SENG 696 Agent-Based Software Engineering

Multi-Agent Daily Planning System (MADPS)

Report 1B
GAIA Design

Ali Mohammadi Ruzbahani [30261140], Shuvam Agarwala [30290444]

Course
Agent-Based Software Engineering (SENG 696)

Instructor: Professor Behrouz Far

Date: October 7, 2025

Contents

List of Figures

List of Tables

1. Methodology & Overview

1.1 Introduction

The Multi-Agent Daily Planning System (MADPS) represents a comprehensive solution to the challenge of intelligent daily task scheduling for individual users. In contemporary professional and personal environments, individuals face the complexity of managing numerous tasks with varying priorities, interdependencies, and time constraints, while simultaneously contending with frequent disruptions and shifting availability. MADPS addresses these challenges through a multi-agent architecture that combines optimization algorithms with adaptive learning to generate personalized, high-quality schedules.

The system is designed around the premise that effective planning requires not only satisfaction of hard constraints, such as deadlines and task dependencies,but also alignment with user-specific patterns of energy, preference, and working style. Traditional scheduling tools treat all users identically and fail to adapt to individual rhythms or learn from historical behavior. MADPS bridges this gap by incorporating machine learning components that continuously refine their understanding of user characteristics, enabling the system to propose schedules that are both feasible and personally optimized.

1.2 Methodology Selection: GAIA

This design employs the GAIA (Generic Architecture for Information Availability) methodology, a systematic approach specifically developed for the specification and design of multi-agent systems. GAIA provides a rigorous framework for decomposing system-level goals into agent roles, defining interaction protocols, and specifying the services that agents provide. Unlike object-oriented or service-oriented methodologies, GAIA explicitly addresses the unique characteristics of agent-based systems: autonomy, proactivity, social ability, and adaptability.

The selection of GAIA for MADPS is motivated by several factors. First, GAIA's goal-driven decomposition ensures complete traceability from high-level system objectives to concrete agent responsibilities and computational services. This traceability is essential for validating that the implemented system fulfills its intended purpose. Second, GAIA's emphasis on interaction protocols and organizational structures provides a clear blueprint for inter-agent communication, reducing ambiguity during implementation. Third, GAIA supports formal specification of agent behavior through liveness and safety properties, enabling verification that agents operate correctly under both normal and exceptional conditions.

GAIA structures the design process through a series of interconnected models. The Role Model identifies the key responsibilities, permissions, and knowledge requirements of each agent. The Interaction

Model formalizes the protocols through which agents communicate, specifying message types, sequences, and failure handling. The Services Model details the computational functions each agent provides, including algorithms, inputs, outputs, and performance criteria. Finally, the Acquaintance Model maps the communication topology, clarifying which agents interact and for what purposes. This structured approach ensures that MADPS is designed systematically, with clear separation of concerns and well-defined interfaces.

1.3 System Architecture and Agent Roles

MADPS employs a three-agent architecture augmented by supporting components. Each agent is assigned a distinct area of expertise, promoting modularity and enabling parallel development and testing. The User Profile Agent (UPA) serves as the personalization engine, maintaining models of user energy patterns and task preferences. By analyzing historical completion data and explicit user feedback, the UPA continuously refines its predictions, enabling the system to propose schedules that align with individual working rhythms. The Task Management Agent (TMA) functions as the custodian of task integrity, managing the complete lifecycle of tasks from creation through completion. It enforces structural constraints such as dependency acyclicity and provides versioned task graphs to other agents. The Planning & Adaptation Agent (PAA) acts as the core decision-maker, responsible for generating initial schedules and rapidly adapting them in response to disruptions. The PAA employs a hybrid optimization strategy, combining genetic algorithms for global optimization with greedy local repair for fast responses to minor disruptions.

These agents interact through a message bus that decouples sender and receiver, promoting scalability and fault tolerance. Supporting components include a UI layer for user interaction and a Logger/Monitor for system observability. This architecture balances autonomy, each agent maintains its own state and decision logic, with coordination, as agents collaborate through well-defined protocols to achieve system-level goals.

