TX²

Technical Solution:
Timers and Timing in Message Broker

Message Broker, and messaging in general are asynchronous and do not store state. Message flows are driven by an input, and completely finish when the output has been generated. (Actually, within standard Message Broker, the one exception to this rule is Aggregation. Aggregation stores state, but relies on a database implementation to do this.)

Waits, timers, etc. are therefore a philosophical as well as a technical issue for Message Broker.

Technically, a simple wait (say, implemented as a thread wait in a java node) blocks the thread, and holds resources, a real issue in a transactional system. It is possible to construct a design which tries to avoid this problem by routing to a new flow at the point you want to wait, which at least can then isolate the waits to a separate execution group, with each wait holding a smaller amount of resource; or you can construct a (fairly complex) timing system based on aggregation timeouts; but in all these cases the basic underlying problem still remains.

It is therefore instructive to look at the reasons for having timers, and to consider alternatives that fit better into a transactional asynchronous environment. There are times however, when you either need to time-out a pending event you are waiting for, or you need to retry an operation after a fixed time.

Taking a high level view of the requirements, we see that actually a very simple new construct in Message Broker can provide the underlying facility to implement several levels of new function – from a very simple timer, to a more complex state check with timeouts and scheduling function.

This simple building block is a “Heart Beat” or “Tick” input node. This input node generates a pseudo-message (no real message or real queue is required for this function) every 10 seconds (or whatever rate is set in a property). Hence, this input node, in a flow, can cause that flow to execute at set intervals.

The node uses practically no resource, as it waits as any other input node waits. What the flow does when triggered is entirely determined by the flow (or flows) that use it; but for example, the following are easy to implement:

1. For a simple timer / scheduler function, the flow checks a database table: TIMERS, containing 3 columns – Timestamp, Queue Name, and Message. If the current time is greater than or equal to the timestamp in the row, a message is send to the queue specified (and deletes the row from the TIMER database).
2. Flow A, in an error situation (not recommended for normal path) needs to retry after 5 minutes …….
3. A full "check for completion or timeout" service – extend the database in (1) to also include a "SELECT condition" (on some user database). The timer flow then not only triggers the message if the time-out value has passed; but also checks if the pending event has happened yet, by executing a "SELECT COUNT" based on the value in the database row. This timer flow can then re-trigger a flow when a database has been updated (for example), or if a timeout occurs. ***For an implementation of this function, see the accompanying article "Event Consolidation***".

The Heart Beat Input node itself is very simple. It has a main "run" method which simply waits and then generates a message, and a setter and getter method for the interval property. The node executable plug-in, and toolkit additions, can be acquired from TX Squared; see www.txsquared.com for more information.

