How to Design Applications That Can Run Connected or Disconnected in a Seamless Manner

Ashok Malhotra (ashok.malhotra@oracle.com)

A desirable characteristic for applications, especially on devices with intermittent connectivity, is the ability to run equally well in a connected or disconnected state. This document analyzes best practices and makes architectural recommendations on how to design such applications.
This is an early draft and may have serious holes. There is also a great deal of detail that needs to be added. Please comment.

Consider an application that is installed on your laptop. This is not essential but makes the exposition simpler. The installed app contains what I call the Initial JavaScript and possibly other things. When the application is started, the Initial JavaScript first checks if the laptop is connected to the Web. It then looks at the resources the application requires and checks if it has access to a minimum set of resources. It may also have hints as to where to find these resources if disconnected: AppCache, IndexedDB or perhaps a special format used, for example, for calendars or a local SQL database. If the Initial JavaScript cannot find the minimum set of resources it cannot continue but if it finds enough resources it starts by invoking the Operational JavaScript which runs the app and must be included among the required resources.

As the application runs and the user interacts with it, it makes changes to its resources. If all the resources are available it may change them directly. If they are not, it changes the resources available locally and stores a local record which is used to update the Web resources when the laptop connects to the Web. This synchronization is perhaps the most complex part of this story. A proposal for how to do synchronization is described below.

Should locally Created Items be Identified by URIs?

If I create a new calendar appointment on my local calendar, the only one who can access this appointment if my local calendar application. Why does it need to have a URI? Also, if we give it a URI then what happens when the item is synchronized with the global calendar. Does the URI remain the same or does it change? There is potential for confusion in both cases. A simple solution could be to identify the local item with a GUID and give it a URI when it is synchronized.

I checked with Raman, and although he said he was not sure, he seemed to feel that locally created Gmail messages did not have URIs. They were assigned URIs when they were synchronized with the global Gmail database. I am trying to get in touch with Rich Miller, one of the editors of the Calendar API.

Synchronization

Consider a situation where the application requires three resources A, B and C. In the case we want to discuss, it finds A locally and starts running but as the user interacts with it, it eventually needs to make changes to all three resources A, B and C.

For each change that the application makes it writes a Change Record in local storage. Each change record has one part for each resource that associated with the application. In this case, each Change Record has three parts corresponding to resources A, B and C. Some of the Change Record parts may be empty.

Each Change Record part consists of instructions on how to update the resource it applies to. These instructions may be JavaScript or PL/SQL or an XQuery script or any other dialect that the application recognizes and can run.

It is necessary that the updates to the resources can be done in a transactional manner and the changes be visible while the transaction is open[1]. If the transaction aborts the changes disappear. When the application starts, it starts a transaction for resource A (and, in general, for all resources it has access to). As it proceeds, it writes Change Records and applies the parts for resource A (and, in general, for all resources it has access to). Consider a calendar application running locally. When the application starts, it starts a transaction for the local calendar. If the user schedules an appointment, the application updates the local calendar and writes a Change Record for the local calendar, the Web calendar and perhaps a departmental calendar i.e. It writes a change record for all the resources related to that application.

After some time, the user finishes with the application and closes it. The situation now is that his local calendar has been updated but the transaction is still open the changes are visible locally. There are change records for all three calendars in local storage and somewhere there is an event that says these Change Records need to be applied when the user connects.

When the user does connect and has access to all three resources, the open transaction on resource A (and all other resources that the application had access to) is rolled back. This is essential to ensure that all resources start in the same state. Then, the Change Record parts for each resource are gathered together. After this we have a pile of changes that need to be applied to resource A, a pile of changes that need to be applied to Resource B and a pile of changes that need to be applied to Resource C. Transactions are opened on all three resources and the changes are applied. If all goes well we do a two-phase commit [2] of the three resources. After this, all three resources are synchronized and are in a consistent state.

If Synchronization Fails

It is possible for the two-phase commit to fail, for example, if one of the resources loses connectivity. If this happens, the application needs to wait and retry. A more likely cause of synchronization failure is that one of the updates fails for semantic reasons. For example, the user may have scheduled an appointment on a day and time when some other user scheduled a non-breakable meeting.

The application needs to detect such semantic conflicts and send messages to the user informing her of the situation and guiding her towards possible resolutions.

In the worst case, if synchronization fails, all the transactions are aborted and all the resources are in the same state as they were before the application was run. This means that the user loses the work he has done.

Recommendations

Applications designed to run seamlessly in connected or disconnected mode must list the resources they require. Some of the resources can be marked essential and may have hints associated with them as to where to look for them in local storage.

All of the resources participating in the online/offline application must be capable of transactional updates. Moreover, the changes must be visible while the transaction is open. Finally, all the resources must be capable of participating in a two-phase commit protocol. See, for example the X/Open specification.

Explanation of Terms

[1] A transaction is a database capability that bundles several changes together and ensures that either all the changes are applied/committed or that none of the changes are committed. This ensures that the database will not be left in an inconsistent state with only some of the changes (say, the debits) are applied and some (say, the credits) not applied.

[2]Two-phase commit is a protocol that synchronizes transactions among multiple distributed databases. It ensures that all transactions on all participating databases are committed or none are committed. In effect, it is a meta-transaction.