DOI: 10.15514/ISPRAS-2026-38(6)-28



Goldfish: A Collaborative Web-Based Petri Net Editor with Real-Time Synchronization

V.A. Kabakov, ORCID: 0009-0009-3459-2810 <vlankabakov@edu.hse.ru>
A.A. Mitsyuk, ORCID: 0000-0003-2352-3384 <amitsyuk@hse.ru>

HSE University,
11, Pokrovsky Bulvar, Moscow, 109028, Russia.

Abstract. Petri nets are a well-established formalism for modeling concurrent and distributed systems, yet existing Petri net editors are predominantly desktop applications without collaborative editing capabilities. This paper presents Goldfish, a cloud-based web platform for real-time collaborative editing, simulation, and analysis of Petri net models. The core contribution is the adaptation of Operational Transformation (OT) techniques to graph-based Petri net structures: we define a domain-specific operation model comprising 9 primitive operation types, formulate transformation rules that preserve bipartite graph invariants and handle cascading structural dependencies, and combine OT with selective element locking for semantically critical operations. The resulting hybrid synchronization mechanism is estimated to resolve the majority of editing conflicts transparently via OT in typical usage scenarios – where concurrent operations target disjoint model elements – invoking locking only for operations with cascading effects such as element deletion. Complexity analysis confirms that all per-operation synchronization costs are independent of model size. The platform integrates standard Petri net analysis capabilities – reachability graph construction, boundedness verification, and interactive simulation – within the collaborative environment via an asynchronous execution model that avoids blocking concurrent editing sessions. Complexity analysis of the proposed algorithms shows that all per-operation synchronization costs are constant or linear in small parameters (log gap, pending queue size, node degree); these bounds hold independently of model size, supporting scalability with respect to the number of concurrent editors.

Keywords: Petri nets; collaborative editing; real-time synchronization; operational transformation; conflict resolution; web-based modeling; process model visualization.

For citation: Kabakov V.A., Mitsyuk A.A. Goldfish: A Collaborative Web-Based Petri Net Editor with Real-Time Synchronization. Trudy ISP RAN/Proc. ISP RAS, vol. 38, issue 6, part 2, 2026, pp. 117-132. DOI: 10.15514/ISPRAS-2026-38(6)-28.

Acknowledgements. This work is an output of a research project (HSE-BR-2025-024) implemented as a part of the Basic Research Program as HSE University.

Goldfish: веб-редактор сетей Петри для совместной работы с синхронизацией в режиме реального времени

В.А. Кабаков, ORCID: 0009-0009-3459-2810 <vlankabakov@edu.hse.ru>
А.А. Мицюк, ORCID: 0000-0003-2352-3384 <amitsyuk@hse.ru>

Национальный исследовательский университет «Высшая школа экономики» (НИУ ВШЭ),
Россия, 109028, г. Москва, Покровский бул., д. 11.

Аннотация. Сети Петри – формализм для моделирования параллельных и распределённых систем. Существующие редакторы сетей Петри – преимущественно приложения для настольных компьютеров без поддержки совместного редактирования. В данной статье представлен редактор «Золотая рыбка» (Goldfish) – веб-платформа для совместного редактирования, моделирования и анализа сетей Петри в реальном времени. Ключевой результат работы – адаптация методов операционных преобразований (ОП) для использования с графовыми структурами сетей Петри. Определена предметно-ориентированная модель операций, включающая 9 типов примитивных операций. Сформулированы правила преобразований, которые сохраняют инварианты двудольного графа и обрабатывают каскадные структурные зависимости. ОП комбинируются с выборочной блокировкой элементов для семантически критических операций. Разработанный гибридный механизм синхронизации способен разрешить большинство конфликтов редактирования с помощью ОП в типичных сценариях использования, когда параллельные операции направлены на непересекающиеся элементы модели. К блокировкам требуется прибегать только для операций с каскадными эффектами (например, удаление элемента). Платформа Goldfish интегрирует также стандартные возможности анализа сетей Петри – построение графа достижимости, проверку ограниченности и интерактивное моделирование – в совместную среду через асинхронную модель выполнения, которая не блокирует параллельные сеансы редактирования. Анализ сложности предложенных алгоритмов показывает, что все затраты на синхронизацию на одну операцию являются константными или линейными по отношению к малым параметрам независимо от размера модели, что обеспечивает масштабируемость по числу одновременно работающих редакторов.

Ключевые слова: сети Петри; совместное редактирование; синхронизация в реальном времени; операционные преобразования; разрешение конфликтов; веб-моделирование; визуализация моделей процессов.

Для цитирования: Кабаков В.А., Мицюк А.А. Goldfish: веб-редактор сетей Петри для совместной работы с синхронизацией в режиме реального времени. Труды ИСП РАН, том 38, вып. 6, часть 2, 2026 г., стр. 117–132 (на английском языке). DOI: 10.15514/ISPRAS–2026–38(6)–28.

Благодарности. Исследование осуществлено в рамках Программы фундаментальных исследований НИУ ВШЭ (HSE-BR-2025-024).

1. Introduction

Petri nets are one of the most widely used formalisms for modeling concurrent and distributed systems. Their graphical representation combined with rigorous mathematical semantics makes them valuable across diverse domains including workflow management, manufacturing systems, communication protocols, and biological processes [1-2].

