

Отчет

по теме:

«Графоориентированная методология разработки средств взаимодействия пользователя в системах автоматизированного проектирования и инженерного анализа. Обзор аналогичных подходов»

**Выполнил:**

аспирант

Бурча А.А. (РК6 - 11А)

**Научный руководитель:**

к.ф.-м.н., доцент, Соколов А.П.

Москва 2021 г.

Оглавление

[Введение в графоориентированный подход. 3](#_Toc62427775)

[Постановка задачи 5](#_Toc62427776)

[Обзор подходов к работе с СВМ 5](#_Toc62427777)

[Место графоориентированного подхода в классификации парадигм программирования 12](#_Toc62427778)

[Существующие решения задачи об интеграции ЛПР в графовую модель 16](#_Toc62427779)

[Список литературы 18](#_Toc62427780)

# Введение в графоориентированный подход.

Графоориентированный подход заключается в том, что пользователь представляет алгоритм сложного вычислительного метода (СВМ) в виде ориентированного графа. Сложным вычислительным методом будем называть множество вычислительных методов, взаимосвязанных друг с другом по входным и выходным данным. Узлы в таком графе представляют текущее состояние СВМ, а дуги представляют собой функции перехода из одного состояния в другое.

Состояние СВМ – это набор уникальных имен параметров СВМ и их типов данных. Некоторое состояние СВМ можно выразить формулой (1):

Данные СВМ – это конкретные значения параметров СВМ. Данные находятся в состоянии СВМ, если значения параметров имеют те типы данных, которые определяются состоянием СВМ. Данные в состоянии можно выразить формулой (2):

Данные, которые не находятся в состоянии , можно выразить формулой (3):

так как типы данных параметров данных не соответствует типам данных состояния .

Переход из одного состояние в другое обеспечивает функция перехода. Она включает в себя функцию предикат, которая производит проверку состояния данных D при этом переходе и разрешает или запрещает переход. Если переход разрешен, то он производится с помощью функции обработчика.

где – функция перехода из состояния в , - функция-предикат, проверяет *D*, выдает 0 или 1, - функция-обработчик, совершает переход, если = 1.

Также следует заметить, что узел ориентированного графа может являться подграфом, как это показано на рисунке 1.

