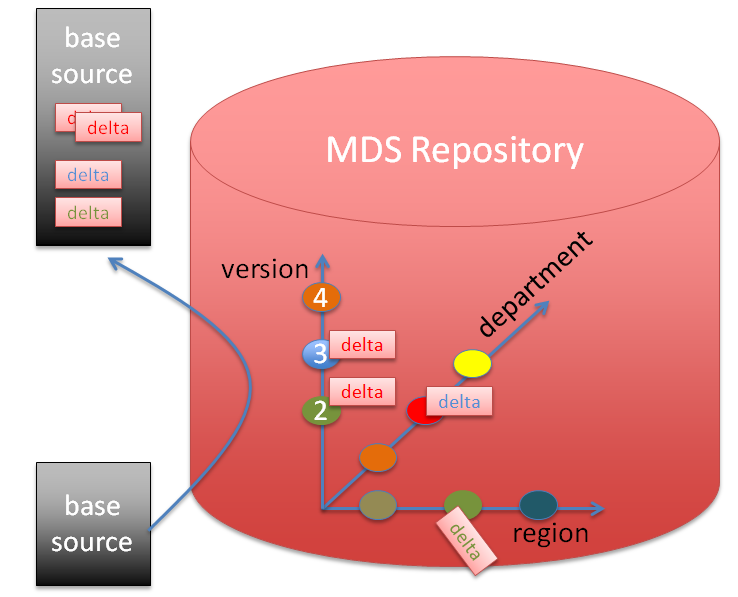


Changes created at run time are stored in MDS. When the application is accessed, these changes are applied on top of the originally deployed sources that the developers created in their design time environment, to complement and refine the application's appearance and functionality.

* WebCenter Page Editor & Composer - with the page editor and composer, privileged users can make instant changes in the layout and composition of pages, including the addition or removal of components, manipulation of existing components and reorganizing components on a page; these facilities stretch so far that manipulation of skins, developer level properties and expressions and the run-time creation of data controls and data bound forms and reports is supported - all without downtime  
  
* ADF Change Persistence - ADF Faces has integrated change persistence facilities; these take care of recording selected changes in the layout and state of various standard UI components and can be leveraged by developers to also include aspects of run time page organization, validation, boilerplate text
* BI (Report) Composer - facilities to allow the creation and manipulation at run time of reports. BI Composer is a simple-to-use wizard that allows users to quickly and easily edit or view the design of Oracle BI EE analyses from a plain ADF or WebCenter Portal application or WebCenter Portal: Spaces. Publishing these reports does not require a formal software re-deployment that might incur downtime on the application as a whole.
* SOA Composer - this composer is used for run time manipulation of Domain Value Maps and Business Rules as well as some aspects of Human Task definitions; through the SOA Composer, the behavior of SOA Composite applications can be tuned at run time.   
  C:\data\SOA_BOOK\Part2_DevelopingCompositeApplications\Chapter8\figures_chapter8\854_EditDecisionTableInSOAComposer.tif  
  Any changes created and committed in the composer are effective immediately. No downtime is required: change, save, done.  
  
* BPM Process Composer - this composer product is used to publish, review and refine BPM process definitions. Unlike the composer products mentioned before, it does not directly impact the run-time definitions of business processes.   
    
  Changes created through this composer are saved to MDS, but require an explicit deployment step to get published to the live environment. This deployment step is a potential cause for temporary unavailability.

More composer products are on their way from the Oracle FMW product development teams, probably to be released in the 12c release of FMW. These tools will help with even more advanced run time editing, for example of ADF artifacts like business components, data controls and task flows.

Note that the run time modifications supported through the Page Editor & Composer and ADF Change Persistence can be further specialized through the customization mechanism: the run time manipulations can be associated with specific user groups, regions, time slots or other conditions to be established at run time. The business not only can apply changes more or less instantaneously, they can also target these changes at carefully defined groups of end users. This of course means that the impact of changes applied at run time can be limited to a very carefully selected user community. As a special trick: we can define a 'version' customization layer.

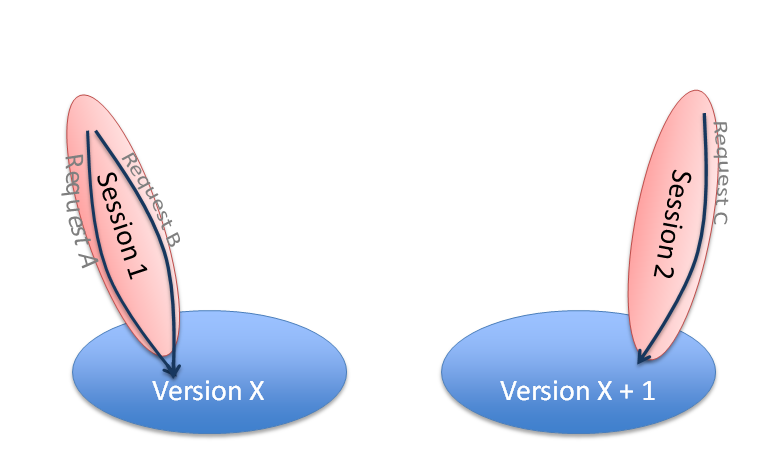


Run time changes could be associated with a specific version value in this layer. Users or user groups can be associated with the latest version from which they should see the changes. The illustration shows the case where the current user session is exposed to all changes associated with version 2 and 3 (but not those from version 4) - in addition to departmental and regional customizations that are not linked to specific versions.

