УДК 378:004
DOI: 10.25206/978-5-8149-3873-2-2024-354-360

Сетевая технология программирования визуальных интерфейсов
A network technology for visual interfaces programming

П. А. Васёв
Институт математики и механики им. Н.Н. Красовского УрО РАН, г. Екатеринбург, Россия

P. A. Vasev
N.N. Krasovskii Institute of Mathematics and Mechanics of the Ural Branch of the Russian Academy of Sciences, Ekaterinburg, Russia

Аннотация. В работе предлагается технология управления визуальными образами, такими как пользовательские интерфейсы и сцены визуализации. Технология базируется на сетевом взаимодействии. Это позволяет реализовать программные компоненты, отвечающие за визуализацию, на одних языках и библиотеках, а общую компоновочную и логическую части на других. Это в свою очередь обеспечивает возможность запуска проекта в разных средах (веб и настольные приложения), привлечение более широкого круга разработчиков, переносимость проекта между разными поколениями графических технологий.

***Ключевые слова*:** визуализация, архитектура программного обеспечения, пользовательские интерфейсы

Annotation. The paper proposes a technology for managing views, such as user interfaces and visualization scenes. The technology is based on network interaction, so it is possible to describe the software components responsible for direct visualization in some languages and libraries, and the general layout and logical parts in others. This makes it possible to launch the project in different environments (web and desktop applications), attract a wider range of developers, and portability of the project between different generations of graphics technologies.

***Ключевые слова*:** visualization, software architecture, graphical user interfaces

I. Введение и постановка задачи

Технологии графических интерфейсов [1] и библиотек стремительно меняются. Это, например, означает, что проект, написанный 10 лет назад, становится морально устаревшим. Под моральным устареванием понимается то, что появляются технологии с более логически выверенными программными интерфейсами, лучшими привязками к оборудованию, лучшей образностью, и т.п. Например, на смену когда-то впечатляющей технологии QML пришли новые технологии, такие как React или Kotlin Compose Multiplatform.

Но тоже самое произойдёт и с этими новыми технологиями. Важным аспектом многих подобных технологий является то, что они предлагают сквозное решение, которое охватывает все основные аспекты программ. Это означает, что если программу реализовывать с использованием подобных технологий, то она вскоре морально устареет вместе с этими технологиями. Это является проблемой.

Что делать? Человечество решает этот вопрос, постепенно двигаясь к разделению графических программ на слои. В целом, по мнению автора, такое разделение наблюдается следующее:

1. Графический слой – элементы управления, визуальные образы, т. е. компоненты непосредственно взаимодействующие с оконной системой и графическими библиотеками уровня Vulkan/OpenGL/etc.
2. Логический слой – абстракции более высокого уровня, в которых программируются прикладные модели. Этот слой опирается на графический слой, использует его в своих алгоритмах.
3. Компоновочный слой — увязывает элементы графических и логических частей в более крупные абстракции, формирует итоговый вид программы.
4. Дополнительные прикладные части и библиотеки.

Например, следующие технологии опираются на подобное разделение: среда X Window System, парадигма Model-view-controller, клиент-серверные веб-технологии и их вариации, такие как Ruby on Rails, и уже упоминавшийся QML.

Имеющиеся решения не идеальны. Как отмечено выше, многие технологии являются сквозными, т. е. предполагают реализацию всех слоёв в рамках одной экосистемы. Другие являются достаточно сложными для

взаимодействия слоёв, например веб-технологии уровня HTML+Javascript подразумевают для взаимодействия слоёв применение низкоуровневых протоколов (Websocket, Server sent events и т. п.).

Одним из кардинальных методов решения обозначенной проблемы является переход на другой уровень абстракции. Например, в среде SciVi [2] предлагается описывать предметную область в форме онтологии, а затем среда генерирует коды для всех вышеупомянутых слоёв и связи между ними.

Другим возможным решением является подход, когда слои разделены явно, и взаимодействуют с помощью сетевых технологий. Это позволяет описывать слои и даже части слоёв на разных языках программирования. А это, в свою очередь, позволяет составлять программы из разных языков и что самое главное – обновлять слои независимо друг от друга. При этом ключевое дополнительное требование – чтобы взаимодействие слоёв было удобным для программиста.

Этот подход реализуется, например, в платформе Логос и описан в работе [3].

Настоящая работа также посвящена исследованию этого подхода. Необходимо отметить, что каждая реализация подразумевает некоторую собственную программную модель (сущности, их взаимосвязи и взаимодействия). В работе представляется оригинальная модель, которая, по мнению автора, решает обозначенную проблему.

