Description

Title

System and Method for Closed-Loop Dosing Engine with ICD-10-Specific Control Overlays

2. Detailed Description of the System

2.1 Base TraceLoop Dosing Engine

The TraceLoop closed-loop dosing engine is an automated control system that continuously monitors patient data and adjusts therapeutic drug delivery in real-time. It consists of a network of sensors (e.g., biochemical monitors, vital sign inputs) and actuators (infusion pumps or dosing devices) governed by a central control algorithm. In operation, the engine maintains target physiological parameters by comparing live measurements to desired setpoints and computing dose adjustments. For example, if blood glucose rises above a set threshold, the TraceLoop controller can increase insulin infusion until the glucose falls into the target range. Multiple control loops run concurrently for different therapies (such as glycemic control, electrolyte balance, blood pressure management), with each loop dynamically titrating a specific drug or fluid to the patient's needs. This architecture provides a flexible platform capable of integrating additional decision layers and safety constraints without manual intervention.

2.2 Baseline Risk Model and Control Constraints

In the baseline TraceLoop system, patient risk factors inform the dosing strategy primarily through scalar weights and fixed safety limits. A risk assessment module computes a numerical risk index for adverse events based on patient attributes (e.g., age, organ function, comorbidities), which in turn modulates the aggressiveness of the control algorithm. Higher risk scores automatically bias the dosing commands toward more conservative values – for instance, by reducing the maximum rate of change of an infusion or imposing tighter thresholds for alerts. Additionally, the base engine enforces general control constraints: each drug has predefined dose ceilings and minimal dosing intervals that the loop will not exceed under normal operation. These baseline guardrails are static and globally applied, ensuring safe operation but not yet tailoring therapy to specific diagnoses beyond this one-size-fits-all risk adjustment. In essence, prior to the introduction of ICD-10 overlays, the dosing engine's adaptive behavior was limited to generic risk-based scaling and hard-coded exclusion rules.

2.3 Preliminary Multi-Loop Conflict Handling

Even without diagnosis-specific overlays, the TraceLoop architecture includes a foundational mechanism for handling interactions between simultaneous control loops. When multiple therapy channels are active, the system employs a hierarchy of priorities and simple rule-checks to avoid harmful conflicts. For example, core safety logic may mark certain drug pairs or actions as mutually exclusive – preventing, say, a vasopressor and a vasodilator from being infused through the same hardware channel at once[1]. Likewise, a life-critical infusion (such as an antidote or emergency pressor) can temporarily override non-critical dose adjustments on other loops. These preliminary conflict-handling rules are implemented in the base control unit as fixed conditions or priority flags, ensuring that obvious contradictions are averted. However, such an approach is inflexible and does not account for the nuanced ways different clinical diagnoses influence therapeutic trade-offs. This limitation is addressed by the introduction of a structured ICD-10 attribute overlay layer, described next, which generalizes conflict resolution and dosing adjustments based on each specific diagnosis.

2.4 ICD-10 Attribute Overlay

FIG. 2.4E illustrates the integration of an ICD-10 attribute overlay module within the TraceLoop engine. Patient diagnosis codes feed into this overlay logic layer, which modulates the core dosing algorithm's output to the drug delivery actuators. In effect, each active diagnosis in a patient's profile contributes a set of modification rules that dynamically shape the control parameters in real-time. This allows the dosing engine to account for condition-specific requirements and precautions on top of the baseline risk model.

FIG. 2.4A conceptually shows that the closed-loop control output is derived from both a baseline risk-weighted calculation and the superimposed diagnosis-specific overlays. While the baseline risk model provides a scalar adjustment (e.g., a general dampening of dose changes for a high-risk patient), the ICD-10 overlay contributes discrete control-layer influences tied to particular conditions. Each diagnosis may alter therapy delivery in ways beyond what a simple risk score would – for instance, by imposing a stricter dose cap for a drug, adding a precondition (gating) before an action can occur, or coordinating multiple loops in the presence of synergistic or antagonistic conditions. The overlay thus functions as a condition-aware control input, refining dosing decisions to align with clinical best practices for each specific diagnosis.

For each diagnosis, one or more overlay attributes can be defined to influence the dosing engine's behavior. In an exemplary embodiment, these attributes include:

- **Dose ceiling modifier:** Adjusts the maximum allowable dose or infusion rate for a given therapy when the diagnosis is present. This can lower a dose limit (for safety in vulnerable conditions) or raise it (to ensure efficacy in tolerant conditions), overriding the default limits set in §2.2.
- **Gating rule:** Introduces conditional logic that must be satisfied before a dose is delivered or adjusted. A gating rule may require certain physiological criteria to be met (e.g., a lab value threshold) or enforce a delay under specific diagnoses. It effectively gates the actuation of a control loop until the condition is appropriate.
- **Synergy behavior:** Defines coordinated adjustments between two or more control loops due to the diagnosis. Synergy attributes cause the engine to consider cross-loop effects –

for example, automatically pairing an insulin dose with a dextrose infusion in a patient with hyperkalemia, or scaling one drug's effect in tandem with another to achieve a combined therapeutic goal.

- **Mutual exclusion constraint:** Specifies that certain therapies or actions should not occur simultaneously in the context of the diagnosis. This overlay attribute will disable or suspend one control loop's activity if a conflicting condition or treatment (identified via another code or drug class) is active, thereby preventing dangerous combinations.

FIG. 2.4B illustrates an overlay compilation algorithm by which the system merges multiple active ICD-10-based modifiers into a unified set of control commands. When a patient has more than one diagnosis, the TraceLoop engine iterates through each active overlay rule in turn, compiling their effects on the base dose calculation. The algorithm evaluates all dose ceiling adjustments, gating conditions, synergy links, and exclusions contributed by the active diagnoses and combines them systematically. In some implementations, this is done by sequentially applying rules in order of priority or severity, updating the candidate dose after each rule. If two overlays affect the same parameter (e.g., two conditions each lower the dose ceiling), the compilation logic applies the more restrictive outcome (in this case, the lower ceiling) to ensure patient safety.

