Zombies in BizTalk Server

Luigi Pampaloni – LPXSoft Ltd

What is a zombie?

- A zombie message is a message that was routed to a running orchestration from the MessageBox and was "in flight" when the orchestration ended. An "in flight" message is a message that has been routed to a service instance and so is in a MessageBox queue destined for the service instance. Since the message can no longer be consumed by the subscribing orchestration instance, the message is suspended and marked with a ServiceInstance/State value of "Suspended (Non-resumable)".

- A zombie service instance is an instance of an orchestration which has completed while a message that was routed to the orchestration instance from the MessageBox was still "in flight". Since the orchestration instance has ended, it cannot consume the "in flight" messages and so is suspended and marked with a ServiceInstance/State value of "Suspended (Non-resumable)".

When a zombie occurs:

In sequential convoys with non-deterministic endpoints – In this scenario, a master orchestration schedule is designed to handle all messages of a certain type in order to meet some type of system design requirement. These design requirements may include ordered delivery, resource dispenser, and batching. For this scenario, the tendency is to define a while loop surrounding a listen with one branch having a receive and the other having a delay shape followed by some construct which sets some variable to indicate that the while loop should stop. This is non-deterministic since the delay could be triggered, but a message could still be delivered. Non-deterministic endpoints like this are prone to generating zombies.

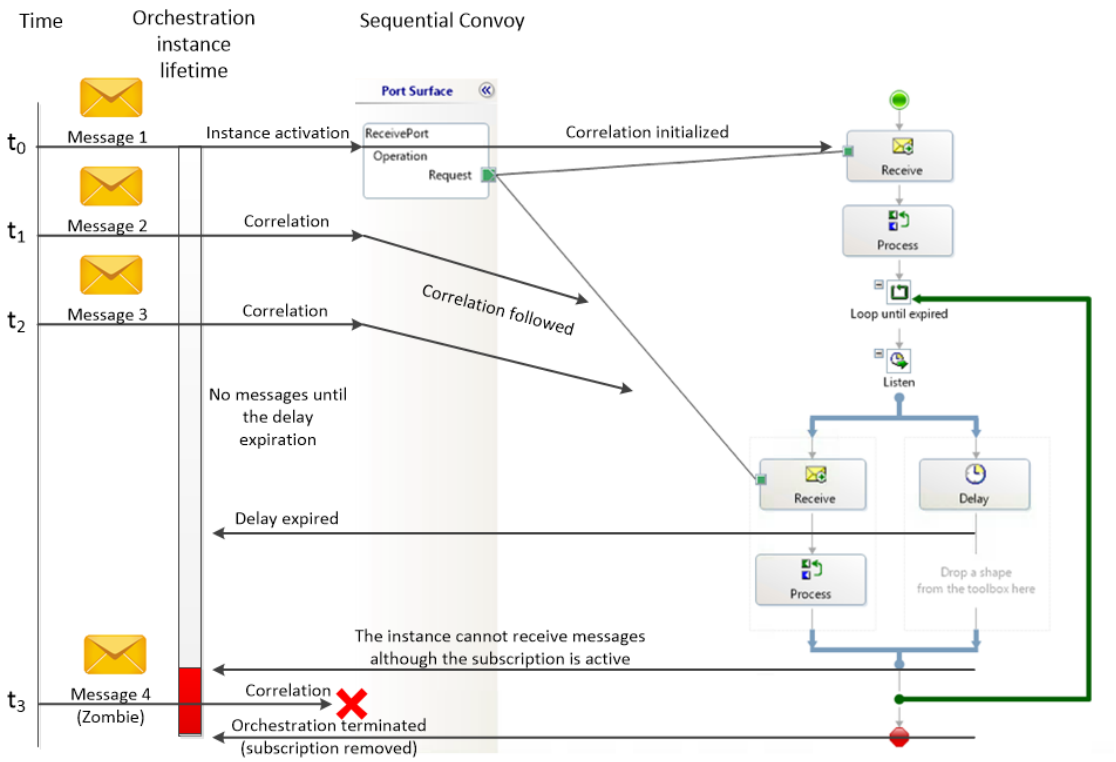
What you see in BizTalk Management Console:

When a zombie service instance is suspended in Microsoft BizTalk Server 2013 R2 the following error message is generated:

```
0xC0C01B4C The instance completed without consuming all of its
messages. The instance and its unconsumed messages have been
suspended.
```

Sequential convoy that generates zombies

The following diagram shows why zombie messages are generated.



t_0 – A message arrives in the MessageBox and an activation subscription is present on the subscription table. The BizTalk service activates an instance of the orchestration that starts to process the message. A subscription for correlated messages is added to the subscription table

t_1 – A correlated message arrives in the MessageBox and a correlation subscription is present on the subscription table. The message is delivered to the existing orchestration instance for processing

t_2 – A correlated message arrives in the MessageBox and a correlation subscription is present on the subscription table. The message is delivered to the existing orchestration instance for processing

t_2 to t_3 – No messages are delivered and the delay expires. The listen shape is terminated and the loop exits. Orchestration starts the termination process

t_3 – A correlated message arrives in the MessageBox and the correlation subscription is still present on the subscription table. The orchestration instance is not listening for messages but not terminated yet. A zombie message is generated