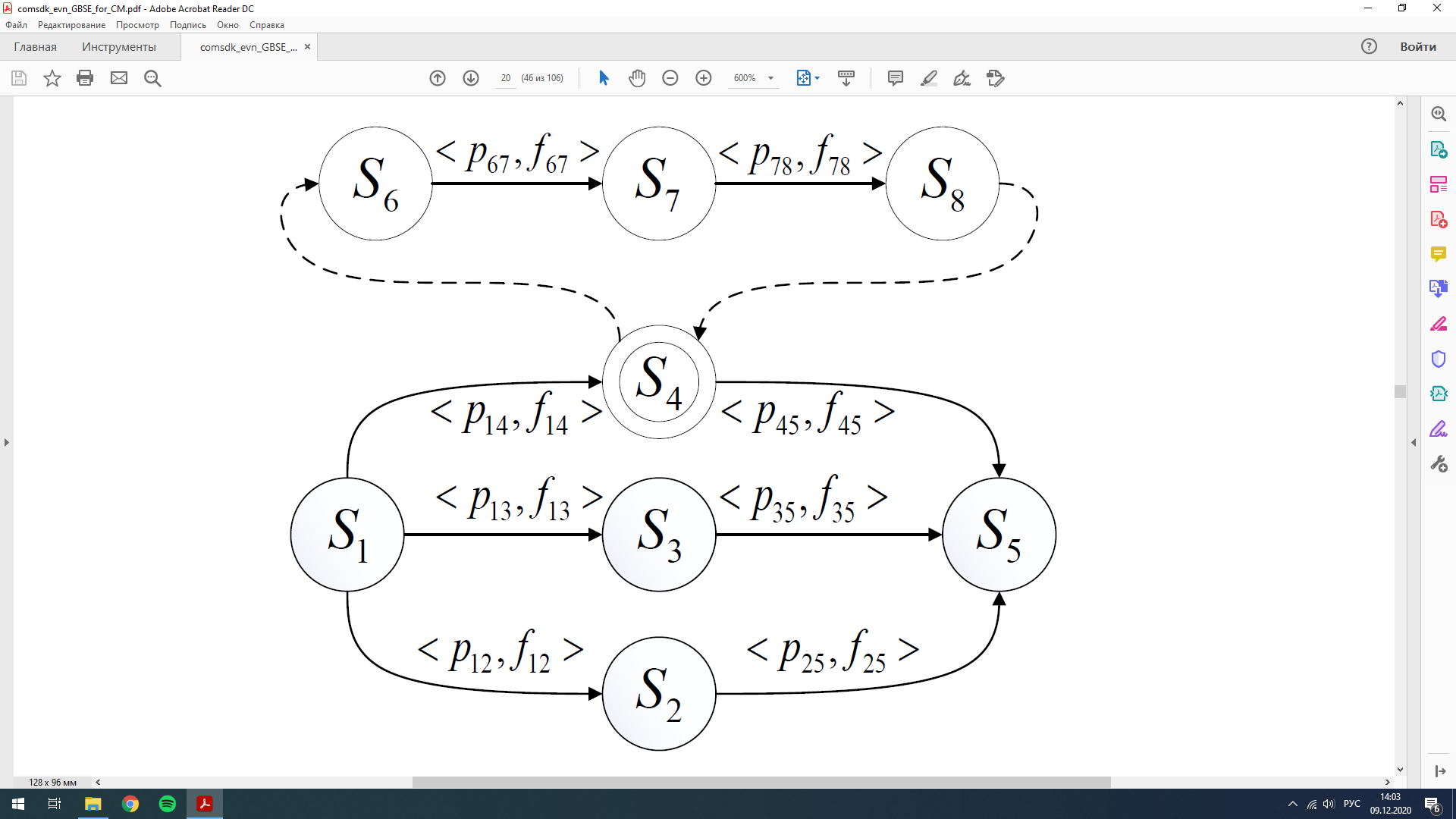


Рис 1. Графовая модель с иерархической вложенной графовой моделью. В данном случае состояние S4 заменяется на графовую модель, состоящую из трех состояний S6, S7 и S8 и соответствующих функций перехода. [1]

Цель подхода – обеспечить инженера-математика набором инструментов для организации своего кода, позволяющих уменьшить трудозатраты на его создание, отладку, дальнейшие верификацию, валидацию и поддержку. Такой подход позволяет инженеру-математику использовать графовую модель алгоритма как уже готовую реализацию СВМ.

В разработке индустриального ПО такой подход называется low-code development (разработка с низким использованием программного кода).

Среди преимуществ low-code платформ можно выделить:

* Быстрая разработка приложений.
* Быстрое развертывание.
* Исполнение и управление приложениями.
* Декларативная, высокоуровневая разработка.
* Возможность моделирования данных, разработки интерфейса и бизнес-логики.
* Пользовательское конфигурирование приложений.

# Постановка задачи

Если система получает входные данные и производит необходимые расчеты по заданному алгоритму без участия человека, то она автоматическая. Однако интерес представляет именно разработка автоматизированной системы, то есть той, в которой участвует лицо, принимающее решения. Это связано с тем, что, как правило, не все необходимые входные данные можно определить в начальный момент времени.

Задача заключается во включении лица принимающего решения в графо-ориентированный подход. Необходимо разработать механизм передачи управления ЛПР.

Эта задача подразумевает следующие пункты:

1. Разработка функций перехода, которые позволят пользователю взаимодействовать с графовой моделью через графический интерфейс.

2. Визуализация текущего состояния графовой модели для пользователя во время выполнения.

3. Синхронизация действий нескольких пользователей в системе с функциями перехода, контролируемыми пользователем.

# Обзор подходов к работе с СВМ

При проведении современных исследований возникает необходимость автоматизировать процессы решения сложных вычислительных задач. Достижение подобной цели не представляется возможным без формально определенного метода организации процессов в автоматизированной системе. Проектирование, создание и сопровождение подобных систем является трудоемкой задачей, для решения которой применяют инструментальные средства и среды разработки автоматизированных систем (CASE-системы) [2].

