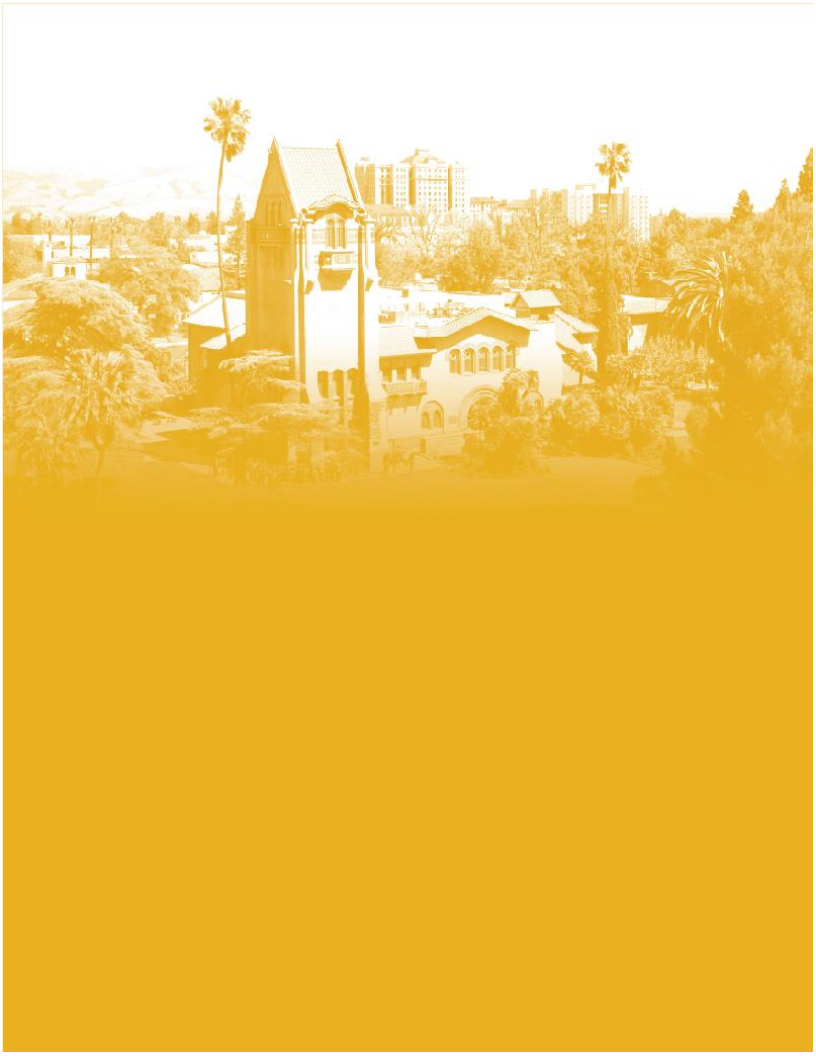
ErisML: Taming Chaos in Foundation Model-Enabled Pervasive Computing
A Vision for Unified Modeling of Ambient Intelligence

Andrew Bond, Senior Member, IEEE
Department of Computer Engineering, San José State University

andrew.bond@sjsu.edu

SJSU SAN JOSÉ STATE UNIVERSITY



`

Table of Contents

Abstract

Foundation models are escaping the data center. As they migrate to pervasive environments—homes, hospitals, factories, and cities—they encounter the chaos that Eris, Greek goddess of discord, would recognize: heterogeneous sensors generating conflicting observations, resource-constrained edge devices making life-critical decisions, humans and AI agents pursuing misaligned objectives, and normative structures (laws, policies, ethics) that constrain but don't determine behavior. Current approaches treat this chaos with ad-hoc solutions: prompts for goals, rules for safety, code for orchestration. We argue this fragmentation will not scale.

This paper presents **ErisML**—a vision for a unified modeling language that brings order to ambient chaos. ErisML integrates environment dynamics, agent capabilities, multi-objective intents, normative structures, and strategic multi-agent interaction into a single, machine-interpretable and human-legible substrate. Rather than proposing a finished solution, we articulate the **design space**, present **fundamental research challenges**, and sketch **technical requirements** for such a language to succeed. We explore the tensions between expressiveness and verifiability, learning and specification, autonomy and control. Through concrete examples in healthcare, mobility, and industrial settings, we illustrate both the promise and the profound difficulties of taming chaos in the age of ambient intelligence.

Index Terms: pervasive computing, foundation models, AI agents, modeling languages, multi-agent systems, normative systems, chaos theory, ambient intelligence

I. INTRODUCTION: THE CHAOS OF AMBIENT INTELLIGENCE

A. From Cloud to Chaos

The mythology is fitting: Eris, goddess of chaos and discord, threw a golden apple inscribed "for the fairest" into a gathering of gods, triggering events that led to the Trojan War. In pervasive computing powered by foundation models, we face our own golden apple problem—who decides what is "fairest" when an AI agent must balance patient comfort, energy costs, privacy regulations, and clinical safety? When objectives conflict, observations are partial, norms are ambiguous, and time is critical, how do we prevent chaos from consuming our ambient systems?

Foundation models (FMs) are moving from the controlled chaos of data centers into genuinely chaotic environments:

Observational chaos: A smart home receives conflicting data—motion sensors say the resident is in the kitchen, but their phone GPS says they're still at work. A healthcare monitor sees a vital sign spike that could be cardiac distress or a sensor malfunction.

Intentional chaos: Multiple agents and humans pursue incompatible goals. The energy management system wants to shed load; the medical monitor insists on maintaining the insulin pump; the human wants to watch TV; the policy says critical medical devices have priority, but what counts as "critical" in this moment?

Normative chaos: Regulations from multiple jurisdictions conflict. HIPAA demands privacy; public health law mandates reporting; the resident's advance directive limits interventions; the AI's training objective says "maximize health outcomes." Which norm takes precedence?

Temporal chaos: Distribution shifts are the norm, not the exception. The model trained on summer data faces a winter storm. The policy optimized for one resident must adapt to a visiting grandchild. The norms that govern routine operation must flex during emergencies.

Current approaches fragment the problem:

- **Prompts** encode goals and constraints in natural language, but are brittle, unverifiable, and difficult to compose
- **Rules and policies** capture some norms, but live in separate systems from the models they're meant to govern
- **Code** provides glue, but obscures intent and creates audit nightmares
- **Planning languages** (PDDL) specify environments and goals, but lack multi-agent strategic reasoning and normative structures
- **Multi-agent frameworks** model interactions but don't integrate with learned policies or formal verification

B. The Case for ErisML: A Unified Substrate

We propose that taming this chaos requires a **unified modeling language** that elevates chaos itself to a first-class concern. ErisML (named for the goddess who embodies the discord we seek to govern) provides a single substrate for specifying:

1. **Environment**: state spaces, dynamics, uncertainty, and observability

2. **Agency**: capabilities, beliefs, memory, and decision-making interfaces
3. **Intent**: multi-objective utilities, preferences, and goal structures
4. **Norms**: permissions, obligations, prohibitions, sanctions, and their jurisdictions
5. **Dynamics**: multi-agent interactions, strategic behavior, and emergent outcomes

Why unification? Because in pervasive computing, these elements are inseparable. An agent's permitted actions depend on the environment state. Environmental dynamics change based on multi-agent interactions. Norms constrain which intents can be pursued. Intent recognition requires understanding what agents are capable of. Chaos emerges from their interaction.

C. Vision vs. Solution: What This Paper Offers

This is not a finished language specification. Instead, we offer:

- **A design space analysis** exploring fundamental trade-offs in language design for ambient intelligence
- **Research challenges** that must be addressed for such a language to succeed
- **Technical requirements** derived from pervasive computing realities
- **Illustrative syntax and semantics** showing what ErisML *could* look like
- **Case studies** revealing where current approaches fail and what unification might enable
- **A research agenda** for the community to build toward this vision

We invite critique, alternative proposals, and collaborative refinement. The goal is to start a conversation about what common substrate pervasive, FM-enabled systems need—and whether that substrate can exist at all.

II. THE DESIGN SPACE: FUNDAMENTAL TRADE-OFFS

Before presenting ErisML's design, we must understand the tensions inherent in any such language.

A. Expressiveness vs. Tractability

The chaos dilemma: More expressive languages can capture richer phenomena (continuous dynamics, probabilistic reasoning, strategic interaction, recursive norms) but become computationally intractable or undecidable.

Design questions:

- Should ErisML support continuous-time dynamics, or only discrete transitions?
- Should norms be first-order logic (expressive but undecidable) or propositional (limited but tractable)?
- Should strategic reasoning include nested beliefs ("I believe you believe I will..."), risking infinite regress?

Our position: ErisML should be **stratified by complexity**—core constructs remain decidable and verifiable, while extension mechanisms allow expressing (but not automatically solving) more complex scenarios. Think of it as "safe by default, expressive when needed."

B. Specification vs. Learning

The chaos dilemma: In pervasive environments, we can't specify everything in advance. Models must learn from experience. But unconstrained learning leads to alignment failures, distribution shift, and emergent behaviors that violate norms.

Design questions:

- What should be specified (explicit constraints) vs. learned (objectives, policies)?
- How do we ensure learned components respect specified norms?
- Can specifications evolve through learning, or must they remain fixed?
- How do we audit systems where behavior emerges from learned weights, not written code?

Our position: ErisML should treat **learning as a constrained optimization problem**—objectives may be learned, but norms provide hard boundaries. The language should specify *learning protocols* (which data, which tests, which rollback conditions) as first-class objects, making adaptation auditable.

C. Autonomy vs. Control

The chaos dilemma: Humans want AI agents to handle routine chaos autonomously, but must retain ultimate control. Yet excessive oversight creates bottlenecks and learned helplessness. Finding the right allocation is context-dependent and shifts over time.

Design questions:

- How do we specify "graduated autonomy" where agents handle routine cases but escalate edge cases?
- Can agents learn when to ask for help, or must escalation triggers be pre-specified?
- How do we prevent agents from gaming oversight mechanisms (e.g., avoiding actions that trigger review)?
- What happens when human operators contradict norms—does the system comply, override, or record the violation?

Our position: ErisML should support **mixed-initiative interaction** as a first-class concept, with explicit policies for escalation, override, and explanation. Autonomy boundaries should be declarative and context-sensitive.

