

Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 23.Б11-мм

Поддержка алгоритмов на языке Python в среде моделирования SMM-3NG

КОЗЫРЕВА Софья Александровна

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
старший преподаватель кафедры информационно-аналитических систем, Смирнов К. К.

Санкт-Петербург
2025

Оглавление

Введение	3
1. Постановка задачи	5
2. Обзор	6
2.1. Обзор существующих решений	6
2.2. Обзор используемых технологий	6
2.3. Выводы	8
3. Описание решения	9
3.1. Реализация Python-агента	9
4. Эксперимент	11
4.1. Условия эксперимента	11
4.2. Результаты	11
Заключение	13
Список литературы	14

Введение

Система математического моделирования – программное обеспечение, которое позволяет симулировать различные процессы, исполнение которых позволяет изучать поведение физической системы. Преимущества такого подхода очевидны: не требуется наличие реального оборудования для проведения экспериментов, появляется возможность тестирования системы без риска выхода ее из строя. Компания “ООО Софтком” разрабатывает такую систему математического моделирования. Ядро написано на C++, и занимается диспетчеризацией агентов – процессов, внутри которых выполняются алгоритмы.

На данный момент существует несколько идей по улучшению системы, которые носят исследовательский характер. Для внедрения в систему необходимо проверить их жизнеспособность, реализуемость на практике. Одной из таких идей является реализация агента на языке, отличном от C++ (например, Python), и оценка, насколько легкой будет интеграция. В случае успеха в системе смогут беспрепятственно выполняться алгоритмы, написанные на разных языках.

В прошлом году эта задача выполнялась студентом в качестве учебной практики [8] для предыдущей версии системы (smm-2ng). Тогда алгоритмы выполнялись в отдельных потоках, единственным способом запуска Python-кода был его вызов из C++. Возникла проблема с преобразованием пользовательских типов данных между языками. Одним из вариантов решения стало использование ASN.1 [1] – язык для описания структур данных. Он принимается за некоторый общий стандарт, преимущество которого – независимость от языка программирования и аппаратной архитектуры. ASN.1 имеет свой набор типов данных, и строгие правила их кодирования и декодирования.

Новая версия системы математического моделирования (smm-3ng) была разработана с использованием ASN.1 и разделением агентов на отдельные процессы. Из-за значительных изменений в архитектуре требуется новая реализация агента на, например, Python, с последующим анализом результатов. Анализ поможет определить, необходима ли еще

одна итерация архитектуры, или можно переходить к улучшению других компонентов системы.

1. Постановка задачи

Целью работы является реализация агента на Python для системы математического моделирования на C++. Для ее выполнения были поставлены следующие задачи:

1. сформулировать требования к библиотеке ASN.1 на Python;
2. на основе требований выбрать библиотеку;
3. реализовать агента, включая алгоритмы, которые будут выполняться;
4. провести эксперимент, чтобы измерить производительность симуляции.

2. Обзор

Цель данного обзора – обосновать выбор технологий, использованных для реализации агента на языке Python, включая взаимодействие с другими агентами и ядром системы моделирования с помощью ASN.1, и описать их функциональность.

2.1. Обзор существующих решений

В предыдущей версии системы (smm-2ng) наблюдались проблемы, связанные с интеграцией алгоритмов, написанных на Python, в ядро системы, реализованное на C++. Основными трудностями являлось преобразование пользовательских типов данных между языками и необходимость ручного управления метаданными плагинов. Каждый плагин должен был самостоятельно описывать свои возможности (имя, входные и выходные параметры, типы данных) через JSON, что приводило к несогласованности и усложняло автоматизацию. Для организации взаимодействия между основным процессом и дочерними процессами использовались `fork()` и `pipe()`, в которых исполнялись Python-алгоритмы. Такая архитектура позволяла разделить исполнение кода на независимые процессы. Однако управление множеством дескрипторов и ручное закрытие лишних файловых дескрипторов усложняло разработку и сопровождение, а также ограничивало гибкость системы.

2.2. Обзор используемых технологий

2.2.1. Архитектура smm-3ng

Новая версия системы (smm-3ng) решает эти проблемы за счет нескольких принципиальных изменений в архитектуре. Во-первых, была реализована идея раздельного исполнения агентов: теперь все экземпляры алгоритмов запускаются в отдельных процессах. Во-вторых, агенты и ядро системы обмениваются ASN.1-данными через TCP-соединения. Эти данные передаются частями в виде PDU (Protocol

Data Unit). В системе предусмотрено несколько типов PDU, каждый из которых имеет собственное назначение и представление в ASN.1. Все данные кодируются и декодируются в формате BER (Basic Encoding Rules)/DER (Distinguished Encoding Rules). Таким образом, применение ASN.1 устраняет проблему сложного преобразования пользовательских типов данных, поскольку теперь данные передаются в едином формате. В-третьих, метаданные алгоритмов представлены в формате ASN.1, обеспечивая ядру и агентам возможность легко извлекать информацию о типах и именах аргументов методов внутри алгоритмов.

