Problem

During design of NSI CS version 2 the majority of error states were removed from the state machine and a new set of notifications were added to inform the uRA of issues with a reservation that did not directly effect the new state machine. Removing the majority of error states help simplify the state machine, and reduce the spontaneous state machine transitions previously triggered by error conditions.

As a result of this new strategy, there is no way for a uRA to determine if an error notification was generated against a reservation if that uRA is not listening to notifications on that reservation. This situation may occur if the RA is using a polling mechanism, or is recovering from a communications problem with the peer provider NSA. As you can imagine, this is a serious issue that needs to be addressed before NSI CS version 2 can be declared ready for prime time.

To help remedy this issue, it was agreed that a new mechanism should be added to the existing NSI CS version 2 protocol that will allow an RA to query a reservation and determine if there have been any error notifications raised against that reservation. Specifically, we need to allow for the handling of ErrorEventType, ReserveTimeoutRequestType, and messageDeliveryTimeout notifications as these will require corrective action from the uRA. The dataPlaneStateChange notification need not be handed as this state is modeled in the dataPlaneStatus element already available in the reservation query.

Four solutions will be proposed and the recommended solution identified in the conclusion.

Solution #1a

This solution uses the existing query commands to return an ordered list of notifications received by the provider NSA for the reservation queried. Notification ordering is maintained from the most recently received error notification to the oldest being stored by the provider NSA. Figure 1 below shows the query confirmed structure with the notification list included. In this case, a reservation query operation would return a complete list of notifications in the query confirmed response with no additional operations required.

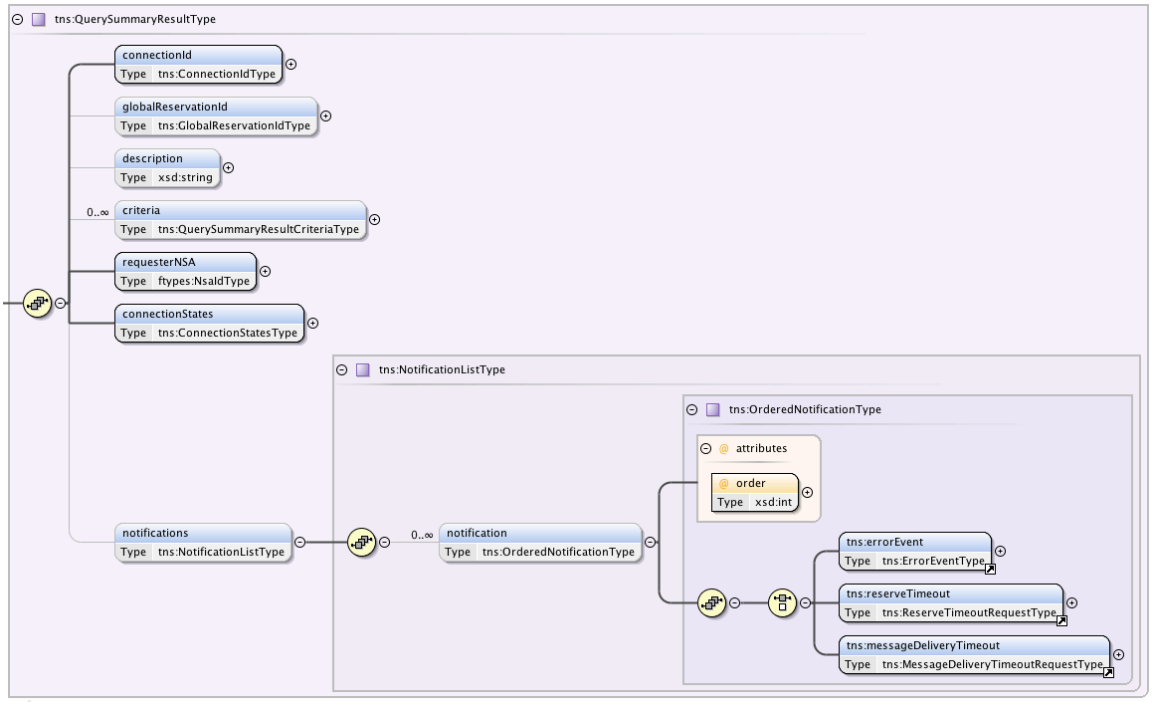

Figure 1 – Query with notification list returned in results.

