

Разработка компонентов графоориентированного программного каркаса для реализации сложных вычислительных методов

Тришин Илья Вадимович, группа РК6-81Б

МГТУ им. Н.Э. Баумана

Россия, Москва, – 20 июня 2022



Введение

↳ Актуальные прикладные задачи и проблемы программной реализации их решений

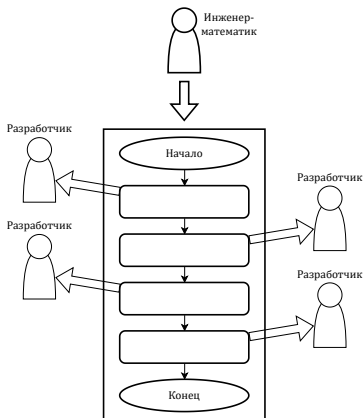
- Задачи математического моделирования.
 - Задачи анализа данных.
 - Задачи проектных инженерных расчётов.
 - и т.д.
-
- **Практически значимая программная реализация методов решений прикладных задач, как правило, является нетривиальной**



Введение

↳ Программная реализация прикладных решений в команде

- Наличие в команде нескольких человек и распределение обязанностей по разработке отдельных этапов между ними положительно сказывается на качестве реализуемого ПО



- Введение новых разработчиков в проект влечёт за собой необходимость описывать для них логику реализуемого решения.
- Целесообразно включить разработку описания логики решения в процесс его программной реализации.



Введение

↳ Современные средства, направленные на упрощение реализации решений прикладных задач

Среди современных средств упрощения разработки можно выделить следующие:

- научные системы управления потоком задач.
- платформы малокодовой разработки (LCPD).

Основной принцип

Использование формальных описаний алгоритма решения при его непосредственной программной реализации.

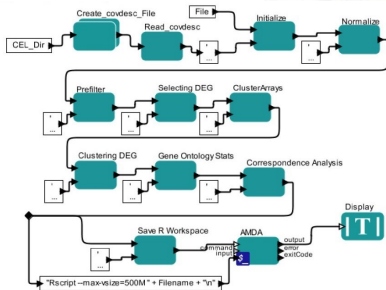


Рис. 1: Пример описания алгоритма в научной системе управления потоком задач Kepler

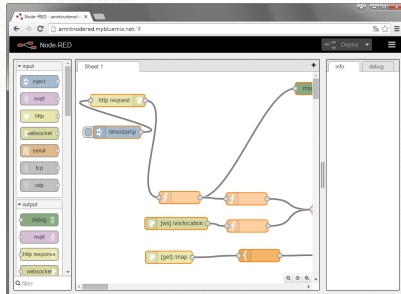


Рис. 2: Пример описания алгоритма в LCPD Node-RED



Введение

↳ Подходы к построению описаний логики решения прикладной задачи

- Можно выделить следующие подходы:
 - ▶ Диаграммы потоков данных (DFD).
 - ▶ Диаграммы потока управления.
 - ▶ Сетевые графики
 - ▶ и т.д.
- В данной работе в первую очередь рассматривается подход, получивший название **Graph-Based Software Engineering (GBSE)**.



Введение

↳ Диаграммы потоков данных

- Описание алгоритма представляет собой ориентированный граф.
- С его вершинами связываются процессы обработки данных.
- Его рёбра определяют пути данных между процессами.

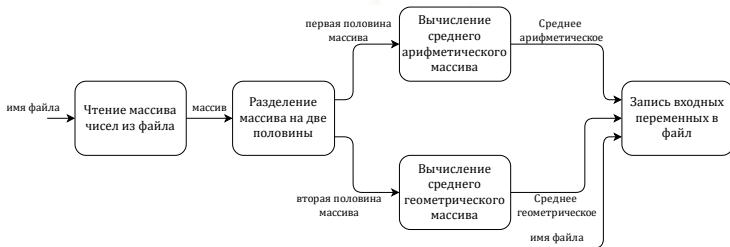


Рис. 3: Пример диаграммы потоков данных, описывающей вычисление среднего арифметического и среднего геометрического двух половин массива целых чисел с последующей записью результатов в файл