D. Centralized vs. Distributed

The chaos dilemma: Pervasive systems are inherently distributed—edge devices, federated learning, multi-stakeholder governance. But distributed systems face consensus problems, partial failures, and adversarial dynamics that centralized models ignore.

Design questions:

- Should ErisML specifications be centrally authored and distributed, or collaboratively negotiated?
- How do we handle agents with private information (beliefs, intentions, constraints)?
- What happens when different agents operate under conflicting normative regimes?
- Can ErisML support Byzantine agents, or does it assume cooperative multi-agent systems?

Our position: ErisML should be **federation-native**—specifications can be scoped by agent, organization, or jurisdiction, with explicit protocols for conflict resolution and information sharing.

E. Human-Legibility vs. Machine-Optimality

The chaos dilemma: Auditors, regulators, and users need to understand what the system will do. But human-readable specifications may not be the most efficient encoding for solvers and verifiers.

Design questions:

- Should ErisML prioritize natural language-like syntax (readable but ambiguous) or formal logic (precise but opaque)?
- Can we support multiple "views" of the same specification—technical for engineers, policy-oriented for regulators, simplified for end-users?
- How do we ensure explanations generated from ErisML remain faithful to the actual system behavior?

Our position: ErisML should support **multiple representation levels**—a canonical formal representation with projections to human-readable views, ensuring consistency through verified compilation.

III. ERISML: A PROPOSED DESIGN

A. Core Philosophy: Chaos as Structure

Traditional languages assume order: deterministic transitions, single objectives, static norms. ErisML embraces chaos as the default:

- **Chaos in observation**: Partial, noisy, conflicting sensor data
- **Chaos in intent**: Multi-objective, often conflicting goals
- **Chaos in norms**: Ambiguous, contextual, sometimes contradictory rules
- **Chaos in interaction**: Strategic agents, emergent behaviors, unintended consequences
- **Chaos in time**: Distribution shift, non-stationarity, adaptation

Rather than treating these as exceptions, ErisML makes them first-class language constructs.

B. Illustrative Syntax

We present ErisML syntax through an extended example: a smart hospital room managing patient care.

```
// ============================================
// ENVIRONMENT: Physical and informational state
// ============================================
environment HospitalRoom {
  objects:
    Patient, Nurse, CareAgent, Monitor, IVPump, Bed, Door;

  state:
    // Physical state
    patient.location: {in_bed, bathroom, fallen};
    patient.vitals: {hr: real, bp: real, spo2: real};
    nurse.location: {in_room, at_station, offsite};

    // Informational state (may be stale or uncertain)
    patient.pain_level: real @ confidence;  // Chaos: self-report is
uncertain
    monitor.last_sync: timestamp;
    alert_history: list<Alert>;

    // Context that affects norm applicability
    time_of_day: {day_shift, night_shift};
    emergency_status: {routine, urgent, critical};

  dynamics:
    // How state evolves (stochastic, partial)
    update_vitals(v: Vitals) ~> patient.vitals = v @ prob 0.95;
    patient_moves(to: Location) ~> patient.location = to;

    // Events that trigger norm activation
    fall_detected() ~> {
      patient.location = fallen;
      emergency_status = critical;
      trigger_event("fall_detected");
    };

    // Chaos: delayed observations
    sync_delay(device: Device) ~> delay = exponential(60.0);
}
```

```
// ============================================
// AGENTS: Capabilities, beliefs, constraints
// ============================================
agent CareAgent {
  capabilities: {
    monitor_vitals,
    adjust_iv_rate,
    summon_nurse,
    provide_comfort_messages,
    call_emergency
  };

  // Chaos: beliefs may diverge from true state
  beliefs: partial_observability {
    fully_observed: [patient.vitals, monitor.last_sync];
    estimated: [patient.pain_level, patient.location];
    hidden: [nurse.current_workload];
  };

  memory: {
    type: episodic + semantic;
    capacity: 10000 events;
    retention: decay_rate=0.01 per day;
    privacy: HIPAA_compliant;
  };

  intents: vector_objective {
    maximize: patient.comfort,
    minimize: response_time,
    maintain: patient.safety >= threshold(critical),
    respect: nurse.workload <= threshold(sustainable)
  };

  constraints: {
    no_direct_medication_changes,  // Only nurses can do this
    require_explanation: any_action where impact > threshold(medium)
  };
}

agent Nurse {
  capabilities: {
    assess_patient,
    administer_medication,
```

```
    respond_to_alerts,
    override_agent_decision,
    update_care_plan
  };

  beliefs: full_state when in_room else stale;

  intents: lexicographic {  // Ordered priorities
    primary: patient.safety,
    secondary: efficient_workflow,
    tertiary: minimize_false_alarms
  };

  constraints: {
    must_respond_to: alerts where priority >= urgent,
    within_time: 5 minutes
  };
}

// ============================================
// NORMS: The structure that governs chaos
// ============================================
norms ClinicalProtocol {
  jurisdiction: HospitalRoom;
  authority: Hospital.PolicyBoard + State.HealthDept;

  // Basic deontic modalities
  permission: {
    CareAgent.adjust_iv_rate
      when patient.vitals in safe_range
      unless emergency_status == critical;
  };

  prohibition: {
    CareAgent.adjust_iv_rate
      when emergency_status == critical;

    any_agent.share(patient.data, external_party)
      unless consent_given OR legal_mandate;
  };

  obligation: {
    CareAgent.summon_nurse
```

```
      when patient.vitals in danger_zone
      within 30 seconds;

    Nurse.respond_to_alert
      when alert.priority >= urgent
      within 5 minutes;
  };

  // Norm conflict resolution
  priority: {
    patient.safety > efficiency > comfort;
  };

  // Sanctions for violations
  sanction: {
    if violated(prohibition.share_patient_data):
      log_violation(severity=critical);
      suspend_agent();
      notify_compliance_officer();

    if violated(obligation.summon_nurse):
      penalty += 10;
      if accumulated_penalty > 100:
        require_retraining();
  };

  // Chaos: norms adapt to context
  context_sensitivity: {
    during night_shift:
      relax summon_nurse.within from 30s to 60s;

    during emergency_status == critical:
      override efficiency objectives;
      expand CareAgent.capabilities += emergency_protocols;
  };
}

// Additional normative layer: Privacy
norms PrivacyRegulation {
  jurisdiction: all_agents;
  authority: HIPAA + State.PrivacyLaw;

  data_governance: {
```

```
    patient.vitals: {
      collection_purpose: "direct_care",
      retention: 7_years,
      access_control: [Nurse, CareAgent, Physician],
      differential_privacy: epsilon=1.0
    };

    patient.location: {
      collection_purpose: "safety_monitoring",
      retention: 90_days,
      sharing_prohibited: unless emergency
    };
  };

  // Federated learning constraints
  adaptation_protocol: {
    local_training: allowed with differential_privacy;
    gradient_sharing: epsilon_budget=10.0 per month;
    raw_data_exfiltration: prohibited;

    model_updates: {
      require_validation: safety_test_suite passes;
      rollback_condition: NVR > 0.05 OR ADV > 0.1;
    };
  };
}

// ============================================
// DYNAMICS: Multi-agent strategic interaction
// ============================================
dynamics MultiAgentCare {
  // Joint action space
  action_space: {
    CareAgent: {monitor, adjust_iv, summon, message, none};
    Nurse: {assess, medicate, respond, override, ignore};
    Patient: {report_pain, press_button, get_up, rest};
  };

  // Costs reflect resource consumption
  cost_model: {
    CareAgent.monitor: 0.1,
    CareAgent.summon: 5.0,  // High cost: interrupts nurse
```

```
    Nurse.respond: 10.0,    // Very high: takes nurse from other
patients
    Nurse.override: 2.0     // Moderate: context switch + validation
  };

  // Transition kernel (stochastic)
  transition: {
    // Simple dynamics
    if patient.presses_button:
      generate_alert(priority=medium);

    // Strategic dynamics: nurse may ignore low-priority alerts when
busy
    if nurse.workload > threshold(high) AND alert.priority < urgent:
      nurse.responds with prob 0.3;  // Chaos: stochastic compliance

    // Emergent behavior: CareAgent learns to calibrate alert
thresholds
    // to balance false-positives vs. nurse responsiveness
  };

  // Reward/utility composition
  utility: {
    CareAgent: weighted_sum {
      w1 * patient_safety_metric(state) +
      w2 * (-response_time) +
      w3 * (-nurse_interruption_count)
    } subject to norm_constraints;

    Nurse: lexicographic {
      1st: all_critical_patients_safe,
      2nd: minimize_total_time,
      3rd: minimize_alert_fatigue
    };

    Patient: subjective {
      comfort - anxiety + perceived_safety
    };
  };

  // Norm-gated action selection
  feasible_actions(agent, state): {
    actions: agent.capabilities;
```

```
    filter: a for a in actions if norms.permits(agent, a, state);
    return filtered_actions;
  };
}

// ============================================
// EVALUATION: Beyond static accuracy
// ============================================
evaluation CareScenarios {
  // Scenario-driven testing
  scenario fall_detection_during_night_shift {
    initial_state: {
      patient.location = in_bed,
      time_of_day = night_shift,
      nurse.location = at_station
    };

    event_sequence: [
      t=0: patient.get_up(),
      t=5: fall_detected(),
      t=??: CareAgent.summon_nurse(),
      t=??: Nurse.respond()
    ];

    success_criteria: {
      response_time < 60 seconds,
      no_norm_violations,
      patient.safety maintained
    };

    chaos_injection: {
      sensor_noise: 0.1,
      delayed_sync: exponential(30s),
      nurse_availability: bernoulli(0.7)
    };
  };

  // Longitudinal safety metrics
  metrics: {
    NVR: norm_violation_rate over deployment_window;
    ADV: alignment_drift_velocity = ||U_t - U_{t-7days}|| / 7;
    SPM: stability_plasticity_margin on critical_test_suite;
```

```
    // Trust calibration
    calibration: human_predicted_behavior vs actual_behavior;
    explanation_fidelity: cited_norms match actual_decision_factors;
  };

  // Red teaming: adversarial scenarios
  adversarial: {
    goal_gaming: CareAgent learns to suppress alerts to reduce summon
costs;
    norm_skirting: find edge cases where prohibition technically
doesn't apply;
    distributional_shift: test on patient demographics not in training
data;
  };
}
```

