Проектирование и архитектура унифицированного представления событийно-непрерывных моделей

**Аннотация:** В статье рассматривается разработка унифицированного формата представления гибридных (событийно-непрерывных) моделей и архитектуры распределённой системы моделирования на его основе. Предлагаемое решение состоит из трёх компонентов: клиентского интерфейса, сервера-слушателя и сервера-вычислителя. Для взаимодействия используется gRPC с описанием интерфейсов через Protocol Buffers, а сами сервисы упакованы в контейнеры Docker. Описываются основные требования к формату моделей (поддержка дискретных и непрерывных компонентов, модульность, переносимость, однозначность), логическая структура данных (сущности «модель–состояние–событие–переменная») и преимущества предложенного подхода – в частности, высокая масштабируемость, отказоустойчивость и гибкость интеграции. Разработанное решение упрощает совместное использование моделей на разных платформах и позволяет динамически масштабировать вычислительные ресурсы при реализации сложных имитационных экспериментов.

**Ключевые слова:** событийно-непрерывные модели, унифицированное представление, распределённая система, gRPC, Protocol Buffers, Docker, масштабируемость, отказоустойчивость.

# Введение

Симуляция сложных технических и социально-экономических систем часто требует учёта как непрерывных процессов (описанных дифференциальными уравнениями), так и дискретных событий (резких переключений, срабатываний датчиков, отказов оборудования) [5]. Такие гибридные модели (событийно-непрерывные модели, СНМ) находят применение в промышленности, телекоммуникациях, киберфизических системах и управлении (например, моделирование автоматизированных производственных линий, сетевого трафика или систем «Интернет вещей» При этом в современных вычислительных средах растёт интерес к распределённым архитектурам вместо монолитных решений: это обеспечивает горизонтальное масштабирование, гибкое использование облачных ресурсов и упрощение интеграции компонентов разных вендоров.

В таких условиях унифицированное представление событийно-непрерывных моделей играет ключевую роль: оно должно обеспечить совместимость моделей между различными вычислительными платформами и ускорить внедрение новых модулей в систему [4][8]. Формализация единого формата существенно упрощает обмен моделями и результатами их вычислений между сервисами и подразделениями, а также повышает воспроизводимость экспериментов [6]. Для реализации этого подхода предлагается распределённая архитектура, в которой gRPC используется в качестве высокопроизводительного протокола удалённого вызова процедур. По словам разработчиков gRPC, клиентское приложение может вызывать метод на удалённом сервере так же, как на локальном объекте, что упрощает создание распределённых сервисов [2]. При этом в качестве интерфейсного языка и формата обмена сообщениями применяются Protocol Buffers – зрелый механизм сериализации структурированных данных от Google [2].

Современные тенденции контейнеризации (в частности, Docker) позволяют упаковывать сервисы вместе со всеми зависимостями в лёгкие изолированные контейнеры, что обеспечивает консистентную среду выполнения для приложения независимо от машины-хоста [1][11]. Это существенно упрощает развёртывание, версионирование и масштабирование вычислительных компонентов. В итоге создание единого формата моделирования в связке с микросервисной архитектурой даёт стратегическое преимущество: система становится более гибкой при интеграции, а при росте нагрузки – масштабируемой и отказоустойчивой.

Цель данной работы – разработать унифицированное представление событийно-непрерывных моделей и описать его реализацию в распределённой системе с использованием технологий gRPC, Protocol Buffers и Docker. Для достижения этой цели решаются следующие задачи: сформулировать требования к формату моделей (поддержка дискретных событий и непрерывных процессов, модульность, переносимость, корректный обмен данными), спроектировать трёхуровневую архитектуру (клиент–сервер-слушатель–сервер-вычислитель) с протоколами взаимодействия, реализовать прототип и оценить его характеристики.