Введение

↳ pSeven – реализация диаграмм потоков данных

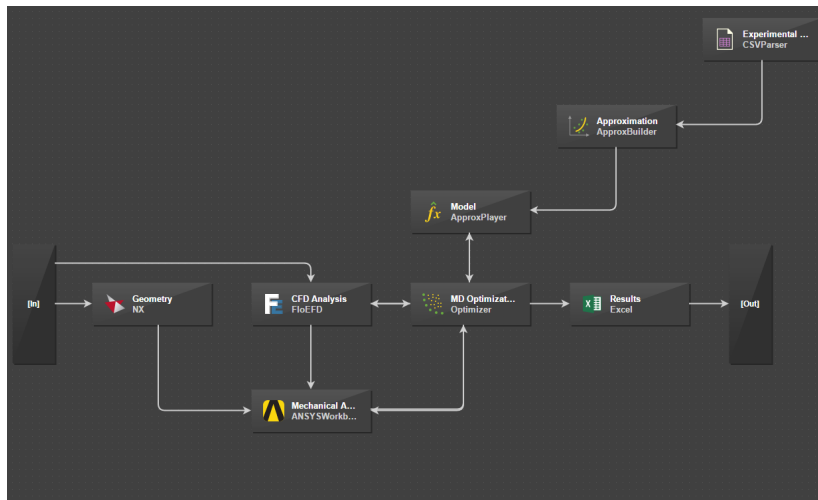


Рис. 4: Графический пользовательский интерфейс pSeven



Введение

↳ Результаты сравнения. Выявленные достоинства объекта разработки

Основные достоинства comsdk по сравнению с pSeven:

- Нет необходимости указывать входные и выходные данные при описании алгоритма.
- По умолчанию поддерживаются алгоритмы, подразумевающие взаимодействие с пользователем

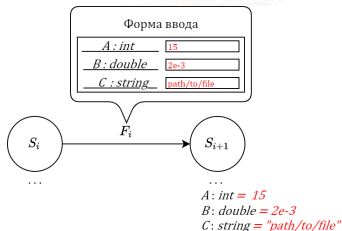


Рис. 5: Пример получения данных от пользователя при помощи генерируемой формы ввода

- Результат применения – компилируемая программа с возможностью запуска на различных платформах.



Введение

↳ Результаты сравнения. Выявленные недостатки объекта разработки

Основные недостатки comsdk по сравнению с pSeven:

- Отсутствие поддержки матричных типов данных.
- Отсутствие средств визуализации результатов расчётов.
- Отсутствие возможности использования при расчётах распределённых вычислительных систем;



Введение

↳ Цели и задачи разработки

Цель

Разработать обновлённые программные средства для создания и интерпретации графовых описаний вычислительных методов в программном каркасе comsdk.

Задачи

1. Сформировать требования к алгоритму, выполняющему этапы алгоритма по его описанию, составленному по методологии GBSE.
2. Спроектировать структуры данных для описания и представления описаний алгоритмов и их элементов в программном каркасе comsdk.
3. Разработать алгоритм обхода графовых моделей с использованием спроектированных структур данных.
4. Представить интерфейсы или реализации разработанных алгоритмов и структур данных на языке C++.



Постановка задачи

↳ Основные понятия GBSE. Состояния данных

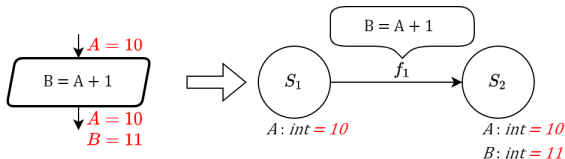
$$\underbrace{S = \left\langle \begin{array}{l} A: \text{int} \\ B: \text{double} \\ C: \text{string} \end{array} \right\rangle}_{\text{Состояние данных}} \quad \underbrace{D \multimap S = \left\langle \begin{array}{l} A: \text{int} = 10 \\ B: \text{double} = 2e-3 \\ C: \text{string} = \text{"path/to/file"} \end{array} \right\rangle}_{\text{Данные } D \text{ в состоянии } S}$$

- Состояние данных определяет, какие переменные какого типа должны быть определены на данном этапе алгоритма.
- Данные алгоритма модифицируются по ходу его выполнения.

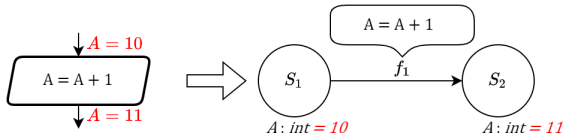


Постановка задачи

↳ Основные понятия GBSE. Функции-обработчики



Вариант 1. Функция-обработчик модифицирует состояние данных



Вариант 2. Функция-обработчик модифицирует только сами данные

- Функции-обработчики отвечают за обработку данных и их перевод из одного состояния в другое.