Для эффективного взаимодействия с подобной системой исследователю необходим графический пользовательский интерфейс, в котором модель организации вычислений может быть представлена визуально. Большие перспективы в автоматизации процесса решения сложных задач открывает перед исследователем возможность прямого взаимодействия с вычислительной моделью: остановка вычислительного процесса на определенном этапе, изучение обрабатываемых данных, просмотр истории изменения обрабатываемых данных, возврат к определенному этапу вычислений, ввод дополнительных параметров на определенной стадии вычислений и т.д. О необходимости подобных решений писал   
B. McCormic [3].

Для достижения подобной цели в GBSE необходимо решить множество технических задач по дополнению существующих инструментов. Одной из главных задач является дополнение функционала в существующий SDK для разработчиков графоориентированной методологии в плане обеспечения возможности приёма, передачи и интерпретации сообщений к графовой модели. Это позволит не только получать статус текущего решения, но также реализовать процесс отладки привычный для большинства инженеров-разработчиков, вводить дополнительные данные во время вычислительного процесса и др. Подобные возможности могут предоставить новый пользовательский опыт для пользователей графоориентированной методологии, поскольку в таком случае графовая модель становится не только инструментом для формального описания алгоритмов сложных вычислительных задач, но и инструментов описания алгоритмов решения сложных задач, но и реализации бизнес-логики пользователя. На основе таких решений можно построить систему инженерного анализа с возможностью интерактивного взаимодействия с графовой моделью и организации вычислительных экспериментов с помощью графического пользовательского интерфейса. Для заложения основ построения бизнес-логики на основе GBSE и решения задачи получения статуса решения и последующей визуализации в данной работе рассматриваются работы методы визуализации и организации сложных вычислительных задач.

Основная идея решения сложных вычислительных задач сводится к их декомпозиции на более мелкие процессы [2]. Методики, используемые в CASE, предполагают организацию процесса решения в виде иерархической структуры, которую можно представить в виде диаграмм. Например, диаграммы потоков данных (DFD), граф-схемы, диаграммы перехода состояний. Такое описание позволяет выделять структурные единицы приложения в виде функций или подпрограмм и связывать их между собой в определенной последовательности. Продолжением идей структурного описания проектируемого приложения можно считать объектный подход, в котором стали применять методы объектно-ориентированного программирования, что впоследствии привело к методологии CBD. Эта технология основывалась на унификации интерфейса для интеграции программного обеспечения в сложную систему. Языком описания для CBD является UML. Для визуализации процессов используются диаграммы последовательностей, диаграммы состояний, диаграммы деятельности. Метод организации и выполнения процессов основанный на CBD описан Robert D. Bjornson и StephenB. Weston [4]. В идее метода лежит организация вычислительный процессов, которые состоят из готовых компонентов, в виде диаграммы потока данных. Для создания и редактирования диаграммы предусмотрен графический интерфейс, в котором можно определить связи между процессами посредством указания точек входа и выхода обрабатываемых данных.

Метод разработки программного обеспечения при помощи проектирования моделей, не зависящих от предметной области, запатентовали David Talby Scott и David McMaster [5]. Для описания организации процессов составляется UML-модель, которая интерпретируется для последующей генерации исходного кода на основе шаблонов. Подобная программная разработка может применяться для создания программного обеспечения для реализации сложных вычислительных методов решения задач. Подход к построению моделей обработки данных вычислительных экспериментов предложили российские исследователи Леонтьев, Тарасов, Харитонов [6]. Для построения моделей используются сети Петри. Модель вычислительного эксперимента состоит из моделей вычислительного и управляющего процессов. Построение этих моделей происходит раздельно. Модель вычислительного процесса строится автоматически на основе дерева событий. Модель управляющего процесса строится на определении шаблонов реакции на события. Подобный подход позволяет построить гибкую структуру вычислительного эксперимента и получать сообщения о статусе эксперимента от управляющего процесса.