II. Теория: модель управления визуальными образами.

Сущности модели и их взаимодействие следующие. Компонента - это логический процесс в смысле Хоара [4]. Над компонентами вводится отношение "родитель-дети». Исполнитель – процесс операционной системы (ОС), работающий на некотором оборудовании. Каждая компонента выполняется в некотором заданном исполнителе.

Канал — сущность для передачи данных. Вводится операция записи сообщений в канал. Вводится операция подписки (реакции) на сообщения в канале. Каналы сопоставляются пространству имён, которое является общим (глобальным) для всех исполнителей. Каждой компоненте сопоставляется набор входных и выходных каналов. Компонента взаимодействует с внешним миром посредством этих каналов.

Связь между двумя каналами — специальный процесс, который производит передачу сообщений записываемых в первый канал во второй канал. Список операций в модели: Открытие канала: open_channel: channel_id → channel, где channel_id — имя канала в общем для всех исполнителей пространстве имён. Возвращает локальный объект для операций с каналом (локальный в смысле доступный в том исполнителе и в том языке программирования, в котором выполнена операция open_channel).Запись сообщения в канал: put: channel, msg → void, где channel — локальный объект для операций с каналом, msg — сообщение в некоторой форме. Технически в зависимости от реализации операция put может поддерживать передачу дополнительных уведомлений о ходе отправки сообщения, например посредством обещаний (promise).Создание реакции:react: channel, fn → void, где channel — локальный объект для взаимодействия с каналом, fn — функция, которая будет вызываться средой при (после) записи следующего значения в канал. В канал можно записать сообщение, находясь в одном исполнителе, а получать эти сообщения — в этом же или других исполнителях. При записи сообщения его получают все подписавшиеся стороны. Название реакция выбрано не случайно, оно соответствует термину "реакция" из модели Lingua Franca [5]. Есть и аналогичные термины, например "подписка на сообщения". Создание связи: create_link: source_id, target_id → local_link_objectгде source_id это имя канала-источника в общем для всех исполнителе пространстве имён, target_id — имя канала-приёмника, local_link_object — локальный объект связи для последующих операций с ней (удаления). После создания связи все записи в канал-источник приводят к копированию этих сообщений в канал-приёмник.

Создание дерева компонент в заданной точке исполнения: create_component: description, runner_id, target_id → root_id, где runner_id – идентификатор исполнителя, в котором следует создать компонент, target_id – идентификатор компонента, в список "детей" которого добавить создаваемый компонент, а description — описание создаваемого компонента. Это описание рекурсивно и определяет компонент и поддерево его вложенных компонент в рамках иерархии "родитель-дети". Структура описания description: type_id — идентификатор типа. Исполнитель должен уметь создавать компоненты этого типа .name — локальное имя объекта (необязательно) .params — список (словарь) значений параметров функции-конструктора компоненты, а также список значений, которые следует записать во входные каналы компонента после его создания .items — список описаний вложенных компонент (описания такой же структуры как description).Создаваемому компоненту назначается уникальный идентификатор root_id. Входные и выходные каналы, соответствующие компоненту, а также вложенные компоненты, получают идентификаторы вида "root_id/subitem_id". Код, который вызвал операцию create_component, получает в результате идентификатор root_id и может, используя его, обратиться к каналам компоненты и его вложенным компонентам. Компоненты, таким образом, представляются

для внешнего мира как набор входных и выходных каналов. Также извне можно обратиться к вложенным компонентами, которые в свою очередь также представляются как набор каналов, и т. д.

Дополнительно к перечисленным выше операциям в модели вводятся и обратные операции, которые в данном тексте пропущены для сокращения описания: закрыть канал, отменить подписку на сообщения, удалить связь, удалить дерево компонент. Порядок работы подразумевается следующий. Каким-либо образом запускаются процессы исполнители (на локальной машине или на наборе машин, связанных сетью). Часть этих исполнителей подразумеваются графические, способные создавать компоненты пользовательского интерфейса и (или) сцен визуализации. Другая часть — логические. Некоторый логический компонент (или просто функция) инициирует создание необходимых деревьев компонент в заданных исполнителях. Также он создаёт связи между каналами этих компонент. Дополнительно логический слой подписывается на сообщения в каналах графических компонент для получения сигналов от пользователя. Получая эти сигналы, логический слой формирует в итоге сообщения для входных каналов компонент пользовательского интерфейса, создаёт новые компоненты, и так далее.