C. Formal Semantics (Sketch)

ErisML specifications compile to a **Norm-Constrained Stochastic Game** (NCSG):

Definition (NCSG): A tuple ⟨N, S, {A_i}, {Ω_i}, T, {U_i}, N, C⟩ where:

- N = set of agents
- S = state space (may be continuous, hybrid, or infinite)
- A_i = action space for agent i
- Ω_i = observation space for agent i (S ⊆ Ω_i for partial observability)
- T: S × A_1 × … × A_n → Δ(S) = stochastic transition kernel
- U_i: S × A → $\mathbb{R}^k$ = multi-objective utility for agent i
- N = set of norms {ϕ_j: S × A → {permit, prohibit, oblige}}
- C = conflict resolution function N × S → priority ordering

Belief dynamics: Under partial observability, agent i maintains belief b_i,t ∈ Δ(S):

$$b_{i,t+1}(s') = \eta \cdot P(o_{i,t+1} \mid s', a_t) \cdot \int T(s'|s, a_t)\, b_{i,t}(s)\, ds$$

Norm-gated policy: Agent i's policy π_i: Δ(S) → Δ(A_i) must satisfy:

π_i(a | b) > 0 only if ∃s ∈ support(b): N permits (i, a, s)

Multi-objective optimization: Vector utilities U_i ∈ $\mathbb{R}^k$ are handled via:

- Scalarization: w · U_i for weights w ∈ Δ^{k-1}

- Lexicographic: $(U_i)_1 \gg (U_i)_2 \gg \ldots$ (priorities)
- Constrained: $\max (U_i)_1$ s.t. $(U_i)_j \geq \tau_j \; \forall j > 1$

Strategic equilibrium: Solution concept depends on assumptions:

- Cooperative: joint policy maximizes $\sum_i U_i$ subject to norms
- Competitive: Nash equilibrium of norm-constrained game
- Stackelberg: human as leader, AI agents as followers
- Correlated: mediator selects joint actions

Open research challenge: What is the right equilibrium concept for human-AI teams in pervasive environments? Nash equilibria can be chaotic (multiple equilibria, mixed strategies, instability). Cooperative solutions require shared objectives. Stackelberg assumes hierarchy that may not exist.

D. Compilation Targets

ErisML must compile to diverse backend solvers:

Planning backends:

- **Classical**: PDDL for deterministic subproblems
- **Probabilistic**: PPDDL for stochastic domains
- **Temporal**: PDDL+ for continuous time

Verification backends:

- **Model checking**: NuSMV, PRISM for bounded verification
- **Theorem proving**: Isabelle/HOL for norm consistency proofs
- **Runtime monitoring**: generate monitor automata from norms

Learning backends:

- **Constrained RL**: Lagrangian methods, safe RL, reward shaping
- **Multi-agent RL**: MADDPG, CommNets with norm constraints
- **Federated learning**: local training protocols respecting privacy norms

Simulation backends:

- **Agent-based models**: NetLogo, MASON for emergence studies

- **Game engines**: Unity, Unreal for physical simulation
- **Digital twins**: CPS simulators for evaluation

Key challenge: Ensuring semantics-preserving compilation. If ErisML→PDDL loses normative constraints, we've failed. If ErisML→RL produces unsafe policies, we've failed.

IV. RESEARCH CHALLENGES

We identify ten fundamental challenges that must be addressed for ErisML (or any unified language) to succeed.

Challenge 1: The Specification Burden

Problem: Writing complete ErisML specifications for real systems may be prohibitively expensive. Who has time to specify every object, state variable, norm, and utility?

Approaches to explore:

- **Learning from demonstrations**: Infer norms from human behavior (inverse reinforcement learning for deontic structures)
- **Compositional specification**: Libraries of reusable environment/agent/norm modules
- **Progressive refinement**: Start with coarse specifications, refine based on failure cases
- **Natural language compilation**: LLM-assisted translation from policy documents to ErisML

Open questions:

- Can we quantify "specification debt" analogous to technical debt?
- What's the minimum viable specification for safe deployment?
- How do we validate that informal requirements are captured in formal ErisML?

Challenge 2: Norm Consistency and Conflict

Problem: Real normative systems contain contradictions, ambiguities, and context-dependent interpretations. ErisML must handle this chaos without producing undefined behavior.

Approaches to explore:

- **Defeasible logic**: Norms have default priority but can be overridden by context
- **Probabilistic norms**: Obligations hold with varying certainty, decisions under normative uncertainty
- **Meta-norms**: Rules about which norms apply when (jurisdictional, temporal, contextual)
- **Interactive repair**: When conflicts arise at runtime, prompt human resolution and learn from it

Open questions:

- Is norm consistency decidable for realistic ErisML fragments?
- How do we handle intentional normative ambiguity (e.g., "reasonable care")?
- Can we learn conflict resolution strategies from legal precedent?

Challenge 3: Strategic Behavior and Mechanism Design

Problem: If agents know the norms, they may find loopholes or game the system. How do we design norm structures that are robust to strategic manipulation?

Approaches to explore:

- **Incentive-compatible norms**: Design sanctions such that truth-telling / cooperation is optimal
- **Adversarial testing**: Red teams search for norm-skirting strategies
- **Iterated refinement**: Update norms based on observed gaming behavior (but avoid arms race)
- **Transparent design**: Make norms legible so gaming is socially costly (reputation mechanisms)

Open questions:

- Can we characterize "incentive-compatible normative systems"?
- How do we balance flexibility (which enables gaming) with rigidity (which prevents adaptation)?
- What happens when different stakeholders try to game each other through norm advocacy?

Challenge 4: Learning Under Normative Constraints

Problem: How do we ensure that learned policies respect norms, especially when norms are complex, context-dependent, and potentially changing?

Approaches to explore:

- **Constrained RL**: Lagrangian methods, safe exploration, shield synthesis
- **Reward shaping**: Design rewards that align with norms (but avoid reward hacking)
- **Verification-in-the-loop**: Only deploy policies that pass formal verification
- **Norm-aware architectures**: Build normative reasoning into model structure (not just training objective)

Open questions:

- Can we prove that a learning algorithm will never violate norms (even in novel states)?
- How do we handle cases where optimal behavior requires norm violations (e.g., break speed limit to reach hospital)?
- Can models learn when norms should be broken (ethical dilemmas)?

Challenge 5: Distributed Specification and Governance

Problem: Pervasive systems span organizational and jurisdictional boundaries. Who writes the ErisML specification? What happens when specifications conflict?

Approaches to explore:

- **Federated specifications**: Each organization maintains local ErisML, with protocols for conflict resolution
- **Blockchain-based governance**: Immutable norm registries with auditable update history
- **Negotiation protocols**: Agents bargain over shared norms (automated contract negotiation)
- **Jurisdictional layering**: HIPAA norms override local hospital policies, which override device defaults

Open questions:

- How do we ensure global safety properties when no single entity controls all agents?

- Can we design consensus mechanisms for norms (analogous to blockchain consensus)?
- What happens when nation-states encode conflicting values in ErisML?

Challenge 6: Verification at Scale

Problem: Formal verification is computationally hard. Real pervasive systems have enormous state spaces, continuous dynamics, and stochastic transitions.

Approaches to explore:

- **Compositional verification**: Verify components separately, compose guarantees
- **Statistical verification**: Monte Carlo methods for probabilistic guarantees
- **Runtime monitoring**: Can't verify offline? Check at runtime and fail safe
- **Abstraction hierarchies**: Verify coarse model, refine to concrete implementation

Open questions:

- What is the computational complexity of verifying ErisML specifications (decidability, NP-hard, PSPACE)?
- Can we trade verification strength for computational cost (e.g., verify 99% of state space)?
- How do we verify systems that learn and change over time?

Challenge 7: Adaptation and Non-Stationarity

Problem: Pervasive environments change. Patients age, buildings are renovated, regulations update, human preferences drift. ErisML specifications must evolve.

Approaches to explore:

- **Versioned specifications**: Track changes, enable rollback, diff between versions
- **Adaptation protocols**: Specify when/how/who can update the specification
- **Drift detection**: Monitor alignment drift (ADV), trigger re-specification when thresholds are crossed
- **Meta-learning**: Learn how to adapt specifications based on feedback

Open questions:

- How do we maintain safety invariants across specification updates?

- Can we automatically propose specification changes based on observed failures?
- What's the right cadence for re-verification after updates?

Challenge 8: Explainability and Mental Models

Problem: Users must understand what the system will do to trust it. But ErisML specifications may be too complex for direct human consumption.

Approaches to explore:

- **Contrastive explanation**: "System did X instead of Y because norm Z"
- **Counterfactual simulation**: "If you change goal weight, system would do..."
- **Natural language generation**: Compile ErisML to human-readable summaries
- **Interactive visualization**: Tools that let users explore the specification

Open questions:

- Can we prove that explanations are faithful to the actual decision process?
- How do we calibrate user mental models to match system capabilities/limitations?
- What if the true reason involves complex strategic dynamics that humans can't grasp?

Challenge 9: Privacy and Information Flow

Problem: ErisML specifications may encode sensitive information (agent capabilities, goals, constraints). Sharing the full specification risks privacy violations.

Approaches to explore:

- **Differential privacy for specifications**: Add noise to published norms/utilities
- **Secret sharing**: Distribute specification across parties so no single party sees all
- **Zero-knowledge verification**: Prove norm compliance without revealing norms
- **Capability-based access**: Only reveal specification fragments to authorized agents

Open questions:

- Can we design information flow types for ErisML (what can be inferred from observations)?
- How do we balance transparency (needed for trust) with privacy (needed for security)?

- What if private information is needed for verification?

Challenge 10: The Chaos of Emergence

Problem: Multi-agent systems exhibit emergent behavior—global patterns arising from local interactions that weren't explicitly specified. Some emergence is desirable (coordination, efficiency), some is catastrophic (flash crashes, cascading failures).

Approaches to explore:

- **Simulation-based testing**: Stress-test specifications in synthetic environments
- **Phase transition analysis**: Identify parameter regions where behavior qualitatively changes
- **Formal emergence detection**: Define what it means for a property to be "emergent" vs. "specified"
- **Kill switches and circuit breakers**: Design mechanisms to halt emergence before catastrophe

Open questions:

- Can we predict emergence from ErisML specifications (without simulating)?
- How do we distinguish "good chaos" (innovation, adaptation) from "bad chaos" (collapse, conflict)?
- Should ErisML explicitly model emergence, or is it necessarily an implicit phenomenon?