***Multiple Production Versions***

After having reduced our challenge of zero-downtime application upgrades substantially in the previous section, by introducing the extended concept of run-time manipulation to replace a large portion of design time development effort followed by a formal deployment step, we will now turn to those changes that can not be realized at run time. For these changes, redeployment of the application is unavoidable.

In the theoretical case of instantaneous responses from applications, we could argue that the roll out of a new version of an application is akin to the flip-of-the-switch: request A is handled by version X, the switch is flipped to bring version X+1 live and the next request B is taken care of by the new version. This, however, is not the reality. Requests take longer to process than no time at all. Besides, requests live in the context of a session - this applies to database as well as application server. This means that replacing version X with a new version X+1 in a throw-the-switch manner would not only potentially interrupt pending non-zero duration requests but would also - and far more harmfully - destroy all pending sessions. This would definitely not constitute zero-downtime application upgrade.

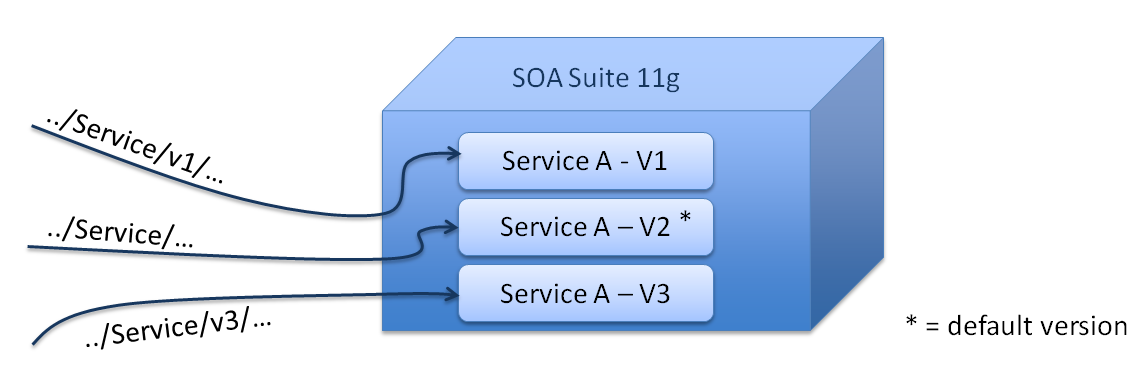


In order to not lose pending interactions - requests, sessions or running BPEL or BPM process instances - during application upgrades, there clearly is a need for the (temporary) co-existence of two versions of applications - the old one with potentially the pending conversations and the new one to deal with all newly initiated conversations.

Various products in Fusion Middleware have somewhat different ways of supporting this multi-version situation. We will look at SOA Suite for SCA composite applications and WebLogic for Web Services and Web applications (including ADF and WebCenter Portal applications).

*BPM and SOA Suite with concurrent versions*

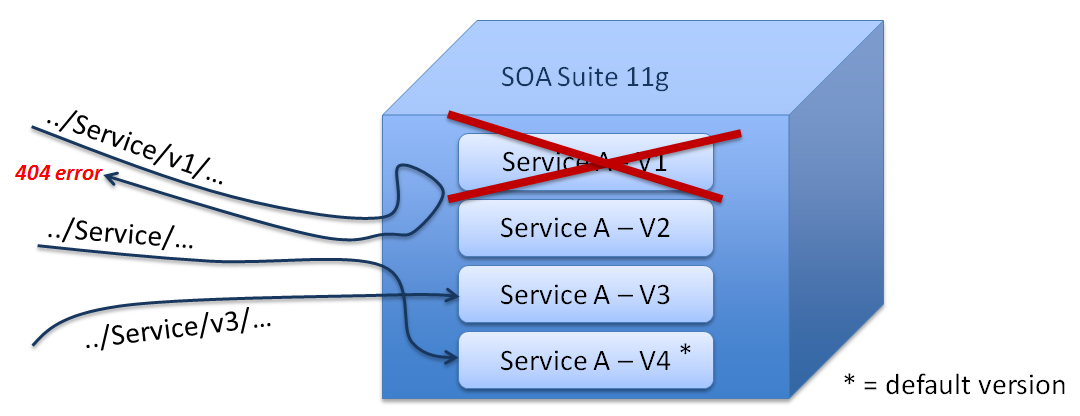
A request to the SOA Suite may result in a (really) long running BPEL or BPM process. That is one reason why the SOA Suite supports multiple versions of service definitions in parallel. Version 1 of some SOA composite application (exposed as service A) can be quite old - yet it may still correspond to running or past instances and therefore still be meaningful. It may even still be invoked by consumers who for some reason have not been able to migrate to the more recent versions V2 or V3. See the next illustration for this situation.



When a composite application is redeployed, it can overwrite an existing (version of that) composite application or it can be created as a new version, next to other versions. When a composite overwrites an existing version, all current instances associated with that version are considered *stale* from that moment on. This means that their execution is aborted.

One version of every composite application can be designated the default version. Any request to the service exposed by the composite application that does not refer to a specific version is routed to this default version. Note that typically most service calls will be without version indication and will therefore end up in the current default version. In the previous illustration, the default version is V2. The request to the service end point url without version indication is routed to this default version - unbeknownst to the consumer.