1.4 Design Philosophy and Objectives

The design of MADPS is guided by three overarching objectives that balance competing demands. The first objective is to maximize user satisfaction by producing schedules that respect all hard constraints while optimizing alignment with user priorities and energy patterns. This requires sophisticated constraint satisfaction and scoring mechanisms that weigh multiple factors. The second objective is responsiveness and adaptability. Real-world planning is inherently dynamic, with tasks overrunning estimates, new urgent tasks arriving, and availability windows shifting. MADPS must respond to these disruptions rapidly without destabilizing the entire schedule, a requirement that drives the two-tier replanning strategy employed by the PAA. The third objective is personalization through learning. By incorporating feedback loops that update energy forecasts and preference weights, MADPS improves its alignment with user needs over time, transforming from a generic scheduling tool into a personalized planning assistant.

These objectives impose specific performance requirements. Initial plan generation for a typical day (up to 40 tasks) must complete within two seconds to provide interactive responsiveness. Local replanning for minor disruptions must achieve a median latency of 150 milliseconds or less, with 95th percentile latency under 500 milliseconds, ensuring that the system feels immediate and fluid during use. All committed schedules must be free of hard-constraint violations, and schedule churn, the extent to which replanning alters previously committed blocks, must remain below a configurable threshold to prevent user frustration. Finally, energy alignment scores should demonstrate week-over-week improvement, validating the effectiveness of the learning mechanisms.

1.5 Scope and Assumptions

MADPS is scoped as a single-user planning assistant, addressing the personal productivity domain rather than multi-user coordination or resource allocation scenarios. The system assumes that users can provide reasonable estimates of task effort and priority, that task dependencies form directed acyclic graphs without cycles, and that user availability windows are known or can be specified in advance. The system is designed for planning horizons of a single day, though the architecture supports extension to multi-day planning.

Certain limitations are acknowledged. The current design does not integrate with external calendar systems, though the architecture is extensible to accommodate such integration in future iterations. The system relies on user-provided effort estimates rather than automatically estimating task duration, a simplification that reduces complexity but places burden on the user. The learning components face a cold-start problem, requiring an initial data collection period before personalization becomes effective. For very large task sets exceeding 100 tasks per day, performance may degrade below target thresholds, though such scenarios are outside the typical use case.

1.6 Report Structure

The remainder of this report systematically elaborates the MADPS design through the GAIA framework. Chapter 2 establishes system goals and traces them to agent roles and services, providing a complete requirements-to-implementation mapping. Chapter 3 presents the Role Model, formally specifying each agent's responsibilities, permissions, and knowledge. Chapter 4 visualizes the system architecture through component diagrams. Chapter 5 details the Interaction Model, defining protocols, message sequences, and pre/post-conditions for agent communication. Chapter 6 focuses on the PAA's internal behavior, presenting flowcharts and statecharts that capture its decision logic. Chapter 7 specifies the Services Model, documenting algorithms, inputs, outputs, and failure modes for each service. Chapter 8 defines the data model through entity-relationship diagrams and data dictionaries. Finally, Chapter 9 presents the Acquaintance Model and summarizes the design.

1.7 Design Objectives

MADPS aims to deliver three core capabilities that distinguish it from conventional scheduling tools:

- Produce feasible, high-quality schedules that maximize priority satisfaction under deadlines, dependencies, and availability constraints.
- Re-plan rapidly following disruptions while minimizing schedule churn and maintaining constraint satisfaction.
- Personalize schedules by learning user-specific energy curves and preference weights, continuously improving fit over time.

2. Goals & Traceability

2.1 Goal Set

- **G1** Maximize priority satisfaction under constraints.
- **G2** Minimize re-plan latency and plan churn after disruptions.
- **G3** Personalize plans using learned energy/pref models.

2.2 Goals → Roles → Services (Traceability)

Goal	Primary Role(s)	Key Services
G1	PAA, TMA	GeneratePlan, ValidateConstraints, CommitSchedule
G2	PAA	RePlanFast (local repair), GlobalOptimize (GA), DeltaPropose
G3	UPA, PAA	LearnPreferences, ForecastEnergy, PlanWithEnergy

Table 2.1: Goals-to-Roles-to-Services traceability.

2.3 Acceptance Criteria

- Initial plan for a typical day ($\leq$ 40 tasks) in $\leq$ 2s; local re-plan p50 $\leq$ 150ms; p95 $\leq$ 500ms.
- No hard-constraint violations (deadlines, double booking); conflicts resolved before commit.
- Energy alignment score improves week-over-week (non-decreasing median).

