Введение

В данной версии LifeGraph OS (LGOS) система рассматривается как единая распределённая виртуальная машина, работающая на уровне сети. Пользователь не имеет локальной автономной копии данных и не может существовать вне сети LGOS. Браузер пользователя является вычислительным и хранилищным узлом, а глобальное состояние LGOS определяется совокупностью всех активных узлов и протоколом согласования.

1. Базовая модель: сеть как виртуальная машина

LGOS реализует модель, в которой вся сеть узлов выступает как одна виртуальная машина. Каждый узел сети представляет собой экземпляр LGOS в браузере пользователя. Узел хранит и обрабатывает лишь часть общего графа, но логически этот граф един. Узлы связаны через P2P-протоколы и обмениваются операциями над графом, поддерживая согласованное глобальное состояние.

Физически узлы распределены по различным устройствам и сетям, но на логическом уровне они образуют единую вычислительную ткань. Код LGOS определяет правила, по которым узлы:

– хранят фрагменты графа; – обмениваются изменениями; – выполняют вычисления (агенты, запросы, трансформации); – восстанавливают состояние после отказов и отключений.

1. Глобальное состояние: распределённый граф

Основой данных LGOS является монолитный с точки зрения логики граф. Он не хранится целиком ни на одном узле, но каждый узел хранит некоторую его часть. Узлы являются хранителями подграфов и реплик. Логический инвариант: любой элемент графа имеет единую логическую идентичность и единую консистентную историю изменений в рамках протокола.

Доступ к графу реализуется через операции (events, mutations). Узлы не оперируют сырыми снапшотами, а применяют и распространяют операции. Это создаёт основу для согласования через CRDT-подобные структуры или журнал операций с гарантией сходимости.

Каждое изменение над графом формализуется как операция над узлом и/или связью: создание, обновление, удаление, изменение метаданных, изменение структуры связей. Операции идентифицируются глобальными ID, могут иметь версионность и временные метки. Узлы обмениваются операциями по принципу gossip-протокола: каждое изменение постепенно достигает всех заинтересованных реплик.

1. Модель согласованности и единичности состояния элемента

LGOS не реализует жёсткий глобальный «лок» на элемент, чтобы не блокировать сеть и не зависеть от минимальных задержек. Вместо этого используется модель eventual consistency с логическим единым состоянием. Это означает, что в каждый момент времени разные узлы могут иметь различные префиксы журнала изменений для одного и того же элемента, но протокол гарантирует, что при обмене операциями все честные узлы в итоге получат одинаковую версию состояния.

Единичность состояния элемента достигается не через централизацию, а через:

– глобально уникальные идентификаторы элементов; – глобально уникальные идентификаторы операций; – детерминированные правила применения операций (коммутативные, идемпотентные, ассоциативные операции в духе CRDT); – строгую модель причинности и разрешения конфликтов.

За счёт этого на логическом уровне у элемента существует одна итоговая версия, к которой сходятся все реплики. При этом сеть не блокируется на время согласования, поскольку каждый узел работает с текущим локальным представлением состояния.

1. Узел LGOS в браузере пользователя

Каждый браузерный экземпляр LGOS является полноценным узлом распределённой виртуальной машины. Узел включает в себя несколько ключевых компонентов.

Во-первых, это runtime ядро LGOS, отвечающее за:

– хранение локального фрагмента графа (подграф + реплики); – ведение журнала операций, известного узлу; – подписки на изменения и реактивное обновление представлений; – взаимодействие с P2P-слоем.

Во-вторых, это вычислительный слой: агенты и обработчики запросов. Они могут выполняться в браузере с использованием WebAssembly и WebGPU. Узел может брать на себя вычислительные задачи для сети, если пользователь дал согласие делиться ресурсами. Вычисления включают обработку запросов к графу, запуск языковых моделей, выполнение агентных сценариев, анализ и трансформацию данных.

В-третьих, это слой интерфейса. UI является лишь проекцией глобального графа, а не отдельным набором сущностей. Все представления (редактор, таймлайн, карта, структура проекта) являются вычислимой функцией от текущего известного узлу состояния графа.

1. Сетевой слой и P2P-протоколы

LGOS использует P2P-сетевой слой поверх WebRTC или аналогичных технологий. Потребности сети включают:

– установление P2P-соединений между узлами; – маршрутизацию сообщений и операций; – gossip-распространение изменений графа; – обнаружение и подключение новых узлов; – возможность передачи вычислительных задач между узлами.

