FreeTON Recurring Payments

Contest submission

Steven de Oliveira
Contact: Steven de Oliveira @Stevendeo

1 Introduction

This document will describe the submission proposed by the OCamlPro team for solving the DevEx contest on recurring payments. The goal of this contest was to provide smart-contracts (SC) for deploying and managing custom services and subscriptions to the said services. These SCs must be compatible with TON's Crystals, TIP3 & DePools. Also, they must be written in Solidity or C++, the FreeTON SC's languages.

The code of our submission is available on Github:

https://github.com/OCamlPro/ocp-freeton-subscriptions

2 Contributions of this submission

This subscription contains the different contracts and their associated Debots. Our submission contains the source of the smart contracts and those of the Debots, as well as deployment scripts using **`ft`** and a high-level specification (added to this report).

2.1 Smart Contracts

- Buildable.sol : a contract defining a suitable interface for deployable contracts
- Builder.sol : a contract defining a suitable interface for contracts deploying contracts
- Constants.sol : constants used in the framework
- RecurringPaymentsRoot.sol : the root contracts ; it deploys the different services
- ServiceList.sol : the mapping of services deployed by a user
- ServiceListBuilder.sol : the deployer of "ServiceList"s
- Subscription.sol : the subscriptions

- SubscriptionBuilder.sol : the subscription builder
- SubscriptionManager.sol : the contract for services
- SubManagerBuilder.sol : the deployer of services
- Wallet.sol : the Crystal wallet for subscriptions
- WalletBuilder.sol : the deployer of Crystal wallets

2.2 Debots

- RootDebot.sol : the debot for communicating with the RecurringPaymenrsRoot contract.
- SubscriptionManagerDebot.sol : the debot for managing services
- SubscriptionDebot.sol : the debot for managing subscription

3 Details

3.1 Workflow

The workflow of the Recurring Payments is divided in 4 different phases.

1. Service deployment

A **Service Provider** deploys a service from the Root contract. The latter calls a the SubscriptionManagerBuilder contract that deploys a new service with the payment plan (duration & cost of the subscription) and a short description of the service. We call **tick** the duration of a payment plan as a time unit. Once deployed, the Service is added to a list of services registering all the services owned by a Service Provider.

2. Subscription

Any user may subscribe to a service by sending it a request ; the user becomes a **subscriber**. A Crystal Wallet & a Subscription are deployed on the chain. The Wallet will own all the funds of a given subscription, and only the Subscription contract will have access to its content. Initially, the Wallet balance is 0.

3. Filling account

A subscriber interacts with the Wallet only through the Subscription contract. It transfers it funds that are put on the Wallet.

4. Claiming / Canceling / Pausing

The Subscription holds two main information : the end of the last paid tick (`m_start`) and the wallet balance (`m_wallet_balance`). As time passes, some funds are progressively locked on the wallet. These locked funds can be claimed by the provider ; while the rest can be claimed by the subscriber.

The locked funds can be calculated from `m_start` & `m_wallet_balance`. First, we calculate the number of `locked ticks`, i.e. `ticks` that have been subscribed (but not claimed by the provider yet). The number of locked ticks is equal to the minimum between the number of payable ticks (balance divided by subscription cost) and the number of ticks from the last paid tick until now (now – m_start, all divided by subscription period). The locked amount is the number of locked ticks times the cost of the subscription.

Hence, it is possible to abstract the automation of the recurring payment with the TON clock. The effect of claiming (the provider claiming its payment), canceling and pausing (the subscriber wishing to stop the recurring payments and claim its unlocked funds) are based on the locked ticks.

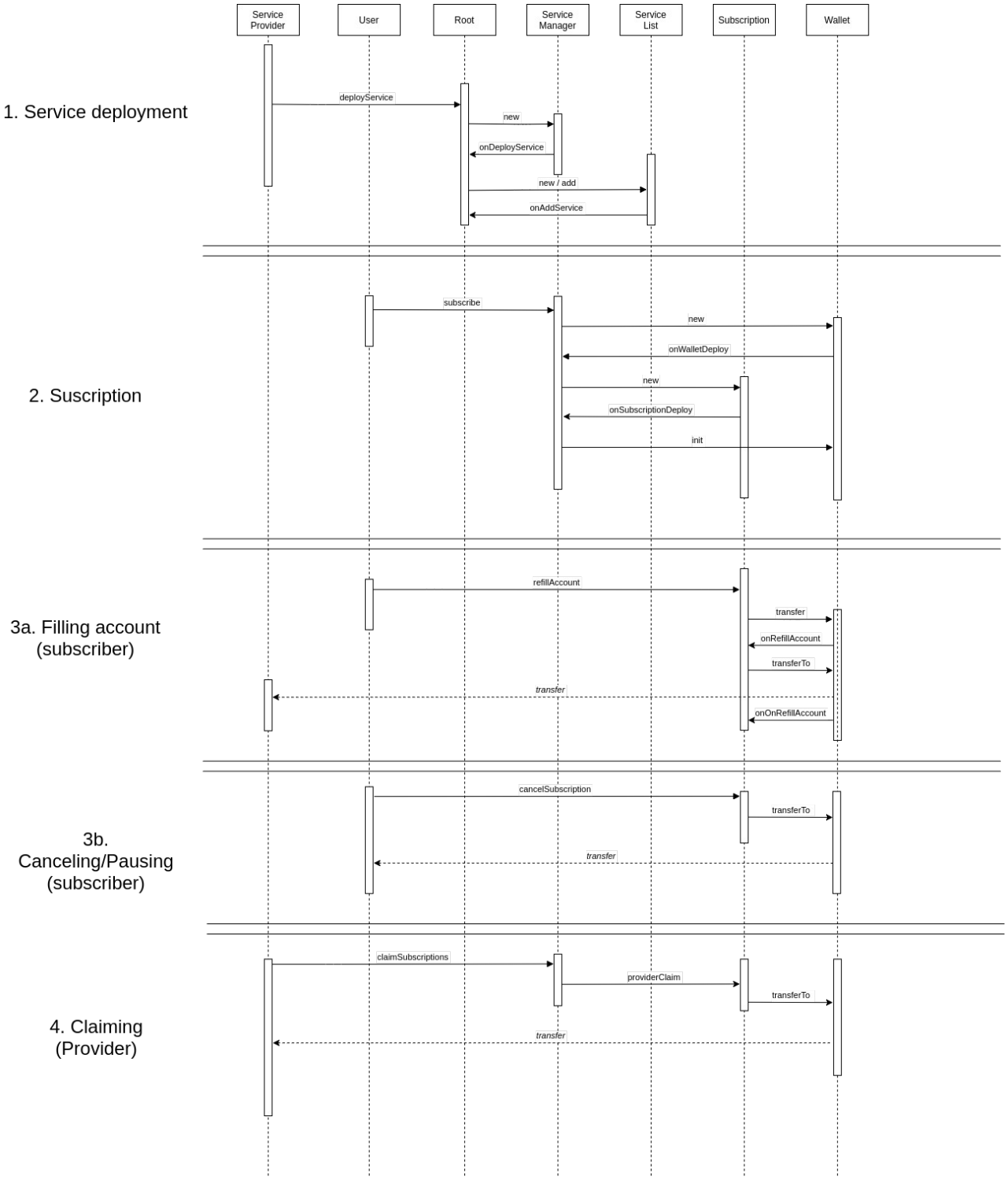

Figure 1: High-level specification