Работа программы идёт сообразно алгоритму, описанному в логическом слое, а её визуальное представление реализуется в графическом слое. В результате становится возможным, например, написать исполнитель на языке QML в котором реализовать компоненты графического слоя (пользовательского интерфейса), а логический слой – в программе на языке Python. Можно затем заменить графический слой QML на слой на web-технологиях, при этом программа логического слоя останется неизменной. Также можно заменить и программу логического слоя, например, переписав её на другом языке. Можно заменить не весь логический слой, а его часть, и, например, вынести его в другой исполнитель / язык. Необходимо отметить, что связи между каналами это крайне существенный элемент, и он находится на том же уровне, что и компоненты. Программа в представленной модели является по сути направленным графом из процессов, ребра в котором — это связи между каналами процессов.Также необходимо отметить, что возможна высокоэффективная реализация связей. Например, пусть создаётся пара компонент на некотором исполнителе, и два их канала связываются. Эту связь можно технически реализовать таким образом, что запись в канал будет приводить к прямому вызову функций реакции на втором канале без сетевого взаимодействия. Это позволяет описывать наборы компонент и связей между ними внешним образом, а их выполнение при этом будет столь же эффективно как обычное взаимодействие кодов внутри процесса ОС.

III. Результаты экспериментов.

Была разработана прототипная среда, реализующая вышеуказанную модель и обеспечивающая сетевую коммуникацию для программ на языках: Javascript (в версиях для web и для nodejs), Python, C++, QML.

Было реализовано две программы: специализированная ГИС и программа визуализации параллельных вычислений. Рассмотрим результаты их разработки.

Специализированная ГИС была разработана в контексте того, что имелась задача и имелся большой технологический задел кодов на QML и Qt. Однако, по мнению автора, к этим технологиям следует выдвинуть ряд претензий (некорректность терминологии, устаревшая версия языка Javascript в QML, ряд неудачных дизайнерских решений, приводящих к трудновыявляемым ошибкам, например, возможность доступа к свойствам вне текущей области видимости выше по иерархии, другие неудобства), и поэтому нецелесообразно создавать новые продукты с их использованием. Было решено опробовать предлагаемую технологию. При этом для логического слоя был выбран язык Python по причине его распространённости. Ожидается, что к проекту на Python будет относительно легко подключать дополнительных разработчиков.

Графический слой был реализован на QML и Qt: объекты имеющегося технологического задела были обёрнуты в оболочки, и представлены как компоненты модели. Сущности и операции модели были реализованы в форме объектов C++ с мостами в QML посредством способов Qt (слоты, сигналы и прочее). Сетевая часть на C++ была реализована на базе библиотеки Mongoose [6] [7].

Примеры созданных компонент графического слоя (для реализации пользовательского интерфейса):

- Ряд (row) — компонент в роли контейнера, размещает вложенные компоненты горизонтально.

6 Этот метод реализации оказался очень громоздким. Его мотивация была получить "заодно" реализацию и для C++. Этого удалось достичь, но в совокупности с "мостами" в Qt/QML получился внушительный объем кода. Если делать проект заново, то автор предпочёл бы провести всю реализацию внутри QML, например на базе библиотеки QML Websocket.

7 Библиотека Mongoose оказалась крайне неэффективной. Например, в ней принимаемые и передаваемые данные копируются несколько раз в промежуточных буферах, прежде чем быть переданными программе или ОС. Поэтому для C++ рассматривается вариант переписать реализацию на библиотеке C++ Asio.

- Колонка (column) — компонент в роли контейнера, размещает вложенные компоненты вертикально.
- Окно (dialog) – компонент в роли контейнера, отображает вложенные компоненты в отдельном окне.
- Кнопка (button), текстовое поле (field), текстовая надпись (text), и т. п. – компоненты, реализующие элементы графического интерфейса пользователя.
- Набор картографических слоёв (map_group) — компонент для управления отображением картографической информации, взаимодействует с мышкой и клавиатурой пользователя и показывает содержимое вложенных компонент.
- Отображение растровых и векторных карт (show_image) — компонент картографического слоя, применяется как вложенный в map_group.
- Отображение векторных данных (show_vector) — компонент картографического слоя, применяется как вложенный в map_group.