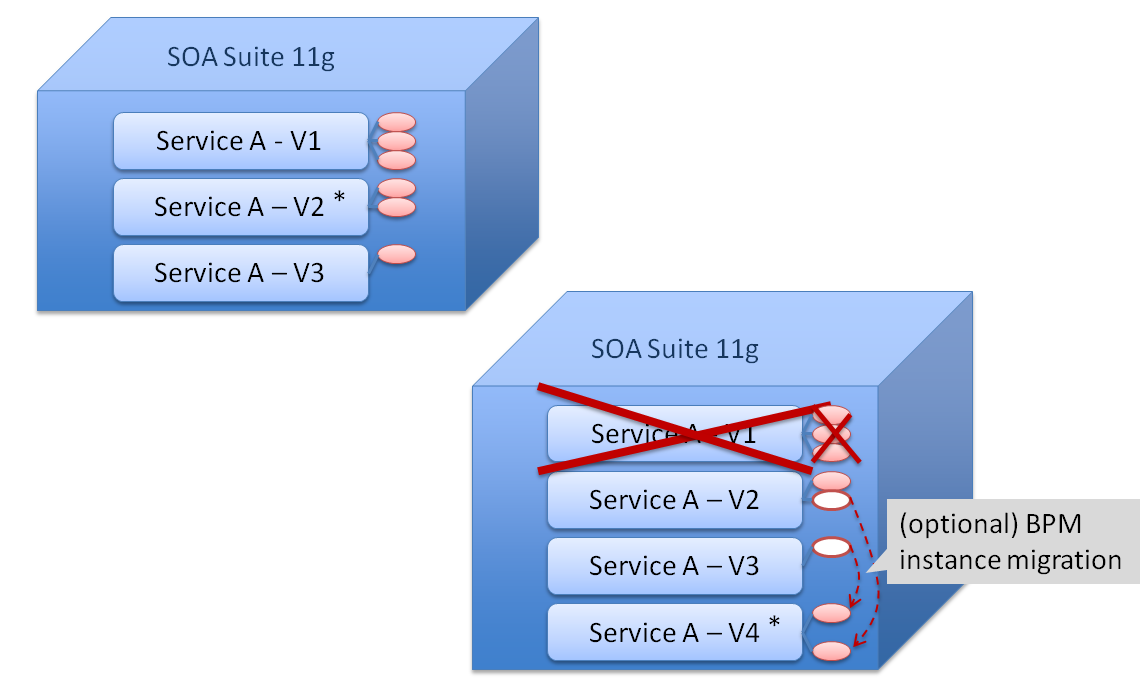
The next illustration shows the situation after Service A V1 has been discarded and V4 of Service A has been deployed - and that version has been designated the default version.



The Service End Point URL including the V1 indication will now result in an error - that end point does no longer exist. The URL without version indication - the one always resolved to the default version of the service - is routed to the newly deployed V4 version of Service A. Note that the client using that URL does not even need to be aware of the fact that the request is now handled by a different service version - as long as the contracts are the same between the versions. The URL with V3 designation is handled as before by V3 of the composite application.

Clearly this multi version mechanism of the SOA Suite makes it possible to deploy a new version of a composite application without interrupting existing conversations with the current or previous versions of the application. Through the use of the *default* indication - which can be set during deployment or at some other time - the administrator can control which version of the composite application is the 'live' one (presuming that most consumers will not include a specific version indication in their service calls).

The next illustration shows instances for each of the versions V1, V2 and V3 of the composite application. When a version is overridden - the corresponding instances are aborted (if running) and turned to the status *stale*. The latest release for BPM introduced an advanced feature: instance migration. This feature allows existing instances of an older version of a BPM process definition to be upgraded - under certain conditions- to the new version of that same process. That would allow already running process instances to behave for the remainder of their duration according to the newly deployed version of the process - the version that was deployed after the instances were initiated.

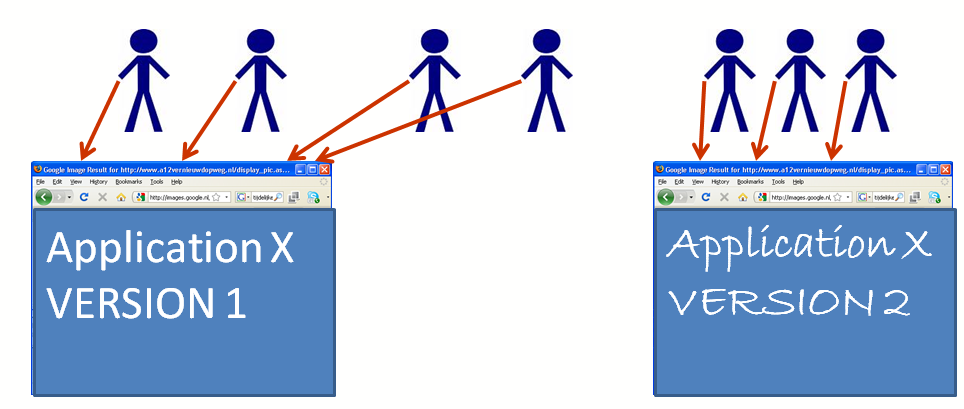


Unfortunately this instance migration functionality is not (and apparently will not) be available for long running BPEL process instances. For these instances there is no remedy than have them run their course according to the old version's definition. A best practice: avoid long waits in BPEL processes. If they occur according to the process logic, than split the BPEL process at that point. This decreases the burden with new versions of the process definition as well as upgrades of the SOA Suite infrastructure itself.

*WebLogic Production Redeployment for Web Applications and Web Services*

WebLogic Server is used to deploy and run Web applications - such as ADF and WebCenter Portal applications - as well as Web Services. For all of these, WebLogic supports a mechanism called *production redeployment*. This too is a method for (temporary) multi-version support.