FIG. 2.4D depicts a representative conflict-resolution graph employed to resolve any contradictory or overlapping rules among the active diagnosis overlays. In this model, each node represents a diagnosis-specific overlay rule set, and directed edges indicate override or influence relationships between diagnoses. For example, an edge from node A (a higher-priority condition) to node B denotes that if diagnoses A and B are both present, A's rules will supersede B's. In one embodiment, the conflict-resolution graph operates within a hierarchical arbitration logic having multiple safety layers, where higher-priority loops can programmatically cancel or delay lower-priority actuations[2], and mutually exclusive guards are encoded to prevent simultaneous execution of incompatible actions[1]. This structured approach (which may be implemented in software and firmware) guarantees a deterministic merge of all active rules. For instance, if a patient's profile includes an active hemorrhage diagnosis alongside an anticoagulation requirement, the overlay graph ensures that the anticoagulant infusion is automatically suppressed under the hemorrhage condition. The result is a harmonized control output that accounts for all relevant diagnoses without need for manual intervention.

FIG. 2.4C provides several clinical examples of the ICD-10 overlay in action, corresponding to the representative scenarios detailed in Table 1 below. These examples illustrate how the overlay schema tailors the closed-loop control logic to specific diagnoses. For instance, in a patient with diabetic ketoacidosis, the insulin dosing loop is automatically gated by the patient's potassium level – the overlay prevents insulin infusion if serum K^+ is too low, reflecting standard DKA protocols. In another example, a patient with hypo-osmolality (hyponatremia) triggers an overlay that caps the rate of sodium correction to prevent osmotic injury, regardless of how aggressively the base loop might otherwise try to correct the sodium. Further scenarios are tabulated in Table 1, demonstrating the breadth of adaptive behaviors enabled by various ICD-10 overlays. Each row in the table maps a bedside control loop and a diagnosis to the specific control-layer modifiers that the overlay applies.

ICD-10 Code (Diagnosis)	Dose Ceiling Modifier	Gating Rule	Synergy Behavior	Mutual Exclusion (if any)
E10.10 – Type 1 diabetes w/ ketoacidosis (no coma)	High initial insulin dose allowed (to overcome insulin resistance)	Hold insulin if $[K^+] <$ 3.3 mmol/L	Auto-initiate KCl and fluid co-infusion with insulin	—
E87.5 – Hyperkalemia	—	Require dextrose co-infusion with insulin	Use insulin–dextrose pairing to drive K^+ into cells	—
E87.1 – Hypo-osmolality (Hyponatremia)	Limit Na^+ increase to ≤8 mmol/L per 24 h (caps hypertonic saline rate)	No hypertonic bolus unless severe symptoms (otherwise slow infusion)	Coordinate with fluid restriction (limit free water intake)	Block concurrent hypotonic-fluid infusions
E87.0 – Hyperosmolality (Hypernatremia)	Limit Na^+ decrease to ≤10 mmol/L per 24 h (caps hypotonic fluid rate)	Stop hypotonic infusion once Na^+ nears safe range (prevent overshoot)	Sync IV and enteral free-water delivery (if available)	No hypertonic saline while Na^+ remains elevated
I50.9 – Heart failure (CHF), unspecified	Max IV fluid rate reduced (e.g., ≤250 mL/h)	Pause fluid infusion if signs of pulmonary edema	Prefer vasopressor/inotrope support over fluid loading for hypotension	Avoid aggressive fluid challenges by default
I61.9 – Intracerebral hemorrhage (ICH)	Cap blood pressure target (e.g., MAP ≤110 mmHg) by limiting pressor dose	Do not elevate BP above target; suppress pressors if BP adequate	Deep sedation loop engagement to prevent agitation-driven BP spikes	Halt any anticoagulant infusion algorithms
K25.0 – Gastric ulcer,	—	—	Elevate blood transfusion priority (prompt	Stop any heparin/antico

ICD-10 Code (Diagnosis)	Dose Ceiling Modifier	Gating Rule	Synergy Behavior	Mutual Exclusion (if any)
acute with hemorrhage			earlier transfuse trigger)	agulant infusion
G47.33 – Obstructive sleep apnea (OSA)	Reduce maximum opioid infusion rate (e.g., 50% usual max)	Auto-pause opioid infusion if apnea or desaturation detected	Utilize multimodal analgesia (add non-opioid analgesics to reduce opioid need)	Avoid concurrent sedative infusion unless necessary
Z79.891 – Long-term current use of opiates	Increase opioid dose cap (~150% of standard)	—	Consider adjunct infusion (e.g., low-dose ketamine) to enhance analgesia	—
N18.6 – End-stage renal disease (ESRD)	Halve infusion rates for renally cleared drugs	Extend interval between dose adjustments (slower titration)	Coordinate dosing with dialysis sessions (prevent drug removal timing issues)	Avoid nephrotoxic infusions (e.g., contrast dye)
K72.90 – Hepatic failure, unspecified (no coma)	Halve infusion rates for hepatically metabolized drugs	—	Prefer alternative medications cleared renally if possible	Block hepatotoxic drugs (e.g., acetaminophe n infusion)
J45.50 – Severe persistent asthma	Limit beta-blocker infusion dose; use cardio-selecti ve agents only	Suspend beta-blocker infusion if bronchospas m occurs	Use alternative rate control (e.g., calcium channel blocker) if needed	No non-selective beta-blocker infusions
Z88.5 – Personal history of narcotic allergy	—	—	Switch to non-opioid analgesic infusion loop	Block any opioid analgesic infusions

ICD-10 Code (Diagnosis)	Dose Ceiling Modifier	Gating Rule	Synergy Behavior (alternate therapy)	Mutual Exclusion (if any)

The above examples are illustrative of the many possible control adaptations enabled by the ICD-10 overlay framework. By disclosing these representative scenarios, this specification enables and encompasses all analogous permutations of diagnosis-specific control modifications via the same schema and rule-merge logic. In practice, any individualized, dynamic control-layer modulation based on a patient's diagnoses – even for conditions not explicitly listed in Table 1 – is achieved by defining the corresponding ICD-10 overlay entries as taught herein. Accordingly, the accompanying claims should be interpreted to cover any such integration of diagnosis-based rules into a closed-loop dosing system. The ICD-10 attribute overlay approach thus provides a generalizable and extensible mechanism for personalized, condition-aware dose control that can be expanded to new diagnoses or modified guidelines without departing from the scope of the invention.