Решение сложных вычислительных задач обусловлено запуском решения на производительных вычислительных машинах. Обычно к методам высокопроизводительных вычислений относят:

1. параллельные вычисления;

2. распределенные вычисления;

- GRID;

- Облачные вычисления (PaaS, IaaS, SaaS).

GRID-системы применяются учеными для вычислительных задач, когда производительности кластеров из мощных серверов недостаточно [7]. Алекперов Р.К. [8] в своем исследовании рассмотрел организацию распределенных вычислительных сред на основе GRID-технологии и проанализировал специфические особенности этих задач. В своей работе он привел практические примеры, когда распределенные вычисления помогли решать задачи, требующие десятки лет вычислений, за несколько недель. Одной из самых популярных программных систем для запуска вычисления и развертывания GRID-сетей является BOINC, которое используется в десятках научных проектов.

В работе ученых Farkas Z. и Kacsuk P. описана разработка графического пользовательского интерфейса для взаимодействия с различными видами GRID-сетей [9]. В системе предусмотрен сервис для мониторинга выполнения задач в рамках архитектуры и визуализация статуса выполнения. Графический интерфейс представляет из себя поток работ (workflow), который графом описывает структуру GRID-сети, позволяет взаимодействовать с вычислительной системой и визуализировать состояние выполняемых задач. В системе существует Workflow Editor, который позволяет менять структуру вычислительной системы. Взаимодействия с различными видами GRID-сетей реализуется через промежуточный слой. Для каждой архитектуры написаны скрипты, которые переводят действия пользователя в графическом интерфейсе на язык конкретной архитектуры, а также обратно интерпретируют ответы от GRID-систем в workflow.

На сегодняшний день для проведения сложных вычислений многие ученые, исследовательские лаборатории и организации пользуются облачными платформами, которые предоставляют свои вычислительные мощности под любые нужды. Большинство облачных платформ предоставляет пользователю графический интерфейс (workflow) для организации процесса вычислений в виде диаграмм потока данных или граф-схемы [10]. Подобные методы реализуются за счет использования промежуточного программного обеспечения, которое связывает готовые программные компоненты или набором библиотек, с помощью которых можно создавать пользовательские вычислительные процессы и исполнять их в облаке.

В настоящее время организация сложных вычислений все чаще проводится с использованием облачных технологий, поскольку подобных подход снимает большую часть ответственности по инфраструктурному обеспечению вычислительной системы и обходится дешевле, чем использование или разработка GRID-сетей. Существующие облачные вычислительные платформы в основном узкоспециализированы на решении задач машинного обучения [11]. Подобный подход позволяет использовать вычислительную платформу для широкого класса задач.

Все найденные методы были основаны на организации процессов решения в виде графов, сетей Петри или потоков данных. Визуализировать процессы с подобной структурой удобно с помощью соответствующих диаграмм. Для визуализации процесса решения в рассмотренных методах были предусмотрены механизмы отслеживания состояния и управления исполнением процессов(workflow monitor), в которых можно:

− увидеть детали выполнения процессов;

− увидеть историю выполнения процессов;

− остановить, прекратить или возобновить исполнение решения.

Подобные механизмы реализуются с помощью создания параллельного управляющего процесса, который может обмениваться сообщениями с исполняющим процессом. Такой подход позволяет разделить обязанности и не усложнять логику работы исполняющего процесса.

При решении сложных ресурсоемких задач зачастую вычислительных мощностей локального персонального компьютера уже не хватает и возникает потребность в использовании многопроцессорных удалённых машин. Для успешного задействования мощностей удаленного вычислительного кластера необходимо решить задачи получения соединения, передачи данных и команд по определенному протоколу, сериализации и десериализации данных, распределения нагрузки на процессоры и др [12]. Над решением подобных задач работало много исследователей, а также производителей программного и аппаратного обеспечения.

В 70-x - 80-х годах появился ряд реализаций подхода удаленного запуска процедур (RPC). Подобные технологии предлагали подход запуска удаленных вычислений таким образом, чтобы это было похоже на вызов обычной процедуры в языках программирования. Среди основных можно выделить DCOM, CORBA, SOAP(XML-RPC) и др. На сегодняшний день существуют более современные аналоги, такие как Thrift, Finagle, gRPC.