The advantage of this solution is that with a single query we get all supported notifications returned, however, this is at a cost of message size as including all notifications can be expensive. In addition, the requester NSA would need to receive the full list at each query, even though they may only be interested in the most recent notification received. Reducing the list to the most recent notification only can result in a loss of valuable notifications if the RA polling interval is not quick enough to catch each notification as they arrive.

Solution #1b

This solution is an optimization of solution #1a where we introduce modeling of notifications with individual identifiers, then provide additional control of the notifications returned through a modification to the query operation. Figure 2 shows the notificationId element added to the query results for each of the notifications returned. This notificationId can be used in subsequent reservation query operations to reduce the list of returned notification results.

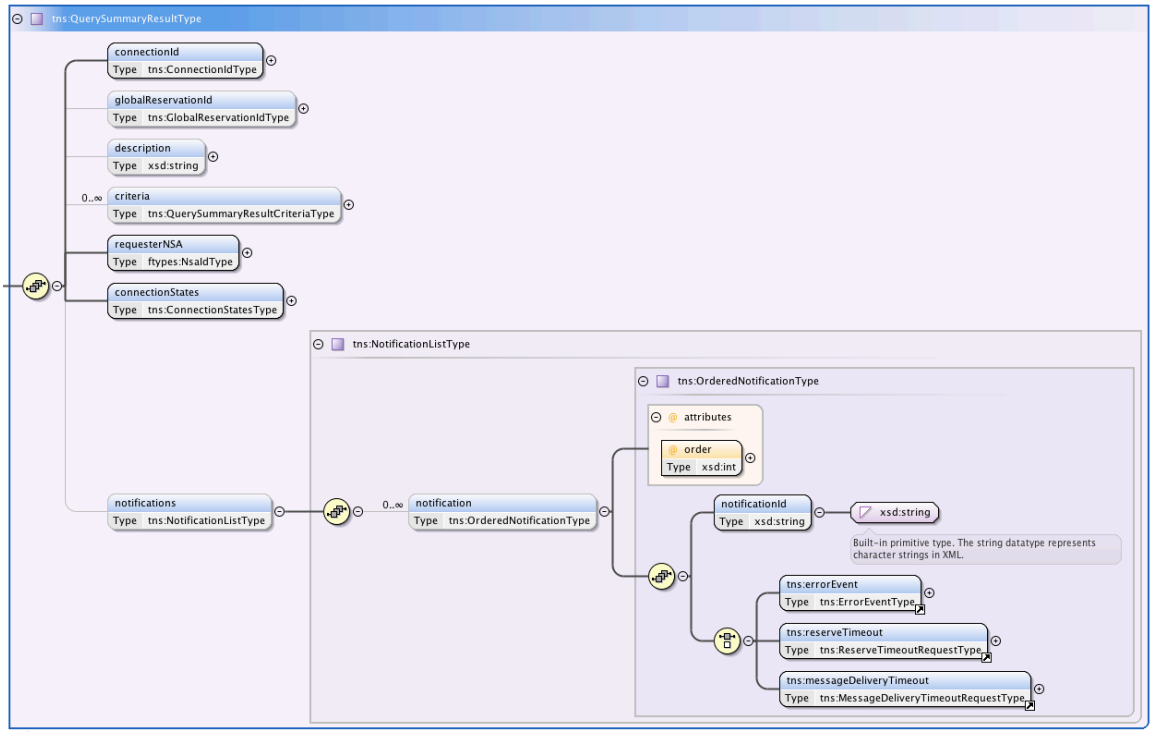

Figure 2 - Query notification confirmation querySummaryResultType.

In Figure 3, we see the introduction of notificationId to the existing querySummary operation. This notificationId conveys to the provider NSA additional constraints on returning query results. When the notificationId is specified in the query request it the provider NSA will only return notifications newer than the supplied notificationId. If no notificationId is specified in the query operation, then all notifications for the connectionId will be returned. This will allow the client to retrieve the initial notification list and then only retrieve any new notifications when they occur.