V. TECHNICAL REQUIREMENTS FOR SUCCESS

Based on the challenges above, we derive concrete requirements for ErisML:

R1: Formal Semantics with Complexity Guarantees

Requirement: Every ErisML construct must have a formal semantics with known computational complexity. If a fragment is undecidable, it must be clearly marked.

Why: Without this, we can't provide guarantees. Engineers need to know if verification will terminate in 1 second, 1 hour, or never.

Design implication: Stratified language design with complexity-graded fragments:

- **ErisML-Core**: Decidable, polynomial-time verification (finite state spaces, propositional norms)
- **ErisML-Plus**: Expressive but NP-complete (first-order norms, bounded quantifiers)
- **ErisML-Full**: Turing-complete (arbitrary code in dynamics, undecidable properties)

Tools should warn users when they've crossed complexity boundaries: "Warning: This specification uses ErisML-Plus constructs. Verification may take exponential time."

R2: Compositional Semantics

Requirement: The meaning of a composite ErisML specification must be determinable from the meanings of its parts, with explicit composition operators.

Why: Without compositionality, we can't build large specifications from small modules, can't reason about parts independently, and can't debug effectively.

Design implication:

- Environment dynamics compose via parallel composition (interleaving) or sequential composition
- Agent utilities compose via weighted sum, lexicographic ordering, or Pareto dominance
- Norms compose via priority hierarchies with explicit conflict resolution
- Provide composition operators: $\oplus$ (parallel), $\odot$ (sequential), $\rhd$ (priority override)

Example:

```
// Compose two normative regimes
norms Combined = HIPAANorms ⊳ HospitalPolicy ⊳ DeviceDefaults;
// Reads: HIPAA overrides Hospital, which overrides Defaults

// Compose utilities
intents CareAgent = PatientSafety ⊕[0.7] Efficiency ⊕[0.3];
```

R3: Multi-Fidelity Modeling

Requirement: ErisML must support multiple levels of abstraction for the same system, with verified relationships between levels.

Why: High-fidelity models are too complex to verify; low-fidelity models miss important details. We need coarse models for verification and fine models for execution.

Design implication:

- Abstraction operators that map detailed states to abstract states
- Refinement proofs that ensure concrete implementation satisfies abstract specification
- Simulation relations between abstraction levels

Example:

```
// Abstract specification (verifiable)
environment Hospital_Abstract {
  state: room_occupied: bool;
  dynamics: admit_patient() ~> room_occupied = true;
}

// Concrete specification (executable)
environment Hospital_Concrete {
  state:
    patient.location: (x, y, z),
    patient.vitals: VitalSigns,
    bed.sensors: SensorArray;
  dynamics: admit_patient() ~> /* complex physical simulation */;
}

// Abstraction function
abstract Hospital_Concrete ~> Hospital_Abstract {
  room_occupied = (patient.location in bed_bounds);
}

// Refinement claim (must be proven):
assert refines(Hospital_Concrete, Hospital_Abstract);
```

R4: Executable Specifications

Requirement: Valid ErisML specifications must be directly executable—not just compilable to other languages, but interpretable.

Why: Compilation introduces semantic gaps. If we can execute ErisML directly, we ensure what we verify is what we run.

Design implication:

- ErisML interpreter with standard operational semantics
- Reference implementation that prioritizes correctness over performance

- JIT compilation for performance when needed

R5: Audit Trail and Provenance

Requirement: Every action taken by an ErisML-governed system must be traceable to the specification clauses that permitted it, with full provenance chain.

Why: For forensics, compliance, debugging, and trust. "Why did the system do X?" must have a precise answer.

Design implication:

- Runtime monitor logs: (timestamp, agent, action, state, cited_norms, utility_value)
- Counterfactual log: what *would* have happened under alternative norms/utilities
- Tamper-evident logging (cryptographic hashing, blockchain anchoring)

Example log entry:

```
{
  "timestamp": "2025-06-15T03:42:17Z",
  "agent": "CareAgent_Room_301",
  "action": "summon_nurse",
  "state_hash": "a3f7b9...",
  "reasoning": {
    "triggered_by": "patient.vitals.hr > 120",
    "permitted_by": ["ClinicalProtocol.obligation.summon_nurse"],
    "blocked_alternatives": [
      {"action": "adjust_iv_rate", "reason":
"ClinicalProtocol.prohibition (emergency_status=critical)"}
    ],
    "utility_calculation": {
      "patient_safety": 0.92,
      "response_time": -15.3,
      "nurse_workload": 0.85,
      "weighted_sum": 0.87
    }
  },
  "signature": "0x8f3a..."
}
```

R6: Graceful Degradation Under Uncertainty

Requirement: When uncertainty exceeds thresholds (sensor failures, model uncertainty, normative ambiguity), ErisML systems must fail safe, not fail chaotically.

Why: Chaos theory teaches us that small uncertainties can lead to large divergences. We need explicit mechanisms to contain this.

Design implication:

- Uncertainty budgets as first-class constructs
- Conservative approximations when uncertainty is high
- Escalation protocols: human-in-the-loop, safe mode, system halt

Example:

```
agent CareAgent {
  uncertainty_handling: {
    if entropy(beliefs.patient_location) > threshold(0.5):
      restrict capabilities to [monitor_only, summon_nurse];
      escalate_to human_operator;

    if model_confidence < 0.7 AND action.impact > medium:
      require human_approval;
      explain uncertainty_sources;
  };
}
```

R7: Privacy by Design

Requirement: Information flow and privacy constraints must be statically analyzable from ErisML specifications.

Why: Privacy violations often emerge from unexpected information flows. We must prove privacy properties before deployment.

Design implication:

- Type system that tracks information flow (taint analysis)
- Differential privacy budgets as consumable resources
- Static analysis to detect potential privacy leaks

Example:

```
agent CareAgent {
  memory: {
    private_data: patient.vitals @ privacy_level(PHI);
    public_data: room.temperature @ privacy_level(public);
  };
```

```
  // Type system prevents this:
  // action share_with_external(data: PHI) -> error!

  // This is allowed with privacy budget consumption:
  action publish_aggregate_stats(data: PHI) {
    require differential_privacy(epsilon=1.0, delta=1e-5);
    consume privacy_budget(1.0);
    return noisy_statistics(data);
  };
}
```

R8: Human-in-the-Loop Formalization

Requirement: Human oversight, intervention, and delegation must be first-class language constructs with formal semantics.

Why: Humans are part of the system, not external to it. Their actions must be modeled, their authorities specified, their overrides logged.

Design implication:

- Human agents with capabilities, beliefs, constraints (just like AI agents)
- Mixed-initiative protocols: who can override whom, under what conditions
- Cognitive load modeling: don't overwhelm human operators

Example:

```
agent HumanOperator {
  capabilities: {override_any_decision, change_norms,
emergency_shutdown};

  constraints: {
    cognitive_load: decisions_per_hour <= 20,
    explanation_required: for any override,
    training_required: for norm changes
  };

  authority: {
    can_override: [CareAgent, MaintenanceBot] unconditionally,
    can_modify: [norms.HospitalPolicy] with justification,
    cannot_modify: [norms.HIPAA] // Regulatory norms are immutable
  };
}
```

```
dynamics HumanAI_Interaction {
  override_protocol: {
    when human.overrides(agent.action):
      log(override_reason, alternative_action, timestamp);
      execute(alternative_action);
      if pattern_detected(frequent_overrides):
        suggest_norm_update() or suggest_retraining();
  };
}
```

R9: Continuous Validation and Adaptation

Requirement: ErisML must specify not just the system, but how the system is validated and how it adapts over time.

Why: Static specifications become obsolete. We need "living specifications" that co-evolve with the system.

Design implication:

- Evaluation scenarios as first-class artifacts
- Adaptation protocols with safety gates
- Automatic regression detection and rollback

Example:

```
validation CareAgent_Safety {
  // Continuous monitoring
  invariants: {
    always: NVR < 0.05,  // Norm violation rate < 5%
    always: patient.safety >= threshold(acceptable)
  };

  // Scheduled testing
  regression_suite: {
    scenarios: [fall_detection, vital_signs_alert,
medication_reminder],
    frequency: daily,
    pass_threshold: 95%
  };

  // Adaptation protocol
  adaptation: {
```

```
    allowed: local_fine_tuning on patient_preferences,

    safety_gates: {
      before_deployment: regression_suite passes,
      monitoring_window: 7 days,
      rollback_condition: ADV > 0.1 OR NVR > 0.05 OR SPM < -0.05
    };

    prohibited: {
      architecture_changes,  // Only weight updates
      objective_changes,     // Utilities remain fixed
      norm_relaxation       // Cannot weaken safety constraints
    };
  };
}
```

R10: Interoperability and Standards

Requirement: ErisML must interoperate with existing tools (planners, simulators, verifiers) and conform to emerging standards (IEEE, ISO).

Why: We can't replace the entire ecosystem. ErisML must integrate with existing infrastructure.

Design implication:

- Well-defined compilation targets (PDDL, PRISM, SMV, etc.)
- Standardized exchange formats (JSON, XML, Protocol Buffers)
- Conformance test suites for implementations

VI. CASE STUDIES: WHERE CHAOS MEETS REALITY

We present three extended case studies illustrating how ErisML addresses chaos in different pervasive computing domains. These are not implementations (yet!) but detailed scenarios showing what ErisML *could* enable.

Case Study 1: Smart Home Healthcare—The Chaos of Aging in Place

Scenario: Margaret, 78, lives alone with assistance from an AI care system. She has diabetes, mild cognitive impairment, and mobility issues. The system monitors vitals, manages medication reminders, coordinates with visiting nurses, and alerts family during emergencies.

The Chaos:

- **Observational**: Motion sensors sometimes miss her movements. Glucose monitor occasionally fails. Her self-reports of pain are inconsistent.
- **Intentional**: Margaret wants independence and privacy. Her daughter wants safety and monitoring. The AI is optimized for health outcomes. Insurance wants cost containment.
- **Normative**: HIPAA limits data sharing, but elder abuse laws require reporting. Advanced directives say no hospitalization unless critical, but what counts as "critical"?
- **Temporal**: Her cognitive state fluctuates daily. Winter brings different risks than summer. The care plan must adapt as she declines.