Despite the maturity of the formalism, its tool support has not kept pace with the shift toward cloud-based collaborative work. A survey of existing Petri net tools – including CPN Tools [3-4], PIPE [5], Carassius [6-7], TAPAAL [8], CPN IDE [9] and web-based editors such as PetriPad [10] and the PES editor [11] – reveals the following limitations:

- **Platform dependency.** Most editors are desktop applications for specific operating systems (*e.g.*, Carassius [6] and CPN IDE [9] support only Windows), creating barriers for cross-platform use;
- **No real-time collaboration**. None of the surveyed tools support synchronous multi-user editing – a feature standard in modern tools such as Google Docs [12] and Figma [13];

- **Limited web accessibility.** Existing web-based Petri net editors [10-11] provide only basic editing without analysis, simulation, collaboration, or versioning;
- **No integrated versioning.** Users must rely on external version control systems not designed for graphical model artifacts.

While general-purpose collaborative editing techniques – Operational Transformation (OT) [14-15] and Conflict-free Replicated Data Types (CRDTs) [16] – are well-studied for text and generic graphical structures, *applying them to Petri net models introduces domain-specific challenges*. Petri nets are bipartite graphs with structural invariants: arcs may only connect places to transitions or vice versa, deleting a node must cascade to incident arcs, and concurrent modifications must not violate these constraints. Standard OT transformation rules for text insertion/deletion or generic shape manipulation do not address these concerns.

This paper presents Goldfish, a collaborative web-based Petri net editor that addresses these challenges. The contributions are:

1) A **domain-specific operation model** for Petri net editing, comprising 9 primitive operation types organized by structural element (places, transitions, arcs), and **transformation rules** that preserve bipartite graph invariants and handle cascading structural dependencies (Section 5);
2) A **hybrid conflict resolution mechanism** combining OT for the majority of operations with selective element locking for semantically critical operations (*e.g.*, deletion with cascading effects), achieving transparent conflict resolution via OT for the estimated majority of operations in typical editing sessions, where users work on structurally disjoint parts of the model (Sections 5, 7);
3) An **asynchronous analysis integration** model that embeds standard Petri net analysis (reachability, boundedness, simulation) within the collaborative environment without blocking concurrent editing sessions (Section 6);
4) A **complexity analysis** of all proposed algorithms, showing $O(1)$ per pairwise operation transformation, $O(\Delta v)$ server-side log catch-up, and $O(\deg(e))$ lock scope computation, confirming that synchronization costs scale with concurrency parameters rather than model size (Section 7).

2. Background

This section provides the definitions and motivates the choice of synchronization approach.

2.1 Petri Nets

A Petri net is a tuple $N = (P, T, F, W, M_0)$, where P is a finite set of places, T is a finite set of transitions. $(P \cap T = \varnothing)$, $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation, $W : F \to \mathbb{N}^+$ is the weight function, and $M_0 : P \to \mathbb{N}_0$ is the initial marking [1].

A transition t is enabled at marking M if $\forall p \in {}^\bullet t : M(p) \geq W(p,t)$. Firing t yields $\mathrm{M'(p) = M(p) - W(p, t) + W(t, p)}$ for all $p \in P$.

Two properties are relevant to the analysis capabilities of Goldfish. A marking M is *reachable* from $\mathrm{M_0}$ if there exists a firing sequence transforming $\mathrm{M_0}$ into M. A place p is $k-$*bounded* if $M(p) \leq k$ for all reachable markings; a net is bounded if all places are bounded.

Computing the full set of reachable markings is EXPSPACE-hard in general [17], motivating the bounded exploration approach in our implementation.

2.2 Collaborative Editing: OT vs. CRDTs

Operational Transformation [14-15] resolves concurrent edits by transforming operations so that applying them in any order yields the same final state. OT requires a central server for operation ordering but allows domain-specific transformation rules.

CRDTs [16] are data structures where concurrent operations converge automatically without transformation. CRDTs are well-suited for generic data types but encoding domain-specific structural invariants (*e.g.*, bipartite graph constraints) into CRDT merge semantics is non-trivial [18].

We chose OT as the primary mechanism because it allows us to define *domain-specific transformation rules* that explicitly handle Petri net structural constraints (bipartiteness, cascading arc deletion). This is supplemented with selective locking for operations where OT-based resolution would produce unintuitive results. The rationale for this hybrid approach is detailed in Section 5-D.

3. Related Work

3.1 Petri Net Tools

Table 1 compares representative Petri net tools along dimensions relevant to collaborative modeling.

CPN Tools [4] provides the most comprehensive analysis for Colored Petri Nets but requires local installation and has no collaboration support. PIPE [5] is cross-platform via JVM but single-user. Carassius [7], developed at HSE University, offers extensive analysis for business process modeling but is Windows-only. TAPAAL [8] specializes in timed-arc Petri nets. PetriPad [10] explored web-based editing but was never publicly released and lacks analysis or collaboration features. The PES editor [11] provides basic web-based editing without collaboration.

No existing tool combines real-time multi-user editing, integrated analysis, version history, and web accessibility.

Table 1. Comparison of Petri Net Editing Tools.

Tool	Platforms	Web	Max users	Net types	Analysis	Versioning	PNML
CPN Tools	2	No	1	3	5+	No	Yes
CPN IDE	1	No	1	1	5+	No	Yes
PIPE	3+	No	1	4	4	No	Yes
Carassius	1	No	1	2	5+	No	Yes
TAPAAL	3+	No	1	2	3	No	Yes
PES Editor	Any	Yes	1	1	1	No	No
PetriPad	Any	Yes	1	1	0	No	No
Goldfish	**Any**	**Yes**	**50+**	**2***	**4**	**Yes**	**Yes**

* Classical P/T nets; TVAW-Nets [20] support planned.