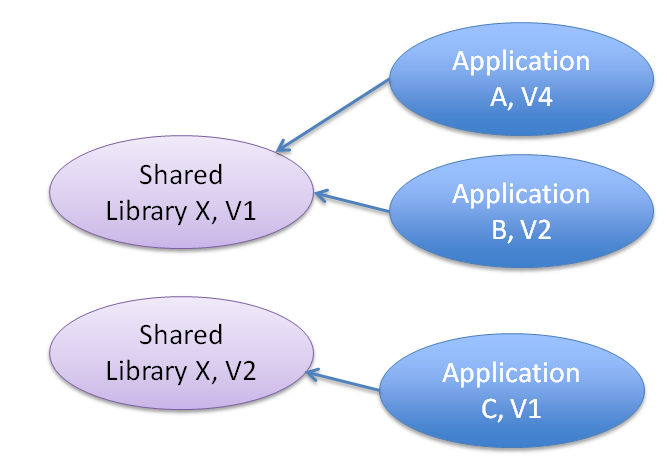
Suppose we have an Application called X. It has been deployed some time ago and is being used quite heavily. A new version of this application has been developed. If the application X has version information included - the WebLogic-Application-Version property in the file ApplicationRoot\meta-inf\MANIFEST.MF has been set - it qualifies for production redeployment. This means that when the new version of Application X is deployed as a redeployment of Application X, WebLogic ensures that for a limited period of time it will have both version 1 and version 2 of the application running, side by side. Version 2 is used for all new sessions. Version 1 is kept around for existing sessions. Only requests in the context of sessions that were created against version 1, before version 2 had been deployed, will be serviced by version 1. The old version is either kept around for a fixed period of time or until all sessions have been concluded. The next illustration shows the situation some time after the new version has been *production redeployed*: some new sessions have been created on version 2 and some pre-existing sessions are still connected to version 1.



In this scenario, none of the users suffers unavailability of the application. Either the user starts a new session on version 2 or the user's session continues running on version 1. No planned downtime!

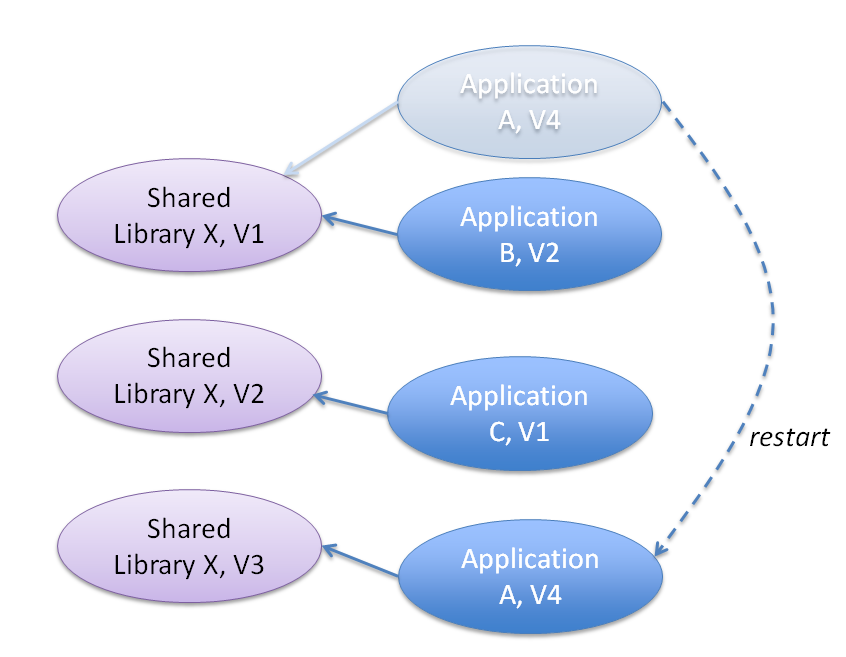
*Multi-version support for Shared Libraries*

WebLogic supports *Shared Libraries*. These libraries are deployments of EAR, WAR or JAR files with JEE resources. Libraries do not expose invokable elements - such as web pages or web services. However, all resources in a shared library can be used by real applications that expose such elements. Any shared library can be used by multiple applications.



WebLogic supports multiple versions of a shared library, coexisting in the same domain. One application may use one version of the shared library, while two other applications use a later version of the shared library. Applications stipulate in a deployment descriptor which version (or just the latest one available) of the shared library they want to hook up with.

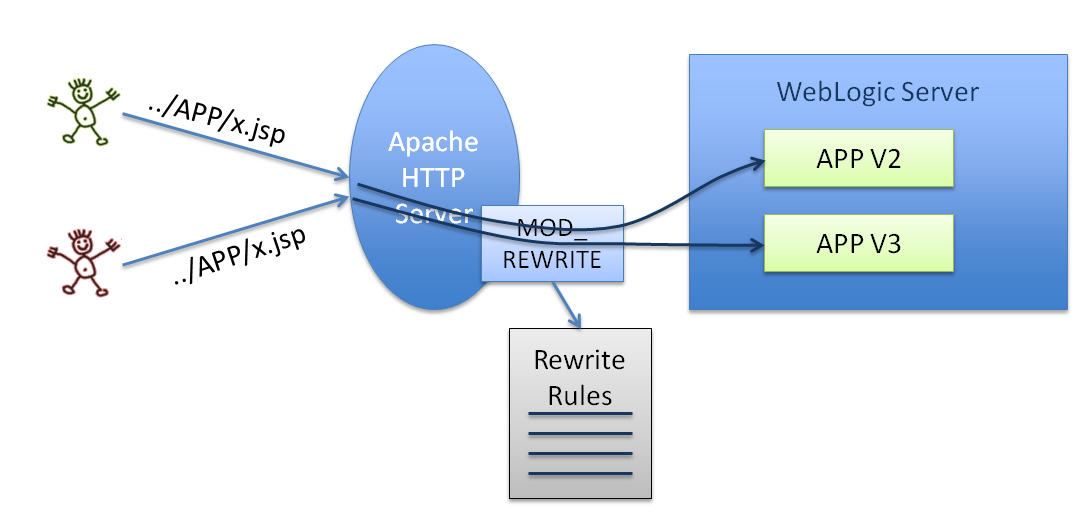
This may seem like a good way for deploying new versions of specific resources such as JSP files, images or Java Classes. Since multiple versions can co-exist, deployment of a new version of the library should not cause any interruption of service. And indeed, that is true. However, the resources in the shared library are only introduced into running applications with a dependency on the shared library after a stop and restart of those applications. And that of course means temporary unavailability of the application and the loss of existing sessions. Probably shorter than for a redeployment of the application as an alternative way to introduce new versions of resources, but still not ideal for zero downtime application upgrade.