# Основные требования к представлению СНМ

Ключевым требованием к унифицированному формату является гибридность: он должен однозначно описывать и непрерывные изменения системы, и мгновенные дискретные переходы между состояниями. Поэтому представление включает в себя информацию о параметрах системы, которые изменяются плавно во времени, задающихся формулами вида (1), и о событиях, задающих резкие переключения.

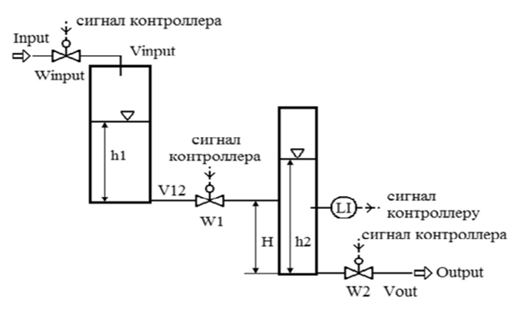


Рисунок Пример системы, совмещающей динамические и дискретные характеристик

Одним из типичных примеров гибридной системы является конфигурация с двумя резервуарами, оснащёнными инерционными клапанами, представленная на рисунке 1. Данная система включает два резервуара, соединённых трубопроводной сетью с клапанами, управление которыми осуществляется посредством контроллера. В состав системы входят: клапан, регулирующий подачу жидкости в первый резервуар; клапан, контролирующий выход жидкости из второго резервуара; а также регулирующий клапан, установленный между резервуарами. Рассматривается режим функционирования, при котором изначально жидкость поступает исключительно в первый резервуар, затем с определённой задержкой открывается клапан второго резервуара. Впоследствии система переходит в режим автоматического регулирования уровня жидкости во втором резервуаре путём управления выходным клапаном. Все клапаны открываются и закрываются постепенно — с плавным изменением расхода между двумя фиксированными значениями при постоянной скорости до завершения переходного процесса. Примеры некоторых дифференциальных уравнений данной модели можно найти в системе уравнений для уровней воды в баках (2), при этом в этой системы присутствуют и строго дискретный параметры, как скорость протекания воды между баками (3).

(2)

(3)

Кроме того, формат должен обеспечивать формальную однозначность и независимость от платформы: он должен быть нейтральным к операционной системе, языкам программирования и СУБД, что облегчает обмен моделями между разнородными компонентами. Это достигается с помощью чётких схем данных, определённых через сериализуемые сообщения Protocol Buffers, и модульной структуры, при которой одно и то же описание может обрабатываться на клиенте и на вычислительном сервере без преобразований.

Другие важные требования включают поддержку различных типов данных (булевых флагов, перечислений, вещественных параметров и т.д.), возможность динамического обновления модели (например, добавление новых событий или переменных во время эксперимента), а также гарантии корректного обмена данными в распределённой системе. Для этого формат данных должен содержать служебную информацию (идентификаторы моделей, версионирование, метаданные) и механизм валидации, позволяющий убедиться в консистентности полученных структур [9]. В итоге унифицированное представление должно позволять без дополнительных преобразований передавать описания СНМ между различными компонентами системы через протокол gRPC, включая возможность сериализации в JSON или Protobuf.

Основные требования к унифицированному представлению СНМ:

* Поддержка гибридной природы: обеспечение описания как непрерывных процессов, так и дискретных событий.
* Формальная однозначность: строгая схема данных (например, на основе Protobuf), гарантирующая корректную интерпретацию без неоднозначностей.
* Модульность и переносимость: нейтральность к платформе и СУБД; возможность использования формата в разных языковых и аппаратных контекстах.
* Поддержка различных типов данных: дискретные и непрерывные параметры, составные структуры, механизмы версии и валидации.
* Совместимость с RPC-протоколами: включение полей и механизмов, необходимых для обмена данными через gRPC (метаданные, идентификаторы сессий и т.д.).

Все перечисленные требования направлены на повышение интероперабельности и масштабируемости системы. Формат СНМ служит общим языком описания сложных процессов, что снижает затраты на интеграцию новых модулей и ускоряет разработку моделей.