Постановка задачи

↳ Основные понятия GBSE. Функции-предикаты и функции перехода

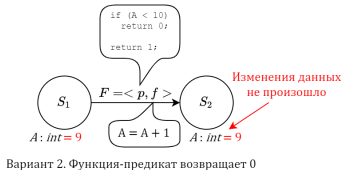
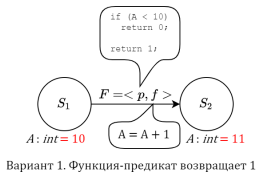


Рис. 6: Принцип работы функции-предиката



Рис. 7: Блок-схема логики функции перехода

- Функции-предикаты отвечают за предварительную проверку данных перед их обработкой.
- Функция перехода – составная функция $F = \langle p, f \rangle$, содержащая в себе функцию-предикат p и функцию-обработчик f .



Постановка задачи

↳ Основные понятия GBSE. Функции-селекторы

- Функции-селекторы отвечают за проверку условий при ветвлении алгоритма.

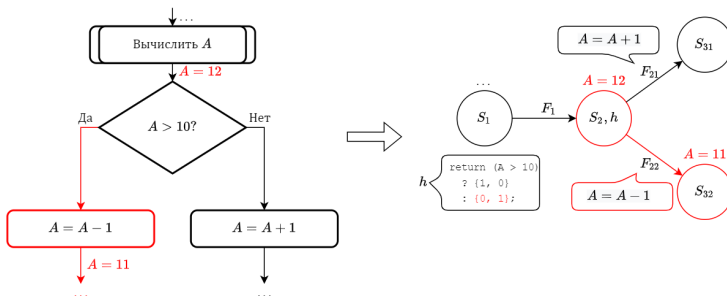


Рис. 8: Принцип работы функций-селекторов. h – функция селектор. Красным показана ветвь алгоритма, которая будет выполнена.



Постановка задачи

↳ Основные понятия GBSE. Графовая модель

- Графовая модель сложного вычислительного метода описывает его логику в виде ориентированного графа.
- Узлам ставятся в соответствие состояния данных.
- Рёбрам ставятся в соответствие функции перехода.

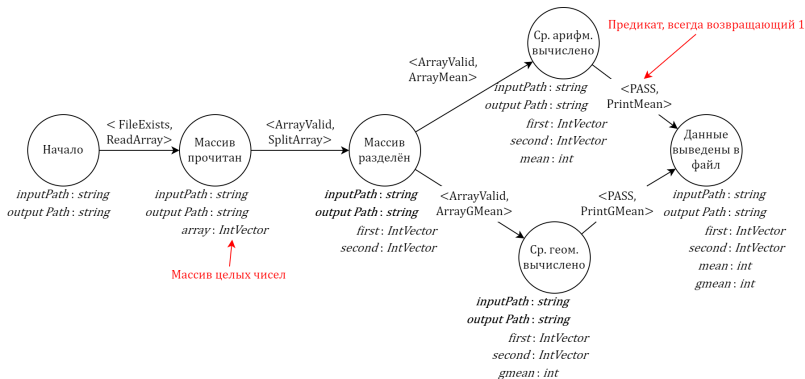
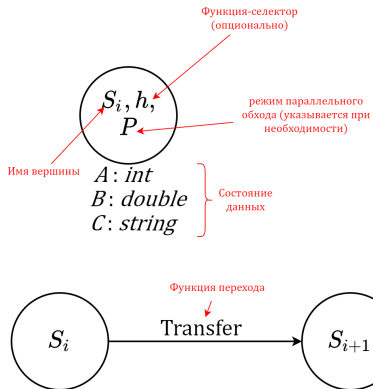


Рис. 9: Пример графовой модели, описывающей вычисление среднего арифметического и среднего геометрического двух половин массива целых чисел с последующей записью результатов в файл



Постановка задачи

↳ Основные понятия GBSE. Графовая модель



Атрибуты вершины:

1. имя;
2. состояние данных;
3. селектор;
4. режим параллельного обхода исходящих ветвей.

Атрибуты ребра:

1. функция перехода.