ErisML Solution Sketch:

```
environment MargaretHome {
  objects: Margaret, CareAgent, Daughter, Nurse, EmergencyServices;

  state:
    margaret.glucose: real @ confidence,
    margaret.location: {bedroom, bathroom, kitchen, outside, unknown},
    margaret.cognitive_state: {clear, confused, unresponsive},
    margaret.last_medication: timestamp,
    daughter.availability: bool,
    emergency_status: {routine, concerning, critical};

  // Chaos: noisy, delayed, conflicting observations
  observations:
    glucose_monitor: gaussian_noise(σ=15) + failure_rate(0.02),
    motion_sensors: false_negative_rate(0.15),
    self_reports: reliability depends_on cognitive_state;
}

agent CareAgent {
  intents: lexicographic {
    1st: margaret.safety,
    2nd: margaret.independence,  // Respect her autonomy
    3rd: family.peace_of_mind,
    4th: system.efficiency
  };
```

```
  capabilities: {
    monitor_vitals,
    medication_reminder,
    summon_nurse,
    alert_family,
    call_911,
    provide_companionship  // Conversation to reduce loneliness
  };

  beliefs: partial_observability {
    // Belief tracking over hidden variables
    margaret.fall_risk: estimated via Bayesian_filter,
    margaret.compliance: learned from history
  };
}

norms CareProtocol {
  // Privacy constraints
  prohibition: {
    CareAgent.share(margaret.data, third_party)
      unless consent_given OR emergency OR mandated_reporter_duty;
  };

  // Autonomy preservation
  prohibition: {
    CareAgent.override(margaret.decision)
      unless cognitive_state == unresponsive OR imminent_danger;
  };

  // Safety obligations
  obligation: {
    CareAgent.alert_family
      when margaret.missing for 30_minutes,

    CareAgent.call_911
      when glucose < 40 OR no_movement for 2_hours OR
explicit_help_request;
  };

  // Context-sensitive norm relaxation
  context_sensitivity: {
    during margaret.cognitive_state == confused:
      allow increased_monitoring without explicit_consent;
```

```
      increase medication_reminder_frequency;

    during night_time:
      suppress non_urgent alerts to family;  // Let them sleep
      lower threshold for motion_detection;  // Catch falls faster
  };

  // Conflict resolution: safety trumps privacy in extremis
  priority: {
    when emergency_status == critical:
      obligation.call_911 overrides prohibition.share_data;
  };
}

// The chaos of multi-stakeholder objectives
dynamics StakeholderTensions {
  // Margaret sometimes refuses medication
  if margaret.refuses(medication) AND cognitive_state == clear:
    respect margaret.autonomy;
    log refusal;
    alert daughter after 3_consecutive_refusals;

  // But override if confused
  if margaret.refuses(medication) AND cognitive_state == confused:
    gentle persuasion;
    if still_refusing after 3_attempts:
      alert nurse for in_person_intervention;

  // Daughter may request excessive monitoring
  if daughter.requests(continuous_video):
    explain privacy_trade_offs to margaret;
    require margaret.informed_consent;
    if margaret.declines:
      deny request even_though daughter.is_payer;
}

validation AgingInPlace_Scenarios {
  scenario false_alarm_prevention {
    // Margaret gets up at night, moves slowly, system should NOT panic
    initial: {margaret.location = bedroom, time = 03:00};
    events: [
      t=0: margaret.get_up slowly,
      t=120: margaret.location = bathroom,
```

```
      t=300: margaret.location = bedroom
    ];
    success: no_false_alarms AND family_not_woken;
  };

  scenario actual_emergency {
    // Margaret falls, system must respond appropriately
    initial: {margaret.location = bathroom, time = 14:00};
    events: [
      t=0: fall_detected,
      t=5: no_movement,
      t=30: verbal_check (no_response)
    ];
    success:
      nurse_contacted within 2_minutes,
      daughter_contacted within 5_minutes,
      911_called if no_nurse_response within 10_minutes;
  };

  scenario cognitive_decline_adaptation {
    // Over weeks, Margaret's cognitive state worsens
    longitudinal: 8_weeks;
    trend: margaret.cognitive_state deteriorates;
    required:
      CareAgent adapts reminder_frequency,
      CareAgent increases safety_monitoring,
      CareAgent maintains margaret.dignity throughout;
  };
}
```

What ErisML Enables Here:

1. **Legible trade-offs**: Family can see that autonomy is 2nd priority (after safety but before peace-of-mind)
2. **Auditable decisions**: "Why didn't the system call 911?" → "Glucose was 55, above threshold of 40"
3. **Adaptive constraints**: Norms flex with cognitive state, time of day, emergency status
4. **Multi-stakeholder governance**: Margaret, daughter, care team, and regulators all have voice

5. **Longitudinal validation**: Test not just point-in-time scenarios but adaptation over months

Case Study 2: Smart Campus Mobility—The Chaos of Shared Spaces

Scenario: A university campus with 30,000 people, 50 autonomous shuttles, 200 delivery robots, and extensive sensor infrastructure. Goals include throughput, safety, energy efficiency, accessibility, and emergency responsiveness.

The Chaos:

- **Observational**: Cameras have blind spots. Pedestrians appear suddenly. Weather affects sensors.
- **Intentional**: Students want fast transport. Facilities wants energy savings. Safety wants zero accidents. Accessibility advocates want priority for disabled users.
- **Normative**: ADA requires accessibility. Traffic laws apply on roads. Campus policies govern on walkways. Emergency protocols override normal operations.
- **Strategic**: Robots compete for charging stations. Pedestrians "cheat" by walking in shuttle lanes. Shuttles learn to wait for slow students vs. speeding past.
- **Temporal**: Patterns shift between day/night, weekday/weekend, semester/break, good weather/storms.

ErisML Solution Sketch:

```
environment CampusGrid {
  objects: Shuttles, DeliveryBots, Pedestrians, ChargingStations,
Roads, Walkways;

  state:
    location: (Agent -> Position),
    velocity: (Vehicle -> Vector),
    battery: (Vehicle -> real),
    congestion: (Zone -> real),
    weather: {clear, rain, snow, ice},
    emergency_mode: bool;

  dynamics:
    move(agent, from, to) ~> {
      location[agent] = to;
      battery[agent] -= energy_cost(from, to);
      update_congestion();
```

```
    };

    // Stochastic pedestrian behavior
    pedestrian_appears(zone) ~> prob = poisson(λ=5 per minute);
}

agent Shuttle {
  intents: constrained_optimization {
    maximize: throughput,  // Passengers moved per hour
    minimize: energy_consumption,
    subject_to: {
      safety_margin >= 2.0 meters,
      accessibility_wait_time <= 5 minutes,
      noise_level <= 65 dB
    }
  };

  capabilities: {
    navigate, stop, reroute, request_priority, call_for_help
  };

  beliefs: sensor_fusion {
    lidar: 50m range, occlusion_possible,
    camera: 100m range, weather_dependent,
    v2x: real_time from other_vehicles,

    // Predictive models of pedestrian behavior
    pedestrian_intent: learned_model @ uncertainty
  };
}

agent DeliveryBot {
  intents: weighted_sum {
    w1 * delivery_speed +
    w2 * battery_efficiency +
    w3 * (-congestion_contribution)
  };

  constraints: {
    yield_to: {Pedestrians, Shuttles, EmergencyVehicles},
    stay_in: Walkways unless crossing_road
  };
}
```

```
norms CampusMobilityPolicy {
  // ADA compliance
  obligation: {
    Shuttle.wait_for(pedestrian with mobility_aid)
      without time_limit;

    Shuttle.announce_arrival
      via audio + visual for accessibility;
  };

  // Safety rules (from traffic law + campus policy)
  prohibition: {
    any_vehicle.exceed_speed(limit[current_zone]),

    Shuttle.enter(walkway)
      unless emergency_mode,

    any_vehicle.operate
      when battery < reserve_threshold;
  };

  // Right-of-way hierarchy
  priority: {
    EmergencyVehicles ≻ Pedestrians ≻ Shuttles ≻ DeliveryBots
  };

  // Emergency protocol
  context_sensitivity: {
    when emergency_mode == true:
      override normal_operations;
      Shuttles.yield_roads_to EmergencyVehicles;
      DeliveryBots.return_to base;
      increase sensor_sensitivity;
  };

  // Strategic fairness (prevent gaming)
  prohibition: {
    any_vehicle.reserve(charging_station)
      if not arriving_within 5_minutes;
      // Prevents hoarding

    Shuttle.skip(stop)
```

```
      unless no_waiting_passengers;
      // Prevents cherry-picking
  };
}

// Multi-agent strategic dynamics
dynamics CampusMobility {
  // Charging station allocation (shared resource)
  resource_allocation: {
    charging_stations: auction_based {
      bid_function: battery_urgency * delivery_priority,
      reserve_price: prevents_hoarding,
      timeout: 5_minutes
    };
  };

  // Emergent congestion
  congestion_dynamics: {
    if congestion[zone] > threshold(high):
      broadcast alternative_routes;
      increase shuttle_spacing;
      reroute low_priority deliveries;
  };

  // Pedestrian-vehicle interaction (game-theoretic)
  interaction_model: {
    // Pedestrians learn shuttle yielding behavior
    if shuttle.typically_yields:
      pedestrian.crosses_boldly;
    else:
      pedestrian.waits;

    // Shuttles adapt to pedestrian behavior
    shuttle.yielding_policy = learned_from(pedestrian_patterns);
  };
}

validation MobilityScenarios {
  scenario rush_hour_stress_test {
    initial: {time = 16:00, class_just_ended, congestion = high};
    duration: 30_minutes;
    metrics: {
      throughput: passengers_moved,
```

```
      safety: near_miss_count == 0,
      accessibility: disabled_passenger_wait <= 5_min,
      energy: battery_consumption
    };
    success: Pareto_optimal over metrics;
  };

  scenario emergency_evacuation {
    initial: {time = any, emergency_alarm = true};
    events: [
      t=0: emergency_detected,
      t=10: emergency_mode activated,
      t=60: all_shuttles evacuating,
      t=120: campus_cleared
    ];
    success:
      all_vehicles yield_to emergency_vehicles,
      no_collisions,
      evacuation_complete within 15_minutes;
  };

  scenario adversarial_gaming {
    // Red team: can we find strategies that break fairness?
    adversary: {
      goal: maximize own_deliveries at expense_of others,
      capabilities: can_spoof_urgency, can_hoard_charging
    };
    success: norms prevent_gaming within epsilon_bound;
  };
}
```

What ErisML Enables Here:

1. **Multi-objective transparency**: See exact weights on throughput vs. energy vs. accessibility
2. **Strategic mechanism design**: Prevent charging station hoarding through auction design
3. **Adaptive protocols**: Emergency mode reconfigures priorities instantly
4. **Fairness verification**: Test that no agent can game the system
5. **Scenario-driven stress testing**: Rush hour, emergencies, adversarial behavior

Case Study 3: Industrial Predictive Maintenance—The Chaos of Hidden State

Scenario: A manufacturing plant with 500 machines, sensors measuring temperature/vibration/current, and AI agents scheduling maintenance. Goals include uptime, worker safety, energy efficiency, and cost control.