Для начального соединения используется сигнальный сервис (signaling server). Это не традиционный бэкенд в смысле хранения и вычислений, а вспомогательный узел, который помогает браузерам обменяться сетевой информацией (ICE-кандидаты, SDP) и установить прямое соединение. Этот сервис не является единой точкой истины и может быть реплицирован или заменён.

После установления соединения узлы обмениваются операциями над графом напрямую. При необходимости данные могут проксироваться через релей-узлы, если прямое соединение невозможно.

1. Распределённые вычисления и использование ресурсов пользователей

LGOS предусматривает, что каждый активный узел может делиться своей вычислительной мощностью с сетью. Если у пользователя открыто окно приложения, соответствующий узел может выполнять часть задач сети: обработку запросов, запуск агентов, индексирование данных, генерацию представлений, дистрибуцию патчей.

Механизм распределённых вычислений включает:

– протокол назначения задач (task assignment), – модель доверия и верификации результатов, – эвристики по нагрузке и приоритетам, – учёт согласия пользователя и ограничений по ресурсам.

Результаты вычислений могут быть переизбыточно проверены другими узлами или валидирующими узлами, чтобы исключить злонамеренное или некорректное поведение.

1. Репликация данных и узлы-маяки

Для повышения надёжности LGOS использует модель репликации данных. Определённые узлы могут выступать в роли «маяков» (сидов) для отдельных пространств: личных, проектных, гильдийных. Узлы-маяки держат полный или приоритетный подграф и обеспечивают доступность данных тогда, когда обычные клиентские узлы временно не активны.

Узлы-маяки не являются центральными серверами в классическом смысле, а выступают как постоянно доступные участники сети. Они могут управляться организациями, сообществами или самим проектом LGOS. Репликация на несколько маяков и на клиентские узлы формирует отказоустойчивую структуру хранения.

1. Обновление системы и контроль разработчика

Поскольку LGOS не опирается на централизованный бэкенд, управление системой осуществляется через версионирование клиентского кода и протоколов. Разработчик контролирует:

– версию ядра LGOS; – формат графа и набор допустимых типов узлов и связей; – протокол обмена операциями; – форматы и версии агентов; – сетевые протоколы и поведение узлов.

Обновления распространяются как новые версии статических артефактов (JS/WASM/модели), размещённых на статическом хостинге или CDN. Узлы при запуске проверяют версию ядра, при необходимости загружают новую и при необходимости выполняют миграции структуры графа и журналов операций.

Миграции выполняются на стороне узлов: новый код содержит процедуры преобразования локального фрагмента графа к новой схеме. При этом протокол операций должен быть спроектирован так, чтобы узлы разных версий могли взаимодействовать в переходных периодах, пока сеть не перейдёт на новую версию.

1. Безопасность, доверие и ограничения

Отказ от локальности и наличие только сетевой модели означает, что пользователь не владеет полной самостоятельной копией данных. Это требует продуманной модели безопасности и доверия. Необходимо обеспечить шифрование данных в транзите и на узлах, контроль над правами доступа, разграничение пространств, а также прозрачность устройства протокола.

Модель угроз должна учитывать:

– подмену данных в злонамеренных узлах; – попытки искажения вычислений; – атаки на сигнальный слой; – попытки изоляции части сети (сетевые разбиения).

Часть угроз решается криптографическими методами (подписи, верифицируемые журналы, шифрование), часть — избыточностью реплик и многократной проверкой вычислений, часть — архитектурой протокола, допускающей деградацию, но не полный отказ.

1. Технологический стек и конкретные модули реализации

Данный раздел описывает конкретные технологии, используемые в LGOS, и роль каждой из них в архитектуре.

WebGPU — основной вычислительный слой для запуска SLM-моделей, параллельной обработки графа и выполнения агентных вычислений прямо в браузере. Используется для: • инференса моделей в формате ONNX; • сложных графовых запросов; • параллельных вычислений для индексирования, агрегаций и трансформаций.

WASM (WebAssembly) — слой низкоуровневого sandbox-исполнения, в котором работают: • ядро LGOS (критические функции графа); • модули CRDT и журналов операций; • парсеры PDF/DOCX/HTML (pdf.js, pdf-parse, Mammoth) для локального ingestion контента.