2.5 Medication-&-Demographic Context Overlay (MD-Overlay)

Adds a fourth control-layer input that fuses (i) active-medication intelligence and (ii) foundational patient demographics with the ICD-10 Attribute Overlay (§ 2.4) and baseline Risk Model (§ 2.2).

(a) Active-Medication intelligence.
The engine queries the enterprise e-MAR / pharmacy database at configurable intervals $\langle \Delta t\_Rx \rangle$. For every drug *d* on the patient's prescription list **PRx**, it imports:

Field	Symbol	Use in MD-Overlay
Drug ID & route	idd, route	Maps to control loop(s) that deliver or antagonize *d*.
Last-fill date	tfill	Computes **recency score** $\rho d = \exp[-\lambda \cdot (now - tfill)]$ with $\lambda \approx 0.03\ day^{-1}$.
SIG / daily dose	σd	Cross-checks TraceLoop dose proposals for cumulative-exposure caps.

Known allergy / ADR flags	ADRd	Converts to *hard mutual-exclusion edge* (force-block).

ρd (0 … 1) enters the risk equation as a multiplicative factor on any overlay rule referencing *d*: recent fills (ρ ≈ 1) are treated as active co-medication; remote fills (ρ → 0) are down-weighted but **never** ignored—clinicians must reconcile status. The audit log records ρd per cycle, prompting chart review when below a configurable alert threshold (default 0.25).

(b) Demographic scalars.
Age (A), weight (W), height (H), sex (S) feed a demographics vector **D** that augments τ[time_to_harm] and ω[reversibility]:

- **Pediatric < 18 y** ⇒ halve τ for hypoglycaemia & fluid-shift loops (faster decompensation).
- **BMI > 35** ⇒ widen ω for positional airway-risk loops (harder reversal).
- **Elderly > 75 y** ⇒ global dose-rate factor α=0.7 (slower titration) unless overridden by ICU-stat order.

Figure 2.5A (to be supplied) shows the additive "ring" architecture: baseline Risk core → ICD-10 overlay → MD-Overlay → final arbitration. Figure 2.5B depicts ρd decay curves and alert thresholds.

(c) Generalized deployment.
Figure 2.5C outlines system scope from ambulatory infusion centers to med-surg wards and ICUs. A single canonical **LOOP+OVERLAY schema** drives any drug-fluid administration node:

- Outpatient biologic infusion (e.g., infliximab): MD-Overlay flags concomitant immunosuppressants; τ scaled by infusion-reaction risk.
- Dialysis unit EPO dosing: ESRD ICD-10 weight + ρd for iron supplements adjust anemia-loop target.
- Home parenteral nutrition: demographics (weight trajectory) modulate lipid infusion ceiling.

Thus the invention expressly **covers all medication-fluid administrations** across the health system, not merely ICU devices, by virtue of its data-agnostic overlay pipeline and-loop schema.

> **Claim support (excerpt)** – "…wherein the control overlay further incorporates a medication-recency coefficient ρ computed from prescription-fill timestamps, and

demographic scalars derived from age, weight, height, and sex, the combined overlay being applicable to any automated drug- or fluid-delivery system within the healthcare enterprise."

Personalized ICD-10–Weighted TraceLoop Dosing Engine Specification

1. System Architecture and Overview

1.1 Closed-Loop Dosing Engine Context: The TraceLoop dosing engine is a hierarchical, closed-loop control system designed for high-acuity medical therapy. It continuously monitors multiple physiological **sensor loops** and drives corresponding **therapeutic actuators** (e.g., infusion pumps, ventilator valves) to maintain patient homeostasis. Each sensor-actuator pair operates as a **control loop** (also called a "factor" or channel) that can request dosing adjustments. These loops compete for shared resources (e.g. multiple drugs sharing one IV line), so the engine implements a **conflict arbitration model** to decide which loop's command takes precedence at any moment. The architecture supports **multi-actuator coordination**: at each control cycle, the engine evaluates all loops and can simultaneously drive different actuators in parallel, selecting the highest-priority compatible loop for **each** actuator conflict group. This ensures, for example, a ventilator setting and an infusion pump can be adjusted concurrently as long as they do not conflict on the same hardware.

1.2 Conflict Groups (Actuator Mapping): Each control loop is tagged with a **conflict_group**, denoting the physical actuator or output channel it targets . A conflict group acts as a namespace for shared actuators (e.g., "IV_PUMP_1", "VENTILATOR") . All loops in the same conflict group vie for the same hardware, so only one loop in each group may execute at a time. The dosing engine maintains a **priority queue per conflict group**; loops in different groups do not block each other and can run concurrently . This structure maps the multi-loop control problem into separate per-actuator arbitration problems, simplifying real-time scheduling. *(See FIG. C for an illustrative Conflict-Group Actuator Arbitration Map, showing loops grouped by actuator and the arbitration decision per group.)*

1.3 Rule Database and Canonical Schema: All control logic is derived from a **canonical rule table** that encodes each loop's attributes and relationships. Each loop is represented as a row with standardized fields (columns) capturing its identity, risk factors, and conflict relationships. At runtime, the engine ingests this table (e.g., from a CSV or database) and **canonicalizes** it into an internal graph representation . This graph-based model is used to enforce dependencies and make arbitration decisions deterministically. The canonical schema includes (but is not limited to) the following key fields for each loop (factor):