На момент начала работы система позволяет обмениваться данными между агентами, написанными на C++. На (рис. 1) представлена общая архитектура, количество агентов выбрано произвольно, явных ограничений на него нет.

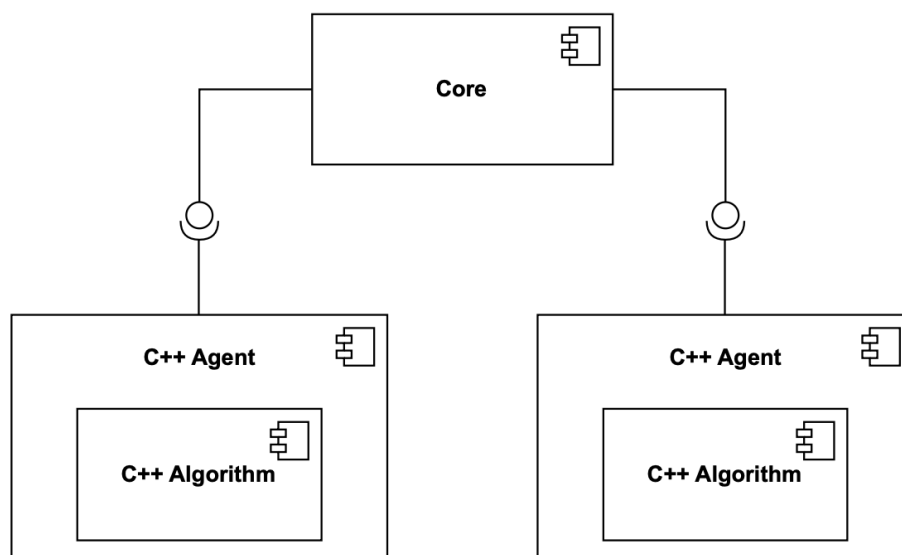


Рис. 1: Диаграмма компонентов системы (пример).

2.2.2. Выбор библиотеки для работы с ASN.1 в Python

Чтобы преобразовывать PDU из ASN.1 в Python и обратно было принято решение использовать одну из существующих библиотек, которая должна удовлетворять следующим критериям:

- поддержка python3;

- поддержка нужных форматов кодирования/декодирования (BER, DER);
- удобство для интеграции с существующей системой (поддержка ее типов данных);
- дополнительным преимуществом является автоматическая генерация Python-кода из схем ASN.1.

На основе данных критериев были выбраны три библиотеки: `pyasn1` [3], `pycrate` [4] и `asn1tools` [2], каждая из которых обладает подробной документацией.

Библиотека `pyasn1` предоставляет полный контроль над данными и их представлением. Такое преимущество является и недостатком: все ASN.1-схемы нужно описывать вручную, что увеличивает объем подготовительных работ.

В `pycrate` Python-классы генерируются автоматически на основе ASN.1-схем. После декодирования поля объекта доступны как атрибуты или элементы словаря. Кроме того, `pycrate` предоставляет возможности для работы не только с ASN.1, но и с другими протоколами, из-за чего ее размер значительно больше, чем у `pyasn1` и `asn1tools`.

В библиотеке `asn1tools` схемы загружаются динамически, то есть компилируются на лету. В отличие от других библиотек, данные представляются в виде словарей, а не объектов. Такой вариант кажется более удобным, учитывая не слишком высокую сложность существующих ASN.1-схем.

2.3. Выводы

По итогам обзор была выбрана библиотека `asn1tools`, которая не требует ни ручного написания классов, ни их генерации. Это сокращает количество шагов для получения Python-кода. Кроме того, после изучения работы с этими библиотеками, API `asn1tools` оказался наиболее интуитивно понятным.

3. Описание решения

3.1. Реализация Python-агента

Передача данных в виде ASN.1-схем дает возможность алгоритмам на разных языках использовать параметры и результаты друг друга (рис.2).