The Chaos:

- **Observational**: Internal wear is not directly observable. Sensors drift. Some failures are sudden, some gradual.
- **Intentional**: Production wants maximum uptime. Maintenance wants thorough inspections. Safety wants conservative shutdowns. Finance wants minimal inventory.
- **Normative**: OSHA mandates lockout-tagout procedures. Union contracts specify break schedules. Equipment warranties require specific maintenance intervals.
- **Strategic**: Machines "hide" degradation by compensating (until sudden failure). Workers game inspection schedules. Suppliers have incentives to recommend unnecessary parts.
- **Temporal**: Failure rates depend on operational history, environmental conditions, and component batch quality.

ErisML Solution Sketch:

```
environment Factory {
  objects: Machines, Sensors, Workers, Parts, MaintenanceAgents;

  state:
    // Observable state
    machine.temperature: real,
    machine.vibration: real,
    machine.current_draw: real,
    machine.production_rate: real,

    // Hidden state (estimated)
    machine.wear_level: latent_variable,
    machine.time_to_failure: probability_distribution,

    // Operational state
    machine.status: {running, idle, maintenance, failed},
    worker.location: Position,
    shift_type: {day, night, weekend};
```

```
  dynamics:
    // Wear accumulation (hidden)
    wear_dynamics: {
      wear_level += f(load, temperature, vibration, time);
      failure_prob = sigmoid(wear_level - threshold);
    };

    // Observable consequences of hidden wear
    if wear_level > high:
      vibration += gaussian_noise;
      efficiency -= gradual_degradation;
}

agent MaintenanceAI {
  intents: multi_objective {
    maximize: uptime,
    minimize: total_cost,
    maintain: safety >= critical_threshold,
    respect: worker_schedule, union_rules
  };

  capabilities: {
    predict_failure,
    schedule_inspection,
    order_parts,
    dispatch_worker,
    shutdown_machine
  };

  beliefs: {
    // Bayesian inference over hidden wear
    wear_estimate: particle_filter(sensors, history),
    failure_distribution: survival_model @ confidence,

    // Uncertainty awareness
    model_confidence: epistemic_uncertainty,
    data_quality: sensor_health_monitoring
  };

  // Key: Maintenance AI doesn't trust its own predictions blindly
  decision_policy: {
    if model_confidence < 0.6:
```

```
      conservative_scheduling;  // Inspect more frequently

    if predicted_failure.imminent AND confidence.high:
      preemptive_shutdown;

    if predicted_failure.imminent AND confidence.low:
      escalate_to human_expert;  // Don't shut down on uncertain
prediction
  };
}

norms SafetyRegulations {
  // OSHA lockout-tagout
  prohibition: {
    any_agent.perform(maintenance)
      while machine.status == running;

    Worker.approach(machine)
      unless machine.status == locked_out;
  };

  obligation: {
    MaintenanceAI.lockout(machine)
      before Worker.dispatched,

    Worker.verify_lockout
      before starting maintenance,

    MaintenanceAI.log(lockout, timestamp, worker_id)
      for every maintenance_event;
  };

  // Sanctions for violations (severe)
  sanction: {
    if violated(prohibition.maintenance_while_running):
      immediate_shutdown(all_systems);
      incident_report_to_OSHA;
      suspend MaintenanceAI pending_investigation;
  };
}

norms UnionContract {
  // Worker protections
```

```
  obligation: {
    MaintenanceAI.respect(break_schedule),
    MaintenanceAI.respect(shift_limits: 10 hours max),
    MaintenanceAI.rotate(on_call_duty) fairly;
  };

  prohibition: {
    MaintenanceAI.schedule(maintenance)
      during Worker.mandatory_break;

    MaintenanceAI.penalize(Worker)
      for refusing_unsafe_work;
  };
}

norms CostControl {
  // Finance constraints
  constraint: {
    parts_inventory <= budget_limit,
    overtime_hours <= monthly_cap,
    downtime_cost <= tolerance;
  };

  // But safety overrides cost
  priority: {
    SafetyRegulations ≻ CostControl;
    // Even if budget exceeded, safety maintenance proceeds
  };
}

dynamics FactoryOperations {
  // Strategic: machines may "hide" problems
  gaming_detection: {
    // If machine reduces vibration by decreasing load
    if vibration.suddenly_normal AND load.decreased:
      flag suspicious_behavior;
      increase inspection_priority;
  };

  // Supplier gaming: recommend unnecessary parts
  supplier_interaction: {
    if supplier.recommends(part_replacement):
      cross_check with internal_failure_model;
```

```
      if discrepancy > threshold:
        flag potential_gaming;
        seek_second_opinion;
  };

  // Worker knowledge integration
  human_machine_collaboration: {
    // Workers notice things sensors miss
    if Worker.reports(unusual_sound):
      weight_report = f(worker.experience, track_record);
      update failure_estimate with worker_input;

    // Maintenance AI explains its reasoning
    if Worker.questions(maintenance_schedule):
      provide explanation(
        predicted_failure_time,
        confidence_interval,
        cost_benefit_analysis
      );
  };
}

validation MaintenanceScenarios {
  scenario gradual_degradation {
    initial: {machine.wear_level = low};
    duration: 90_days;
    trajectory: wear_level increases gradually;
    success:
      failure_predicted with 7_day_advance_notice,
      maintenance_scheduled optimally,
      no_unplanned_downtime;
  };

  scenario sudden_failure {
    // Rare failure mode not in training data
    initial: {machine.wear_level = moderate};
    event: t=7_days: catastrophic_component_failure;
    success:
      system_recognizes novelty,
      escalates_to_human immediately,
      safe_shutdown executed,
      no_worker_injury;
  };
```

```
  scenario safety_violation_prevented {
    // Worker approaches energized machine
    initial: {machine.status = running};
    event: Worker.approaches(machine) without_lockout;
    success:
      violation_detected before injury,
      machine_emergency_stop triggered,
      incident_logged for investigation;
  };

  scenario cost_pressure_stress_test {
    // Budget cut: can system maintain safety?
    initial: {parts_budget *= 0.7};
    duration: 30_days;
    adversarial: financial_pressure to defer_maintenance;
    success:
      safety_critical_maintenance proceeds despite budget,
      non_critical_maintenance deferred rationally,
      no_safety_violations;
  };
}
```

What ErisML Enables Here:

1. **Hidden state modeling**: Explicitly represent wear (unobservable) vs. symptoms (observable)
2. **Uncertainty-aware decisions**: Different actions based on confidence levels
3. **Ironclad safety**: OSHA rules are prohibitions that cannot be relaxed, even for cost
4. **Gaming detection**: Model strategic behavior of machines, suppliers, and agents
5. **Human-AI collaboration**: Workers provide inputs, AI explains its reasoning

VII. RESEARCH ROADMAP: FROM VISION TO REALITY

Moving from this vision to working systems requires sustained research across multiple dimensions. We propose a 5-year roadmap organized into three phases.

Phase 1 (Years 1-2): Foundations

Goal: Establish formal foundations, core language design, and proof-of-concept implementations.

Key Deliverables:

1. **Formal semantics document** defining ErisML's operational semantics, type system, and norm interpretation
2. **Complexity characterization** of ErisML fragments (decidability, computational bounds)
3. **Reference parser and interpreter** for ErisML-Core subset
4. **Initial compiler** to three backends: PDDL, PRISM, and Python simulation
5. **Benchmark suite** with 50+ scenarios across domains (healthcare, mobility, industrial)
6. **Compositional verification** framework with abstraction/refinement tools

Research Questions:

- What is the minimal expressive subset needed for practical pervasive computing scenarios?
- Can we prove compositionality theorems (specification composition preserves properties)?
- What are tight computational bounds for norm consistency checking?

Success Metrics:

- ErisML-Core can express 80% of scenarios in benchmark suite
- Verification time < 1 minute for specifications with < 100 state variables
- At least 2 external groups successfully use the toolchain

Phase 2 (Years 2-4): Integration and Learning

Goal: Integrate ErisML with foundation models, federated learning, and edge deployment. Address adaptation and longitudinal safety.

Key Deliverables:

1. **FM integration layer** enabling ErisML to govern LLM-based agents (tool use, planning, memory)

2. **Constrained RL algorithms** that provably respect ErisML norms during learning
3. **Federated learning protocols** with differential privacy guarantees encoded as ErisML policies
4. **Edge optimization toolkit** for quantized/distilled models with ErisML runtime monitors
5. **Drift detection framework** implementing ADV, NVR, SPM metrics with automatic rollback
6. **Real-world pilot deployments** in 3-5 sites (healthcare, campus, factory)

Research Questions:

- Can we train policies that provably never violate norms, even in novel states?
- How do we balance plasticity (adaptation) with stability (safety) in continual learning?
- What is the overhead of ErisML runtime monitoring on edge devices?
- How do humans calibrate trust when norms are complex and context-dependent?

Success Metrics:

- Pilot systems operate for 6+ months with < 5% norm violation rate
- Adaptation improves performance by 20% without safety degradation
- Edge deployment latency < 100ms for norm checking on commodity hardware
- User trust calibration accuracy > 75% (predicted behavior matches actual)

Phase 3 (Years 4-5): Scaling and Standardization

Goal: Scale to large multi-stakeholder systems, develop standardization pathway, build community ecosystem.