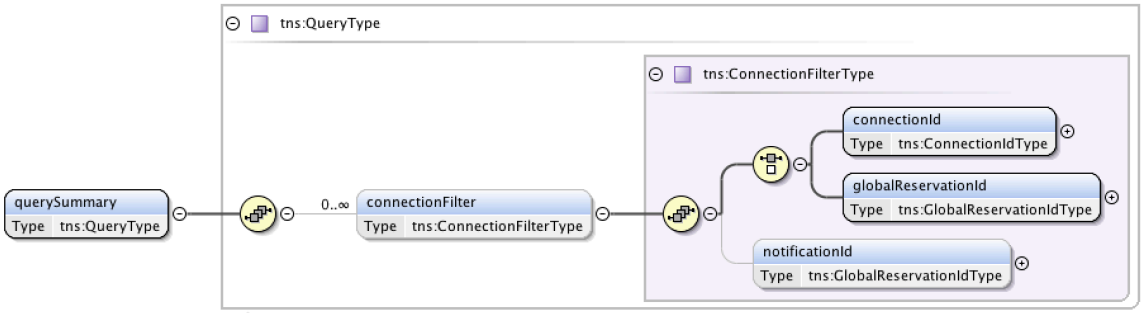

Figure 3 - querySummary operation for notification retrieval.

This solution addresses the concern of ongoing notification message overhead we introduced with solution 1a by providing an additional level of query control. However, we have overloaded the query request/response by introducing the notification messages and control mechanism. If we seriously consider plans to introduce a notification service in NSI CS version 3.0, then modification of the query

request may not be the best choice of solutions, since we do not have clear separation between reservation information retrieval and notification retrieval.

Solution #2a

This solution takes a different approach to providing access to notification messages, and thereby reduces the query overhead associated with returning a full list of notifications messages in the query result. Instead returning a list of notification messages in the reservation query confirmation, a list of unique notification identifiers associated with the notifications is returned. These identifiers can in turn be used to retrieve the individual notifications through use of a new notification query operation. In this solution the provider NSA generates a notification identifier that is unique within the context of that provider NSA. A separate notification query operation is introduced that allows the requester NSA to retrieve the error notifications of interest. Figure 4 shows the query results structure for this solution.