Постановка задачи

↳ Требования к алгоритму обхода

- Обход графовой модели – процесс выполнения функций перехода при движении по графовой модели от начальной вершины к конечной
- Должно быть реализовано условное ветвление с применением функций-селекторов.
- Должна быть предусмотрена возможность использовать различные вычислительные ресурсы для параллельного обхода независимых ветвей модели

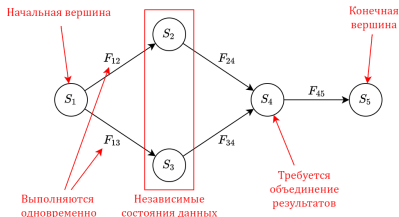


Рис. 10: Пример графовой модели, подразумевающей параллельный обход ветвей

- Режим параллельного исполнения в вершине S_1 определяет, какие ресурсы будут задействованы для выполнения ветвей $S_1 \rightarrow S_2 \rightarrow S_4$ и $S_1 \rightarrow S_3 \rightarrow S_4$



Программная реализация

↳ Функциональные структуры данных

- Были разработаны классы, представляющие функции-предикаты (Predicate), обработчики (Processor) и селекторы (Selector) и функции перехода (Transfer).
- Для хранения данных был использован универсальный ассоциативный массив Anymap.

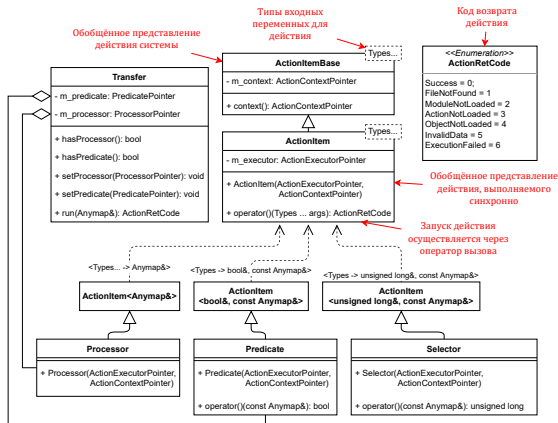


Рис. 11: UML-диаграмма разработанных функциональных структур данных



Программная реализация

↳ Информационные структуры данных

- Вершины графовой модели реализованы в классе Node.
- Рёбра графовой модели реализованы в классе Edge.

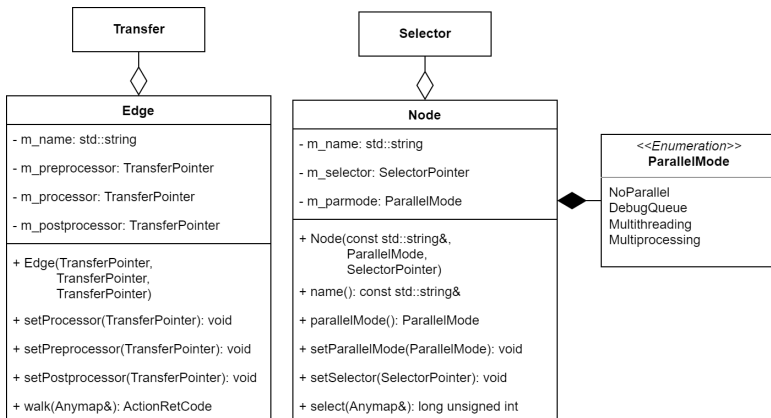


Рис. 12: Разработанные структуры данных, представляющие элементы графовой модели



Программная реализация

↳ Информационные структуры данных

- За программное представление графовых моделей отвечает класс Graph.
- Для объединения возможностей взаимодействия с вершинами и рёбрами и обращения к топологии графа спроектированы вспомогательные структуры данных NodeOp (операция с вершиной) и EdgeOp (операция с ребром).

Graph
- m_edges: std::vector<Edge> - m_nodes: std::vector<Node> - m_links: std::vector<std::vector<int>>
+ Graph() + node(unsigned): NodeOp + edge(unsigned): EdgeOp + addNode(const std::string& ParallelMode, SelectorPointer): int + addEdge(TransferPointer, TransferPointer, TransferPointer): int + connect(unsigned, unsigned, unsigned): void + run(Anymap&): ErrorCode

NodeOp
- m_graph: Graph& - m_index: unsigned
+ NodeOp(Graph&, unsigned) + incoming(): std::vector<EdgeOp> + outgoing(): std::vector<EdgeOp> + parents(): std::vector<NodeOp> + children(): std::vector<NodeOp> + inside(): Node& + operator=(const Node&): NodeOp&

EdgeOp
- m_graph: Graph& + m_index: unsigned
+ EdgeOp(Graph&, unsigned) + start(): NodeOp + end(): NodeOp + inside(): Edge& + operator=(const Edge&): EdgeOp&



- Интерфейс ExecutionContainer отвечает за отслеживание выполнения параллельных ветвей графовой модели.
- Интерфейс ExecutionBranch отвечает за обход одной ветви графа и хранение вспомогательной информации о нём.
- ExecutionBranch уведомляет контейнер по завершении каждой функции перехода в обходимой ветви.