Архитектура распределённой системы

Предложенная архитектура моделирования СНМ состоит из трёх основных компонентов:

Клиентский интерфейс – отвечает за приём модели от пользователя и отправку её в систему. Как правило, это веб- или настольное приложение, которое формирует запросы (например, в формате JSON или Protobuf) к серверу-слушателю.

Сервер-слушатель – gRPC-сервис, принимающий входящие запросы от клиента. Он обрабатывает описание модели, выполняет первичную валидацию и инициацию вычислительных задач. Сервер-слушатель может порождать или назначать вычислитель (сервер-вычислитель) для выполнения тяжёлых расчётов. Таким образом, он выполняет роль координатора и балансировщика нагрузки. Использование gRPC позволяет организации клиент-серверного взаимодействия: клиент может вызывать методы этого сервера удалённо, как если бы они выполнялись локально [2].

Сервер-вычислитель – компонент, выполняющий собственно моделирование, решая уравнения непрерывной части и обрабатывая события согласно логике модели. Вычислитель взаимодействует с сервером-слушателем и/или с клиентом через gRPC: он может получать новые задачи, отправлять промежуточные результаты или уведомления об окончании расчёта. Каждый вычислитель упакован в отдельный контейнер Docker, что позволяет легко масштабировать их число в зависимости от нагрузки. Упаковка в контейнеры гарантирует, что среда выполнения является идентичной на любом хосте [1]. В сочетании с оркестрацией (например, Kubernetes) это даёт возможность динамически разворачивать или удалять вычислительные узлы без простоя системы [12].

Изображение выглядит как диаграмма, текст, линия, План

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок Архитектура распределённой системы моделирования СНМ (клиент–сервер-слушатель–сервер-вычислитель)

Взаимодействие между компонентами организовано через четко определённые интерфейсы gRPC [7]. С помощью Protocol Buffers задаются схемы сообщений (например, запрос на вычисление следующего шага моделирования, ответ с результатами или статусом) и сервисов. Благодаря этому клиент может, например, вызвать метод ComputeModel (ModelData) на удалённом вычислителе так же легко, как обычную функцию. Использование Protobuf обеспечивает эффективную бинарную сериализацию данных и строгое согласование версий: схема может эволюционировать, сохраняя совместимость старыми клиентами [14]. В частности, gRPC и Protobuf позволяют автоматически генерировать код-клиент и код-сервера на разных языках, что упрощает расширение системы [13].

Таким образом, архитектура системы обеспечивает высокую масштабируемость и отказоустойчивость. Новые серверы-вычислители могут появляться по мере необходимости (например, при появлении в очереди новых модельных задач), а сбой одного из них не влечёт за собой крах всей системы – остальная часть продолжает работать. Кроме того, такой подход упрощает интеграцию сторонних сервисов: достаточно реализовать gRPC-интерфейс протокола обмена, чтобы подключить новый компонент. Широкое распространение gRPC и контейнерных технологий подтверждается на практике: например, по мнению экспертов, использование Docker обеспечивает консистентную среду для приложения [1], а gRPC уже становится популярным выбором для построения сервисной архитектуры [1].

# Логическая структура данных

Для описания событийно-непрерывных моделей в базе данных или других хранилищах используется следующая логическая схема. Основные сущности:

Модель – обобщённое описание системы или процесса. Модель включает параметры и начальные условия, описывающие её конфигурацию.

Состояние – конкретная конфигурация модели в момент времени. Каждое состояние характеризуется набором значений переменных.

Событие – мгновенное изменение состояния системы. Событие фиксирует переход системы из одного состояния в другое (например, «датчик сработал», «узел вышел из строя», «изменился режим работы»). Оно содержит информацию о том, какие переменные и как поменялись. События позволяют отслеживать историю эволюции системы и служат точками синхронизации между дискретной и непрерывной частями.