In order to have the application adopt the new version of the shared library without loss of service, the application should be 'production redeployed' as described above, even if it does not contain any changes itself - other than a slightly higher version number. The "new version" of the redeployed application will use the latest artifacts from the latest version of the shared library.

*URL rewriting*

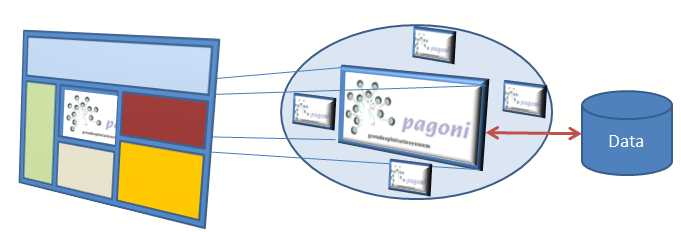
Prolonged use of parallel multiple versions is not supported by WebLogic's Production Redeployment feature: all new sessions will be initiated on the new version, if you like it or not. If you do not, there is another option - which requires a little more work on your part but offers more flexibility through the use of URL rewriting. For example using Apache HTTP server in front of WebLogic and then leveraging its mod\_rewrite plugin, you can reroute requests for http://domain:port/application/path?someOptionalParameters to the version specific application URL such as http://domain:port/application\_v2/path?someOptionalParameters.



The actual rewrite logic is written in instructions in a configuration file. When the file is changed, the modifications are automatically reloaded so a server restart is not required. Note: the rewrite instructions can be quite sophisticated, for example looking at the client from which the request originates in order to determine to which URL to forward the request and checking whether the request is one initiating or continuing a session.

*Portlets*

Portlets are basically stand-alone web applications. As such, everything said about production redeployment on WebLogic of Web Services, Web applications and ADF applications should apply to Portlets on WebLogic as well. This means that for a short time after deployment, existing sessions can continue to use the old version of the portlet while new sessions will use the new version.



Even without production redeployment, the effects on the consuming applications of redeployment of a portlet are typically somewhat more limited than with full application redeployment, thanks to the loosely coupling that exists between the consuming application and the portlet. While redeploying the new portlet version , part of the consuming page - the portlet frame- may temporarily be unavailable (and existing sessions can be disrupted) but application on the whole is not affected. Depending on the nature of the portlet, the user - and the SLA - may not consider this a loss of availability of the application.

*Oracle Service Bus*

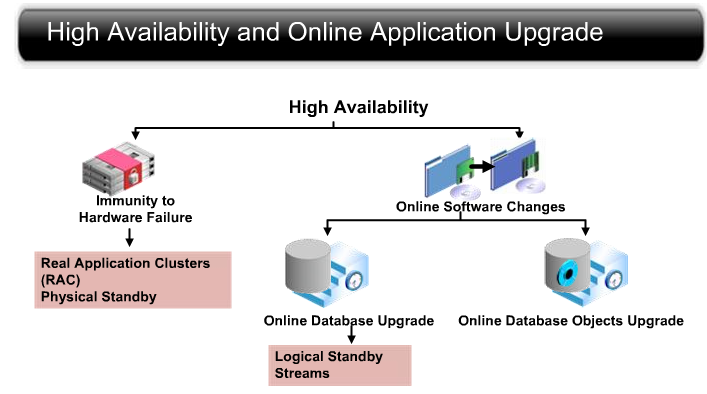
Oracle Service Bus does not have a multi-version concept for services. Given the fact that OSB interactions are stateless and (therefore) short-running, support for multiple versions is not deemed necessary. It does have a mechanism to prevent the service from appearing unavailable during redeployment. This mechanism involves completing running requests prior to and queuing up new requests during redeployment. To achieve multiple versions of services, these would have to be deployed as different proxy services. Through the Service Registry or some other form of end point virtualization, applications can be directed to the version they require - similar to the URL rewriting for web applications discussed above.

***Upgrading the Database objects***

So far this article discussed the upgrade to later versions of middleware artifacts like web applications, web services and SOA Composite applications with BPEL or BPM processes. It has not yet mentioned the database, yet most applications in our corner of the world rely on database objects such as tables, views and PL/SQL packages. A new version of the middleware artifacts may very well have a need for new versions of database objects as well. Web pages may rely on additional or modified columns in views or tables and Composite applications may need to invoke new procedures or functions in the PL/SQL APIs of the application. Therefore, no discussion of smart application upgrading aiming for zero-down time can be complete without taking the database into consideration.