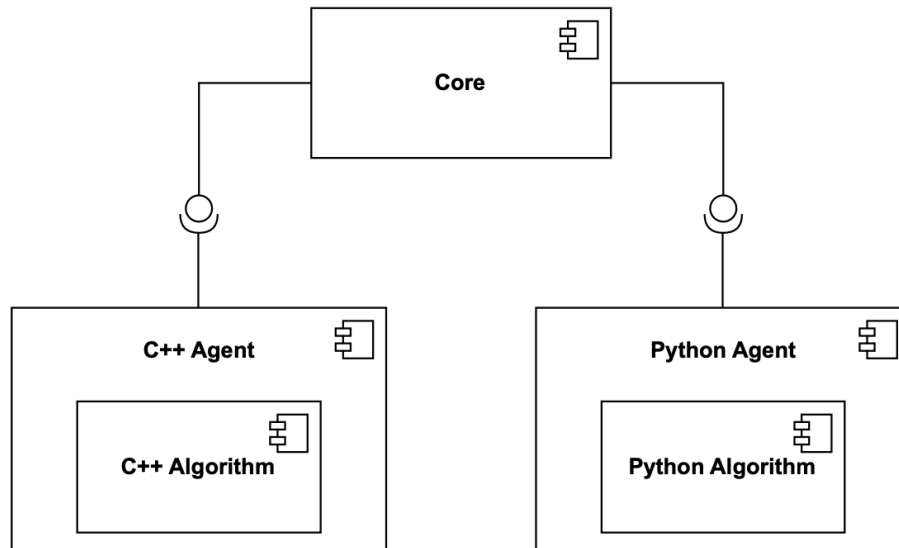


Рис. 2: Внедрение агента на новом языке.

3.1.1. Передача Python-данных в системе

Для кодирования и декодирования ASN.1-схем создается объект `asn1_compiler`. С помощью него реализованы функции отправки и получения PDU. Помимо этих двух базовых функций, для них присутствуют несколько оберток, которые передают информацию о состоянии агента в системе (например, `sendAckPDU`, `sendDonePDU`). Для передачи данных по сети используются функции из стандартного модуля `socket` [5]. Чтобы контролировать порядок передачи байтов, используется модуль `struct` [6]; несмотря на то, что принятым в сетевых интерфейсах является big-endian, в системе уже используется little-endian, на что указывает контрольный символ `<`.

3.1.2. Интеграция Python-агента с системой

Агентам системы известен IP-адрес ядра и его порт. Основная функциональность агента содержится в `start_agent`. Для начала создается сокет, прослушивающий входящие соединения. Далее агент должен подключиться к ядру и послать регистрационное сообщение. Оно содержит информацию об адресах и запущенном алгоритме.

Сам алгоритм теоретически может представлять собой любые действия над данными. Здесь важнее показать, что агенты вообще могут обмениваться информацией, даже если их алгоритмы написаны на разных языках. В качестве примера реализован простой `consumer`, который принимает данные от `producer`, проверяет, является ли входное значение целым числом, и ничего не возвращает.

Агент получает информацию о своих зависимостях от ядра в виде PDU. Он узнает, с кем соединяться и по каким параметрам. Эта информация хранится в конфигурационном файле. В таком PDU хранится в том числе IP-адрес ядра и порт другого агента для установления взаимодействия. Далее, с помощью специальных PDU системы (`SetupConnectionsPDU`, `ConnectionPDU`) агенты могут обмениваться реальными данными, которые используются в алгоритмах.

3.1.3. Конфигурация агентов и взаимодействие с ядром

Файл `main.py` передается в конфигурационный файл для запуска агента. Сам конфигурационный файл передается ядру при запуске. Этот `config` представляет собой JSON-файл, где для каждого агента (`modules`) указаны: путь до нужного файла (например, до `main.py`), алгоритм, имя его экземпляра и параметры (начальные данные агентов).

4. Эксперимент

Было проведено сравнение производительности двух симуляций: в обоих использовался алгоритм **producer** на C++, который увеличивает счетчик на 1 и отправляет число в качестве данных. В одной симуляции использовался алгоритм **consumer**, написанный на C++ (уже существующая реализация) – он проверяет, является ли тип данных целым числом, и выводит его в консоль, а в случае неверного типа выдает сообщение. Во второй симуляции агент исполнял алгоритм **consumer** на Python.

4.1. Условия эксперимента

Для проведения эксперимента использовались:

- процессор Intel(R) Celeron(R) N5095 @ 2.00GHz с 4 ядрами;
- операционная система Ubuntu 22.04.5 LTS (Linux);
- оперативная память – 8Гб.

Также была установлена минимальная частота процессора – 800 МГц.

В системе можно регулировать время симуляции. В качестве тестовых данных были взяты значения 1, 5, 10, 25, 50, 75 секунд. Время исполнения для каждой точки измерялось 10 раз. Измерения проводились с помощью команды **time** в Linux, в качестве результата измерения было взято значение графы **real**, которое отражает общее время выполнения программы.

4.2. Результаты

График можно увидеть на (рис.3). Для каждой серии измерений посчитана стандартная ошибка, она не превышает 5%.