Логический слой был реализован на языке Python 3. Слой определяет состав и группировку компонент графического слоя (см. выше), посылает данные в каналы этих компонент, реагирует на сообщения в их выходных каналах (например, движение мышки, клики по картографическим слоям, нажатие на кнопки, и т.п.)

Для работы представленной модели в среде Python была разработана специальная библиотека на базе модуля asyncio. Сущности и операции модели были реализованы в форме объектов. Пример использования кода Python показан на рис. 1.

Программа была успешно реализована и применяется на практике. Планируется разработка варианта графического слоя с применением веб-технологий, например, на базе библиотеки отображения карт OpenLayers.

```
############  создание таблицы из 10 полей и кнопки
### формирование описания description
lst1 = []
for x in range(0,10):
    d = gui.elem(type="row",params={"spacing":10},items=[
          gui.elem( type="text",params={"text":"Поле "+str(x)}),
          gui.elem( type="field",params={"value":""},"id"="field_"+str(x)),
        ])
    lst1.append(d)
lst1.append( gui.elem(type="button",params={"text":"ПОЕХАЛИ"},id="gobtn"))
d = gui.elem(type="column",params={"spacing":5},items=lst1)


### выполнение операции create_component на исполнителе gui
col2 = await gui.create_component( target_id="app",description=d )


############ реакция на нажатие кнопки
def go(msg):
    print("ЕДЕМ!")
col2.item("gobtn").open_channel("clicked").react( go )
```

Рис. 1. Пример кода на языке Python. Код формирует описание графических компонент, выполняет операцию по созданию компонент согласно этому описанию, и добавляет реакцию к выходному каналу одной из компонент.

Визуализация параллельных вычислений. Разрабатывается параллельная версия одного алгоритма обучения с подкреплением. На начальном этапе разработки оказалось, что версия имеет низкий коэффициент распараллеливания. Чтобы понять причины этого, было принято решение визуализировать её работу. Вид отображения был разработан следующий (см. пример на рис. 2):

- каждая задача (в смысле задач графа задач [6]) отображается отрезком вдоль оси OX. Положение отрезка на оси X соответствует времени начала и конца выполнения задачи, положение на оси Z соответствует номеру параллельного исполнителя;
- дополнительно изображаются передачи данных между процессами, поскольку передача может занимать значительное время, и это хотелось бы видеть. Параллельная версия использует схему мастер-рабочий, и коммуникация осуществляется только между мастером и рабочими. Поэтому чтобы не захламлять изображение, было решено показывать передачу вертикальными отрезками от плоскости рабочих Y=1 к плоскости Y=0, что означает пересылку данных на вход рабочему от мастера и в обратном направлении.

Графический слой реализован на веб-технологиях, на базе Javasript и DOM API. Трёхмерная составляющая реализована на базе библиотеки ThreeJS. Были разработаны компоненты:

- Контейнеры, элементы пользовательского интерфейса — реализованы такие же как в примере выше (для колонок и рядов использована технология CSS Flexbox).

- Рендеринг (render) — компонент, инкапсулирующий процесс рендеринга сцены.
- Сцена (scene) — компонент в роли контейнера, реализует древовидную структуру для компонент трёхмерной графики.
- Точки, отрезки, треугольники (points, lines, mesh) – компоненты трёхмерной графики, предназначены для отображения соответствующих графических примитивов.

Компоненты графических примитивов points, lines, mesh для управления составом векторных данных предоставляют на вход каналы двух видов: 1) по установке данных целиком, и 2) по модификации данных, например, по добавлению части данных. Это позволяет менять состав векторных данных, не пересылая их целиком каждый раз по сети.

Логический слой был реализован на языке Python 3. Это было обусловлено тем, что обучение (однопоточная и параллельная версии) были разработаны именно на этом языке. В уже имеющуюся программу были добавлены операции представленной модели для управления трёхмерной сценой.

А именно, реализация следующая. При начале вычисления задачи исполнитель замеряет текущее время. По завершению он вычисляет время, затраченное на задачу. Затем высылается сообщение в канал «решённые задачи». Специальный модуль логического слоя («рисователь») обрабатывает эти сообщения и формирует координаты вершин для сцены. Затем он пересылает эти координаты посредством сообщений в каналы «модификация данных» (см. выше) компонентам графического слоя, ответственным за рендеринг.

Программа запускается, запускает сетевую среду выполнения, переходит к вычислениям и показывает веб-ссылку для запуска визуализации. Пользователь может в любой момент открыть по этой ссылке визуализацию в браузере, программа по ходу работы обновляет графическое представление. Таким образом, была получена визуализация работы параллельного вычисления в режиме онлайн [7], т. е. по ходу вычисления.