3.2 Collaborative Editing Platforms

Google Docs [12] uses OT for text; Figma [13] uses a CRDT-inspired approach for vector graphics; Yjs [20] provides a CRDT framework for generic collaborative applications. None of these address the structural invariants of graph-based mathematical models. In particular, the bipartite constraint, arc endpoint validity, and cascading deletion semantics of Petri nets require domain-specific transformation rules beyond what generic OT/CRDT frameworks provide.

4. System Architecture

4.1 Overview

Goldfish uses a modular monolithic architecture with event-driven components for real-time functionality. The system is organized into six bounded contexts following Domain-Driven Design [21]: (1) User Management (OAuth 2.0, profiles, sessions), (2) Team Management (teams, roles, RBAC), (3) Project Management (lifecycle, metadata), (4) Petri Net Editor (element manipulation, validation), (5) Real-time Synchronization (WebSocket communication, OT, locking), and (6) Analysis and Simulation (property verification, simulation). Fig. 1 shows the high-level architecture.

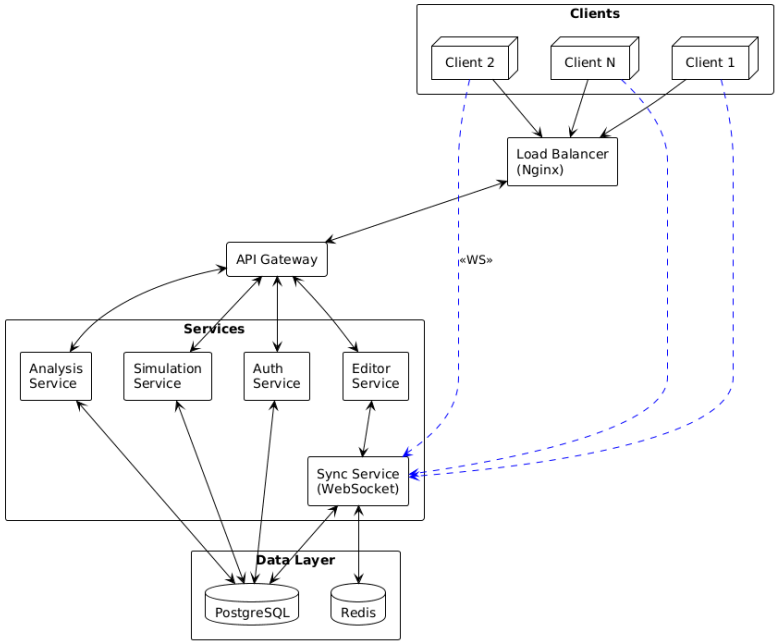


Fig. 1. High-level architecture of Goldfish.

4.2 Data Model and Technology Stack

The Petri net model is stored using a hybrid approach: relational columns for project metadata and JSONB columns for the graph structure, enabling flexible schema evolution and atomic updates. Fig. 2 shows the core domain entities.

The technology stack comprises: Vue.js 3 with TypeScript and Konva.js for canvas rendering (frontend); Kotlin with Ktor, JOOQ, and Kotlinx.coroutines (backend); PostgreSQL with JSONB and Redis (persistence); WebSocket for real-time and REST for CRUD (communication); Docker and Nginx (infrastructure).

4.3 Scalability

Horizontal scaling is achieved through: (1) stateless API servers with session state in Redis, (2) WebSocket session affinity via consistent hashing on project ID (ensuring collaborators on the same model connect to the same server), (3) PostgreSQL read replicas for read-heavy queries, and (4) Redis Cluster for distributed caching.

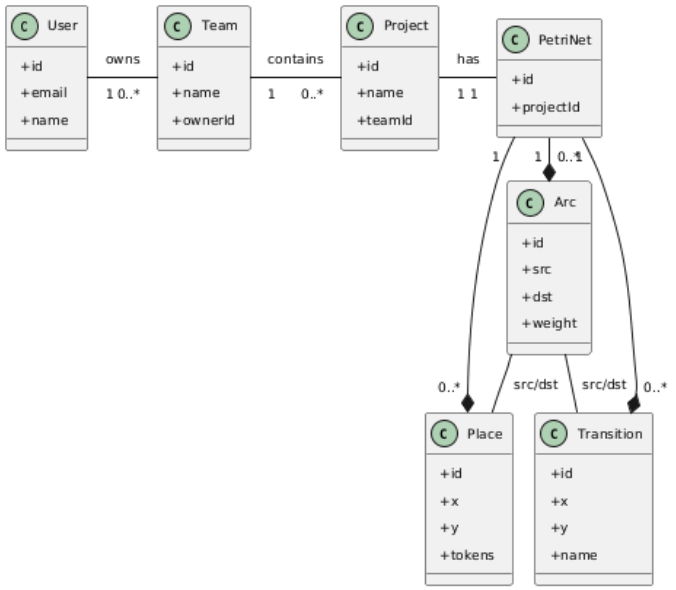


Fig. 2. Core domain model.