Нельзя не упомянуть архитектурный подход к взаимодействию компонентов распределенного приложения в сети — REST [13], который очень хорошо себя зарекомендовал как способ обмена данными и командами в клиент-серверной архитектуре. Поскольку REST использует такие стандарты, как HTTP, URL, JSON и XML, послать запрос на удаленный запуск процедуры на сервере не представляет большого труда.

Реализации RPC предлагали не только способы удаленного вызова, но и технологии сериализации-десериализации данных и даже шифрования, однако для передачи больших данных, которые использовали ученые и исследователи, производительность была недостаточной. Для научных наборов данных, имеющих большой объем, таких как данные, получаемые от спутников, или численные модели климата, погоды и океанов, были разработаны специальные бинарные стандарты сериализации, например HDF [14] и netCDF [15].

Для передачи разнородных данных по сети особую популярность приобрел язык разметки XML, однако, программная обработка XML может оказаться неоправданно затратной, по сравнению с работой с данными более простой структуры. В таком случае разработчики рассматривают средства, изначально ориентированные на данные, такие как INI, YAML, JSON.

Несмотря на большое количество технологий удаленного запуска процедур, одним из самых практичных до сих пор считается удаленный доступ к вычислительному кластеру и запуску на его мощностях необходимых вычислений. Существуют специальные программные системы для запуска вычислений на удаленных кластерах, которые используют протоколы SSH, FTP и SFTP. Например, программа COMSOL Desktop [16] позволяет с помощью графического интерфейса запустить задачу на расчет, просматривать текущий статус вычислений и получить результирующий файл, в котором содержаться результаты вычислений.

Запуском вычислительного процесса на кластере занимается специальная программа — система управления кластером, главными задачами которой является запуск вычислительных процессов, балансировка нагрузки на CPU и GPU, защита от сбоев, максимально быстрый возврат результата вычислений. Примерами таких систем могут служить Microsoft Cluster Server, Beowulf, Condor, MOSIX, EnFuzion, PBS [17] и др. Многие системы представлены в виде набора дополнений к ядру операционной системы, позволяющее распределять процессы между узлами в составе кластера.

Несмотря на большую производительность, развертка GRID-сети не является самым лучшим решением для проектов с ограниченными ресурсами и задачами не требующих множества часов вычислений. Более рациональным вариантом может оказаться задействование облачных платформ для запуска вычислительных задач. На сегодняшний день существуют платформы для запуска решателей от всемирно известных производителей программного обеспечения, таких как Siemens, ANSYS, MSC и др. Подобное решение позволит исследователю сосредоточится на решении прикладных задач, а представитель сервиса берет на себя обязательства по предоставлению необходимых вычислительных мощностей с предустановленным программным обеспечением.

# Место графоориентированного подхода в классификации парадигм программирования

Говоря о графоориентированном подходе как о подходе к разработке ПО, будет уместно рассмотреть классификацию парадигм программирования и обозначить в ней место графоориентированного подхода.

Рассмотрим два больших класса парадигм программирования: императивное и декларативное программирование.

Императивное программирование — это парадигма программирования (стиль написания исходного кода компьютерной программы), для которой характерно следующее:

* в исходном коде программы записываются инструкции (команды);
* инструкции должны выполняться последовательно;
* данные, получаемые при выполнении предыдущих инструкций, могут читаться из памяти последующими инструкциями;
* данные, полученные при выполнении инструкции, могут записываться в память.

При императивном подходе к составлению кода (в отличие от функционального подхода, относящегося к декларативной парадигме) широко используется присваивание. Наличие операторов присваивания увеличивает сложность модели вычислений и делает императивные программы подверженными специфическим ошибкам, не встречающимся при функциональном подходе.

Основные черты императивных языков:

* использование именованных переменных;
* использование оператора присваивания;
* использование составных выражений;
* использование подпрограмм;
* и др.

Декларативное программирование — парадигма программирования, в которой задаётся спецификация решения задачи, то есть описывается, что представляет собой проблема и ожидаемый результат.