Пример полученной визуализации показан на рис. 2. Эта визуализация была использована для понимания работы и последующей оптимизации параллельного алгоритма.

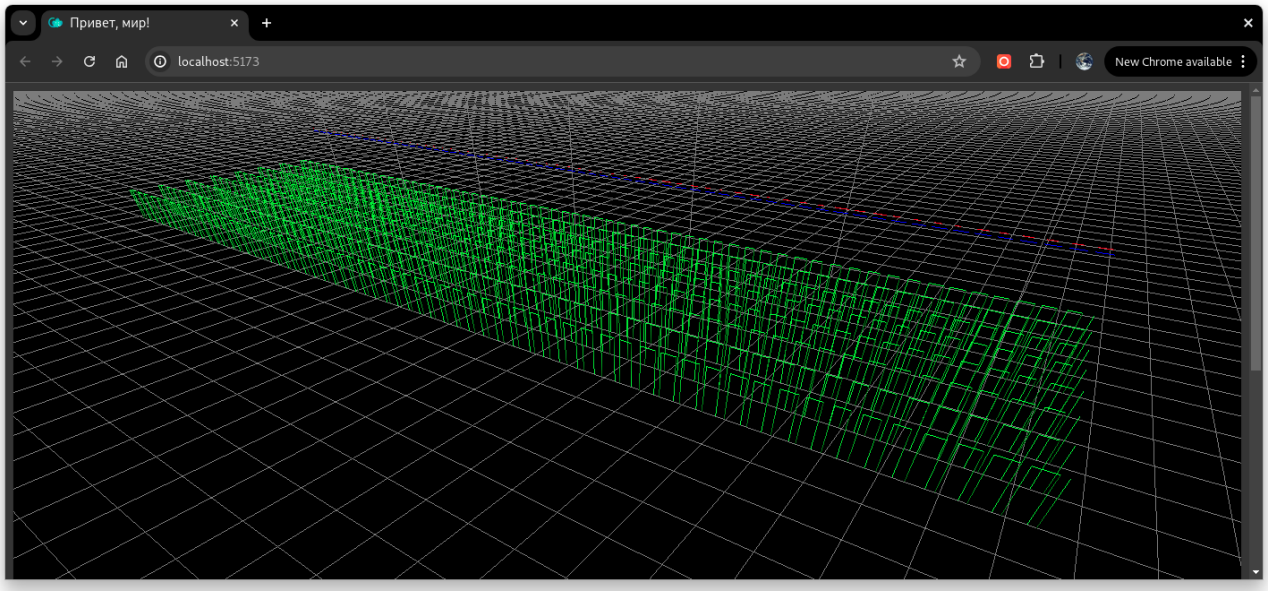

Рис. 2. Пример визуализации, полученной с помощью представленной технологии. Показана работа 8 рабочих процессов. Время идёт слева направо. Рабочие расположены каждый на своей оси Z (ближе-дальше к экрану). Горизонтальная линия на оси рабочего — выполнение задачи. Вертикальная — получение рабочим входных данных от мастера и высылка ему результата. На дальнем плане синие отрезки — ожидание мастером всех рабочих, красные — одновременное выполнение мастером оценки текущей ситуации.

IV. Обсуждение результатов.

Отличие предложенной модели от модели X Window System следующее. В последней компоненты графического слоя выполняются в основном процессе программы, и они отправляют графические примитивы в программы, работающие на графическом оборудовании (на «сервер» в терминологии X Window). В обратном направлении высылаются команды пользователя — сигналы от мыши и клавиатуры. В предложенной модели посылаются не графические примитивы, а высокоуровневые сообщения для компонент графического слоя,

которые работают в процессах на графическом оборудовании. В обратном направлении также движутся высокоуровневые сообщения («нажата кнопка», «изменено значение бегунка» и т. п.).

Среди отличий предложенной модели от модели Логос, представленной в работе [3], в первую очередь необходимо выделить следующие. В работе [3]:

– вместо возможности прямого доступа к каналам компонент, в описание компонент (см. поле description выше) вносятся связи локальных каналов компонент с глобальными каналами. Такой подход позволяет, в том числе, программировать динамику взаимодействия графических компонент сразу на уровне описания посредством передачи сообщений между компонентами опосредовано через глобальные каналы. Направления связей можно указывать в обе стороны — от локальных к глобальным и наоборот.