5. Adapting OT for Petri Net Structures

This section presents an adaptation of Operational Transformation to the domain of Petri net editing. We define a domain-specific operation model, formulate transformation rules that account for structural invariants of Petri nets, and describe the hybrid OT/locking mechanism.

5.1 Domain-Specific Operation Model

In contrast to text-based OT where operations reduce to insert/delete/retain, Petri net editing requires operations that reflect the graph-based structure. We define an operation as a tuple $o = (id, uid, \tau, v, type, params)$ where id is a UUID, uid identifies the user, τ is a server-assigned logical timestamp, v is the base model version, and $type \in \mathcal{T}$ with associated $params$.

The key design decision is the partitioning of the operation type set into three groups mirroring Petri net structure:

$$\mathcal{T} = \mathcal{T}_P \cup \mathcal{T}_T \cup \mathcal{T}_A$$

Table 2 lists all 9 primitive operations. This decomposition enables conflict detection based on element type: operations in $\mathcal{T}_A$ have *structural dependencies* on elements targeted by $\mathcal{T}_P$ and $\mathcal{T}_T$ (because arcs reference places and transitions as endpoints), which standard text or shape OT does not model.

We define the *target set* $tgt(o)$ as the set of element IDs affected by operation o. For most operations, this is simply $\{o.id\}$. For CreateArc, $tgt(o) = \{o.id, o.src, o.dst\}$ capturing the structural dependency on endpoint nodes. Fig. 3 illustrates the operation model taxonomy.

Table 2. Primitive Operation Types for Petri Net Editing.

Group	Type	Parameters
$\mathcal{T}_P$ (Places)	CreatePlace	$id, (x, y), label, m_0$
	MovePlace	$id, (x', y')$
	DeletePlace	id
$\mathcal{T}_T$ (Trans.)	CreateTrans	$id, (x, y), label$
	MoveTrans	$id, (x', y')$
	DeleteTrans	id
$\mathcal{T}_A$ (Arcs)	CreateArc	id, src, dst, w
	UpdateArc	id, w'
	DeleteArc	id

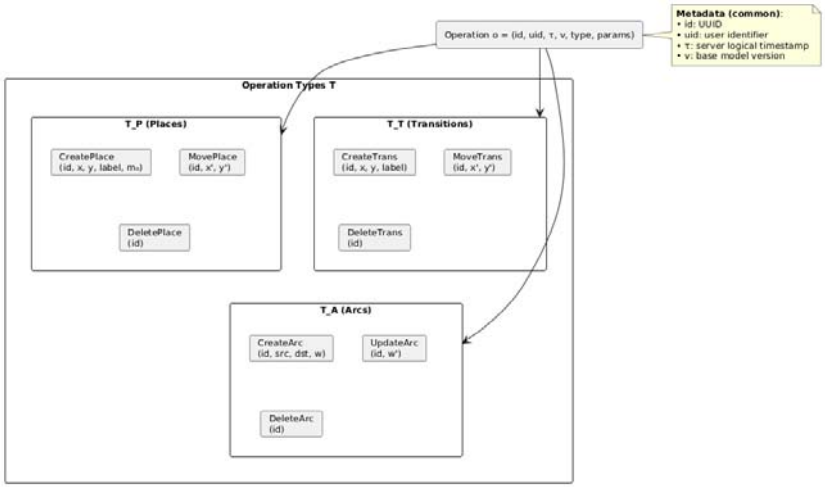


Fig. 3. Operation model: 9 primitive types partitioned by Petri net element.

5.2 Transformation Rules for Petri Net Operations

The transformation function $T(o_1, o_2)$ adjusts operation o_1 given that o_2 has already been applied. We denote by $o_\perp$ the null operation (no effect). The transformation rules are organized into three categories reflecting increasing conflict severity.

1. *Non-Conflicting Operations (Disjoint Targets).*

   If two operations affect entirely different elements, no transformation is needed:

   $$T(o_1, o_2) = o_1 \quad \text{if} \quad tgt(o_1) \cap tgt(o_2) = \varnothing$$

   This covers the common case of users working on different parts of the model and accounts for the majority of concurrent operations in practice.

2. *Delete-Dependent Conflicts (Petri Net Specific).*

   This category is specific to graph-based structures and has no direct analogue in text OT. Two subcases arise:

   Direct conflict. If o_2 deletes element e and o_1 also targets e (*e.g.*, move or property change), then o_1 becomes meaningless:

   $$T(o_1, o_2) = o_\perp \quad \text{if} \quad o_2.type \in \{Delete^*\} \wedge o_1.id = o_2.id$$

   Structural dependency (cascading). This is the rule that distinguishes Petri net OT from generic graph OT. If a user creates an arc to a node that another user concurrently deletes, the arc creation must be nullified because its endpoint no longer exists:

   $$T(o_a, o_d) = o_\perp \quad \text{if} \quad o_a.\text{type} = CreateArc \ \wedge o_d.\text{id} \in \{o_a.\text{src}, o_a.\text{dst}\}$$

   Conversely, when a delete operation is applied, all arcs incident to the deleted node are removed as a cascading side effect. This preserves the well-formedness invariant that every arc references two existing, type-compatible endpoints (place ↔ transition).

