Active Data Services: Real time data change notification in ADF Faces

Frank Nimphius
Oracle Corporation

Schlüsselworte:

Ajax, ADF, JSF, Push

Introduction

Did you know that the rich Internet is a member of the same disconnected web that serves traditional web application well for so many years? Despite of all the UI richness, communication in Web 2.0 is still based on the Http protocol that sends a server response in return to a preceding client request. What about Ajax you ask? No, Ajax did not change this at all. It only did change the user perception by hiding the request and response cycle from the end user's eye, pretending instant responses. However, to accomplish the mission to replace desktop applications, rich web applications not only need look like desktop applications, the need to behave like desktops. This implies the ability to server change information from the server to the client without the client asking for it.

Ajax-push, Comet, streaming, poll, long-poll and web sockets are some of the technology keywords used by the Ajax community to describe the technical implementation of the reverse web in which the server updates the client with data changes as they occur. A common requirement of Oracle customers for example is to be notified about database changes performed by other users. For this, a change event needs to be broadcasted from PLSQL to Java and from the middle tier to the web client. Using the Oracle RDBMS, data change notification can be received in Java using the Oracle 11g JDBC thin driver. This however only makes the middleware aware of the data change but does not yet notify the client. Like in mobile telecommunications of the late 90ies, the "last mile" to the client is the hardest to do. To update clients with server side changes in considerable close to real time, rich Internet application developers implement one or more of the acronyms listed above on top of the Http protocol, using manual coding or the help of development frameworks.

Active Data Services (ADS) is a framework in Oracle ADF Faces that is designed for pushing server side changes to the client, without locking the developer in to a chosen approach.

Real time: Mastering acronyms and their meaning

To web application users it must appear as if the web reinvents itself once a year. The results are more interactive user interfaces, increasing performance through partial page refreshes and a desktop like usability patterns that allow users to become as productive using web applications then using real desktop clients. One technical detail though hasn’t changed in the past and is unlikely to change in the near future; the http protocol. The http protocol is based on the request - response principle in which the client sends a query to the server and the server responds with the requested data. Between requests, there is no connection maintained between the client and the server that would allow server side logic to sent more data unasked. If at all, changes in the underlying data layer used by an application are first detected within the next client request. Hopefully it doesn’t come to you by surprise when we state that in the modern time of Web 2.0 and Rich Enterprise Applications (REA)

the web still is disconnected and stateless. You may object saying that there are applications on the web that you frequently use and that update their client UI with server side changes without you doing anything. And right you are. There is truth in both, our statement and your observation. The question therefore is how they did it and if you can do the same with ADF Faces RC. In the following, we'll have a look at the options that are available in Ajax and other implementation technologies of Rich Internet Applications (RIA) to implement automatic UI refreshes.

By chance

The "by chance" approach queues a message on the server and waits for the next client request to apply the content changes to the rendered parts of the UI. This implementation does not qualify for any real time messaging but is good to use when model data changes between user requests. An example for this is dependent lists, in which a value change in the parent list refreshes the choice in the detail list. If well implemented, then the user query does not need to explicitly query the changed data to mark the associated user interface components as a refresh candidate.

Polling

Using polling, the client frequently sends a client request to the server to check for updates. Unlike user client requests, a poll is initiated by the client and usually is configurable for the application developer. The challenge using polling is to find the best balance between the poll frequency and network capabilities. The maximum waiting time for a server side change before it is sent to the client is the duration between two polls. This mechanism is good for model changes that are predictable and that don't need to be displayed immediately or at least close to when they occur. Polling helps where there is no server side support for advanced techniques. However, it tends to suffer from network latency and blocking-IO handling of the servers.