К подвидам декларативного программирования также зачастую относят функциональное и логическое программирование — несмотря на то, что программы на таких языках нередко содержат алгоритмические составляющие, архитектура в императивном понимании (как нечто отдельное от кодирования) в них также отсутствует: схема программы является непосредственно частью исполняемого кода.

Графоориентированный подход включает в себя принципы как императивного, так и декларативного программирования. Черты императивного программирования проявляются при написании модулей графовой модели на языках С++ и Python. Из декларативного программирования используется задание пользователем алгоритма СВМ в виде ориентированного графа.

Похожие принципы демонстрирует подход для создания сложного ПО, который называется реактивным программированием. Реактивное программирование — парадигма программирования, ориентированная на потоки данных и распространение изменений. Это означает, что должна существовать возможность легко выражать статические и динамические потоки данных, а также то, что нижележащая модель исполнения должна автоматически распространять изменения благодаря потоку данных.

К примеру, в императивном программировании присваивание a = b + c будет означать, что переменной a будет присвоен результат выполнения операции b + c, используя текущие (на момент вычисления) значения переменных. Позже значения переменных b и c могут быть изменены без какого-либо влияния на значение переменной a.

В реактивном же программировании значение a будет автоматически пересчитано, основываясь на новых значениях.

Современные табличные процессоры представляют собой пример реактивного программирования. Ячейки таблицы могут содержать строковые значения или формулу вида «=B1+C1», значение которой будет вычислено исходя из значений соответствующих ячеек. Когда значение одной из зависимых ячеек будет изменено, значение этой ячейки будет автоматически обновлено.

Другой пример — это языки описания аппаратуры (HDL), такие как Verilog. Реактивное программирование позволяет моделировать изменения в виде их распространения внутри модели.

Реактивное программирование предлагалось как путь для лёгкого создания пользовательских интерфейсов, анимации или моделирования систем, изменяющихся во времени.

К примеру, в MVC архитектуре с помощью реактивного программирования можно реализовать автоматическое отражение изменений из Model в View и наоборот из View в Model.

Также следует упомянуть такой подход к разработке ПО как Flow Based Programming (FBP) – программирование потоков данных.

Он основан на том, что программная реализация работает по принципам ориентированного графа: она состоит из узлов и дуг. По дугам происходит движение потоков данных. Узлы имеют вход и выход. Когда данные по дуге приходят в вход узла, то узел активируется и выполняет заданные в нем команды. Затем данные попадают из выходного узла на следующую дугу и по дуге могут попасть в другие узлы. Также узлы могут активироваться от внешних воздействий, что позволяет использовать такую методологию для асинхронного или событийно-ориентированного программирования.

Среди систем, основанных на этой методологии, можно выделить Matlab Simulink, Labview, TensorFlow.

LabView – система для построения моделей технических систем и устройств. Для этой цели LabView включает в состав множество специализированных библиотек реализованных компонентов, на базе которых возможно построение новых систем из конкретных технических областей.

Simulink – это графическая среда имитационного моделирования, позволяющая при помощи блок-диаграмм в виде ориентированных графов, строить модели динамических, дискретных, непрерывных и гибридных, нелинейных и разрывных систем. Simulink также ориентирован на решение задач моделирования сложных технических систем и представляет собой модельно-ориентированную систему проектирования. В первую очередь Simulink позволяет создавать графическую модель системы, собрав ее на основе стандартных базовых элементов, которой будет соответствовать конкретная математическая модель системы на основе систем обыкновенных дифференциальных уравнений или уравнений в частных производных.

TensorFlow представляет собой библиотеку с открытым исходным кодом, для построения программных реализаций численных методов. В основе TensorFlow лежат понятия теории графов, а также понятие граф потоков данных. В TensorFlow узлы представляют математические операции, а ребра – многомерные массивы данных (тензоры).

Графоориентированный подход очень похож на FBP. В нем, как и в FBP используется ориентированный граф, однако узлы представляют текущее состояние данных, а дуги представляют собой функции перехода, которые преобразовывают данные.