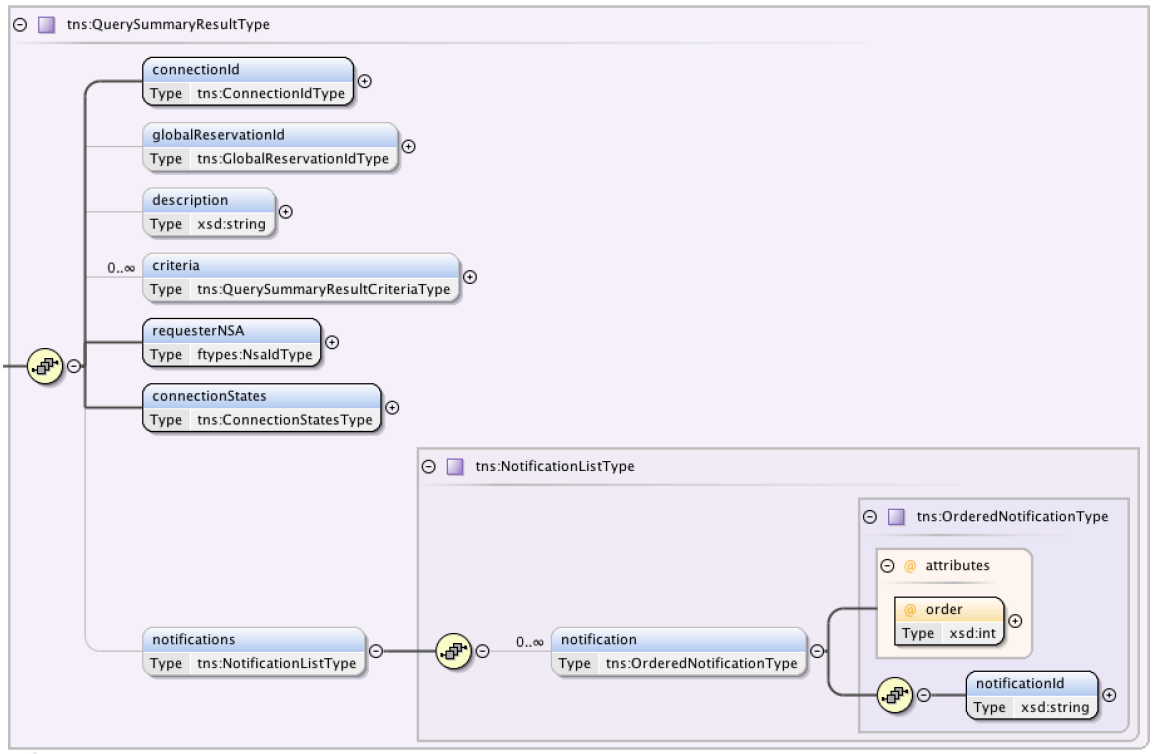

Figure 4 - Query confirmation with notification Id list.

The newly defined queryNotificationSyncConfirmed operation provides retrieval of notifications based on supplied list of notificationId. We model this after the standard querySummarySync operation by accepting a list of notificationIds of interest, with the queryNotificationSyncConfirmed result carrying the matching notification messages. In addition, we support the retrieval of all notifications against a supplied connectionId as a convenience function. Figure 5 shows this operation structure.

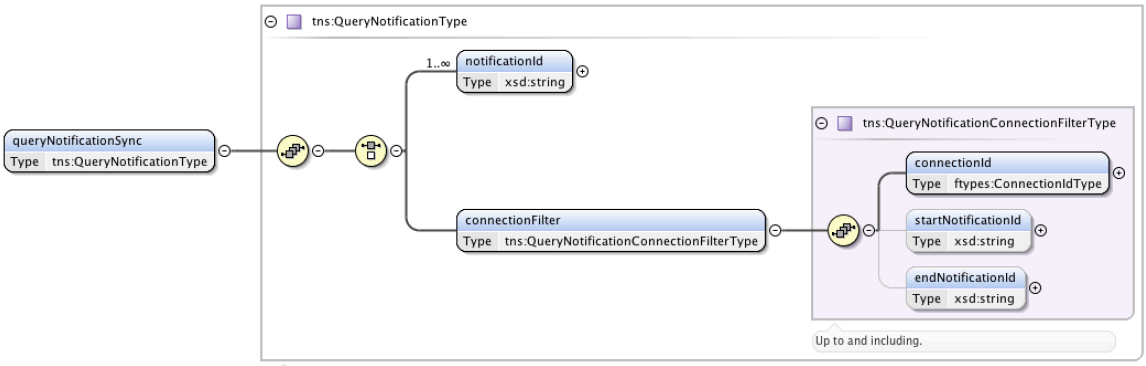

Figure 5 – Query notification operation queryNotificationSync.

The connectionFilter elements allow an RA to retrieve notifications messages based on notificationIds, or by specifying a connectionId and a range of notificationIds. Figure 6 shows the queryNotificationConfirmedType and the ordered notification messages.

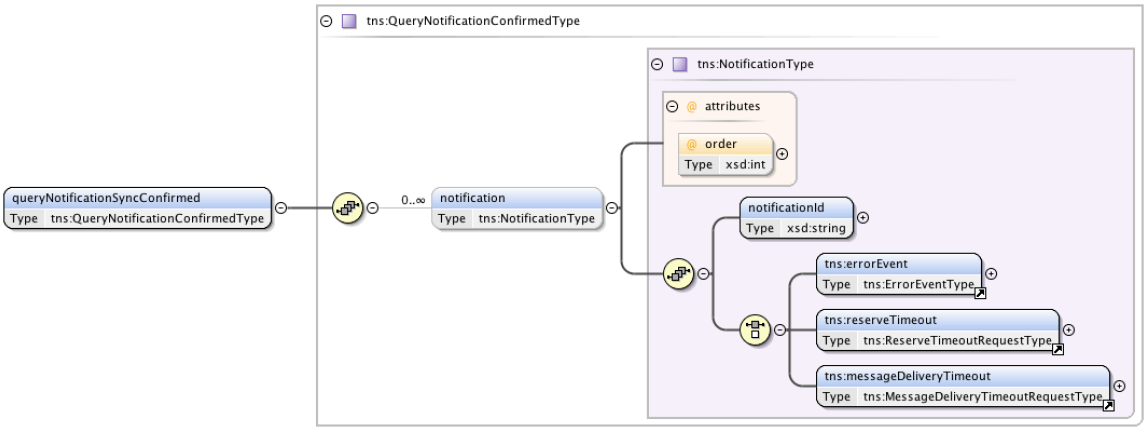

Figure 6 – Query notification confirmation QueryNotificationConfirmedType.