- **id and label:** Unique loop identifier and human-readable name.
- **category / organ_system:** Classification of the loop's clinical domain (e.g., Hemodynamic, Renal) and target organ system.
- **harm_severity:** Integer 1–9 indicating the clinical harm level if the loop's condition remains uncorrected (derived from hazard analysis, e.g. ISO-14971).
- **time_to_harm:** Qualitative urgency indicator (minutes / hours / days) reflecting the worst-case time until patient harm occurs without intervention.
- **reversibility_window:** Qualitative timeframe (short / moderate / long) after which the resulting harm becomes irreversible, even if therapy is applied.
- **conflict_group:** The shared actuator or resource name that this loop uses (e.g., an IV pump channel or ventilator) . Loops sharing a conflict_group cannot execute simultaneously.
- **priority_over:** List of loop id values that this loop **preempts** if both request action (a *hard precedence* relationship) . If loop A is listed in another loop B's priority_over, A will always win priority over B in conflicts.
- **mutually_exclusive:** List of loop IDs that cannot run at the same time as this loop (a symmetric *mutual exclusion* constraint) . If any listed loop is currently active, this loop is blocked (and vice versa).
- **requires_ok:** List of loop IDs that act as gating prerequisites for this loop (a *gating dependency*) . The loop will only execute if all listed loops are in a safe or "OK" state (e.g., a ventilator loop might require that an oxygenation loop is okay before proceeding).
- **synergy_with:** List of loop IDs that have a beneficial co-operation relationship with this loop . If the top-priority loop has a synergy_with partner also pending, the engine may co-schedule them in the same cycle (e.g., two drug infusions that should start together) to achieve a combined effect.

Each loop's row may include additional metadata (sensor modality, clinical rules text, dose limits, etc.), but the above fields define the core **conflict graph** relationships and risk parameters. The rule database can be maintained as a relational schema (e.g., a FACTOR table for loops and a RELATION table for edges as in FIG. 3 of the broader architecture) . **Compile-time validators** check this schema for consistency – for example, ensuring each conflict_group is set, no cycles exist in priority_over, mutual_exclusion lists are symmetric, and all referenced IDs exist . This guarantees the graph is well-formed before the dosing engine runs.

2. Risk Scoring Model with ICD-10 Weighted Logic

2.1 Base Risk Score Calculation: To arbitrate between competing loops, the engine computes a numerical **Risk(L)** score for every loop *L*. This base risk score is derived from the loop's harm severity and timing parameters. In one representative embodiment, the formula is:

Risk(L) = \frac{\textit{harm_severity} \times \tau[\textit{time_to_harm}]}{\omega[\textit{reversibility_window}]} \tag{1}

Here, **τ[ ]** and **ω[ ]** are lookup tables that assign scalar weighting factors based on the time urgency and reversibility classifications . For example, the τ table may define multipliers such that an imminent issue (minutes) = 1.0, hours = 0.5, days = 0.1, reflecting that a loop with minutes to harm is weighted more urgently . Similarly, ω might weight the reversibility window (e.g., short window = 0.75, moderate = 1.0, long = 1.5) so that conditions which are **harder to reverse** (short window) yield a higher Risk score (note the inversely proportional effect in the formula) . The product of harm_severity × τ[…] produces a baseline hazard score, which is then scaled by 1/ω[…] to elevate loops with narrow recovery windows. This Risk(L) is a dimensionless priority metric that **feeds into the arbitration queue** for the loop's conflict group .

2.2 Integration of ICD-10 Diagnosis Weights: A key personalization feature is the incorporation of patient-specific **ICD-10 diagnosis codes** as scalar weight modifiers in the risk model. Each relevant ICD-10 code (chronic condition, comorbidity, etc.) is mapped to a predefined weight factor reflecting how that diagnosis influences the urgency or severity of certain loops. These weights are loaded from a configuration file or database (for instance, a CSV mapping ICD-10 codes to weight values and affected loop categories). The weighting logic combines the real-time risk parameters with these preloaded ICD-based weights as follows:

- For each loop L and each ICD-10 code *d* that the patient has, the engine looks up a weight scalar w_{d,L}. This scalar w_{d,L} (>1.0 for risk-amplifying conditions, or <1.0 for risk-mitigating factors) represents how much having diagnosis *d* should scale the risk of loop L. For example, an ICD-10 code for **chronic kidney disease** might increase the risk weight on a **potassium (K^+) control loop**, since impaired renal function makes hyperkalemia more dangerous and less reversible.

- The loop's final **ICD-adjusted risk** can be computed by multiplying the base Risk(L) by the combined ICD weight factor(s). In one embodiment, a single **aggregate weight factor** \Theta_L is calculated per loop: \Theta_L = \prod_{d \in D_P} w_{d,L}, where D_P is the set of the patient's diagnoses relevant to L. Then Risk_{ICD}(L) = Risk(L) \times \Theta_L. Alternatively, other combinations may be used (such as applying only the maximum weight from among applicable codes, or summing weight adjustments) to ensure no single loop's risk skews excessively. The design is flexible: in all cases, the ICD-10 weights act as **scalars** that modulate the risk score upward or downward.

- **Example:** Consider a loop controlling anticoagulation via heparin (monitoring Anti-Xa levels). If a patient carries an ICD-10 code for *bleeding disorder*, the system may assign

a weight w_{\text{bleed},\,\text{AntiXa}} = 1.3, increasing the risk score for the **Anti-Xa loop** by 30%. This reflects that in a patient prone to bleeding, overshooting anticoagulation is more harmful, thus that loop's priority is effectively elevated to avoid excessive dosing. Similarly, a diagnosis of *acute respiratory distress* might raise the weight on a ventilator oxygenation loop, etc. These weights are determined from clinical data or expert input and are stored in the device as a configurable table.