   Formally, the cascading semantics are defined as follows. Let $Delete^*(e)$ denote the deletion of element $e \in P \cup T$. The server expands this operation into an ordered sequence:

   $$Delete^*(e) \to \left[DeleteArc(a) \,\middle|\, a \in F, e \in \{src(a), dst(a)\} \right] \Big\| \; Delete^*(e)$$

   where "$\Big\|$" denotes sequencing. All implicit DeleteArc operations are appended to the operation log, assigned consecutive version numbers, and broadcast to clients. This ensures that (i) every client's log contains explicit records of all arc removals, and (ii) subsequent transformations of concurrent operations against the log encounter these deletions individually.

3. *Attribute Conflicts (Same Element, Same Property).*

   When two users concurrently modify the same property of the same element (*e.g.*, both move the same place), deterministic resolution is required. We use *server-order precedence*: the operation that reaches the server first wins:

   $$T(o_1, o_2) = o_\perp \quad \text{if} \quad o_1.id = o_2.id \wedge attr(o_1) = attr(o_2)$$

   where the operation with the earlier server timestamp takes effect. The client whose operation was nullified receives a notification and sees the updated state, allowing re-editing if desired. Fig. 4 shows the conflict classification and resolution logic.

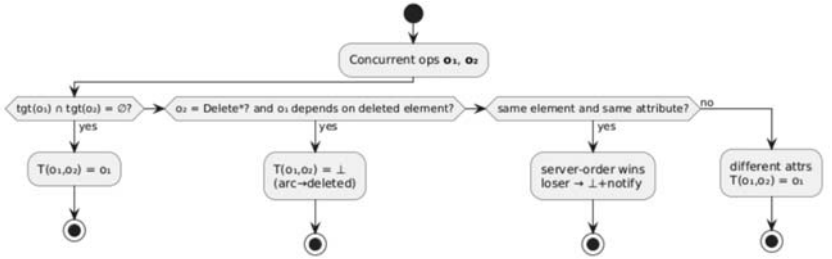


Fig. 4. Transformation rule selection: concurrent operations are classified by target overlap and conflict type, then resolved accordingly.

4. *Convergence Argument.*

   The standard convergence condition for OT (TP1 [15]) requires that for any concurrent operations o_1, o_2:

$$o_2 \circ T(o_1, o_2) \equiv o_1 \circ T(o_2, o_1)$$

   where "≡" denotes state equivalence. We argue informally that the proposed transformation rules satisfy TP1 by case analysis on the conflict categories:

   - **Disjoint targets:** $T(o_1, o_2) = o_1$ and $T(o_2, o_1) = o_2$, so both application orders yield the same state (operations are independent).
   - **Delete-dependent:** If o_2 deletes an element that o_1 depends on, then $T(o_1, o_2) = o_\perp$. Symmetrically, $T(o_2, o_1) = o_2$ (the deletion proceeds regardless). In both orders, the element is deleted and o_1 has no effect, yielding the same final state.
   - **Attribute conflict:** Both $T(o_1, o_2)$ and $T(o_2, o_1)$ resolve to server-order precedence: the operation with the earlier timestamp takes effect, the other is nullified. Regardless of application order, the same single operation determines the final attribute value.

The selective locking mechanism (Section 5-D) further strengthens this guarantee by serializing operations that involve cascading side effects, preventing the concurrent generation of conflicting cascades.

5.3 Client–Server Protocol

The synchronization follows an event-sourcing pattern. The server maintains an authoritative, totally ordered operation log; clients maintain local state for immediate feedback.

Client state: (i) last acknowledged version v_c, (ii) pending queue Q of locally applied but unacknowledged operations, (iii) visual model state. The client sends at most one operation at a time, queuing subsequent ones until acknowledgment.

Server state: (i) authoritative model state, (ii) ordered log $L = [o_1, \ldots, o_n]$ with version numbers, (iii) active sessions.

Protocol steps:

1) Client applies operation o locally (optimistic update) and appends to Q.

2) Client sends o with base version v_c to server.

3) Server transforms o against all log entries since v_c: $o' = T(\ldots T(T(o, o_{v_c+1}), o_{v_c+2})\ldots)$

4) Server assigns next version to o', appends to log, broadcasts to all clients.

5) Originating client removes o from Q, updates v_c.

6) Other clients transform o' against their pending queues and apply.

Fig. 5 shows a concrete protocol sequence.

Optimizations. (1) *Operation grouping:* drag interactions produce many intermediate move events; the client sends only the final position upon mouse release (or after N *ms* of inactivity). (2) *Batch submission:* accumulated pending operations are sent as a batch upon acknowledgment.

