СЛОЖНОСТИ ИСПОЛЬЗОВАНИЯ ДИНАМИЧЕСКОЙ МОДЕЛИ НА C++ В GOLANG-СЕРВИСЕ НА ПРИМЕРЕ РАЗРАБОТКИ ТРЕНАЖЕРА ПО ТЕМЕ «ТЕОРИЯ АВТОМАТИЧЕСКОГО УПРАВЛЕНИЯ»

Тимшин В.Р.[1], Коровкин В.А.[2]
[1]ТПУ, ИШИТР, гр. 8К03, e-mail: vrt2@tpu.ru:
[2]ТПУ, ИШИТР, ассистент, e-mail: alcasar@tpu.ru:

Введение

Использование виртуальной реальности в тренажерах, с одной стороны, позволяет погружать обучающихся в более реалистичную симуляцию, а с другой генерировать большую вариативность взаимодействия со средой. Один из главных недостатков устройств мобильной виртуальной реальности – повышенные требования к вычислительной мощности устройств. Клиент-серверная архитектура позволяет частично решить эту проблему с помощью переноса некоторых процессов моделирования на сторону сервера.

В качестве примера будет использован тренажер, обучающий работе с некоторой системой автоматического регулирования (далее – САР). САР может быть представлена математической моделью, которая с помощью дифференциальных уравнений описывает состояния 40 параметров в реальном времени и обновляет эти параметры 10 раз в секунду.

В большинстве случаев необходимая математическая модель уже разработана и включена в процесс обучения. Для реализации модели могут использоваться различные языки программирования, такие как R, C, C++, и сами реализации могут не подразумевать интеграции с сервисами. Так, в рассматриваемом тренажере САР для описания математической модели используется C++, а в качестве серверной части – Golang.

Описание способов решения проблемы

Можно выделить два основных подхода в решении данной проблемы при том условии, что использование нескольких языков программирование при реализации сервиса нежелательно:

– Реализовать отдельный микросервис на C++ вокруг математической модели;
– Переписать реализацию математической модели на Golang.

При выборе первого способа в системе микросервисов будет находиться сервис, реализованный на C++. Поддержка этой системы с момента включения сервиса требует постоянного наличия разработчика C++ в команде.

Выбор второго способа, с одной стороны, позволяет сохранить неизменность технологий серверной части и фиксировать изменения модели, но, с другой стороны, возникает проблема верификации работы исходной и переписанной моделей. Она может решаться модульным тестированием.

Если применение нескольких языков программирования возможно, существует возможность использования библиотеки CGO [1], позволяющей вызывать функции языка Си. Применение ключевого слова extern позволяет вызывать функции, определенные на C++ [2]. К недостатку использования библиотеки CGO можно отнести необходимость написания и поддержки следующего кода (Листинг 1).

```
/*
#define intgo swig_intgo
typedef void *swig_voidp;

#include <stdint.h>


typedef long long intgo;
typedef unsigned long long uintgo;


typedef struct { char *p; intgo n; } _gostring_;
```

```
typedef struct { void* array; intgo len; intgo cap; } _goslice_;


extern void _wrap_Swig_free_main_f233ffd02779d35b(uintptr_t arg1);
extern uintptr_t _wrap_Swig_malloc_main_f233ffd02779d35b(swig_intgo arg1);
extern uintptr_t _wrap_LIB_NewFoo_main_f233ffd02779d35b(swig_intgo arg1);
extern void _wrap_LIB_DestroyFoo_main_f233ffd02779d35b(uintptr_t arg1);
extern swig_intgo _wrap_LIB_FooValue_main_f233ffd02779d35b(uintptr_t arg1);
#undef intgo
*/
import "C"

import "unsafe"

////////////////////////////////////////////////////////////////////////
func LIB_NewFoo(arg1 int) (_swig_ret uintptr) {
    var swig_r uintptr
    _swig_i_0 := arg1
    swig_r = (uintptr)(C._wrap_LIB_New-
Foo_main_f233ffd02779d35b(C.swig_intgo(_swig_i_0)))
    return swig_r
}
```

Листинг 1. Фрагменты программы, использующей CGO

Из приведенного листинга можно выделить две проблемы:

– Прототипы используемых функций необходимо указать в комментарии в определенном месте программы.

– Использование CGO даже в самых простых случаях требует опыта работы с библиотекой unsafe, использование которой не является хорошей практикой.

Для ускорения разработки можно воспользоваться генератором кода SWIG [3]. Пример работы генератора показан на листинге 1. Для успешной генерации кода необходимо создать файл с прототипами экспортируемых функций (Листинг 2).

```
%module main
%{
    extern void* LIB_NewFoo(int value);
    extern void LIB_DestroyFoo(void* foo);
    extern int LIB_FooValue(void* foo);
%}

extern void* LIB_NewFoo(int value);
extern void LIB_DestroyFoo(void* foo);
extern int LIB_FooValue(void* foo);
```

Листинг 2. Пример содержимого файла, необходимого SWIG для генерации

После использования данного метода при разработке сервиса для САР с математической моделью на C++ можно выделить основные недостатки использования нескольких языков программирования в одном проекте:

– Если используется SWIG, то необходимо изучить ограничение возможностей SWIG и алгоритм его применения.

– Если используется только CGO, то необходимо вручную создавать файл, «соединяющий» два языка программирования.

– Сложность проекта линейно возрастает из-за наличия дополнительного слоя абстракции между двумя языками программирования.

– Для поддержки приложения требуются разработчики со знанием обоих языков программирования.

– Код обоих языков должен предотвращать потенциальные утечки памяти, способные произойти из-за того, что каждый язык программирования обладает своей моделью управления памятью [4].

Заключение

В ходе решения задачи реализации сервиса на основе разработанной математической модели были сделаны следующие выводы:

1. Реализация необходимого функционала на целевом языке программирования – лучший вариант в большинстве случаев, поскольку:
   1.1. Продукт не будет усложнен управлением памятью и дополнительным слоем абстракции.
   1.2. В будущем не будет нужды в разработчике на отличном от целевого языке программирования;
2. Использование нескольких языков программирования в одном приложении целесообразно, если написание промежуточного слоя абстракции требует меньшего вложения ресурсов по сравнению с реализацией уже имеющихся возможностей;
3. Реализация необходимого функционала на нецелевом языке программирования целесообразно, если в будущем планируется использовать этот язык.

Выбор способа для рассматриваемого тренажера определялся следующими факторами:

– Реализация модели на C++ не использовала сторонние математические библиотеки или коллекции библиотеки std. Поскольку нет необходимости поиска библиотек с аналогичным функционалом, разработка ускоряется.

– Использование модели требует разработки системы управления ею в реальном времени. Проектирование этой системы с учетом существования дополнительного слоя абстракции займет больше времени по сравнению с проектированием системы на одном языке программирования.

Исходя из перечисленных факторов и выводов, было принято решение реализации модели на языке Golang.

Список использованных источников

1. Официальная документация по CGO [Электронный ресурс]. – URL: https://pkg.go.dev/cmd/cgo (дата обращения 20.01.2023).
2. Документация по C++: ключевое слово extern [Электронный ресурс]. – URL: https://learn.microsoft.com/ru-ru/cpp/cpp/extern-cpp?view=msvc-170 (дата обращения 26.02.2023)
3. Главная страница SWIG [Электронный ресурс]. – URL: https://www.swig.org/ (дата обращения 26.02.2023).
4. When and how to use CGO [Электронный ресурс]. – URL: https://youtu.be/qvVyT9QjIqc (дата обращения 20.01.2023).