3. Role Model

3.1 Table 3.1: Role Model

Role	Responsibilities (Liveness / Safety)	Permissions	Knowledge	I/O & KPIs
User Profile Agent (UPA)	*Liveness:* $(ReceiveEvents \cdot Update \cdot PublishForecast)^{\omega}$. *Safety:* Forecast $\in [0,1]$; no raw PII exposure; bounded updates.	Read event stream; write model parameters; answer forecast queries from PAA/UI.	Energy-curve parameters; preference weights; feedback history; learning hyperparams.	*In:* events, feedback. *Out:* energy_curve, weights. *KPIs:* MAE of energy forecast; forecast freshness; feedback acceptance rate.
Task Management Agent (TMA)	*Liveness:* $(Validate \cdot Version \cdot PublishGraph)^{\omega}$. *Safety:* Task DAG only; IDs immutable; dependency integrity.	Full CRUD on tasks; publish versioned graphs; validate dependencies and effort bounds.	Task DAG; critical path metrics; effort/priority metadata; version history.	*In:* task CRUD. *Out:* task_graph (ver), validation diagnostics. *KPIs:* validation error rate; time-to-graph; DAG acyclicity checks.
Planning & Adaptation Agent (PAA)	*Liveness:* $InitPlan \cdot (Observe \cdot RePlan)^{*}$. *Safety:* respect hard constraints; no double booking; churn $\leq \theta$.	Read forecasts/graphs; write schedule store; propose/commit via UI; hold locks on committed blocks.	Constraint set; schedule model; scoring weights; churn threshold θ; explanation traces.	*In:* plan request, disruption events. *Out:* schedule, delta, rationale. *KPIs:* plan latency (p50/p95), violations=0, churn$\leq \theta$, user accept rate.

Table 3.1: Role Model for MADPS (Part 1).

Role	Responsibilities (Liveness / Safety)	Permissions	Knowledge	I/O & KPIs
UI / Frontend	*Liveness:* collect inputs, display outputs, route user actions. *Safety:* confirm destructive ops; sanitize inputs.	Request/accept proposals; CRUD tasks; send feedback/events.	User settings; view state; session context.	*In:* proposals, forecasts. *Out:* requests, accepts, feedback. *KPIs:* task entry latency; accept ratio; usability NPS.
Logger / Monitor	*Liveness:* append events/metrics; rotate logs. *Safety:* append-only; redact sensitive payloads.	Write-only from agents; read for analytics.	Event schema; metric dictionary; retention policy.	*In:* events/metrics. *Out:* audit/analytics. *KPIs:* lossless ingest; query latency.

Table 3.2: Role Model for MADPS (Part 2).