Переменная – параметр модели, принимающий значения во времени. Переменные могут быть непрерывными (например, температура, давление, скорость – описываются числами с плавающей запятой и изменяются согласно дифференциальным уравнениям или дискретными (например, статус устройства, перечисляемые коды режимов или булевы флаги – имеющие ограниченный набор значений и меняющиеся по событиям). Также предусматриваются составные типы данных (записи, списки), чтобы объединять взаимосвязанные параметры.

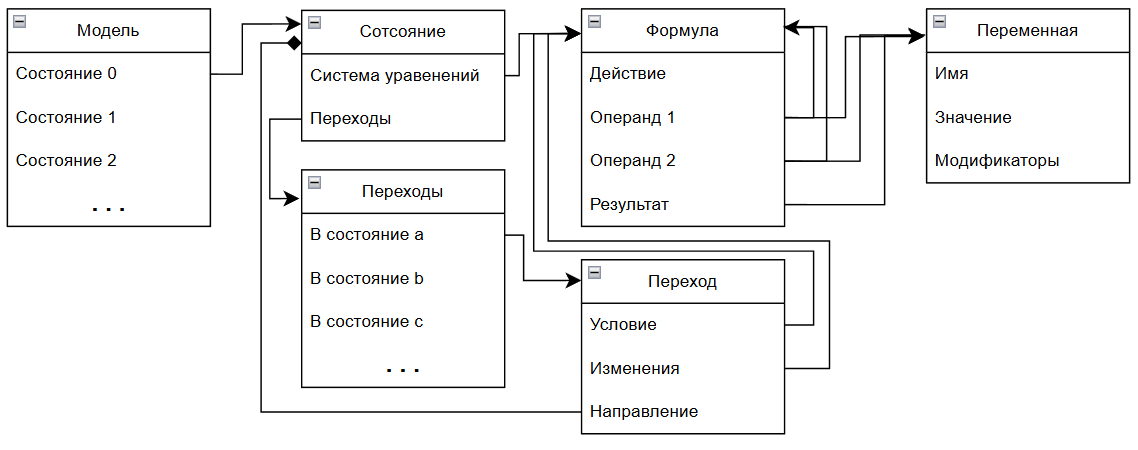


Рисунок Логическая структура данных СНМ: сущности «модель–состояние–событие–переменная» и их связи

Такое разделение на сущности «модель–состояние–событие–переменная» гарантирует целостность описания и удобство хранения. В схеме поддерживаются связи (внешние ключи) между моделями, состояниями и событиями, что предотвращает аномалии и дублирование данных. Для масштабируемости используются методы индексации (например, по временным меткам и идентификаторам моделей), а также возможно распределённое хранение (шардирование) исторических состояний. Выбор подходящего типа данных для каждой переменной (целое, вещественное, перечисление) критически влияет на производительность. Дискретные события описываются компактно, что упрощает поиск ключевых переключений, а непрерывные параметры хранятся в оптимизированных структурах для быстрого вычисления по ним.

В итоге логическая схема данных позволяет гибко моделировать гибридные процессы: дискретные сущности фиксируют резкие переходы системы, а непрерывные параметры отражают «движение» системы между событиями. Например, в технической системе событие «поломка узла» представляется отдельной записью (дискретно), а изменения рабочих характеристик узла перед и после отказа описываются непрерывными переменными. Такая комбинированная модель даёт полное представление об объекте и позволяет эффективно выполнять численное моделирование.

# Технологии взаимодействия и реализации

В разработанном программном комплексе применены современные технологии, обеспечивающие заявленные требования:

gRPC и Protocol Buffers: gRPC используется как фреймворк RPC поверх HTTP/2, обеспечивая низкие сетевые задержки и поддержку двунаправленных потоков. Интерфейсы сервисов описаны в файлах «.proto», где задаётся структура сообщений (моделей, состояний, команд) и сервисов (методов вызова).