The ICD-10 weighting integration makes the risk model **personalized**. It ensures the arbitration engine is sensitive to individual patient conditions: loops addressing more critical patient-specific vulnerabilities gain priority proportional to those vulnerabilities. The weights are scalar and multiplicative, preserving the deterministic nature of the Risk(L) formula while adjusting it per patient profile. Figure B illustrates how an example **risk priority queue** is resorted when ICD-10 weight overlays are applied: a loop that was second in base risk can move to top priority if the patient's diagnosis weight boosts its score (FIG. B shows the **Risk Queue Sorting with ICD-10 Weight Overlay**).

2.3 Risk Priority Queues: Once each loop's Risk(L) (with ICD-10 adjustments) is computed, the engine populates a **priority queue** for each conflict group with the active loops. The risk score serves as the primary sort key. Within each conflict group's queue P^c, loops are ordered by descending risk score . Ties or close scores are broken by secondary rules: e.g., a predefined topological rank (from the priority_over hierarchy) and then by timestamp (oldest request first) . This ensures stable and fair ordering even if two loops have identical risk values. The highest-risk loop that is **eligible** (not excluded by any gating or mutual exclusion rule) will be at the head of the queue, ready for execution. By incorporating ICD-10 weights into Risk(L), the queue inherently reflects patient-specific priorities – effectively, the patient's diagnoses tilt the queue in favor of the most crucial therapy at that time.

(Refer to FIG. B for an example visualization of two priority queues (for two actuators) and how a diagnosis weight shifts the ordering in one queue, highlighting the dynamic reprioritization.)

3. Conflict Graph Arbitration Engine

3.1 Graph-Based Priority Resolution: Internally, the dosing engine represents all loops and their relationships as a directed graph **G(V,E)**, where each vertex v \in V is a control loop and edges E encode the constraints from the rule table . Specifically, four types of edges capture the conflict logic:

- **Precedence edges (priority_over):** A directed edge from loop A to loop B (A → B) indicates A has higher priority and will pre-empt B if both are contending for the same conflict group . These edges yield a partial order (they are kept acyclic by design) that the engine uses as tie-breakers in scheduling. Essentially, priority_over defines a **hierarchy or ladder of precedence** among loops (like an override ladder). If a higher-ranked loop needs the actuator, lower-ranked ones are deferred. This creates a

Conflict Arbitration Ladder as shown in **FIG. A**, where loops are arranged on rungs according to priority. FIG. A conceptually illustrates how a loop higher on the ladder will suppress those below it in real time arbitration.

- **Gating edges (requires_ok):** A directed edge A → B in requires_ok means loop B can run only if loop A is in a satisfactory state (often indicating B addresses a secondary issue that should only activate when A – a primary loop – is under control) . In the graph, a gating edge acts like a dependency: B is *blocked* unless A's condition is resolved or within safe bounds. These edges partition the graph into dependency chains, ensuring, for instance, a corrective loop does not activate until a prerequisite stabilizing loop has given an "OK" signal.

- **Mutual-exclusion edges:** These are bidirectional (undirected) links between loops that cannot operate together . The graph stores them as symmetric connections (A ↔ C) typically because the two therapies would conflict if simultaneous (for example, two drugs that are chemically incompatible or physiological antagonists). The engine will not allow both ends of a mutual-exclusion pair to run in the same cycle. In practice, if one loop is active, the other is temporarily disabled. Mutual exclusions are checked dynamically each cycle.

- **Synergy edges (co-schedule relationships):** These are bidirectional links with a co-scheduling intent . If loop X has a synergy_with link to loop Y (and vice-versa), it means that if both X and Y become active around the same time, the engine should consider running them together for combined effect (within some Δt time window). The graph marks these relationships, and during arbitration, if the top candidate loop has a synergistic partner also queued, both may be executed in the same cycle (this is a **coordinated actuation** feature). An example would be two complementary drug infusions that yield best results if started simultaneously.

Using these edges, the engine constructs a conflict-resolution graph on startup. It performs a **topological sort** of any precedence (priority_over) chains within each conflict group to assign a static rank ordering . Cycles in these relations are disallowed and caught at compile-time (it is not possible for A priority_over B and B priority_over A, etc.) . This sorted order (the "ladder") is used in arbitration as a tiebreaker after risk scoring. The graph also records mutual exclusions and gating which are evaluated dynamically.

3.2 Real-Time Arbitration Algorithm: The core control logic runs in a loop (the control cycle) that continually arbitrates which loops get to execute. Pseudocode for one cycle of the **arbitration algorithm** is as follows (executed independently for each conflict group c):

1. **Gather Enabled Loops:** Start with set P^c = all loops in conflict_group c that are currently **enabled** (i.e. their sensor trigger condition is met or they are requesting actuation) . If no loop in c is active, nothing happens on that actuator this cycle.

2. **Apply Mutual Exclusion:** For each candidate loop L in P^c, check if there exists any other loop M such that (L, M) is a mutually_exclusive pair and M is currently running . If so, remove L from P^c (L cannot run because its mutually exclusive counterpart is already active).

3. **Apply Gating Dependencies:** For each remaining loop L in P^c, check its requires_ok list. If any required loop is not in an "OK" state (i.e., its condition is not yet resolved, or it is actively requesting), then L is gated and cannot run . Such L is removed from P^c for this cycle. This ensures prerequisite conditions are respected.

4. **Sort by Priority (Risk and Precedence):** Sort the final set P^c by **(a)** descending Risk score (taking into account ICD-10 weight adjustments), **(b)** the topological priority rank (if two loops have close risk but one is designated higher via priority_over edges), and then **(c)** by timestamp order of request . This sorting produces an ordered **priority queue** for conflict group c, with the highest priority loop at the head . *(FIG. A's "Conflict Arbitration Ladder" conceptually covers steps 2–4: first knocking out disqualified loops, then ranking the rest.)*

5. **Execute the Top Loop:** The loop at the head of the queue (highest priority) is selected for execution on that actuator for this cycle . The engine dispatches the corresponding actuator command (e.g., open pump valve for loop L's drug and dose). That loop is now considered "running" or active.