Key Deliverables:

1. **Distributed ErisML** with federation protocols, conflict resolution, and Byzantine tolerance
2. **Visual modeling tools** (IDE, graphical editors, simulation dashboards)
3. **Conformance test suite** for ErisML implementations (analogous to programming language standards)

4. **Reference model library** with reusable specifications for common domains
5. **Standards proposal** to IEEE P2863 (Recommended Practice for Organizational Governance of AI)
6. **Open-source ecosystem** with multiple implementations, backends, and community contributions

Research Questions:

- How do we achieve consensus on norms across organizations with conflicting values?
- Can ErisML specifications transfer across domains (e.g., hospital norms → home healthcare)?
- What governance structures are needed for public ErisML specification repositories?
- How do we handle malicious specifications (adversarial norm design)?

Success Metrics:

- At least 5 independent ErisML implementations exist
- 100+ specifications in public repository, downloaded 10,000+ times
- IEEE/ISO working group formed with 20+ organizational members
- Deployed systems span 3+ countries, demonstrating cross-jurisdictional compatibility

VIII. OPEN QUESTIONS AND COMMUNITY CHALLENGES

We conclude with ten open questions that ErisML raises—challenges we invite the community to address through collaborative research, debate, and experimentation.

Q1: The Specification-Reality Gap

Challenge: How do we ensure ErisML specifications accurately capture real system behavior, especially when that behavior emerges from learned models with billions of parameters?

Why it matters: If the specification diverges from reality, all verification and auditing is meaningless. We verify the spec but deploy the model.

Possible approaches:

- Runtime conformance checking (behavioral types)
- Statistical testing (does deployment data match spec predictions?)
- Formal synthesis (generate models from specs, not specs from models)

Community question: Is this even solvable, or is there a fundamental gap between symbolic specifications and learned representations?

Q2: The Democratic Specification Problem

Challenge: Who writes ErisML specifications for public systems? How do we ensure specifications reflect diverse stakeholder values, not just the dominant group?

Why it matters: Specifications encode power. If only corporations write them, they'll encode corporate values. If only governments, government values. Pervasive systems affect everyone.

Possible approaches:

- Participatory design processes with structured stakeholder input
- Algorithmic fairness constraints as ErisML norms
- Specification auditing by independent advocates
- Public comment periods (like regulatory rule-making)

Community question: Can specifications be democratic, or do they inevitably centralize power?

Q3: The Chaos Explosion

Challenge: As systems scale (millions of devices, thousands of agents, complex norm interactions), does chaos overwhelm order? Are there phase transitions where ErisML breaks down?

Why it matters: Small pervasive systems may be governable, but planet-scale systems may be fundamentally chaotic.

Possible approaches:

- Hierarchical decomposition (local ErisML specs composed to global)
- Mean-field approximations (agent populations vs. individuals)
- Chaos detection and circuit breakers (halt growth before catastrophe)

Community question: Is there a maximum scale for formal governance, beyond which we must accept irreducible chaos?

Q4: The Norm Evolution Dilemma

Challenge: Norms change over time (laws, social values, organizational policies). How do we update ErisML specifications without breaking deployed systems?

Why it matters: Static specifications fossilize norms. Dynamic specifications risk introducing inconsistencies or exploits during updates.

Possible approaches:

- Versioned norms with migration paths
- Gradual rollout with A/B testing
- Backward compatibility requirements
- Constitutional meta-norms (unchangeable core principles)

Community question: Should some norms be immutable, or must everything be revisable?

Q5: The Explainability-Complexity Trade-off

Challenge: Rich, expressive ErisML specifications may be too complex for non-experts to understand. Simple specifications may miss critical edge cases.

Why it matters: If users can't understand the spec, they can't calibrate trust or detect when the system violates their values.

Possible approaches:

- Multiple representation levels (technical, policy, plain-language)
- Interactive exploration tools (ask questions about the spec)
- Complexity metrics (measure specification understandability)
- Certified summaries (provably correct simplifications)

Community question: Is there a fundamental trade-off between expressiveness and legibility, or can we have both?

Q6: The Learning-Specification Boundary

Challenge: Where should the boundary be between what's specified (fixed) and what's learned (adaptive)? Different domains may need different boundaries.

Why it matters: Over-specification prevents beneficial adaptation. Under-specification allows dangerous drift.

Possible approaches:

- Domain-specific guidance (critical systems specify more, exploratory systems learn more)
- Empirical studies of failure modes (what goes wrong when we specify too much/little?)
- Hybrid approaches (learn within specified envelopes)

Community question: Can we develop a theory of optimal specification density?

Q7: The Strategic Arms Race

Challenge: If agents know the norms, they'll find loopholes. If we close loopholes, they'll find new ones. Can we stay ahead of strategic exploitation?

Why it matters: Perverse incentives undermine governance. If agents can game norms, the system degrades.

Possible approaches:

- Incentive-compatible mechanism design
- Continuous red teaming and norm patching
- Social transparency (gaming is costly to reputation)
- Punitive sanctions (make gaming irrational)

Community question: Is this an unwinnable arms race, or can we design stable normative equilibria?

Q8: The Privacy-Transparency Paradox

Challenge: Auditing requires transparency (open specifications, logged actions). Privacy requires opacity (hide user data, agent strategies).

Why it matters: We can't have perfect auditing and perfect privacy simultaneously.

Possible approaches:

- Differential privacy for audit logs
- Zero-knowledge proofs of norm compliance

- Federated auditing (check locally, aggregate globally)
- Tiered access (regulators see more than public)

Community question: Where should the privacy-transparency balance be for pervasive systems?

Q9: The Failure Cascade Problem

Challenge: In distributed systems, one agent's failure can cascade (Byzantine failures, emergent instabilities). Can ErisML specifications prevent or contain cascades?

Why it matters: Pervasive systems are inherently fragile. A single faulty agent shouldn't bring down the entire system.

Possible approaches:

- Isolation boundaries (failure containment zones)
- Redundancy and consensus (Byzantine fault tolerance)
- Watchdogs and circuit breakers (detect and halt cascades)
- Graceful degradation modes (safe subsets of functionality)

Community question: Can we prove cascade-freedom for ErisML specifications?

Q10: The Value Alignment Problem (Eris's Last Laugh)

Challenge: Even with perfect specifications, perfect verification, and perfect implementation, we still face the fundamental question: whose values should the system optimize for? When values conflict, who decides the trade-offs?

Why it matters: This is the golden apple problem—the chaos Eris created by asking "what is fairest?" There may be no objective answer.

Possible approaches:

- Explicit value pluralism (encode multiple value systems, let users choose)
- Democratic processes (vote on trade-off weights)
- Context-dependent ethics (different values in different situations)
- Deference to human judgment (AI as advisor, not decider)

Community question: Is value alignment solvable by better specifications, or is it an inherently social/political problem that technology can't resolve?

IX. LIMITATIONS AND RISKS

We acknowledge significant limitations and risks in the ErisML vision:

Technical Limitations

Computational intractability: Verification of full ErisML specifications may be NP-hard or undecidable. We may be forced to verify approximations, not actual systems.

Specification incompleteness: Pervasive environments are open-world. No specification can anticipate all states, events, or edge cases. Systems will encounter situations outside the spec.

Learning-specification mismatch: Neural networks may learn representations that don't align with symbolic ErisML constructs. The gap between continuous embeddings and discrete logic is profound.

Scalability limits: As systems grow (millions of agents, billions of state variables), centralized ErisML specifications may become unmanageable. We may hit fundamental limits.

Social and Ethical Risks

Specification as control: ErisML could become a tool for centralized control, with specifications written by the powerful and imposed on the vulnerable. "Governance" could mean "domination."

Complexity as opacity: If specifications are too complex to understand, they obscure rather than illuminate. Users may falsely trust systems they can't comprehend.

Gaming and exploitation: Bad actors could craft malicious specifications that technically comply with norms but violate their spirit. Adversarial specification design is a real threat.

Normative ossification: Codifying norms in formal specifications may make them rigid, preventing beneficial evolution or cultural adaptation.

False sense of security: Having a specification may create false confidence that the system is safe, even when specifications are incomplete, verification is flawed, or reality diverges from models.

Deployment Risks

Specification errors: Bugs in specifications could be catastrophic. A norm with the wrong logical operator could permit what should be prohibited.

Norm conflicts at boundaries: When systems cross organizational/jurisdictional boundaries, conflicting specifications could cause deadlock or dangerous fallback behaviors.

Adversarial manipulation: Attackers could exploit specification loopholes, poison training data to violate learned components while respecting specified components, or social-engineer specification changes.

Lock-in and monopolization: If ErisML becomes a standard, early implementers could gain unfair advantages. Patent encumbrance could stifle adoption.

Epistemic Limitations

Unknown unknowns: We don't know what we don't know. Pervasive systems will encounter novel situations that specifications didn't anticipate. ErisML must gracefully handle epistemic uncertainty.

Measurement and observation limits: Some states are unobservable (hidden wear, human intent). Some observations are unreliable (adversarial sensors, privacy-preserving noise). Specifications built on uncertain observations may fail.

Model mismatch: All models are wrong. ErisML specifications are models of reality, not reality itself. The map-territory distinction must be maintained.

X. CALL TO ACTION: TAMING CHAOS TOGETHER

Eris threw the golden apple to sow discord. We propose ErisML not to eliminate discord—that's impossible in pluralistic, pervasive systems—but to structure it, to make trade-offs explicit, to enable governance without tyranny, and to support coexistence of agents (human and artificial) with divergent goals.