Long-polling

Like polling, the initial request comes from the client. Instead of polling, using long-polling, the server response is not immediately following the request and instead waits for a server side event, which is why it is called "long-polling". If an event occurs on the server, then the response is sent to the client and the UI is updated. After completing the long poll, a new request is sent from the client to the server and again the response waits for an event to come. Long-polling has its maximum delay for server side events that occur directly after a response is sent to the client as it takes a bit of time for the client to send the next request. Long-polling is a good choice for different network topologies including the use of proxies. You need to be aware though that long-polling uses two connections, one for the push notification and one for the data. The way that long polling is implemented determines the number of opened connections to the server. In praxis though, most solutions for active data support provide channel sharing t reduce the number of opened server connections.

Push

Push, also known as http streaming, starts with a client request opening a connection to the server. With a connection open, the server is able to immediately notify the client about server side changes. To keep the connection alive, the server sends partial responses to the client, which are empty if no server side update is available or contain a message payload if a UI change is required.
Push is more immediate than poll and long poll and suitable for the impatient usecases in which the UI needs to be refreshed as close as possible to a server side change event. Like in long-polling, two

separate connections are used for data and event notification. Push may not work well if older proxies are involved that cache the http content, in which case long-polling could be used as a fallback.

Introduction to ADF Faces Active Data Services (ADS)

In ADF applications you use ADS to build dashboard style read only UIs that actively notify users about server side data changes. Since data updates are performed on the UI component rendering only, you cannot use ADS on UI input components. To add ADS functionality to their application development, application developers implement the `ActiveDataModel` interface or use the Active Data Services proxy framework which decorates the component model to provide active data changes.

ADS provide developers with a configurable choice of notification types they can use to push server side events to the client. The supported notification types in ADS are polling, long-polling and push. In addition, using the ADF Faces RC auto-ppr functionality, the mechanism we described as "by chance" and that will do for most of the usecases is implemented too. The selling point of ADS is its flexibility, the ability to decide for a notification type by configuration and not through hard coding it into an application.

Using ADS in Oracle ADF Faces applications

From an active services perspective, there are two options available for applications to bind active business services to ADS to refresh the UI in response to a change event.

The first option is to use the Active Data Model proxy framework in ADS to decorate the JSF component model. In this scenario data is directly passed on to the ADF Faces components model, without using the ADF binding layer.

The second option is to use the ADF binding layer. As the time of writing, there exist is only one Data Control, which is for Business Activity Monitoring (BAM), with integrated ADS functionality. To bind active services to ADS using the ADF binding layer you may use the POJO (Plain Old Java Objects) Data Control in combination with the Active Data Model proxy. As the time of writing, there is no direct support for ADS push in ADF Business Components (ADF BC). Therefore, Best practice for updating the user interface in response to change events in ADF BC is to use a mix of auto-PPR and poll.

No matter which approach you choose, Active Data Services is defined on the ADF Faces component model, implementing the `ActiveDataModel` interface.

Shown in figure 1, ADF bound UIs have the active data model implemented in the ADF Faces specific ADF binding classes. ADF Faces UIs that are not bound to ADF need to implement ADS in their component model, which using the ADS proxy is the recommended approach for application developers.

The ADS proxy saves developers from implementing the ADS interfaces by decorating the ADF Faces component model with ADS functionality. It delegates active data handling to a convenience implementation of the `ActiveDataModel` interface which listens to data change events from the data layer and interacts with the Event Manager. Using the ADS proxy, developers enable existing JSF component models for active data, which also means that application development may start without ADS in the picture and then to a later point in time activate it.