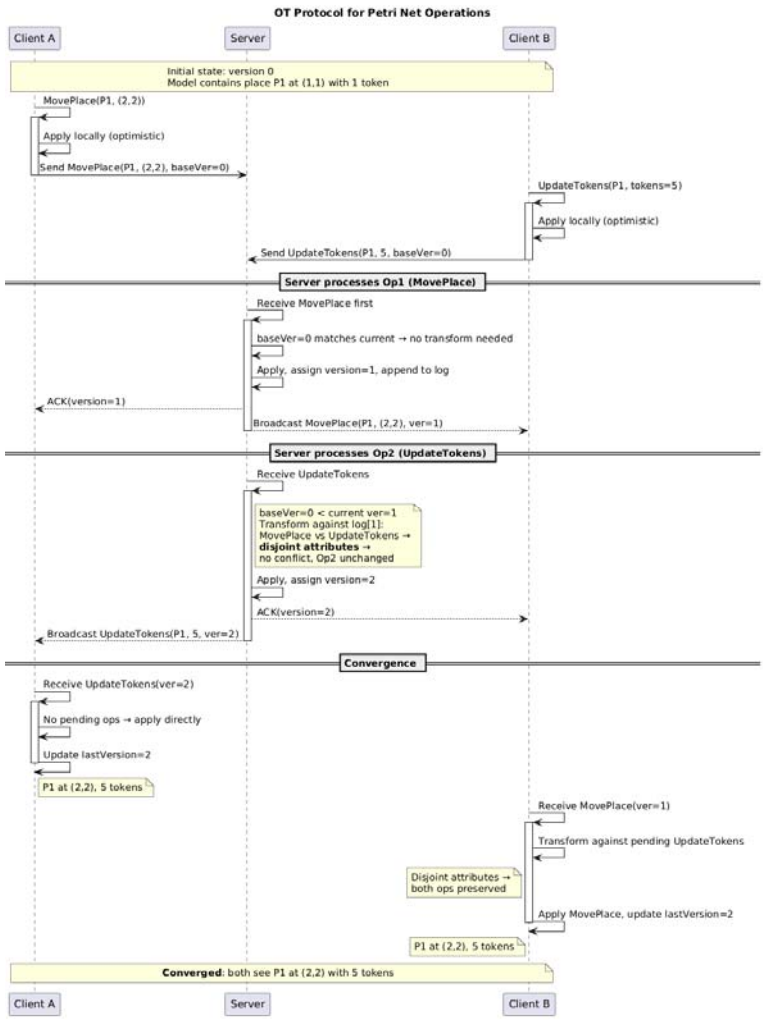


Fig. 5. Synchronization protocol: operation flow between two concurrent clients and the server.

The convergence and causality preservation properties of this protocol follow from the standard OT guarantees [14-15]: the server imposes a total order via version numbers, all clients apply the same deterministic transformation function against the same sequence, and the base-version mechanism ensures causal ordering.

5.4 Selective Locking for Critical Operations

While OT handles most conflicts transparently, certain Petri-net-specific operations benefit from pessimistic concurrency control. The key observation is that *deletion of a node has cascading effects* (removing incident arcs) that are difficult to communicate clearly to other users when resolved purely via OT. Consider: User A creates an arc from place p to transition t, while User B

simultaneously deletes p. OT would correctly nullify the arc creation, but User A would see their newly created arc silently disappear – a confusing experience. We therefore classify operations as *critical* or *non-critical*:

- **Critical:** element deletion (cascading arc removal), structural topology changes. These require a lock before execution.
- **Non-critical:** moves, label edits, token count changes, arc weight updates. These are handled via OT without locking.

The locking protocol (Algorithm 1) acquires a lock on the target element *and all incident arcs* before a critical operation, preventing concurrent modifications to structurally dependent elements.

Algorithm 1 Critical Operation with Selective Locking

Require: Critical operation o, target element e, user u

```
scope ← {e} ∪ IncidentArcs(e)
result ← RequestLock(scope, u)
if result = Granted then
    ApplyLocally(o)
    SendToServer(o)
    AwaitAck()
    ReleaseLock(scope, u)
else
    NotifyUser("Element locked by " ‖ result.owner)
end if
```

Locks are released upon: (i) explicit release after acknowledgment, (ii) inactivity timeout (default 30 *s*), or (iii) Web-Socket disconnection. Lock state is broadcast to all clients, which display visual indicators (colored borders, lock icons) on locked elements.

This hybrid approach ensures that non-critical operations (the vast majority during typical editing) proceed without any locking overhead, while critical operations that could cause cascading confusion are serialized

6. Analysis and Simulation in a Collaborative Context

Goldfish integrates standard Petri net analysis capabilities – reachability graph construction, boundedness verification, and interactive simulation – as described in [1]. The contribution here is not the analysis algorithms themselves (which are well-known) but rather *how they are embedded within a collaborative editing environment*, addressing two challenges: (1) analysis must not block other users' editing sessions, and (2) simulation state must be isolated from the shared model.

6.1 Asynchronous Analysis Execution

Reachability analysis and boundedness verification may require significant computation (EXPSPACE-hard in general [17]). To prevent blocking, analysis runs asynchronously:

1) A user submits an analysis request via REST API.
2) The server snapshots the current model state and creates an analysis task.
3) A worker coroutine processes the task on the snapshot, pushing progress updates via WebSocket (*e.g.*, "explored 5,000 / 100,000 states").
4) Results are cached and delivered upon completion.

The analysis operates on a *snapshot*, not the live model. If the model changes during analysis, the results remain valid for the snapshot version, and the user is informed of the version mismatch. Resource limits prevent abuse: maximum 3 concurrent analyses per user, 5-minute timeout, state space limit of 100,000 markings.

6.2 Simulation State Isolation

Interactive simulation (step-by-step or automatic firing of enabled transitions) modifies the marking of the Petri net. In a collaborative environment, one user's simulation must not alter the shared model state visible to others. Goldfish achieves this by maintaining simulation state in a *session-local copy* of the marking. The user sees token animation and marking changes locally; other users see the original model state. When the simulation ends, the session-local state is discarded.

6.3 Structural Validation During Editing

The system performs incremental validation during editing, checking three invariants in real time: (1) bipartiteness (arcs connect places to transitions or vice versa), (2) endpoint validity (all arc endpoints exist), and (3) weight positivity ($w > 0$). Violations trigger immediate visual feedback. This validation is lightweight and runs synchronously as part of operation application, unlike the heavyweight analysis operations above.