6. **Check for Synergy Co-Schedule:** After selecting the head loop L, the engine checks if L has any synergy_with partner S that is also in the queue P^c and ready . If so (and if the partner's request falls within the allowable time window Δt for co-trigger), the engine will **co-schedule** S – meaning it will execute S's actuator command in parallel this cycle as well . This way two synergistic loops can run together. The co-scheduled loop (S) is effectively a second winner of this arbitration cycle for group c.

7. **Output and Logging:** The chosen actuation(s) are issued to the hardware drivers, and a detailed log entry is created documenting the decision. The log will include the timestamp, which loop(s) were executed (winner), which were not (losers), their risk scores, and which rule or edge determined the outcome . For example, a log might show that loop "ANTI_XA" won over "HEPARIN_PROTAMINE" due to a priority_over edge and higher risk . This provides an **explanation layer** for transparency and post hoc analysis, meeting regulatory expectations for ML-based or automated decision support .

Each cycle (which can be as fast as every few milliseconds in full sensor streaming mode) the above steps repeat for each conflict group. Thus, multiple actuators can be controlled simultaneously by picking one top loop per group each cycle. The result is a real-time, deterministic arbitration that respects both numeric risk priorities and graph-defined safety constraints. FIG. A (Closed-Loop Conflict Arbitration Ladder) and FIG. C (Conflict-Group

Actuator Arbitration Map) together illustrate this process: FIG. C shows how each group's top command is chosen, while FIG. A shows the ladder of checks (priority, mutual exclusion, gating) a given loop must clear to win control.

3.3 Conflict Resolution with ICD-10 Weight Influence: The integration of ICD-10 weights into this engine does not change the steps of the algorithm, but it changes the **inputs to step 4 (sorting)**. By adjusting risk scores as described in Section 2.2, patient-specific factors alter the outcome of the priority sort. In practical terms, an otherwise lower-priority loop can outrank another if the patient's diagnoses indicate it's more critical. The conflict graph edges (priority_over, etc.) still apply; for instance, if a lower-risk loop A has an explicit priority_over edge over loop B, A will win even if B's risk is slightly higher. But since the ICD weighting influences the calculated risk, it can in some cases invert which loop triggers unless a fixed priority edge dictates otherwise. This interplay is depicted in FIG. B, where the **Risk Queue Sorting with ICD-10 Overlay** highlights that the engine's queue ordering (and thus arbitration outcome) is **context-dependent** on the patient's ICD profile. The conflict graph thus effectively becomes **weighted by patient context** through the risk values on the vertices.

Importantly, the ICD-10 weighting does **not** violate any safety constraints: all gating and mutual exclusion rules remain in force, and hard priority_over relationships will still trump risk if specified (ensuring deterministic conflict resolution). The weights simply fine-tune the risk dimension of the decision-making, allowing a more personalized arbitration within the safe graph structure.

3.4 Multi-Actuator and Parallel Arbitration: Because the engine maintains separate queues for each conflict_group, it inherently supports parallel actuation. In a given cycle, each conflict group c will produce at most one winning loop (plus possibly a synergy partner) which gets executed. For example, if conflict_group "IV_PUMP_1" selects a vasopressor loop to run and conflict_group "VENTILATOR" selects a respiratory loop to adjust, both actions occur in parallel. The arbitration algorithm described is applied *per group*, so different groups do not interfere except through any global mutual exclusions defined (e.g., a ventilation loop could be mutually exclusive with a pump loop if, say, they should not occur together during certain procedures – but that would be an explicit rule). The system is deterministic and real-time, able to cycle through decisions fast enough (e.g., 10–100 Hz) to manage dozens of loops simultaneously. The architecture is **scalable**, having been tested with up to 1024 loops/nodes without missing real-time deadlines .

(FIG. C's Conflict-Group Actuator Arbitration Map can be consulted for an overview of multiple conflict groups operating concurrently – it conceptually maps groups to their current highest-priority loops in a snapshot of time.)

4. TraceLoop-Lite (Lab-Driven) vs TraceLoop-Full (Sensor-Streaming)

4.1 TraceLoop-Full – High-Frequency Sensor Streaming: In the full implementation, all sensors provide continuous or high-frequency data (e.g., every second or faster for critical parameters). The engine's control cycle runs very frequently (for instance, every 10 ms in some implementations) to pick up new data and adjust actuators promptly. This **TraceLoop-Full** mode is suitable for ICU environments with advanced sensors (e.g., arterial-line analyzers, real-time sweat or blood monitors). The risk calculation and arbitration steps described in Sections 2 and 3 are executed continuously. Because data is streaming, the engine can quickly detect changes (like a rising potassium or dropping oxygen saturation) and update Risk scores in real-time, triggering the appropriate loop without delay. The closed-loop behavior is tightly coupled to sensor input, achieving minute-by-minute (or second-by-second) homeostasis adjustments. This mode assumes sensors such as continuous glucose monitors, Anti-Xa sensors, etc., are available and feeding data into the system bus in real time.

4.2 TraceLoop-Lite – Scheduled Lab Input Handling: For settings with less sensor infrastructure, **TraceLoop-Lite** operates on discrete lab results and periodic measurements. In this variant, certain loops do not get continuous input but rely on lab values drawn every few hours. For example, an Anti-Xa level might be measured **q6h** (every 6 hours) in a patient on anticoagulation, or serum K^+ (potassium) might be checked **q4h** in a critical care ward. TraceLoop-Lite can ingest these lab results as they become available (e.g., via an HL7 lab interface or manual input) and update the corresponding loop's sensor value. The control algorithm then proceeds on the same principles, but the **timing of loop activation is driven by lab arrival**. If a lab result indicates an out-of-range value, the appropriate loop becomes enabled and gets a Risk score (with ICD weighting as applicable) just like in Full mode.