Изображение выглядит как диаграмма, текст, зарисовка, шаблон

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок Порядок обмена сообщениями в системе

Преимущество Protocol Buffers – строгость схем и высокая скорость сериализации [2]. Сгенерированный код-клиент и код-сервера гарантируют корректность взаимодействия и облегчают поддержку нескольких языков.

Контейнеризация (Docker): Каждый компонент (сервер-слушатель, сервер-вычислитель) упакован в Docker-контейнер. Контейнеры запускаются на единой системе оркестрации, что обеспечивает консистентную окружение и лёгкость развёртывания [1]. Благодаря этому микросервисы могут быть быстро масштабированы – запускаются дополнительные экземпляры вычислителя при возрастании нагрузки, а неиспользуемые контейнеры автоматически удаляются. Docker-контейнеры также изолируют окружение приложения, что упрощает отладку и гарантирует одинаковую работу на разных серверах.

Изображение выглядит как диаграмма, зарисовка, План, Технический чертеж

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок Концепция контейнерной оркестрации (Docker) для системы моделирования СНМ

Механизмы балансировки и отказоустойчивости: В систему интегрированы привычные подходы распределённых вычислений [10]. Например, сервер-слушатель может распределять задачи между несколькими вычислителями в зависимости от их загруженности или географического расположения. При сбое любого контейнера-вычислителя остальная часть системы продолжает работу – оставшиеся узлы обрабатывают очередь заявок. Хранение промежуточных результатов может вестись в надёжном хранилище (Redis, Cassandra и т.д.) для обеспечения персистентности и восстановления после падений. Благодаря этому система демонстрирует высокую отказоустойчивость: выход из строя одного узла минимально влияет на общее выполнение задач.

Примерная последовательность действий в реализации такова: клиент отправляет описание модели серверу-слушателю; сервер-слушатель запускает gRPC-вычислитель в контейнере (или назначает уже запущенный) и передаёт ему данные по gRPC; вычислитель выполняет расчёты (численное решение уравнений + обработка событий) и периодически отсылает результаты или статусы обратно клиенту или в журнал. При этом вся коммуникация формально описана в «.proto» и логика обмена стандартна, что обеспечивает расширяемость (к примеру, легко добавить новые типы запросов – новые RPC-методы – без изменения существующих интерфейсов).

# Преимущества разработанного решения

Предложенная архитектура и формат данных обеспечивают целый ряд положительных свойств системы моделирования:

Масштабируемость: благодаря микросервисному подходу и контейнеризации система легко масштабируется по горизонтали. Новые вычислительные узлы на Docker можно развернуть автоматически, если нагрузка возрастает. Такой подход значительно улучшает производительность при большом объёме моделируемых данных и при большом числе параллельных экспериментов.

Отказоустойчивость: Разбиение системы на независимые сервисы повышает надёжность. Сбой одного компонента не приводит к полному отказу: например, в случае падения одного вычислителя запросы перенаправляются на другие активные контейнеры. Также возможна реализация механизмов повторной отправки запросов и резервирования данных (снэпшоты состояний), что соответствует современным требованиям к устойчивости распределённых систем.

Гибкость интеграции: Унифицированное представление СНМ обеспечивает лёгкую интеграцию с внешними сервисами и инструментами. Любой внешний модуль, поддерживающий gRPC и Protobuf, может использовать формат без трансформаций. Это позволяет подключать к системе сторонние симуляторы или аналитические модули (например, машинного обучения для адаптивного управления), расширяя функционал комплекса.

Портативность и управляемость: Контейнеризация гарантирует единообразие окружения при разработке, тестировании и внедрении. Благодаря этому архитектура остаётся независимой от конкретной ОС и конфигурации хост-систем. В Docker-контейнеры можно встроить все необходимые библиотеки (например, математические библиотеки для решения дифференциальных уравнений), не беспокоясь о конфликтах версий. Это упрощает сопровождение и обновление системы.