GUN.js — распределённая графовая БД для хранения подграфов, репликации и обеспечения offline-first модели. Используется как CRDT-движок, обеспечивающий: • автоматическое слияние состояний; • криптографические подписи через SEA/WebCrypto; • децентрализованный обмен данными между узлами через WebRTC.

WebRTC — сеть прямых P2P-соединений. Обеспечивает: • gossip-распространение операций; • передачу вычислительных задач между узлами; • взаимодействие узлов-маяков с клиентскими узлами.

Transformers.js + ONNX Runtime Web — стек для локального инференса SLM-моделей. ONNX Runtime автоматически выбирает WebGPU или WASM в зависимости от устройства. Модели загружаются и кешируются в IndexedDB.

IndexedDB — локальное хранилище для: • кеша моделей; • журнала операций; • временных реплик подграфов.

TipTap + собственный layout-движок — интерфейсный слой, проектирующий граф в виде блоков, документов, карт, таймлайнов и структур проектов. HTML/DOCX/PDF импортируется и преобразуется в графовые структуры через ingestion-пайплайн.

Signaling Server — лёгкий сервис для обмена SDP/ICE-кандидатами. Не хранит данные, не выполняет вычисления. Может быть любым CDN-хостингом.

Эти технологии образуют полностью клиентскую распределённую вычислительную среду, заменяющую классический бэкенд.

1. Прикладные компоненты LGOS (Chat, Hub, Editor, Messenger, Goals и др.)

LGOS как сетевая виртуальная машина предоставляет общий граф и единый интерфейсный слой, поверх которых реализуются прикладные компоненты. Каждый компонент — это не отдельный модуль с собственным состоянием, а специфическая проекция единого графа, использующая одни и те же механизмы P2P-синхронизации, CRDT-операций и WebGPU-агентов.

Chat (Коммуникационный слой) Chat представляет собой графовое представление диалогов и сообщений. Сообщения существуют как узлы графа с типами «message», «thread», «mention». Синхронизация сообщений происходит через P2P-узлы на основе GUN.js. Chat использует WebRTC-каналы для мгновенного обмена операциями и WebGPU-агентов для локальной обработки сообщений (классификация, sumarization, semantic routing). Chat не хранит свои данные отдельно — каждый диалог является частью общего графа LGOS.

Hub (Центральный центр активности) Hub — это проекция графа, агрегирующая недавно изменённые узлы, новые события, извлечённые документы, задачи, упоминания и активность агентов. Hub использует: • CRDT-операции для фиксации новых изменений; • ingestion-пайплайн (PDF/DOCX/HTML → граф); • WebGPU-агентов для извлечения ключевой информации; • TipTap-рендеринг для отображения блоков сведений. Hub не хранит данные, а вычисляет состояние из журнала графа.

Editor (Документы, структуры, PR-линии) Editor — интерфейсный слой поверх TipTap и layout-движка. Каждая «страница» — это граф узлов: блоки, текстовые элементы, вложенные структуры, ссылки на другие узлы. Комментарии, выделения, PR-предложения (как в GitHub) — это операции над узлами графа. Editor использует WebGPU-агентов для автодополнений, анализа структуры, генерации контента и распознавания смысловых блоков. Все изменения идут как CRDT-патчи.

Messenger (Групповые чаты и p2p-каналы) Messenger расширяет Chat, добавляя: • групповые узлы «room»; • p2p-каналы для передачи файлов; • голосовые и S2ST-каналы под WebRTC; • агентные функции (semantic search, routing). Messenger не имеет собственного хранилища: комнаты и сообщения — это узлы глобального графа LGOS, а маршрутизация идёт через P2P-мультисекции.

Goals (Цели, проекты, задачи) Goals — это проекция графа на типы «goal», «milestone», «task», «event». Вся логика проектного управления — это связи между узлами: «часть», «зависит от», «выполнено», «запланировано». Goals используют WebGPU-агентов для прогнозирования выполнения, приоритизации, временных расчётов и анализа критического пути. Все изменения синхронизируются P2P.