Handling infrequent updates requires additional logic for safety: the engine tracks the **age of the last data point** for each lab-driven loop. If a loop's data is stale (e.g., no new lab for several hours), the system can either **hold** that loop (assuming no news is good news, or that it should wait for fresh data) or apply a conservative extrapolation/decay of risk. One embodiment implements a **risk decay over time**: after each lab-driven actuation, the loop's Risk(L) will gradually decrease over the subsequent hours if no new measurement arrives, reflecting growing uncertainty – unless a domain-specific model is available to predict the trend. This prevents the engine from repeatedly dosing on an old result. Alternatively, the loop can be set to require an updated measurement (treated as **requires_ok** on a "Lab Update" condition) before re-activating. The specific strategy is configurable per loop. The key point is that TraceLoop-Lite uses the same arbitration engine, but in slower motion and with logic to avoid acting on outdated information.

4.3 Real-World Lab Scheduling Integration: The engine is aware of typical lab schedules and can integrate them into the control plan. For instance, if an **Anti-Xa** loop knows that its next lab result is expected at 6:00 AM (q6h schedule), it can time the tapering or pausing of heparin infusion to align with when fresh data arrives. Similarly, for **K^+** management on q4h labs, the engine can administer a potassium correction dose and then suppress further K^+ dosing until the next lab confirms the effect (using a requires_ok dependency on receiving that lab result). This coordination between lab timing and dosing prevents over-correction in the blind interval. Essentially, TraceLoop-Lite leverages the known periodicity: just as it uses **time_to_harm** in risk

scoring, it also respects **time to next data** in scheduling decisions. If a potential harm could occur before the next lab, the engine might act preemptively (with caution), whereas if ample time and a scheduled check is ahead, it might wait. All such logic remains rule-based and transparent.

4.4 Consistency with Full Mode: Importantly, the priority arbitration mechanism is unchanged between Lite and Full variants. Both use the conflict graph and Risk(L) prioritization. The difference lies in input frequency and perhaps in some loop configurations (like longer integration periods or added safety checks for lab-based loops). The ICD-10 weighting is equally applicable in both modes – patient diagnoses weigh into risk scores whether the data arrives continuously or intermittently. Thus, a patient's comorbidity will elevate a loop's priority even if that loop only triggers with lab input. The architecture is modular enough that a hospital could start in TraceLoop-Lite mode (using intermittent lab data) and upgrade to TraceLoop-Full (with continuous sensors) without altering the core arbitration logic – only the data feed changes. Both modes maintain a complete audit trail of decisions (the log notes each actuation along with its input source and timing), fulfilling traceability requirements.

5. Adaptive Learning and Control Optimization

5.1 Time-Windowed Learning ("μ-band" Passive Mode): To enhance personalization beyond static weights, the system includes an embodiment for **time-windowed learning** using so-called *micro-bands* (μ-bands). The idea is to allow the controller to learn patient-specific dose-response characteristics **without risking harm**, by observing system behavior in a safe range. Each control loop defines an inner safe band around the target value (for example, 10% inside the normal high/low limits). When the sensor reading for a loop drifts into this μ-band, the engine does **not immediately correct it with a pump action**. Instead, it holds off active intervention (the loop is temporarily passive) and uses this opportunity to observe how the system responds. These are effectively **training-tier thresholds**: slight deviations which are tolerated for learning purposes . Specifically, the engine records the natural changes and any ongoing actuator outputs from other loops, to estimate cross-coupling.

During these μ-band periods, the engine logs data such as the derivative of the sensor value and correlates it with concurrent actuator changes. For every other loop Y that is active, it computes \Delta \text{dose}Y / \Delta X *(how a change in loop X's sensor correlates with changes in Y's output) . Over multiple observations, these deltas inform a **patient-specific Jacobian matrix* J{patient}** of the system . The Jacobian's entries represent sensitivities between control loops (e.g., how much does adjusting insulin affect glucose level; how does a diuretic dose affect potassium, etc.). After accumulating a sufficient number of observations (for example, n = 30 data points) , the controller can update its internal model parameters (such as the expected effect of a drug) to better match this particular patient.

5.2 Jacobian-Based Adaptation: Once the patient-specific Jacobian J_{patient} is computed, the TraceLoop engine can adjust its dosing algorithms (the control model) accordingly. In practical terms, this means the system transitions from using population-average parameters to

using personalized ones. For instance, if the Jacobian suggests the patient is **less sensitive** to a certain drug (i.e., needs a higher dose for the same effect), the controller can increase that loop's dosing increment or frequency within safe limits. Conversely, if the patient is **highly sensitive**, the controller will scale down dosing to avoid overshoot. This adaptation is done in a **time-windowed, gradual** manner – typically by periodically recomputing the Jacobian and then blending the new coefficients (perhaps using a moving average or Kalman filter approach) so that control remains stable. The "μ-band" strategy ensures these adjustments are based on real observed cause-effect in the patient, not just static heuristics.

Crucially, all such adaptive changes are logged for transparency (each coefficient update can be recorded to the audit trail, complying with good ML practice by preserving a history of model changes). The system can also impose **safety caps** on adaptation – for example, not allowing the learned parameters to exceed certain bounds without clinician review (to prevent learning from any spurious data). Over time, the combination of ICD-10 weighted risk prioritization and Jacobian-based dose tuning yields a controller that is both **risk-aware (what to treat first)** and **response-aware (how strongly to treat it)** for the individual patient. This two-tier personalization (diagnosis-driven prioritization and data-driven gain tuning) is a novel aspect of the TraceLoop engine.

5.3 Embodiment Example – μ-tier Learning: *For illustration:* Suppose the system is managing a **zinc infusion loop** and a **copper infusion loop** in a patient on renal replacement therapy. These two loops can interact (chelating agents or one deficiency affecting the other). The controller sets inner μ-bands around normal Zn levels. When Zn drifts slightly low but within the μ-band, the engine holds the Zn pump and observes. Meanwhile, if the Cu pump is running, the engine notes how Zn level responds to Cu dosing. After several such occurrences, it learns a coupling coefficient between Cu and Zn loops (perhaps high Cu infusion tends to lower Zn). The Jacobian entry J_{Zn, Cu} would be updated. The next time, the system might proactively adjust zinc dosing when copper is infused, compensating for that interaction. This learning happens automatically, leveraging safe fluctuations. In essence, the controller is discovering cross-talk and adjusting its control law, an approach akin to adaptive control or iterative learning control. All of this occurs under the hood, ensuring the patient remains within safe limits (the μ-band ensures no extreme deviation is allowed during learning) .

