Event scheduling

This article explains our approach to modelling time based events in code. It explains how a contract state can expose an upcoming event and what action to take if the scheduled time for that event is reached.

Introduction

Many financial instruments have time sensitive components to them. For example, an Interest Rate Swap has a schedule for when:

- Interest rate fixings should take place for floating legs, so that the interest rate used as the basis for payments can be agreed.
- Any payments between the parties are expected to take place.
- Any payments between the parties become overdue.

Each of these is dependent on the current state of the financial instrument. What payments and interest rate fixings have already happened should already be recorded in the state, for example. This means that the *next* time sensitive event is thus a property of the current contract state. By next, we mean earliest in chronological terms, that is still due. If a contract state is consumed in the UTXO model, then what *was* the next event becomes irrelevant and obsolete and the next time sensitive event is determined by any successor contract state.

Knowing when the next time sensitive event is due to occur is useful, but typically some *activity* is expected to take place when this event occurs. We already have a model for business processes in the form of flows, so in the platform we have introduced the concept of *scheduled activities* that can invoke flow state machines at a scheduled time. A contract state can optionally described the next scheduled activity for itself. If it omits to do so, then nothing will be scheduled.

How to implement scheduled events

There are two main steps to implementing scheduled events:

- Have your `ContractState` implementation also implement `SchedulableState`. This requires a method named `nextScheduledActivity` to be implemented which returns an optional `ScheduledActivity` instance. `ScheduledActivity` captures what `FlowLogic` instance each node will run, to perform the activity, and when it will run is described by a `java.time.Instant`. Once your state implements this interface and is tracked by the wallet, it can expect to be queried for the next activity when committed to the wallet. The `FlowLogic` must be annotated with `@SchedulableFlow`.
- If nothing suitable exists, implement a `FlowLogic` to be executed by each node as the activity itself. The important thing to remember is that in the current implementation, each node that is party to the transaction will execute the same `FlowLogic`, so it needs to establish roles in

the business process based on the contract state and the node it is running on. Each side will follow different but complementary paths through the business logic.

Note

The scheduler's clock always operates in the UTC time zone for uniformity, so any time zone logic must be performed by the contract, using `ZonedDateTime`.

In the short term, until we have automatic flow session set up, you will also likely need to install a network handler to help with obtaining a unique and secure random session. An example is described below.

The production and consumption of `ContractStates` is observed by the scheduler and the activities associated with any consumed states are unscheduled. Any newly produced states are then queried via the `nextScheduledActivity` method and if they do not return `null` then that activity is scheduled based on the content of the `ScheduledActivity` object returned. Be aware that this *only* happens if the wallet considers the state "relevant", for instance, because the owner of the node also owns that state. States that your node happens to encounter but which aren't related to yourself will not have any activities scheduled.

An example

Let's take an example of the interest rate swap fixings for our scheduled events. The first task is to implement the `nextScheduledActivity` method on the `State`.

Kotlin

```
override fun nextScheduledActivity(thisStateRef: StateRef,
                                   flowLogicRefFactory: FlowLogicRefFactory):
ScheduledActivity? {
    val nextFixingOf = nextFixingOf() ?: return null

    val (instant, duration) = suggestInterestRateAnnouncementTimeWindow(index =
nextFixingOf.name,
                                                                        source =
floatingLeg.indexSource,
                                                                        date =
nextFixingOf.forDay)
    return
ScheduledActivity(flowLogicRefFactory.create(TwoPartyDealFlow.FixingRoleDecider::class.java

                                                              thisStateRef, duration),
instant)
}
```

The first thing this does is establish if there are any remaining fixings. If there are none, then it returns `null` to indicate that there is no activity to schedule. Otherwise it calculates the `Instant` at which the interest rate should become available and schedules an activity at that time to work out what roles each node will take in the fixing business process and to take on those roles. That `FlowLogic` will be handed the `StateRef` for the interest rate swap `State` in question, as well as a tolerance `Duration` of how long to wait after the activity is triggered for the interest rate before indicating an error.

Note

This is a way to create a reference to the FlowLogic class and its constructor parameters to instantiate.

As previously mentioned, we currently need a small network handler to assist with session setup until the work to automate that is complete. See the interest rate swap specific implementation `FixingSessionInitiationHandler` which is responsible for starting a `FlowLogic` to perform one role in the fixing flow with the `sessionID` sent by the `FixingRoleDecider` on the other node which then launches the other role in the fixing flow. Currently the handler needs to be manually installed in the node.