Learn (Онбординг, курсы, документация) Learn — слой поверх Hub и Editor. Материалы, задания, связки между разделами и прогресс выполнения — это узлы графа. Learn использует ingestion-модуль для загрузки PDF/Docs, а WebGPU-агенты — для автоматической генерации структур курсов.

Subscription (Экономика доступа и роли) Subscription определяет доступ к узлам через криптографические ключи и ролевую модель: Owner, Admin, Member, Guest. Решение хранится распределённо: доступ определяется схемой ключей SEA/WebCrypto, а не серверной БД.

Все прикладные компоненты LGOS представляют собой разные формы визуализации одного и того же графового состояния, что создаёт монолитную, но гибкую архитектуру.

1. Архитектурные схемы компонентов и связь с распределённой виртуальной машиной

В этом разделе фиксируется уровневое устройство каждого прикладного компонента: от распределённой виртуальной машины LGOS до конкретного UI. Для всех компонентов используется единая четырёхслойная схема:

– Слой распределённой виртуальной машины (VM Layer): P2P-сеть, протокол операций, CRDT-граф, узлы-маяки. – Слой операционной системы LGOS (OS Core Layer): runtime ядро, менеджер графа, планировщик задач, агентный фреймворк, система прав. – Слой доменной логики компонента (Component Domain Layer): типы узлов и связей, доменные операции, правила агрегации и индексации. – Слой интерфейса (UI/View Layer): конкретные представления TipTap/React, взаимодействие пользователя, события UI.

12.1. Компонент Chat

VM Layer. Сообщения, треды, упоминания и вложения представлены как узлы и связи распределённого графа. Каждый узел хранит часть этих данных, реплицируя их по P2P-сети через GUN.js/WebRTC. CRDT-механизм обеспечивает сходимость состояний при асинхронной доставке сообщений.

OS Core Layer. Ядро LGOS ведёт журнал операций типа «send_message», «edit_message», «delete_message», «add_reaction». Планировщик задач маршрутизирует операции в нужные подграфы и узлы-маяки. Агентный фреймворк предоставляет Chat-агентам доступ к данным (поиск, кластеризация диалогов, обнаружение тем).

Component Domain Layer. Специфичные типы узлов: Message, Thread, Channel, Mention, Attachment. Доменные правила: порядок сообщений в треде, логика «прочитано/не прочитано», связь сообщений с задачами, документами и событиями. Здесь же определяются индексы (по участникам, по каналам, по проектам).

UI/View Layer. Интерактивный интерфейс чата: список каналов, треды, панель ввода, инлайн-ответы, реакции. UI подписан на подграф сообщений через реактивный слой LGOS и обновляется при поступлении операций. Агентные результаты (сводки, рекомендации) отображаются в виде подсказок и карточек.

12.2. Компонент Hub

VM Layer. Hub использует глобальный журнал операций и метаданные узлов, не создавая выделенного хранилища. Узлы-маяки агрегируют события (новые документы, изменённые задачи, активности агентов) и распространяют их как мета-операции по сети.

OS Core Layer. Ядро LGOS предоставляет Hub-сервису API для подписки на события: «node_created», «node_updated», «relation_created», «agent_event». Планировщик задач периодически пересчитывает приоритеты и формирует «ленты» активности для разных контекстов (личный, командный, гильдийный).

Component Domain Layer. Определены доменные сущности ActivityItem, Notification, Feed, Digest. Для каждого элемента задаются правила агрегации: какие типы узлов и связей отображать в Hub, как группировать события (по проекту, по типу, по агенту).

UI/View Layer. Визуально Hub представлен как лента карточек активности, панель уведомлений и дашборд ключевых изменений. Все элементы Hub — только проекции на реальные узлы графа; переходы из Hub ведут в соответствующие компоненты (Chat, Editor, Goals).

12.3. Компонент Editor

VM Layer. Документы, разделы, блоки и встроенные сущности (цитаты, таблицы, ссылки на задачи) представлены как деревья и подграфы. CRDT-операции над блоками (insert, delete, split, merge, annotate) распространяются по P2P-сети и сходятся без централизованного сервера.

OS Core Layer. Ядро LGOS предоставляет Editor-движку API: «apply_block_op», «get_document_subgraph», «lock_logical_section» (мягкие логические блокировки для UX). Агентный слой использует подграф документа для автодополнения, рефакторинга, ре-структурирования и для связи с другими узлами.