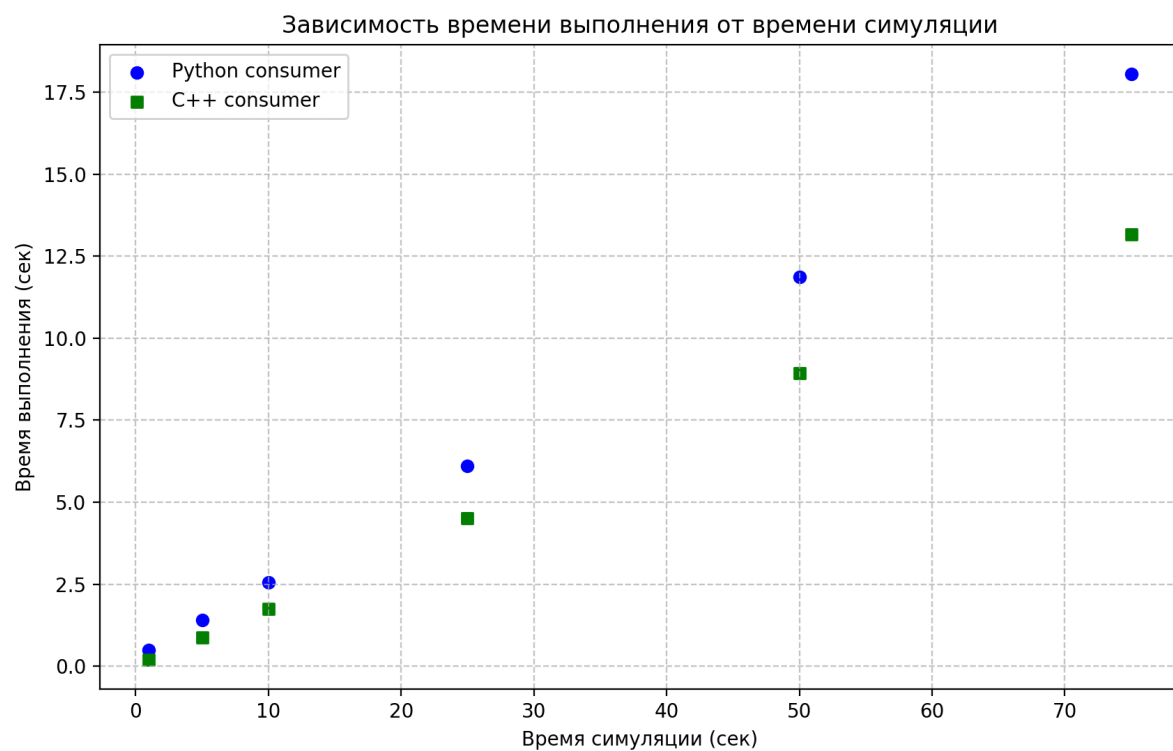


Рис. 3: График производительности программы с consumer-модулями на Python и C++.

Заключение

В результате работы были решены следующие задачи:

- Сформулированы требования к библиотеке ASN.1 на Python.
- Выбрана библиотека на основе требований.
- Реализован агент, включая алгоритм, который агент выполняет.
- Проведен эксперимент, измерена производительность симуляции.

Код проекта доступен в репозитории на GitHub [7].

Список литературы

- [1] ASN.1. — URL: <https://www.bgbm.org/tdwg/acc/Documents/asn1gloss.htm> (дата обращения: 23 июня 2025 г.).
- [2] asn1tools. — URL: <https://github.com/eerimoq/asn1tools> (дата обращения: 23 июня 2025 г.).
- [3] pyasn1. — URL: <https://github.com/etingof/pyasn1> (дата обращения: 23 июня 2025 г.).
- [4] pycrate. — URL: <https://github.com/pycrate-org/pycrate> (дата обращения: 23 июня 2025 г.).
- [5] socket — Low-level networking interface. — URL: <https://docs.python.org/3/library/socket.html> (дата обращения: 23 июня 2025 г.).
- [6] struct — Interpret bytes as packed binary data. — URL: <https://docs.python.org/3/library/struct.html> (дата обращения: 23 июня 2025 г.).
- [7] Репозиторий с исходным кодом. — URL: <https://github.com/sofyak0zyreva/smm-3ng-python> (дата обращения: 23 июня 2025 г.).
- [8] Связь моделей на языке Python с моделирующим ядром на C++. Баранов Артём Артурович. — URL: https://se.math.spbu.ru/thesis/texts/Baranov_Artem_Arturovich_Autumn_practice_2nd_year_2023_text.pdf (дата обращения: 23 июня 2025 г.).