Благодаря совокупности перечисленных факторов разработанное решение соответствует современным стратегическим требованиям к цифровым технологиям: оно обеспечивает высокую производительность моделирования, способен быстро адаптироваться к возрастанию объёмов данных и демонстрирует надёжность при распределённых расчётах. Опыт показывает, что использование gRPC и контейнеров действительно ускоряет разработку и развёртывание распределённых приложений [1][2], что подтверждает практическую значимость выбранного подхода.

# Заключение

В данной статье предложены ключевые принципы проектирования унифицированного представления событийно-непрерывных моделей и распределённой системы их обработки. Унификация формата позволяет описывать гибридные системы целостно, а распределённая трёхуровневая архитектура (клиент–gRPC-сервер–вычислитель) обеспечивает масштабируемость и отказоустойчивость. Использование технологий gRPC/Protocol Buffers упрощает межкомпонентное взаимодействие, а контейнеризация Docker обеспечивает стабильность и гибкость развёртывания. Результатом является высокопроизводительная вычислительная платформа, пригодная для имитационного моделирования сложных систем. Перспективные направления дальнейшей работы включают интеграцию адаптивных алгоритмов балансировки нагрузки, расширение формата моделирования новыми типами процессов и тестирование системы в условиях реального применения.

# Список литературы

1. Мело Т. Running a gRPC service with Docker [Электронный ресурс] // HackerNoon. 2023. URL: https://[1]/running-a-grpc-service-with-docker (дата обращения: 15.04.2025).
2. gRPC overview [Электронный ресурс] // Google Cloud API Gateway Documentation. 2024. URL: https://[2]/api-gateway/docs/grpc-overview (дата обращения: 15.04.2025).
3. Бессонов А.В. Символьная спецификация и анализ программных моделей гибридных систем [Текст] : автореф. дисс. … к.т.н. / А.В. Бессонов. – Новосибирск : НГТУ, 2016.
4. Зайглер Б. П. Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems: монография. – New York: Academic Press, 1974. – 336 с.
5. Фрицсон П. Principles of Object‑Oriented Modeling and Simulation with Modelica 3.3: A Study of the Language and its Applications: монография. – Chichester ; Hoboken: Wiley‑IEEE Press, 2014. – 520 с.
6. Branicky M. S., Borkar V. S., Mitter S. K. A Unified Framework for Hybrid Control: Model and Optimal Control of Hybrid Systems // IEEE Trans. Automat. Control. – 1998. – Т. 43, № 4. – С. 531–546.
7. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) // IEEE Std 1516‑2010. – Piscataway (NJ): IEEE, 2010. – 183 с.
8. Modelica Association. Modelica 3.4: A Unified Object‑Oriented Language for Systems Modeling / Modelica Assoc. – Linköping: Modelica Assoc., 2017. – 1 200 с.
9. OMG. Functional Mock‑up Interface for Model Exchange and Co‑Simulation, Version 2.0 / Object Mgmt. Group. – Needham (MA): OMG, 2014. – 118 с.
10. Boutellier S., Lehtonen M., Pluymers B. Cloud‑based Simulation: New Models for Simulation as a Service // Simulation Modelling Practice and Theory. – 2015. – Т. 59. – С. 36–52.
11. Docker Inc. Docker Documentation. – [Электронный ресурс]. – Режим доступа: https://docs.docker.com/ (дата обращения: 15.04.2025).
12. The Kubernetes Authors. Kubernetes Documentation. – [Электронный ресурс]. – Режим доступа: https://kubernetes.io/docs/ (дата обращения: 13.04.2025).
13. Indrasiri K., Kuruppu D. gRPC: Up and Running: Building Cloud Native Applications with Go and Java for Docker and Kubernetes: практическое руководство. – Sebastopol (CA): O’Reilly Media, 2020. – 202 с.
14. Google. Protocol Buffers: Developer Guide. – [Электронный ресурс]. – Режим доступа: https://developers.google.com/protocol-buffers/docs/overview (дата обращения: 03.02.2025).