7. Complexity Analysis

This section provides asymptotic complexity bounds for the algorithms introduced in this paper: the domain-specific operation transformation (Section 5-B), the client–server synchronization protocol (Section 5-C), and the selective locking mechanism (Section 5-D).

7.1 Pairwise Operation Transformation

The transformation function $T(o_1, o_2)$ proceeds in two steps: (1) conflict detection via target set intersection, and (2) resolution by rule selection.

Since $|tgt(o)| \leq 3$ for all operation types (the maximum is attained by CreateArc, whose target set includes the arc and both endpoints), the intersection $tgt(o_1) \cap tgt(o_2)$ is computed in $O(1)$ time. Rule selection (Section 5-B) involves a constant number of comparisons on operation type and target identifiers. Therefore:

$$T(o_1, o_2) \in O(1) \text{ time and } O(1) \text{ space}$$

7.2 Server-Side Log Catch-Up

When the server receives an operation o with base version v_c and the current log version is v_s, it applies $\Delta v = v_s - v_c$ sequential transformations (Section 5-C, step 3):

$$o' = T(\ldots T(T(o, o_{v_c+1}), o_{v_c+2}) \ldots, o_{v_s})$$

Each transformation is $O(1)$, yielding $O(\Delta v)$ total time per incoming operation. The operation log requires $O(|L|)$ space, where $|L|$ is the log length. Log compaction via periodic model snapshots bounds this to $O(W)$ for a configurable retention window W.

7.3 Client-Side Remote Operation Application

Upon receiving a remote operation o', a client transforms it against its pending queue Q (Section 5-C, step 6). This requires $|Q|$ pairwise transformations, giving $O(|Q|)$ time. Under the single-outstanding-operation policy, $|Q| \leq 1$ in steady state, yielding amortized $O(1)$.

7.4 Broadcast

After committing an operation, the server sends it to all k clients in the session, requiring $O(k)$ message transmissions per operation.

7.5 Selective Locking

Lock scope computation for a critical operation on element e (Algorithm 1, line 1) requires enumerating all arcs incident to e. Let $\deg(e)$ denote the number of incident arcs. Using a hash-map-based adjacency index, this is $O(\deg(e))$. Lock acquisition and release each involve $O(\deg(e))$ hash map operations (one per element in the scope). The aggregate space for the lock registry is $O(|P|+|T|+|F|)$.

7.6 Cascading Deletion

When a *DeletePlace* or *DeleteTrans* operation is committed, all incident arcs must be identified and removed (Section 5-B). Using the same adjacency index, this requires $O(\deg(e))$ time and generates $O(\deg(e))$ implicit *DeleteArc* operations that are appended to the log and broadcast. The total cost of a cascading deletion is therefore $O(\deg(e)\cdot k)$ including broadcast.

7.7 Incremental Structural Validation

Per-operation validation (bipartiteness, endpoint existence, weight positivity; Section 6) involves a constant number of hash map lookups in the element index, yielding $O(1)$ time per operation. Maintaining the element index requires $O(|P|+|T|+|F|)$ space.

7.8 Summary

Table 3 summarizes the asymptotic bounds for the algorithms introduced in this paper. All per-operation costs are constant or linear in small quantities ($\Delta v, |Q|, \deg(e)$) confirming that the synchronization mechanism scales with the number of concurrent editors rather than with model size.

8. Discussion

OT vs. CRDT. We chose OT over CRDTs because OT allows explicit domain-specific transformation rules. Encoding cascading arc deletion semantics into a CRDT merge function is non-trivial: the “delete wins” vs. “add wins” conflict inherent in set CRDTs does not cleanly map to the Petri net case where deleting a node must also remove arcs. Investigating CRDT-based approaches for non-critical operations (moves, label edits) is a direction for future work.

Locking granularity. The current approach locks individual elements and their incident arcs. Coarser granularity (locking a connected component or canvas region) would reduce lock requests but increase contention. Finer granularity (per-attribute locking) would increase parallelism but add protocol complexity. The element-level granularity was chosen as a practical middle ground.

Limitations. The bounds presented are worst-case asymptotic estimates. Practical performance depends on network latency, contention patterns, and model size. The analysis assumes a single-server deployment; in a multi-server setting, cross-node coordination for locking and log replication may introduce additional overhead not captured by the per-operation bounds.

Table 3. Asymptotic Complexity of the Proposed Algorithms.

Algorithm	Time	Space
Pairwise transform	$O(1)$	$O(1)$
Server log catch-up	$O(\Delta v)$	$O(\|L\|)$
Client-side transform	$O(\|Q\|)$	$O(\|Q\|)$
Broadcast	$O(k)$	$O(1)$
Lock scope computation	$O(\deg(e))$	$O(\|P\|+\|T\|+\|F\|)$
Cascading deletion	$O(\deg(e)\cdot k)$	$O(\deg(e))$
Incremental validation	$O(1)$	$O(\|P\|+\|T\|+\|F\|)$

9. Conclusion and Future Work

We presented Goldfish, a collaborative web-based Petri net editor whose core contribution is the adaptation of Operational Transformation to graph-based Petri net structures. The domain-specific operation model and transformation rules preserve bipartite graph invariants and handle cascading structural dependencies that generic OT/CRDT frameworks do not address. The hybrid OT/locking mechanism resolves the majority of conflicts transparently while providing exclusive access for semantically critical operations.