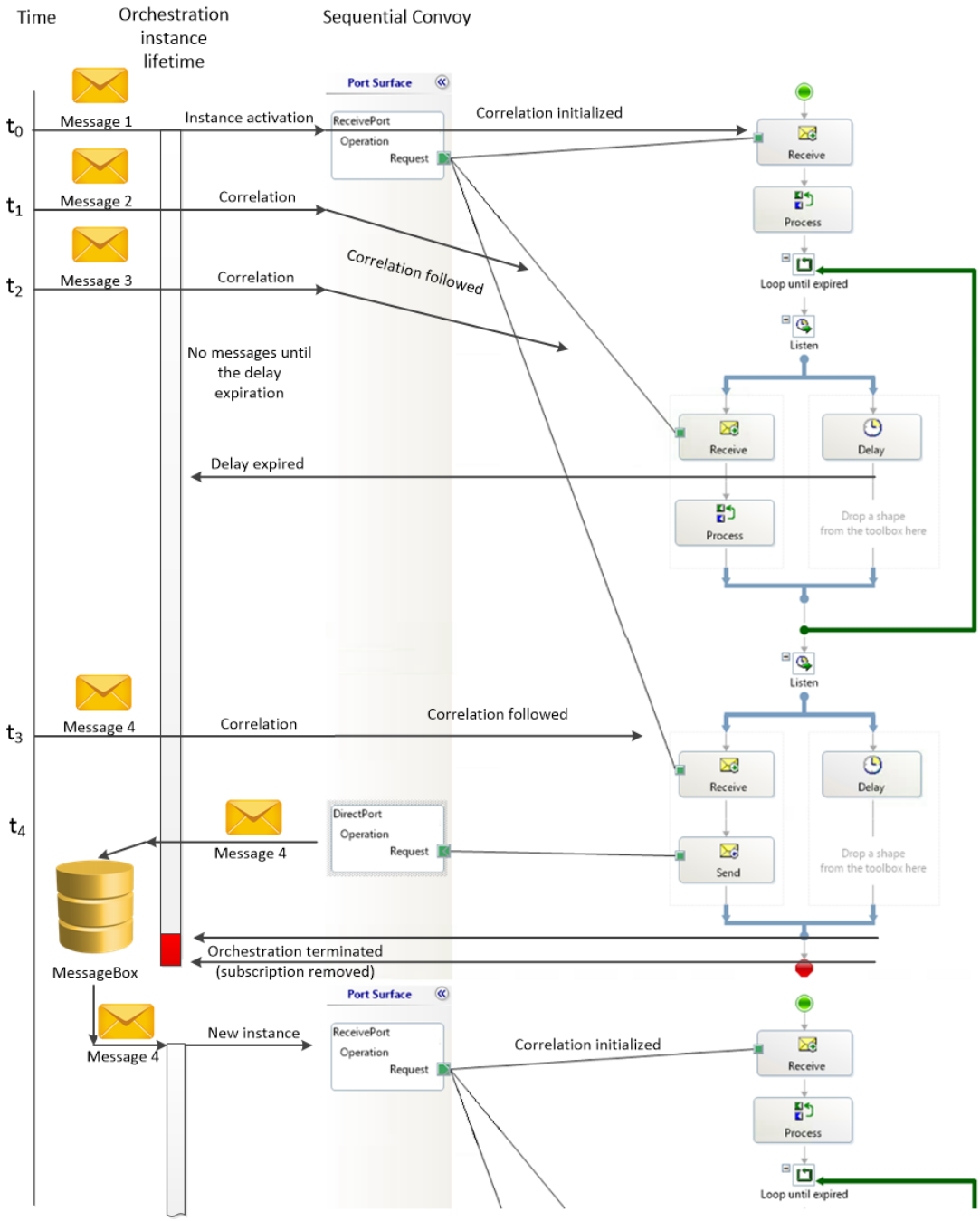
Solution 1 – add a date to the correlation set:

On the correlation set you can add a date field (without time) and the delay is set at every loop to expire after midnight of the day.

Any message received after midnight will start a new orchestration instance while the current instance is terminating without generating any zombie.

Solution 2 – shorten the time in between the listen shape and the orchestration termination:

Another solution is to add an additional listen shape that waits for a short period of time for correlated messages. This allows to shorten the period of time in between the listen shape and the termination of the orchestration instance. This solution reduces the cases of having zombies but you may still have zombies.



t_0 – A message arrives in the MessageBox and an activation subscription is present on the subscription table. The BizTalk service activates an instance of the orchestration that starts to process the message. A subscription for correlated messages is added to the subscription table

t_1 – A correlated message arrives in the MessageBox and a correlation subscription is present on the subscription table. The message is delivered to the existing orchestration instance for processing

t_2 – A correlated message arrives in the MessageBox and a correlation subscription is present on the subscription table. The message is delivered to the existing orchestration instance for processing

t_2 to t_3 – No messages are delivered and the delay expires. The listen shape is terminated and the loop exits. Orchestration starts the termination process

t_3 – A correlated message arrives in the MessageBox and is received by the second listen shape that will send the message back to the MessageBox via a direct port. The orchestration is terminated and the message will generate a new instance of the processing orchestration