Oracle followed this same reasoning, when considering smarter upgrades for Fusion Applications. Downtime for enterprise systems because of application upgrades are becoming increasingly unacceptable to organizations operating globally in a 24/7 economy. Oracle did a lot of work on the database to reduce planned downtime because of infrastructure upgrades - RAC rolling upgrade and hot patching, use of Data Guard or Golden Gate - and through the KSplice acquisition it also targets the Linux Operating System.

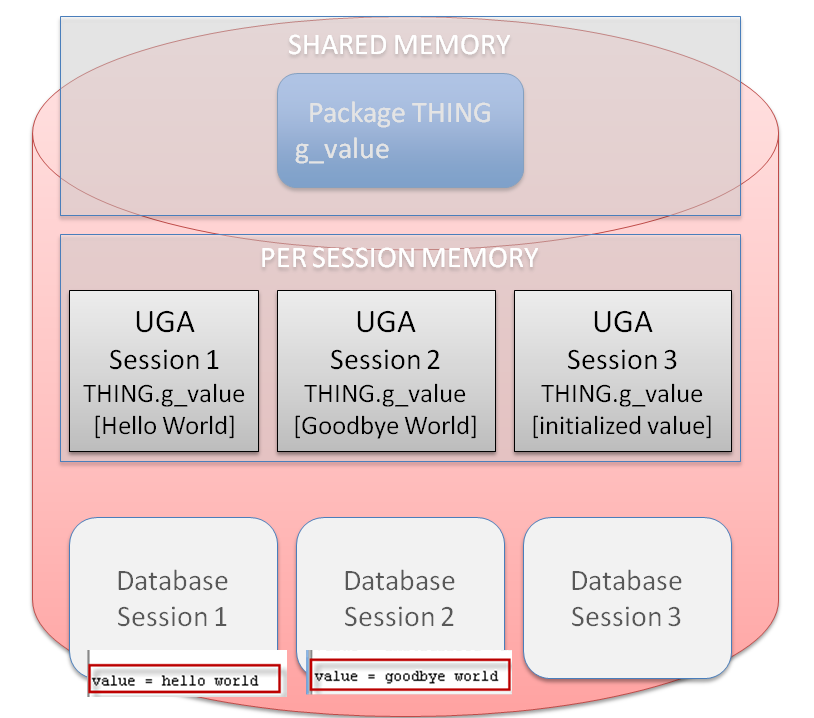
The next figure illustrates Oracle's approach to maximum availability - catering for both unplanned and planned downtime , the latter consisting of system maintenance and application upgrades.



In various database releases, Oracle did a lot of work in supporting online redefinition of database objects, striving to a reduction of the impact of changes both in terms of unavailability (through locks) and invalidation of dependent objects. When a table is redefined online, it is accessible to both queries and DML during much of the redefinition process. The table is locked in the exclusive mode only during a very small window that is independent of the size of the table and complexity of the redefinition, and that is completely transparent to users. Through fine grained dependency tracking, the number of occasions on which dependent objects are rendered invalid is reduced to those situations where it is really justified. In database releases prior to 11g, object dependencies were managed at the object level, so altering an object automatically invalidated all dependent objects. Oracle 11g has more granular dependency management, so only changes that directly affect an object will cause an invalidation.

Additionally, the frequency of the appearance of ORA-04068 (existing state of packages has been discarded) was reduced. This message prior to Oracle Database 10g was shown for calls to a package in sessions that already invoked that package whenever that package had been recompiled or when objects it depends on had been recompiled .

Note however that there are still situations where the message will be shown, and correctly so. When state is associated with a package - through the use of global variables in the package - and the package is recompiled itself, because a new version of the package is introduced , the state of the package is lost for all sessions that had created such state.



The figure overhead shows a package THING with a global variable g\_value. Three different database sessions have each accessed the package and thereby created associated state for the package. This state is stored in the UGA - global area per session. When package THING is recompiled, the associated state for the three database sessions is wiped out. Their highly useful values of *hello world* and *goodbye world* are lost.

Sessions may not have availability issues with the (introduction of the) new version of the package. However, the loss of state because of the upgrade may result in service interruptions. This, combined with the fact that database connection pools such as used from FMW applications do not go together well with session state kept in packages, leads to a suggested practice regarding session related state in the database: use an application context to store such state, rather than global variables in packages.

So instead of:

package body MY\_PACK

g\_important\_global varchar2(1000);

use something like:

package body MY\_PACK

function g\_important\_global

return varchar2

is

begin

   return sys\_context('MY\_PACK\_CONTEXT', ' g\_important\_global');