Future work includes: (1) support for TVAW-Nets [20] with parametric (variable) arc weights; (2) CRDT-based synchronization for non-critical operations; (3) offline editing with reconnection merge; (4) integration with external model checking tools; and (5) user studies evaluating collaborative modeling effectiveness.

References

[1]. Murata T. Petri nets: Properties, analysis and applications. Proceedings of the IEEE, 1989, vol. 77, no. 4, pp. 541-580. DOI: 10.1109/5.24143.
[2]. Diaz M. Petri Nets: Fundamental Models, Verification and Applications. Hoboken, NJ: Wiley, 2009. 613 p.
[3]. Ratzer A.V., Wells L., Lassen H.M., Laursen M., Qvortrup J.F., Stissing M.S., Westergaard M., Christensen S., Jensen K. CPN tools for editing, simulating, and analysing coloured petri nets. In International Conference on Application and Theory of Petri Nets. Berlin: Springer, 2003, pp. 450-462.
[4]. CPN Group, Aarhus University. CPN Tools. Available at: http://cpntools.org/, accessed: 14.03.2026.
[5]. PIPE Development Team. Platform Independent Petri Net Editor (PIPE). Available at: https://github.com/sarahtattersall/PIPE, accessed: 14.03.2026.
[6]. Nikitina N., Mitsyuk A. Carassius: A simple process model editor. Proceedings of the Ivannikov Institute for System Programming of the RAS, 2015, vol. 27, no. 3, pp. 219-236.
[7]. HSE PAIS Laboratory. Carassius: Petri Net Editor. Available at: https://pais.hse.ru/research/projects/carassius, accessed: 01.12.2024.
[8]. TAPAAL Development Team. TAPAAL: Tool for Verification of Timed-Arc Petri Nets. Available at: https://www.tapaal.net/, accessed: 14.03.2026.
[9]. Verbeek E., Fahland D. CPN IDE: An extensible replacement for CPN Tools that uses Access/CPN. In 3rd International Conference on Process Mining (ICPM 2021). CEUR-WS.org, 2021, pp. 29-30.
[10]. University of Hamburg, PAOSE Group. PetriPad: A Web-Based Petri Net Editor. Available at: https://paose.informatik.uni-hamburg.de/paose/wiki/PetriPad, accessed: 14.03.2026.
[11]. VSB Technical University of Ostrava. Petri Net Editor. Available at: https://pes.vsb.cz/, accessed: 14.03.2026.
[12]. Google. Google Docs: Real-time Collaboration. Available at: https://workspace.google.com/resources/real-time-editing/, accessed: 14.03.2026.

[13]. Figma Inc. Figma: The Collaborative Interface Design Tool. Available at: https://www.figma.com/, accessed: 14.03.2026.
[14]. Ellis C.A., Gibbs S.J. Concurrency control in groupware systems. In Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data. New York: ACM, 1989, pp. 399-407.
[15]. Sun C., Ellis C. Operational transformation in real-time group editors: Issues, algorithms, and achievements. In Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work. New York: ACM, 1998, pp. 59-68.
[16]. Shapiro M., Preguiça N., Baquero C., Zawirski M. Conflict-free replicated data types. In Proceedings of the 13th International Conference on Stabilization, Safety, and Security of Distributed Systems. Berlin: Springer, 2011, pp. 386-400.
[17]. Lipton R.J. The reachability problem requires exponential space. Yale University, Department of Computer Science, Research Report 62, 1976. 15 p.
[18]. Xu Y., Sun C., Li M. Achieving convergence in operational transformation: Conditions, mechanisms and systems. In Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work. New York: ACM, 2014, pp. 505-518.
[19]. Lomazova I.A., Mitsyuk A.A., Rivkin A. Typed Petri nets with variable arc weights. In Business Process Management Workshops. Cham: Springer, 2024, pp. 483-495.
[20]. Jahns K. Yjs: Shared Editing Library. Available at: https://yjs.dev/, accessed: 14.03.2026.
[21]. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software, 2003, 359 p.

Информация об авторах / Information about authors

Владислав Анатольевич КАБАКОВ – студент магистратуры факультета компьютерных наук, стажёр-исследователь научно-учебной лаборатории процессно-ориентированных информационных систем НИУ ВШЭ. Области научных интересов: алгоритмы, сети Петри, формальные модели систем, архитектура программных систем, машинное обучение.

Vladislav Anatolyevich KABAKOV – Master's student at the faculty of computer science at the HSE University and research intern at the laboratory of process-aware information systems (PAIS Lab). Research interests: algorithms, Petri nets, formal system models, software system architecture, and machine learning.

Алексей Александрович Мицюк – канд. комп. наук, доцент факультета компьютерных наук, старший научный сотрудник научно-учебной лаборатории процессно-ориентированных информационных систем НИУ ВШЭ. Области научных интересов: алгоритмы, сети Петри, формальные модели систем, архитектура программных систем, анализ и визуализация процессов, взаимодействие человека и машин.

Alexey Alexandrovich MITSYUK – Cand. Sci. (Comp. Sci.), associate professor at the faculty of computer science of HSE University, senior researcher at the laboratory of process-aware information systems (PAIS Lab). Research interests: algorithms, Petri nets, formal system models, software system architecture, process analysis and visualization, and human-machine interaction.