Component Domain Layer. Типы узлов: Document, Section, Block, Comment, Suggestion, Reference. Правила: версия документа, ветки правок (PR-линии), статус предложений (accepted/rejected), связи документа с задачами, встречами и целями.

UI/View Layer. TipTap-редактор, панель структуры, side-панель комментариев и предложений, режим сравнения версий. Пользовательские действия трансформируются в CRDT-операции и сразу транслируются в VM Layer.

12.4. Компонент Messenger

VM Layer. Messenger использует те же узлы Message/Thread, но добавляет Room, CallSession, MediaStream. P2P-сеть обеспечивает не только обмен операциями, но и передачу медиапотоков по WebRTC.

OS Core Layer. Ядро управляет сессиями, правами доступа к комнатам, сигналами «join/leave», а также журналом событий звонков. Агентный слой может анализировать метаданные, но содержимое медиа шифруется end-to-end.

Component Domain Layer. Доменные правила: типы комнат (private, team, guild), управление участниками, история звонков, привязка к проектам и задачам. Messenger также интегрируется с Goals (создание задач из чата/звонка).

UI/View Layer. Список комнат, экраны звонков, групповые чаты, управление устройствами. Все состояния UI зависят от журнала операций Messenger-подмножества графа.

12.5. Компонент Goals

VM Layer. Цели, задачи, вехи и события — узлы графа с богатыми связями: depends_on, part_of, scheduled_for, owned_by. Их структура живёт в распределённой VM и реплицируется между узлами.

OS Core Layer. Ядро предоставляет API для построения графа зависимостей, вычисления критического пути, прогнозирования загрузки ресурсов. Планировщик задач может использовать этот граф для распределения вычислений между узлами.

Component Domain Layer. Типы узлов: Goal, Milestone, Task, Event, Resource. Правила: статусы, приоритеты, дедлайны, связи с документами (Editor) и коммуникациями (Chat/Messenger).

UI/View Layer. Kanban, таймлайны, диаграммы зависимостей, дашборды прогресса. Все представления — чистые функции от подграфа цели/проекта.

12.6. Компонент Learn

VM Layer. Курсы, модули, уроки, задания и прогресс — узлы графа с отношениями prerequisite, part_of, completed_by. Их состояние реплицируется и обновляется через общие механизмы LGOS.

OS Core Layer. Ядро предоставляет Learn-сервису доступ к контенту и прогрессу, обеспечивает контроль прав и взаимодействие с агентами. Агентный слой может формировать адаптивные траектории обучения на основе графа знаний.

Component Domain Layer. Типы узлов: Course, Module, Lesson, Assignment, ProgressRecord. Правила: последовательность прохождения, условия завершения, связи с целями (Goals) и реальными задачами.

UI/View Layer. Доска курсов, страницы уроков, трекер прогресса, рекомендации следующего шага. Всё это — представления над графом Learn-подмножества.

12.7. Компонент Subscription

VM Layer. Модель доступа реализована через криптографические ключи и связи между идентичностями пользователей и пространствами. Узлы-политики (PolicyNode) задают, какие операции разрешены для каких ролей.

OS Core Layer. Ядро LGOS проверяет права при каждой операции, используя криптографические подписи и схему ролей. Централизованная серверная ACL-БД не требуется: решения принимаются на основе распределённого графа прав.

Component Domain Layer. Типы узлов: UserIdentity, Space, Role, Policy. Правила: Owner/Admin/Member/Guest, делегирование прав, временные доступы, платные подписки.

UI/View Layer. Настройки доступа, экраны управления участниками, экраны тарифов/подписок. Все изменения прав немедленно отражаются в поведении остальных компонентов через OS Core Layer.

Заключение

Технически LifeGraph OS в описанной модели представляет собой распределённую виртуальную машину уровня сети, в которой каждый браузерный узел является одновременно интерфейсом, вычислителем и частичной репликой общего графа. Данные и вычисления живут только внутри сети LGOS, а пользователь не имеет локальной автономной копии. Согласованность достигается через операции и CRDT-подобные структуры, P2P-протоколы обеспечивают распространение

изменений и задач, а узлы-маяки и репликация формируют устойчивость и отказоустойчивость. Управление системой осуществляется через версионирование клиентского ядра, протоколов и миграций, без классического централизованного бэкенда.