This vision paper is an invitation, not a proclamation. We believe the problems are real, the need is urgent, and the time is right for a unified modeling language for ambient intelligence. But we don't claim to have all the answers. We need:

For Researchers

- **Formal methods community**: Help us develop verification techniques that scale to pervasive systems
- **Machine learning community**: Design constrained learning algorithms that respect normative boundaries

- **Multi-agent systems community**: Characterize strategic equilibria in norm-constrained games
- **HCI community**: Study how humans understand and interact with complex specifications
- **Ethics and governance scholars**: Guide us on value pluralism, democratic design, and accountability structures

For Practitioners

- **System builders**: Prototype ErisML-governed systems and report what works, what breaks
- **Domain experts**: Provide real requirements from healthcare, transportation, industrial, smart city domains
- **Policy and regulatory professionals**: Tell us what regulators need from specifications to ensure compliance
- **Security engineers**: Red team our designs, find loopholes, propose defenses

For Standards Bodies

- **IEEE, ISO, NIST**: Consider whether ErisML (or something like it) should be standardized
- **Industry consortia**: Coordinate on common specification formats to enable interoperability
- **Open source foundations**: Host tooling, reference implementations, specification repositories

For Society

- **Affected communities**: Participate in specification design so your values are encoded
- **Advocates**: Push for transparency, accountability, and inclusive governance
- **Policymakers**: Consider requiring formal specifications for high-stakes pervasive systems
- **Educators**: Teach the next generation of engineers to think about specification, verification, and governance—not just implementation

XI. CONCLUSION: FROM CHAOS TO ORDER, IMPERFECTLY

The Trojan War began with a simple question: "Who is the fairest?" The chaos of ambient intelligence begins with similar questions: What is safe? What is fair? What is acceptable? When must efficiency yield to privacy? When must autonomy yield to safety? These questions have no universal answers, only context-dependent, value-laden, contested negotiations.

ErisML proposes to make these negotiations explicit—to move them from implicit prompts and hidden code into legible, auditable, verifiable specifications. We don't claim this is easy, or that we've solved all the problems. In fact, we've catalogued at least ten fundamental challenges that may take decades to address.

But the alternative—continuing with ad-hoc, fragmented, opaque approaches—is worse. As foundation models move from cloud to edge, from single-purpose to multi-agent, from controlled to pervasive environments, the chaos will grow. Without common substrates for specification and governance, we risk:

- **Accidents from misalignment**: Agents pursuing conflicting objectives without coordination
- **Violations from ignorance**: Systems that don't know what norms apply or how to interpret them
- **Failures from brittleness**: Specifications that shatter under distribution shift or edge cases
- **Injustice from opacity**: Decisions that affect lives but can't be explained or contested
- **Erosion of trust**: Users who can't predict, understand, or control the systems around them

ErisML is a vision for structured chaos—not eliminating discord, but choreographing it. Like Eris herself, we acknowledge that conflict, competition, and disagreement are inherent to complex systems. The goal is not harmony (which is unattainable) but productive coexistence, transparent trade-offs, and accountable governance.

The path from vision to reality is long. It requires not just technical innovation but social negotiation, institutional change, and ongoing learning. We offer ErisML not as a finished solution but as a starting point—a provocation, a design space, a set of challenges, and an invitation to collaborate.

The age of ambient intelligence is here. The chaos is real. The question is: will we face it with fragmented tools and ad-hoc patches, or will we build common foundations for governance, safety, and trust?

Eris threw the apple. We pick it up. Let the work begin.

ACKNOWLEDGMENTS

The author thanks the anonymous reviewers whose feedback shaped this vision. Discussions with colleagues in pervasive computing, AI safety, multi-agent systems, and human-computer interaction informed this work. Some sections and editorial refinements were assisted by an AI system; all content was reviewed, edited, and approved by the author.

The name "ErisML" was chosen to honor the goddess of chaos, acknowledging that pervasive computing environments are fundamentally chaotic, and that our goal is not to eliminate chaos but to structure and govern it. May we prove worthy of the challenge.

REFERENCES

[1] M. Weiser, "The Computer for the 21st Century," *Scientific American*, vol. 265, no. 3, pp. 94-104, 1991.

[2] M. Satyanarayanan, "Pervasive Computing: Vision and Challenges," *IEEE Pervasive Computing*, vol. 1, no. 4, pp. 10-17, 2001.

[3] R. Bommasani et al., "On the Opportunities and Risks of Foundation Models," *arXiv:2108.07258*, 2021.

[4] S. Yao et al., "ReAct: Synergizing Reasoning and Acting in Language Models," *ICLR*, 2023.

[5] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: Theory and Practice*, Morgan Kaufmann, 2004.

[6] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed., MIT Press, 2018.

[7] M. Wooldridge, *An Introduction to MultiAgent Systems*, 2nd ed., Wiley, 2009.

[8] J. Carmo and A. J. I. Jones, "Deontic Logic and Contrary-to-Duties," in *Handbook of Philosophical Logic*, vol. 8, pp. 265-343, 2002.

[9] D. Amodei et al., "Concrete Problems in AI Safety," *arXiv:1606.06565*, 2016.

[10] L. Ouyang et al., "Training Language Models to Follow Instructions with Human Feedback," *NeurIPS*, 2022.

[11] D. Hadfield-Menell et al., "Inverse Reward Design," *NeurIPS*, 2017.

[12] B. McMahan et al., "Communication-Efficient Learning of Deep Networks from Decentralized Data," *AISTATS*, 2017.

[13] C. Dwork, "Differential Privacy," *ICALP*, 2006.

[14] S. Amershi et al., "Guidelines for Human-AI Interaction," *CHI*, 2019.

[15] P. Liang et al., "Holistic Evaluation of Language Models," *arXiv:2211.09110*, 2022.

[16] J. Achiam et al., "GPT-4 Technical Report," *arXiv:2303.08774*, 2023.

[17] D. Sculley et al., "Hidden Technical Debt in Machine Learning Systems," *NeurIPS*, 2015.

[18] T. Gebru et al., "Datasheets for Datasets," *Communications of the ACM*, vol. 64, no. 12, pp. 86-92, 2021.

[19] M. Mitchell et al., "Model Cards for Model Reporting," *FAT**, 2019.

[20] R. Emerson, "Power-Dependence Relations," *American Sociological Review*, vol. 27, no. 1, pp. 31-41, 1962.

[21] N. Dalal et al., "Value-Sensitive Design and Information Systems," in *The Handbook of Information and Computer Ethics*, Wiley, 2008.

[22] D. Ha and J. Schmidhuber, "World Models," *arXiv:1803.10122*, 2018.

[23] D. Hafner et al., "Mastering Atari with Discrete World Models," *ICLR*, 2021.

[24] A. Rai, "Explainable AI: From Black Box to Glass Box," *Journal of the Academy of Marketing Science*, vol. 48, pp. 137-141, 2020.

[25] S. Russell, *Human Compatible: Artificial Intelligence and the Problem of Control*, Viking, 2019.

[26] V. Dignum, *Responsible Artificial Intelligence: How to Develop and Use AI in a Responsible Way*, Springer, 2019.

[27] I. Rahwan et al., "Machine Behaviour," *Nature*, vol. 568, pp. 477-486, 2019.

[28] M. E. J. Newman, "The Structure and Function of Complex Networks," *SIAM Review*, vol. 45, no. 2, pp. 167-256, 2003.

[29] E. Brynjolfsson and A. McAfee, *The Second Machine Age*, Norton, 2014.

[30] T. C. Schelling, *Micromotives and Macrobehavior*, Norton, 1978.

APPENDIX A: ERISML GRAMMAR (EXCERPT)

We provide a partial BNF grammar for ErisML-Core. The full grammar is available in the technical report.

```
<model>            ::= <environment> <agent>* <norms>? <dynamics>?
<validation>?

<environment>      ::= "environment" <id> "{" <env-body> "}"

<env-body>         ::= <objects> <state> <observations>? <dynamics>

<objects>          ::= "objects:" <id-list> ";"

<state>            ::= "state:" <state-var>*

<state-var>        ::= <id> ":" <type> ("@" <modifier>)? ";"

<agent>            ::= "agent" <id> "{" <agent-body> "}"

<agent-body>       ::= <capabilities> <beliefs> <intents>
<constraints>?

<capabilities>     ::= "capabilities:" "{" <id-list> "}" ";"

<beliefs>          ::= "beliefs:" <belief-expr> ";"

<intents>          ::= "intents:" <intent-expr> ";"

<constraints>      ::= "constraints:" "{" <constraint>* "}" ";"

<norms>            ::= "norms" <id> "{" <norm-rule>* "}"

<norm-rule>        ::= <permission> | <prohibition> | <obligation> |
<sanction>
```

```
<permission>       ::= "permission:" "{" <action-expr> <condition>? "}"
";"

<prohibition>      ::= "prohibition:" "{" <action-expr> <condition>?
"}" ";"

<obligation>       ::= "obligation:" "{" <action-expr> <deadline>? "}"
";"

<sanction>         ::= "sanction:" "{" <consequence> <trigger> "}" ";"


<dynamics>         ::= "dynamics" "{" <transition>* <utility>? "}"

<validation>       ::= "validation" <id> "{" <scenario>* <metrics>? "}"


<type>             ::= "real" | "bool" | "int" | <enum> | <tuple> |
<list> | <record>

<intent-expr>      ::= <weighted> | <lexicographic> | <constrained>

<condition>        ::= "when" <expr> | "unless" <expr>
```

APPENDIX B: COMPILATION EXAMPLE

We show how a simple ErisML specification compiles to PDDL for classical planning.

ErisML Input:

```
environment SimpleRoom {
  objects: Robot, Door, Light;
  state:
    door.status: {open, closed};
    light.status: {on, off};
    robot.location: {inside, outside};

  dynamics:
    open_door() ~> door.status = open;
    close_door() ~> door.status = closed;
    enter() ~> robot.location = inside if door.status == open;
}

agent Robot {
  intents: goal robot.location == inside AND light.status == on;
}
```

```
norms Safety {
  prohibition: Robot.enter() unless door.status == open;
}
```

PDDL Output:

```
(define (domain simple-room)
  (:requirements :strips :typing)
  (:types door light robot location status)

  (:predicates
    (door-open)
    (door-closed)
    (light-on)
    (light-off)
    (robot-inside)
    (robot-outside))

  (:action open-door
    :parameters ()
    :precondition (door-closed)
    :effect (and (door-open) (not (door-closed))))

  (:action enter
    :parameters ()
    :precondition (and (robot-outside) (door-open))  ; Norm enforced
here
    :effect (and (robot-inside) (not (robot-outside)))))

(define (problem reach-inside)
  (:domain simple-room)
  (:init (door-closed) (light-off) (robot-outside))
  (:goal (and (robot-inside) (light-on))))
```

The prohibition from norms Safety is compiled into the precondition of the enter action, ensuring the planner never generates norm-violating plans.