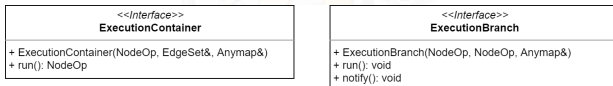


Рис. 13: UML-диаграмма управляющих структур данных



Программная реализация

↳ Алгоритм обхода графовой модели

Приведённый алгоритм реализован в методе `run()` класса `Graph`.

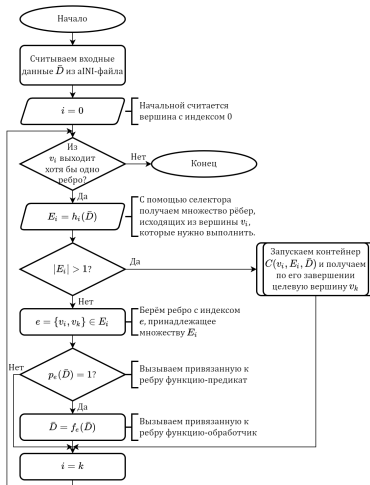


Рис. 14: Общий алгоритм обхода графовой модели



Программная реализация

↳ Алгоритм обхода графовой модели

Приведённые ниже алгоритмы реализованы в методах `run()` классов `ExecutionContainer` и `ExecutionBranch` соответственно.

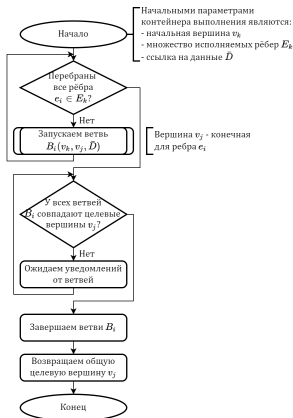


Рис. 15: Блок-схема алгоритма следящей структуры данных “контейнер выполнения”

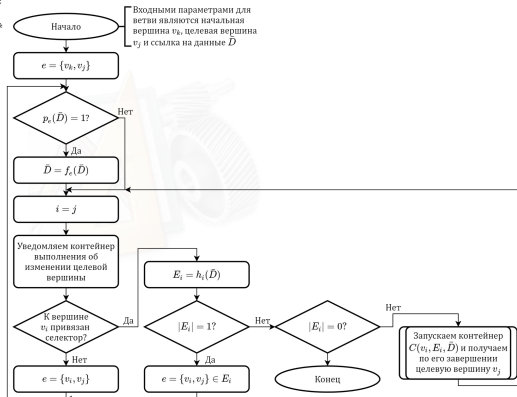


Рис. 16: Блок-схема алгоритма обхода одной ветви графовой модели



Заключение

↳ Анализ результатов

В результате выполнения работы:

- 1) расширены функциональные возможности библиотеки comsdk
- 2) создана новая архитектура классов, позволяющая в дальнейшем упростить процесс автоматического формирования программного представления графовых моделей;
- 3) разработаны структуры данных для программного представления графовых моделей алгоритмов и их элементов;
- 4) был разработан алгоритм, осуществляющий выполнение этапов алгоритма в соответствии с его графовой моделью;



Заключение

↳ Перспективы разработки

Перспективы развития программного каркаса comsdk включают в себя:

- реализации алгоритма обхода графовой модели с задействованием различных вычислительных ресурсов
- интеграцию средства генерации форм ввода¹;
- разработку средства визуализации обрабатываемых данных;
- разработку средства автоматической документации реализуемых алгоритмов;

¹ Соколов А.П Першин А.Ю. Программный инструментарий для создания подсистем ввода данных при разработке систем инженерного анализа // Программная инженерия. 2017. Т. 8, № 12. С. 543–555.



Спасибо за внимание!