This is a very clean solution with little impact on the size of the original reservation query message. By utilizing a list of notification identifiers, we provide a mechanism that permits a requester to check the current list against a previously retrieved list to determine if a new error notification was received. We also provide a mechanism to allow retrieval of all notifications against a connectionId, and to retrieve only new notifications since a specific notificationId. It should be noted that in this solution notifications are modeled as separate entities with their own identifiers but relationships still maintained to the reservation or connection Id. The downside of this solution is that it requires a second query to get at the error notifications.

Solution #2b

This solution is an optimization of solution #2a where we simplify notificationId handling. There are a few key changes:

- We change the definition of a notificationId to be a linearly increasing integer unique in the context of a connectionId, removing the need for a separate ordering attribute.
- We return the most recent notificationId against the connection instead of providing a list of notificationIds in the query result.
- We modify the queryNotificationSync parameters to support only the range based queries.

Figure 7 shows the querySummaryConfirmed with the single optional notificationId. This notificationId is only present if a notification message has been generated against the reservation.

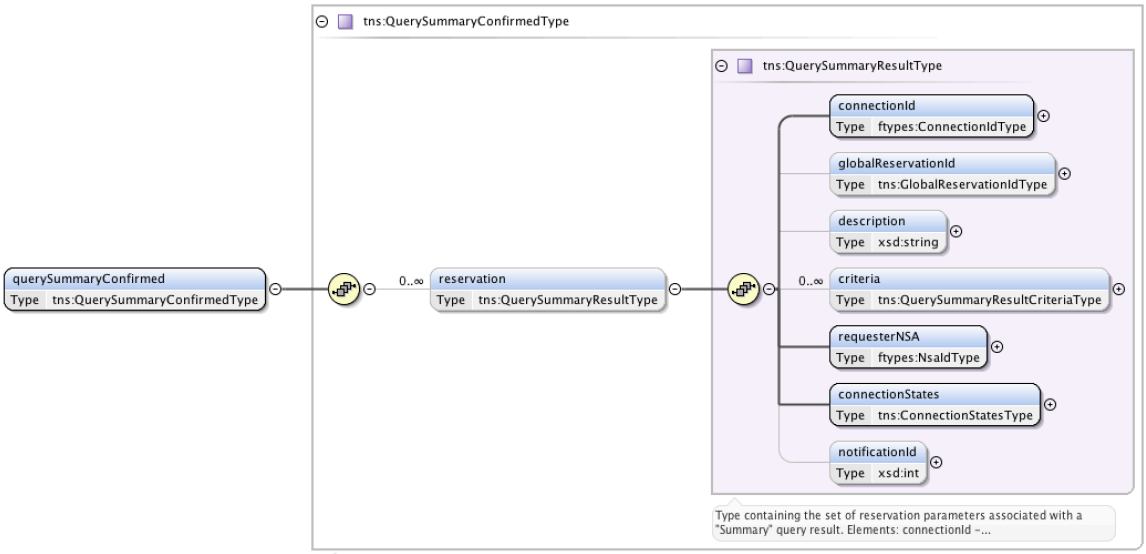

Figure 7 - Query confirmation with single notification Id QuerySummaryConfirmedType.

We simplify the queryNotificationSync operation by removing the ability to query a list of notificationIds since this is no longer required. An RA can control the retrieval of notification messages in multiple ways:

1. Get all the notifications messages against a connectionId by specifying the connectionId but no start or end notificationId.
2. Get a single notification message by specifying connectionId with both start and end notificationId set to the single notificationId.
3. Get all notification messages up to and including the specified notificationId by specifying the connectionId, and end notificationId set to the specified notificationId.
4. Get all notification messages after but also including the specified notificationId by specifying the connectionId, and start notificationId set to the specified notificationId.