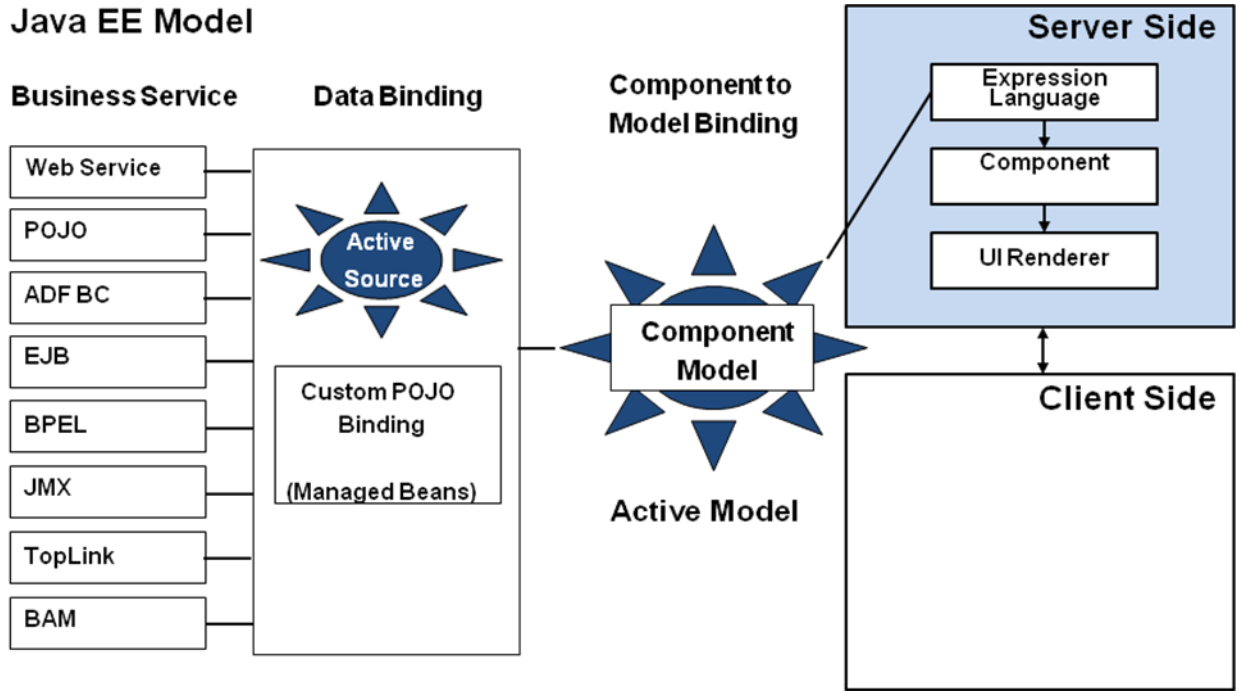

Figure 1: JSF component model for ADF Faces active components

Note: ADS is designed to work with ADF bound and non ADF bound ADF Faces component models. It does not require the ADF binding layer to be present.

Implementing Database update notification using ADF BC

In ADF Business Components, the use case of updating the UI based on database updates can be declaratively implemented in a mix of auto-PPR and client side polling, using the `af:poll` component.

The auto-PPR functionality queues a change request for components in the ADF Faces context but requires a client request to process the UI refresh, which is what the `af:poll` component is used for. In the example in this section, a View Object that is exposed on a shared Application Module in ADF Business Components is used to listen for database table changes to update a list of values. The changes that are detected include the addition and removal of list objects.

The difference of this approach, compared to using the code centric integration with the ADS proxy, is that it is a declarative approach that uses auto-PPR instead of Http streaming.

To implement the database change event notification use case using auto-ppr, it is important to understand the role of shared Application Modules in ADF Business Components. Not all data that are displayed in business applications change frequently. For better performance, this data could be cached and shared across applications or within a session. Use cases in which such data would be used are lookups like LOVs. To define a shared Application Module in ADF Business Components, open the ADF BC project properties and expand the Business Components | Application Module Instances node. You use session sharing for lookup data that is instance specific to the user session. Otherwise, share data application wide.

Note: Working with shared Application Modules in ADF Business Components is fully explained in the “Sharing Application Module View Instances” chapter of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework 11g*.

To push database change notification updates to the user interface, you need to share the Application Module in application scope. This way, the data of View Object instances that are exposed on the shared Application Module become accessible to all user sessions.

Kontaktadresse:

Frank Nimphius
Oracle Corporation

E-Mail frank.nimphius@oracle.com
Internet: www.oracle.com
www.oracle.com/technetwork/developer-tools/adf/learnmore/index-101235.html