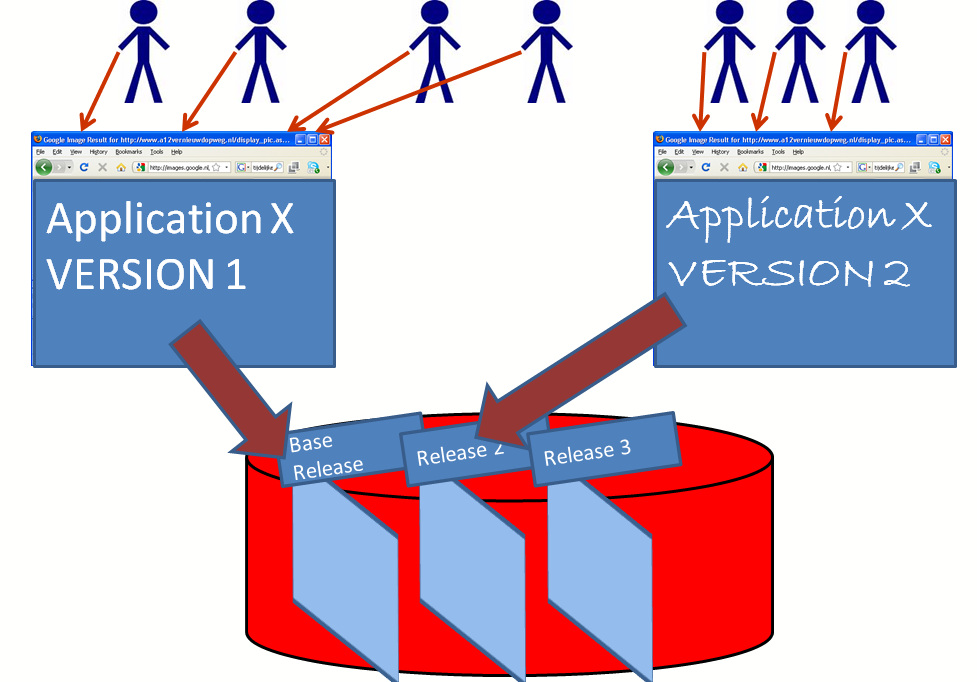
end;

When the package is now recompiled, its state is preserved because its state if stored in an application context that is not affected by the recompilation.

Of course introducing this approach would mean changing quite a bit of existing code in all cases where the global is set/modified. We also need to cater for the fact that application context only stores varchar2 data. However, when uninterrupted service to the users of the application is essential, it is worth considering. And, on the upside there is an interesting bonus in the form of the use of Client Session Based Application Context - where the value of CLIENT\_IDENTIFIER is used to link a session to the state in a particular Application Context. This introduces a way to connect sessions on the middle tier to their database held state.

***Edition Based Redefinition***

For the 11g release of the database Oracle went beyond the reduction of downtime and the ability to redefine database objects with a minimum of unavailability. It implemented in the database a mechanism that is very similar to WebLogic's production redeployment: the ability to have two versions of the application (or in this case: a coherent set of database objects) live at the same time, the old version servicing longer running sessions that started before the new version went live. This mechanism is called Edition Based Redefinition. It is illustrated in the next figure: the two versions 1 and 2 of Application X - temporarily running in parallel on WebLogic just after production redeployment - have need different versions of the database objects. Version 1 relies on the base release of the database objects while version 2 of the application uses the versions of database objects shipped in Release 2, complemented with the objects from version 1 that were not changed for version 2.



The notion of a 'release of database objects' was introduced in Oracle Database 11gR2 and is called 'an edition'. Editions are almost parallel universes in which the same database objects - such as views, packages, types - from the same schema can exist in with different definitions. For example: package SALARY\_MANAGEMENT in Schema SCOTT can exist multiple times - for example once in the Base Edition, once in Edition Release 2 and once in Edition Release 4.

Through Edition Based Redefinition (EBR), we can build up the database objects for a new release in its own *edition* in parallel with and insulated from the versions of those objects that are currently live. Only when we make the new edition available for new sessions created against the database will the new versions of the database objects get accessed. The edition to use is determined at session level, using a simple ALTER SESSION statement that can for example be executed in a LOGON trigger. Obviously, it is important to synchronize the version of the application with the edition in the database that contains the corresponding versions of database objects. Note: a default edition is always indicated; any session not explicitly setting the target edition will use this default.

In the example shown in the figure above, we can build instructions into application X to explicitly set the database edition after establishing the connection to the database. Version 1 would set the Base Edition and version 2 would set the Release 2 Edition. Alternatively, a trick I heard from the product manager for EBR (Bryn Llewellyn), the applications can make use of different JDBC Data Sources on the application server that connect through different database services to the same database schema. A LOGON trigger detects the name of the database service used to connect and selects the session's edition accordingly.

Edition Based Redefinition was introduced in 11gR2 and will be further refined in the upcoming 12c release. It is a mechanism included with all editions (SE One, SE, XE, EE) of the database. It allows a new version of the database side of applications to be deployed, compiled, tested in isolation - without any impact on the live environment. Then this new version can be exposed to [selected] new database sessions. Unlike production redeployment on WebLogic, it is possible to keep various editions around for a long time - not only until all current sessions have completed. It is also possible to have more than two editions; in fact I am not sure there even is a limited to the supported number of editions. Depending on the logic that governs the selection of the edition for a session, many different edition can be in use for different users and their sessions at the same time.

Using EBR, zero downtime upgrade of the database objects that go with FMW applications has become possible. No more planned unavailability need to be suffered because of the compilation of changed packages or the invalid status somewhere halfway during the execution of DDL scripts upgrading the application's database schema.

## Conclusion

Availability is a highly desirable aspect of Fusion Middleware application. Planned downtime therefore sounds quite silly. However, the introduction of changes to applications has typically been associated with the roll out and redeployment of the application causing disruption of service - through loss of existing conversations and unavailability during the deployment process. This article has discussed a number of mechanisms that will help substantially reduce the loss of uptime because of changes to applications.

First of all, many types of changes may be implemented through run time configuration, rather than software change requiring redeployment of application artifacts. This approach requires good design of aspects of applications that may require modification and the construction of run time mechanisms for applying and processing such changes. The design time at run time capabilities of several Fusion Middleware products go a long way in supporting this approach.