This level of control allows an RA to fill in possible gaps it may encounter in the notification message stream when first detecting a notificationId against a reservation, as well as retrieving new notification messages encountered over the

life of the reservation. Figure 8 shows this optimized queryNotificationSync message.

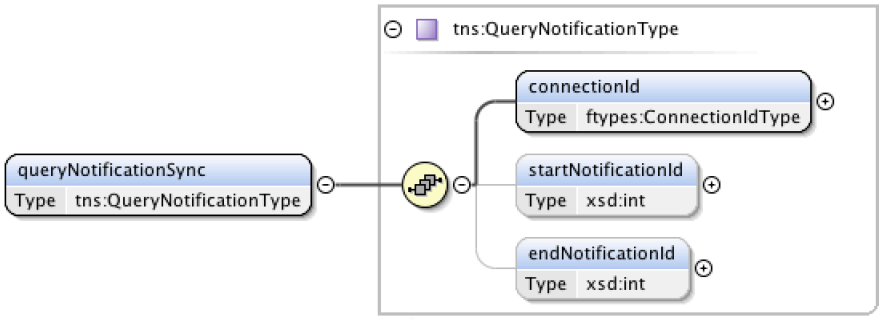

Figure 8 – Query notification operation queryNotificationSyncType.

In this solution the queryNotificationSyncConfirmed message is also modified to provide connectionId context, since the notificationId is no long unique provider wide.

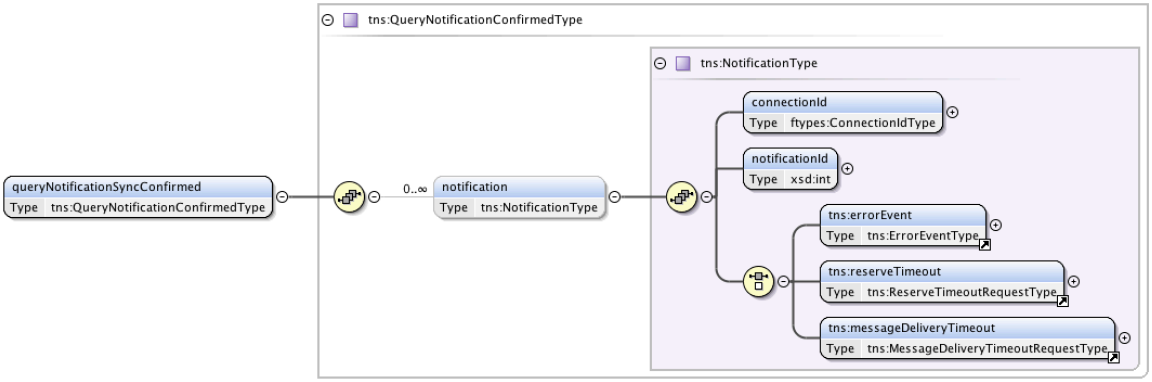

Figure 9 – Query notification confirmation QueryNotificationSyncConfirmedType.

This solution minimizes the impact on the reservation query message confirmation message requiring only a single optional notificationId. In addition, a separate queryNotificationSync operation provides flexibility in accessing all needed notification messages based on a connectionId and a range of notificationId. It has all the advantages of solution #2a, but in an even more compact representation.

Conclusion

Solution #1a would provide a path of least resistance to implementation, as we introduce no new operations and no unique notification identifier; however, the additional size overhead for every query when notifications are present will make this solution extremely expensive.

Solution #1b addresses the issue of notification message overhead by providing notification retrieval control within the reservation query message. The main drawback of this solution is that we are overloading the reservation query operation with notification messages when separation of concerns may be the better path given our plans for NSI CS version 3.0.

Solution #2a optimizes the query operation by including a list of notificationIds against the reservation and allowing the RA to retrieve the full notification messages through a separate notification query operation. This results in both a smaller message size on the initial query of the reservation, and more control on the results returned for a notification query. Well worth the additional code logic required to retrieve the notifications separate from the schedule query.

Solution #2b is a further improvement on solution #2a, modeling notifications in the context of a connectionId and optimizing query capabilities based on projected RA requirements for query behaviors.

Recommendation: Solution #2b. Using a separate notification interface will reduce reservation query overhead and reduce code churn within the NSA when a separate notification service is introduced in version 3.0 of the protocol.