This embodiment addresses one of the challenges of multi-loop systems: interdependence of physiological variables. By combining rule-based conflict arbitration with data-driven Jacobian updates, the TraceLoop dosing engine achieves both **robust safety** and **adaptive performance**.

6. Integration with Canonical Schema and Data Structures

6.1 Canonical Row Schema and ICD-10 Data: The logic described above is fully reflected in the system's data schema. Each loop's row in the canonical table (Section 1.3) contains the base parameters (severity, timings, relationships) which feed the risk engine and graph builder. To incorporate ICD-10 personalization, the implementation adds either an extension table or

extra columns for **diagnosis weights**. One approach is a separate **ICD-10 weight mapping table** keyed by ICD code, listing affected loop IDs or categories and the weight value. When the patient's ICD list is loaded, the engine cross-references this table to retrieve all weights relevant to the loops in the system. These weight factors (Θ) are then applied in the risk calculation before sorting the queues.

An alternative embodiment could store a pre-computed composite weight per loop in the loop's row itself (for example, a field icd_weight_factor that is calculated at patient admission by scanning their diagnoses). This field would initially default to 1.0 and be updated whenever diagnoses change. The advantage of the canonical schema approach is that all necessary information for arbitration (including personalized weights) resides in one data structure that can be hashed, audited, and validated. Indeed, by driving the entire logic from a unified table, the system guarantees consistency: as noted in the advantages, a **single canonical table drives all safety layers** of the system .

6.2 Conflict Graph Implementation: The engine's software (which may run on an embedded controller or a server) transforms the row data into in-memory structures. For example, it may use adjacency lists or matrices to represent the graph of edges (priority_over, requires_ok, etc.). The priority_over and similar lists from each row are parsed into graph edges at startup . The engine then performs the topological sort of each conflict group's precedence subgraph and assigns a **topo_rank** to each loop (perhaps storing it in a dictionary). It also pre-compiles quick lookup sets for each loop's mutually_exclusive partners and gating requirements for efficient check in the cycle. The risk parameters (harm_severity, etc.) are stored and updated per loop; ICD-10 weights are either incorporated by adjusting harm_severity dynamically or by maintaining a separate weight vector applied at runtime. All these data structures are kept in memory for real-time access, while the original table (and any dynamic changes) is preserved for audit.

During operation, when a sensor event comes in (or a lab result in Lite mode), the engine updates the corresponding loop's state (e.g., marks it "enabled" and updates its latest sensor value). Then on the next cycle, the priority queue for that loop's conflict_group is regenerated. This design ensures that any change in inputs (sensors, diagnoses, etc.) is quickly reflected in the decision-making outcome. The conflict graph also enables **rich explainability**: since every decision can be traced back to a combination of edges and risk comparisons, the system can produce human-readable justifications (as in the JSON log example in Section 3.2) . For regulatory and safety purposes, this traceability is crucial.

6.3 Safety and Overrides: The control model includes a manual override layer, as required in critical systems. A supervisor (clinician) can issue a **BLOCK** or **FORCE** command on a specific loop via a secure interface . BLOCK will prevent that loop from running, regardless of risk (the engine will treat it as disabled). FORCE will momentarily elevate a loop to run immediately, bypassing the normal priority logic (useful in emergencies), after which normal logic resumes . These overrides are also represented in the data model (e.g., an override register or flags associated with each loop ID) and the arbitration algorithm (step 0 before enabling loops might check for any override blocks, and after selection, check for any force command). The presence

of overrides is recorded in the logs too. The design ensures that even under override, the conflict safety is maintained (for instance, forcing one loop will still block its mutually exclusive counterpart to avoid direct conflicts).

6.4 Patent-Grade Embodiment Clarity: In summary, the personalized ICD-10–weighted TraceLoop dosing engine combines a *numerical risk model* with a *graph-based arbitration engine* to safely coordinate multiple therapy loops. The described architecture is implementable in software or firmware as a deterministic state machine augmented by adaptive learning. It addresses the challenges of multi-actuator closed-loop control with a novel integration of patient-specific diagnostic weights, ensuring that **what** to treat first is informed by both real-time physiology and known medical history. The use of a unified rule schema with fields like priority_over, mutually_exclusive, requires_ok, etc., provides a clear, auditable blueprint of the logic , which can be readily updated (e.g., adding a new loop or rule via a CSV import) without altering code. The risk scoring with τ and ω lookup tables and ICD-10 scaling offers a transparent equation-driven approach rather than opaque neural nets, making the system's decisions explainable and **patentably distinct** in its combination of features.

Through the mechanisms described – from conflict arbitration ladders (FIG. A) to ICD-weighted risk queues (FIG. B) to conflict-group maps (FIG. C) – this specification lays out a robust control model for an autonomous dosing engine that can be readily embodied in critical care devices. It is intended to meet the rigor of WIPO/USPTO patent standards, with precise definitions and a clear logical flow enabling one skilled in the art to implement the invention. All variations, such as the TraceLoop-Lite and Full modes, and the adaptive μ-band learning, are considered part of the possible embodiments of the core invention, which is defined by the claims (to be drafted separately). The described system achieves deterministic yet patient-tailored closed-loop therapy, pushing the state of the art in automated patient care while maintaining safety and compliance at every level.

Here's **FIG. A — Closed-Loop Conflict Arbitration Ladder**, visually describing how TraceLoop filters and ranks active loops through risk evaluation, safety edges, priority ordering, selection, and audit logging.