# Существующие решения задачи об интеграции ЛПР в графовую модель

В сфере интерактивных развлечений проблема воздействия пользователя на граф потоков данных стоит особенно остро. Один из примеров решения такой проблемы есть в Blueprints и подобных системах. Blueprints — это система визуального скриптинга Unreal Engine 4. Она является быстрым способом создания прототипов ПО для интерактивных развлечений. В этой системе есть узлы, которые настроены на нажатие клавиш и активируют соответствующее события. При срабатывании события выполняются все узлы, которые к нему привязаны. Таким образом, лицо принимающее решение само генерирует событие, которое изменяет путь обхода ориентированного графа. При этом ожидание событий от пользователя не блокирует проход по остальным узлам графовой модели.

То, что предлагается сделать для взаимодействия пользователя с графовой моделью сейчас – генерация пользователем события, которое будет привязано к функции перехода. Если проводить аналогии обработки графов в FBP и GBSE, то можно прийти к выводу, что предлагаемое решение для GBSE эквивалентно существующему решению для FBP. Из этого можно сделать вывод, что такое решение будет работоспособным.

# Список литературы

1. Соколов А.П., Першин А.Ю. Графоориентированный программный каркас для реализации сложных вычислительных методов. Программирование. – T.47, No5– 2019, c. 43-55.
2. Норенков И.П. Основы автоматизированного проектирования, М: Изд-во МГТУ им. Н. Э. Баумана, 2006. - 448 c.
3. McCormic B. Visualization in Scientific Computing // New York: Computer Graphics. 1987, 81 p.
4. US 2004/0056908 Al - Turbo Worx, Inc. (New Haven, CT, US) - Method and system for dataflow creation and execution (2004).
5. US 8,949,772 Bl - Amazon Technologies, Inc., (Reno, NV, US) – Dynamic model based software application development (2015).
6. Леонтьев Д.В., Тарасов В.Г., Харитонов И.Д. Модель обработки данных вычислительных экспериментов // Научный сервис в сети Интернет: труды XX Всероссийской научной конференции. М.: ИПМ им. М.В.Келдыша, 2018. - C. 373–386.
7. Zurek R.W, Martin L.J. GridPP: development of the UK computing grid for particle physics // New York: Journal of Physics G: Nuclear and Particle Physics, 2006, p. 1–20.
8. Алекперов А. Организация распределенных вычислений на базе GRID-технологии // Баку: Искусственный интелект. - 2011. - C. 6-14.
9. Farkas Z., Kacsuk P. P-GRADE Portal: A generic workflow system to support user communities // Elsevier. - Volume 27 (Future Generation Computer Systems), 2010, p. 455–465.
10. Godec, P., Pančur, M., Democratized image analytics by visual programming through integration of deep models and small-scale machine learning // Nat Commun. - Volume 10, 2019, C. 33-40.
11. Kudryavtsev A.O., Koshelev V.K., Izbyshev A.O. Development and implementation of the cloud system for solving problems of high // Proceedings of the Institute for System Programming of Russian Academy of Sciences, Volume 24, 2011, p. 13-33.
12. Tanenbaum, Andrew S. Distributed systems: principles and paradigms. Upper Saddle River, NJ: Pearson Prentice Hall, 2006, 702 p.
13. Erik Wilde, Cesare Pautasso. REST: From Research to Practice. // Springer Science & Business Media, 2011, 528 p.
14. hHDF5 Software Documentation [Электронный ресурс]. – Режим доступа: <https://support.hdfgroup.org/HDF5/doc/>
15. Белоушко, К. Е. Формат NetCDF как стандарт для обмена данными в атмосферных исследованиях // Тезисы II конференции «Базы данных, инструменты и информационные основы полярных геофизических исследований» (POLAR-2012), Троицк, ИЗМИРАН, 2012.
16. Как работать с кластерами через графический интерфейс COMSOL Desktop [Электронный ресурс]. – Режим доступа: <https://www.comsol.ru/blogs/how-to-run-on-clusters-from-the-comsol-desktopenvironment/>
17. Аветисян А.И., Грушин Д.А., Рыжов А.Г. Системы управления кластерами // Труды Института Системного Программирования РАН, Москва, 2002.