Additionally, several FMW components support short or even long term co-existence of different versions of applications. This means that existing conversations can complete even when a new version has been made available. It also means that there does not need to be any time during deployment at which the application is not available. BPM and SOA Suite have multi versioning capabilities that allow prolonged co-existence of two or many more versions of composite applications. BPM also supports instance migration - allowing running process instances to be upgraded to a new process definition. WebLogic supports *production redeployment*, which allows Web applications and Web Services to temporarily - just until all current sessions have been taken care - have two co-existing versions.

Keeping in step with the middleware's capabilities for downtime free application upgrades, Oracle has introduced Edition Based Redefinition in the 11gR2 release of the database. This mechanism too supports prolonged co-existence of different versions of (the database objects supporting) applications. By associating specific versions of middleware application components with specific editions in the database, it becomes straightforward to have parallel versions of an application and consequently interruption free upgrades.

# Resources:

Fusion Column: Design Time at Run Time – Power to the end user, Lucas Jellema in ODTUG Technical Journal, Q2 2011 - http://technology.amis.nl/wp-content/uploads/images/Design-Time-at-Run-Time\_ODTUGTechnicalJournal\_Q2\_LJ.pdf

Instant Agility through Design Time at Run Time - Lucas Jellema, Oracle Open World 2011 - Presentation on Slideshare - http://www.slideshare.net/lucasjellema/instant-agility-in-oracle-fusion-middleware-through-design-time-run-time-oracle-open-world-2011

# James Smith's blog article BPM Composer – Runtime Edit -http://jamessmith73.wordpress.com/oracle-fusion-middleware/oracle-soa-bpm-11g-blogs/oracle-soa-bpm-11g/bpm-composer-runtime-edit/

Manage Agility through Manageability -Lucas Jellema, Introducing Design Time at Run Time - Oracle User Group Holland 2011 - Presentation on Slideshare- http://www.slideshare.net/lucasjellema/manage-agility-through-manageability-introducing-design-time-at-run-time-in-oracle-fusion-middleware

# WebCenter 11gR1 PS3 – Design Time at Run Time with a Vengeance – introducing run time Data Controls and Data Visualizations - Lucas Jellema on AMIS Technology Blog - http://technology.amis.nl/2011/01/16/webcenter-11gr1-ps3-design-time-at-run-time-with-a-vengeance-introducing-run-time-data-controls-and-data-visualizations/

## [Production Redeployment , WebLogic Versioning](http://middlewaremagic.com/weblogic/?p=394) by [JaySenSharma](http://middlewaremagic.com/weblogic/?author=1) on Middleware Magic blog - http://middlewaremagic.com/weblogic/?p=394

Introduction to URL Rewriting with Apache MOD\_REWRITE - blog article by Ross Shannon - http://www.yourhtmlsource.com/sitemanagement/urlrewriting.html

On Signatures and Changing WHERE Steven Feuerstein - Oracle Magazine, September/October 2008 - http://www.oracle.com/technetwork/issue-archive/2008/08-sep/o58plsql-099029.html

Continuous Database Application Evolution with Oracle Database 11gR2 - Lucas Jellema, ODTUG Kaleidoscope 2009 - Presentation on Edition Based Redefinition - http://www.slideshare.net/lucasjellema/edition-based-redefinition-continuous-database-application-evolution-with-oracle-database-11g-release-2

Using Data Guard for minimizing downtime through database upgrades -Oracle technical marketing presentation http://www.oracle.com/technetwork/database/features/availability/311390-133499.pdf

Maximum Availability Architecture (MAA) Best Practices for Planned Maintenance: Online Patching and Rolling Upgrades with Oracle Database -http://www.oracle.com/technetwork/database/features/availability/maa-planmaint-131050.pdf

An Oracle White paper: Zero-Downtime Database Upgrades Using Oracle GoldenGate http://www.oracle.com/technetwork/middleware/goldengate/overview/ggzerodowntimedatabaseupgrades-174928.pdf

Introducing Application Context - AMIS presentation - http://www.slideshare.net/lucasjellema/introducing-application-context-from-the-plsql-potpourri-presentation

***Oracle Documentation:***

# Fusion Middleware Administrator's Guide for Oracle WebCenter 11g Release 1 (11.1.1.5.0) - Using WebCenter Portal Administration Console - managing portlet producers - http://docs.oracle.com/cd/E21764\_01/webcenter.1111/e12405/wcadm\_admin\_apps.htm#CFAFAIDH

# Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite 11g Release 1 (11.1.1.5.0)- Managing SOA Composite Applications - http://docs.oracle.com/cd/E25054\_01/admin.1111/e10226/soacompapp\_mang.htm

## Fusion Middleware User's Guide for Oracle Business Intelligence Enterprise Edition 11g Release 1 (11.1.1) - Using BI Composer to Work with Analyses - http://docs.oracle.com/cd/E23943\_01/bi.1111/e10544/bicomposer.htm#BIEUG11325; also: Adding BI Composer to a WebCenter Portal Application or to WebCenter Portal: Spaces -(http://docs.oracle.com/cd/E23943\_01/bi.1111/e10545/integrating\_center.htm#BIEDV3190) and Adding BI Composer to an ADF Application - (http://docs.oracle.com/cd/E23943\_01/bi.1111/e10545/bi\_composer\_adf.htm#BIEDV3201)

WebLogic Server 12c - Production Redeployment and Web Services - http://docs.oracle.com/cd/E24329\_01/web.1211/e24964/client.htm#i227120

WebLogic 12c Shared Libraries - http://docs.oracle.com/cd/E24329\_01/web.1211/e24368/libraries.htm

Oracle 11gR2 On Line Documentation on Edition Based Redefinition - http://download.oracle.com/docs/cd/E11882\_01/appdev.112/e10471/adfns\_editions.htm