Antecedentes

Dentro del ecosistema de Rust ya existen librerías dedicadas a la simulación por eventos discretos, algunas de estas siguen un formalismo especifico mientras que otras buscan ser una librería de uso general para simulaciones, este último es el caso de la librería desarrollada en este documento por lo tal se estudiaran a fondo las librerías similares, como funcionan y por qué son diferentes a la desarrollada.

Ecosistema de Rust respecto a librerías de Simulación por eventos discretos.

Las de uso general | Las que siguen un formalismo

Nosotros somos de uso general por lo tanto esas se estudiarán a fondo

Mostrar cómo funcionan y el cómo difieren de nosotros.

Solo existen 2 que cumplen los criterios: Desim | Simrs

Simrs: Se define a sí misma como “librería de propósito general de simulaciones que provee mecanismos como planificador, estado, colas, etc.”. Desim en cambio se define como “Una librería que implementa un *framework* de simulación por eventos discretos inspirado por la librería SimPy de Python”.

Hablar de las similitudes y diferencias, que tienen en común y que hacemos que ellos no para justificar una nueva librería.

Las 3 librerías proveen:

* Una forma de insertar y agendar funciones en la simulación las cuales tienen la capacidad de auto agendarse y agendar otras funciones.
* Una estructura principal, la cual el usuario interactúa para realizar el modelo, comúnmente llamada Simulation.
* Un mecanismo interno para asegurar avanzar la simulación evento por evento, conocida como lista de eventos futuros y comúnmente denominada Scheduler.
* Una llave identificadora entregada al usuario al momento de ingresar una función al simulador con la cual pueden identificarla y agendar eventos en momentos específicos de la simulación.

Simrs: No permite pausar y reanudar las funciones, para poder ejecutar otro evento la función debe retornar, eliminando todo el estado actual de la función teniendo que empezar de nuevo, este problema es mitigado introduciendo una estructura llamada State utilizada para guardar y extraer data y se puede usar tanto para evitar la pérdida de información como para permitir el intercambio de data entre dos o más funciones.

Simrs:

* No permite pausar / reanudar funciones
  + Causa que la función empiece desde el inicio sin guardar el estado anterior
  + Utiliza una estructura State para evitar este problema
* Modelar procesos complejos que involucren funciones que entrelazan acciones se vuelve más difícil ya que la cantidad de código aumenta por cada acción que requiera sincronización entre funciones (Control Flow).
  + Ejemplo: función A solicita un recurso y solo puede avanzar cuando función B cree el recurso.
  + No poder pausar y reanudar cuando se solicita y se recibe el producto causa que el programador deba modelar los estados de transición de manera manual guardando el estado con el cual la función finalizo y utilizándolo para saber dónde debe continuar.

Desim: Utiliza generadores experimentales proporcionados por el lenguaje, solo se encuentran disponible en versión nightly del compilador del lenguaje y requiere que sus usuarios también ocupen un compilador nightly y habiliten la funcionalidad de generadores. Ofrece 6 operaciones:

* Timeout (time): Resume el generador después del tiempo especificado
* Event (event, time): Agenda el evento especificado en el tiempo especificado
* Request (resource): Solicita un recurso
* Release (resource): Devuelve un recurso solicitado previamente
* Wait: El generador no es agendado nuevamente hasta que otro generador lo active.
* Trace: Evento usado solo para logs, reagenda el generador inmediatamente.

En comparación Simulator provee 3 operaciones:

* Hold: Equivalente a Timeout
* Passivate: Equivalente a Wait
* Activate: Equivalente a Event con una diferencia clave, el generador que envía este evento es reanudado inmediatamente el cual no es el caso en Desim causando que activar un generador en Wait hace que el que envié el evento también entre en Wait.

A la hora de modelar procesos que simulen concurrencia es difícil de lograr con Desim debido a esta diferencia en la implementación de Passivate / Wait.

[NO SE SI AGREGAR ESTA PARTE]

(Lo siguiente son pensamientos, no texto explicito)

Simrs fue la influencia más grande a la hora de diseñar la API de esta librería, la idea de separar cada responsabilidad de la función en una estructura distinta es algo que se da bastante en el lenguaje rust, se considera idiomático, internamente Simrs se divide en un Container para las funciones, un Scheduler para la lista de eventos futuros, un State para guardar datos y compartirlo entre funciones, que de otra manera se encuentran isoladas, y Simulation que se compone de las 3 estructuras y provee la funcionalidad con la cual el usuario interactúa. Hace no mucho Simrs lanzo su versión 0.2.0 que incluye una nueva interfaz llamada Execute y una implementación de dicha interfaz en la estructura Executor de esta forma la responsabilidad de ejecutar la simulación es extraída de Simulation y en cambio se transfiere a cualquier estructura que implemente la interfaz.