– Реакции можно создавать только в привязке к глобальным каналам.

– Поддерживаются «теги», которые позволяют включать каждую компоненту в несколько множеств, и поддерживаются групповые операции над элементами таких множеств. Например, команда «установить значение поля «value» для всех компонент, входящих в заданное множество.

– На уровне реализации предоставлена возможность загружать описание компонент в форме текста на языке JSON. Реализация логического слоя сделана для языка Python. Поддерживаются версии графического слоя для веб и настольных платформ, которые реализуют большое количество компонент.

Работа [3] активно применяется на практике, но, к сожалению, она недоступна для сторонних разработчиков.

Отличие предложенной модели от модели Yandex Divkit следующее. В Divkit предложена богатая модель встроенной динамики компонент графического слоя. Это сделано потому, что подразумевается большая автономность этого слоя от логического слоя, подход «загрузи и работай». В модели настоящей работы подразумевается более тесное, постоянное взаимодействие компонент графического и логического слоёв.

V. Выводы и заключение

В работе предложено вынести интерфейс взаимодействия графического и логического слоёв программ на сетевой уровень. Это позволяет:

1. Не зависеть от развития и устаревания технологий. При устаревании технологии, на которой написан слой, его можно переписать. В первую очередь это касается графического слоя. Но и логический слой можно переписывать — постепенно. Предложенная технология такова, что обеспечивает возможность работы частей слоя одновременно в разных процессах ОС и на разных языках программирования.
2. Удобно переключать реализации графического слоя, если это ценно для задачи. Например, менять "веб" на "настольную" версию, или варианты с OpenGL на Vulkan.
3. Применять наиболее удобные и подходящие в контексте задачи языки для реализации слоёв. Например, графический слой можно реализовать на веб-технологиях, а логический на Python.
4. Запускать графические программы в многомашинной среде "естественным" образом, благодаря сетевому протоколу взаимодействия. Например, графический слой может выполняться на исполнителях на машинах с мониторами, а логический на исполнителях на других машинах, в том числе на узлах суперкомпьютера.

Технология протестирована на двух прикладных задачах, и показала свою работоспособность. Вместе с тем её можно развивать, некоторые варианты развития показаны в разделе "Обсуждение результатов".

Прототипная реализация технологии выполнена на базе системы параллельного программирования PPK (https://github.com/pavelvasev/ppk).

Автор благодарит многоуважаемых коллег, сотрудников Академии наук, Корпорации Росатом, Уральского федерального университета и других организаций – за поддержку, плодотворное обсуждение и синтез идей.

Список литературы

1. В.Л. Авербух, Развитие человеко-компьютерного взаимодействия // Научная визуализация 12.5: 130 - 164, DOI: https://doi.org/10.26583/sv.12.5.11

2. Ryabinin K., Chuprina S. Ontology-Driven Edge Computing // Lecture Notes in Computer Science. – Springer, 2020. – Vol. 12143. – P. 312–325. DOI: https://doi.org/10.1007/978-3-030-50436-6_23.

3. Губайдулина Е. А. и др., Инструмент разработки специализированных пользовательских графических интерфейсов для проведения комплексного математического моделирования средствами пакета программ ЛОГОС // Тезисы XIX Международной конференции «Супервычисления и математическое моделирование», 20 - 24 мая 2024г., г. Саров, стр. 68. URL: www.cv.imm.uran.ru/e/3241932

4. C.A.R. Hoare, Communicating Sequential Processes , December 4, 2022.

5. Marten Lohstroh, Christian Menard, Soroush Bateni, and Edward A. Lee. 2021. Toward a Lingua Franca for Deterministic Concurrent Systems. ACM Trans. Embed. Comput. Syst. 20, 4, Article 36 (July 2021), 27 pages. DOI: doi.org/10.1145/3448128

6. Васёв П.А., Визуализация работы алгоритма планирования параллельных задач // Труды 33-ей Международной конференция по компьютерной графике и машинному зрению ГрафиКон 2023, 19-21 сентября 2023 г., город Москва. С. 341-353. DOI: https://doi.org/10.20948/graphicon-2023-341-353

7. Pavel Vasev, Analyzing an Ideas Used in Modern HPC Computation Steering // 2020 Ural Symposium on Biomedical Engineering, Radioelectronics and Information Technology (USBEREIT), Yekaterinburg, Russia, 2020, pp. 1-4, doi: https://doi.org/10.1109/USBEREIT48449.2020.9117685