План выполнения курсовой работы по теме «Реализация поддержки асинхронного программирования для фреймворка DSLab»

Содержание

Аннотация

DSLab - программный фреймворк для имитационного моделирования и тестирования распределенных систем.

В проекте используется дискретно-событийный подход описания моделей и приложений, где события обрабатываются в пользовательских функциях (callback-ах). В рамках проекта предстоит добавить возможность управлять событиями асинхронно.

1 Описание проекта и постановка задачи

1.1 Устройство DSLab

В силу широты охвата областей применения фреймворка он организован в виде набора слабо связанных программных модулей, использование которых будет осуществляться через их API. Это даст возможность пользователям фреймворка (исследователям, разработчикам, преподавателям) гибким образом собирать из модулей решения под свои цели, например симуляторы для конкретных типов систем или постановок задач.

Входящие в состав фреймворка модули можно условно разделить на три типа:

1. Базовые, функциональность которых используется остальными модулями (например, реализация дискретно-событийного моделирования)
2. Универсальные, функциональность которых может быть использована в различных предметных областях (например, модели сети);
3. Специализированные, которые заточены под определенную предметную область (например, библиотеки для моделирования облачных инфраструктур, исследования алгоритмов планирования заданий на кластерах или тестирования решений учебных заданий).

Архитектуру DSLab можно схематично представить в виде трех слоев (Рис. 1), включающих модули соответствующего типа. На рисунке также указаны текущие модули и зависимости между ними. Зависимости от dslab-core (от него зависят все имеющиеся универсальные и специализированные модули) не указаны, чтобы не загромождать рисунок. Таким образом, модули могут зависеть от модулей с нижних слоев, но не наоборот.

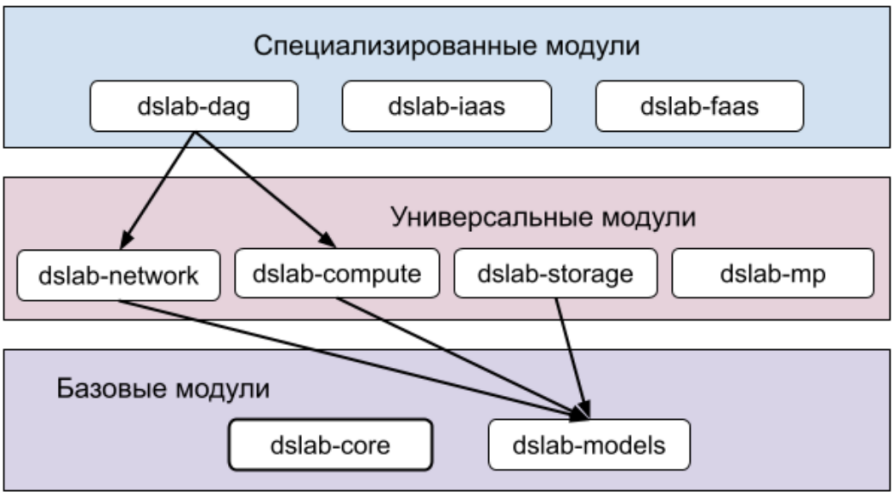

Рис. 1: Архитектура DSLab

Таким образом пользователь при разработке собственной симуляции может либо использовать уже готовые разработанные компоненты, либо реализовывать свои и произвольно их связывать.

Основным процессом создания симуляции является создание событий и написание процессов реагирования на них. Каждое событие представляет из себя следующую структуру:

1. идентификатор события
2. идентификатор компонента, создавшего событие
3. идентификатор компонента, которому это событие предназначено доставить
4. внутреннее время, когда событие должно наступить
5. произвольные данные события («полезная нагрузка»)

Таким образом в процессе симуляции разные компоненты генерируют события друг для друга и с помощью самого базового модуля dslab-core обмениваются ими. Чтобы как-то реагировать на событие, каждый компонент реализует единственный обобщенный метод (`trait EventHandler в языке Rust`), в который dslab-core передает нужное событие. При обработке события компоненты генерируют новые и таким образом цикл симуляции замыкается.

Получается, что сейчас реагировать на пришедшие события можно только в одном месте – в той самой функции-callback которую позовет dslab-core. В случае, когда событий немного – это лаконично выглядит и этим удобно пользоваться. В случае же, если возникает более сложный алгоритм с большим количеством различных событий, и, что важнее, цепочками событий, на которые нужно последовательно реагировать, подобная модель становится не самой удобной.

Хотелось бы иметь альтернативную возможность писать на языке программирования более понятный с первого взгляда алгоритм, ведь с единой точной входа многие последовательные логичные действия оказываются разбросаны по коду и фрагментированы.

Как раз такой возможностью является написание асинхронного кода. Тогда какие-то сложные куски логики можно было бы выносить в отдельные функции, держать их вместе, просто исполнение бы прерывалось в ожидании какого-то события, чтобы продолжить.

1.2 Цель

Добавить возможность пользователю использовать асинхронность при программировании симуляции в DSLab.

2 Актуальность и значимость

Как уже было описано в мотивации постановки цели этой работы, главное преимущество асинхронной модели – удобство при написании сложных многоступенчатых алгоритмов.

Это так же повысит читаемость кода. Процесс принятия решение о сохранении файла в распределенном хранилище в такой парадигме мог бы выглядеть таким образом (псевдокод):

```
async fn add_file_to_storage(some_file) {
    send_file_to_all_replicas(some_file);
    result = wait_for_confirmation_from_all().await;
    if result.has_quorum {
        send_commit_to_all_replicas();
        wait_for_commit_confirmation_from_quorum().await;
        send_ok_message_to_user();
    } else {
        send_reject_message_to_user();
    }
}
```

Посмотрев на функцию можно понять, что она делает, потому что логика последовательна и удобна разбита на подфункции. У нас есть возможность написать такой алгоритм на верхнем уровне, а не обрабатывать сообщения всех типов от всех реплик в одной функции.

3 Существующие работы и решения

Подобный подход уже был реализован в других симуляторах, например в SimGrid, но этот код сложно назвать легко читаемым.

4 Предлагаемые подходы и методы

Чтобы дать возможность пользователю писать асинхронный код, нужно создать саму возможность обрабатывать события асинхронно. Для этого нужно написать свой executor задач в DSLab, который будет исходя из внутренней логики (наступление времени, когда событие нужно доставить получателю) понимать, когда и какую асинхронную задачу нужно разбудить, и дать ей продолжить исполнение.

Для этого нужно реализовать альтернативу dslab-core (или дополнить уже сущетсвующую реализацию), и затем уже пользоваться новым интерфейсом на более верхних уровнях.

К счастью, язык Rust дает возможность пользователям самим управлять процессом рантайма, поэтому с помощью библиотеки futures подобное поведение может быть реализовано.

По асинхронному расту есть официальная документация (еще не в завершенном состоянии, но уже достаточно подробная, чтобы начать с нее).

Изучив предлагаемые там примеры и посмотрев материалы был реализован прототип.

5 Ожидаемые результаты

Поддержана возможность реализовывать алгоритмы в DSLab используя асинхронность. В проект добавлены примеры кода, документация по использованию реализованных методов.

6 План работ

Требования к DSLab, документация https://docs.google.com/document/d/1CprULnQiVSWTXiBkx90CSEi4tgkUhM3R6Ve
Crate futures https://crates.io/crates/futures rust async book https://rust-lang.github.io/async-book/