4. Agent System Architecture

4.1 Component Diagram

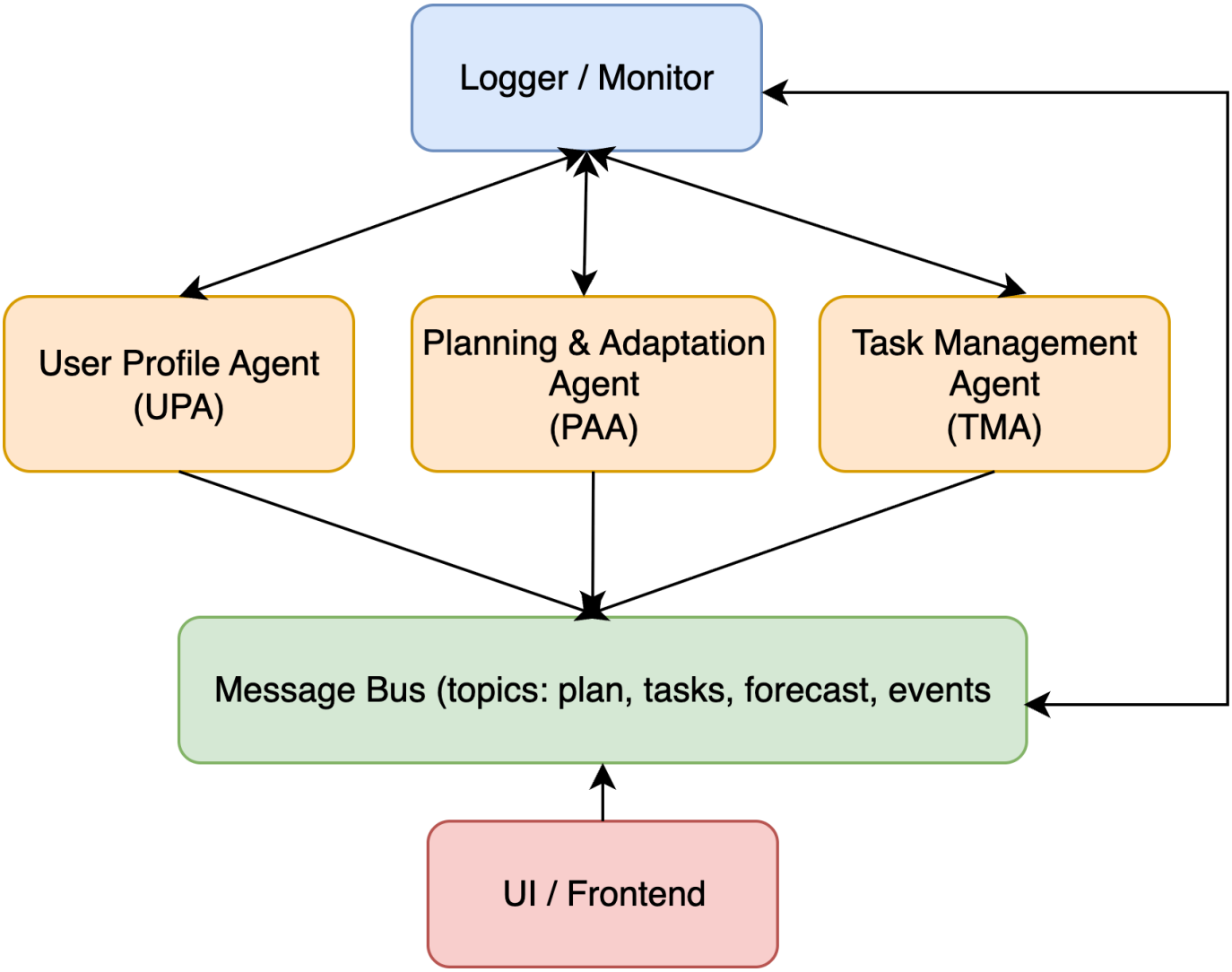

Figure 4.1: High-level component/agent architecture.

5. Interaction Model

5.1 Table 5.1: Protocol Specification

Protocol	Initiator → Responder(s)	Performatives & Data	Timeout / Retry
Plan Request	UI → PAA; PAA → UPA,TMA	REQUEST(plan D); QUERY(forecast D), QUERY(task_graph); PROPOSE(schedule,rationale); ACCEPT / REJECT; INFORM(commit/abort)	2s / 1 retry
Disruption Handling	UI → PAA	INFORM(overrun/new/cancel); PROPOSE(delta); ACCEPT / REJECT; INFORM(updated)	1s / 2 retries
Feedback Loop	UI → UPA	INFORM(feedback payload); optional INFORM(updated forecast) to PAA	async
Task Graph Update	TMA → PAA	INFORM(task_graph, version); ACK; optional re-plan trigger	1s / 2 retries

Table 5.1: Overview of agent interaction protocols.

5.2 Table 5.2: Interaction Model

Protocol	Inputs (from / type)	Outputs (to / type)	Pre-conditions	Post-conditions
Plan Request	UI → PAA : day, availability; PAA → UPA : forecast(D); PAA → TMA : task_graph(D)	PAA → UI : schedule + rationale; UI → PAA : ACCEPT/REJECT; PAA → UI : commit/abort	Valid task DAG; default or fresh forecast available	Committed schedule stored; conflicts=0; explanation attached
Disruption Handling	UI → PAA : overrun/new/cancel + context	PAA → UI : delta schedule; UI → PAA : ACCEPT/REJECT; PAA → UI : updated	Current schedule exists; disruption payload valid	Updated schedule with churn $\leq \theta$; if global, GA result committed
Feedback Loop	UI → UPA : thumbs/notes; completion events via Logger	UPA → PAA : updated forecast/weights	Enough data points in window	Forecast MAE decreases or remains stable; preferences updated
Task Graph Update	TMA → PAA : task_graph(version)	PAA: (optional) re-plan; ACK to TMA	New/edited tasks validated; DAG acyclic	PAA cache updated; schedule may be refreshed

Table 5.2: Descriptive interaction model for MADPS.

5.3 Message Sequence Charts

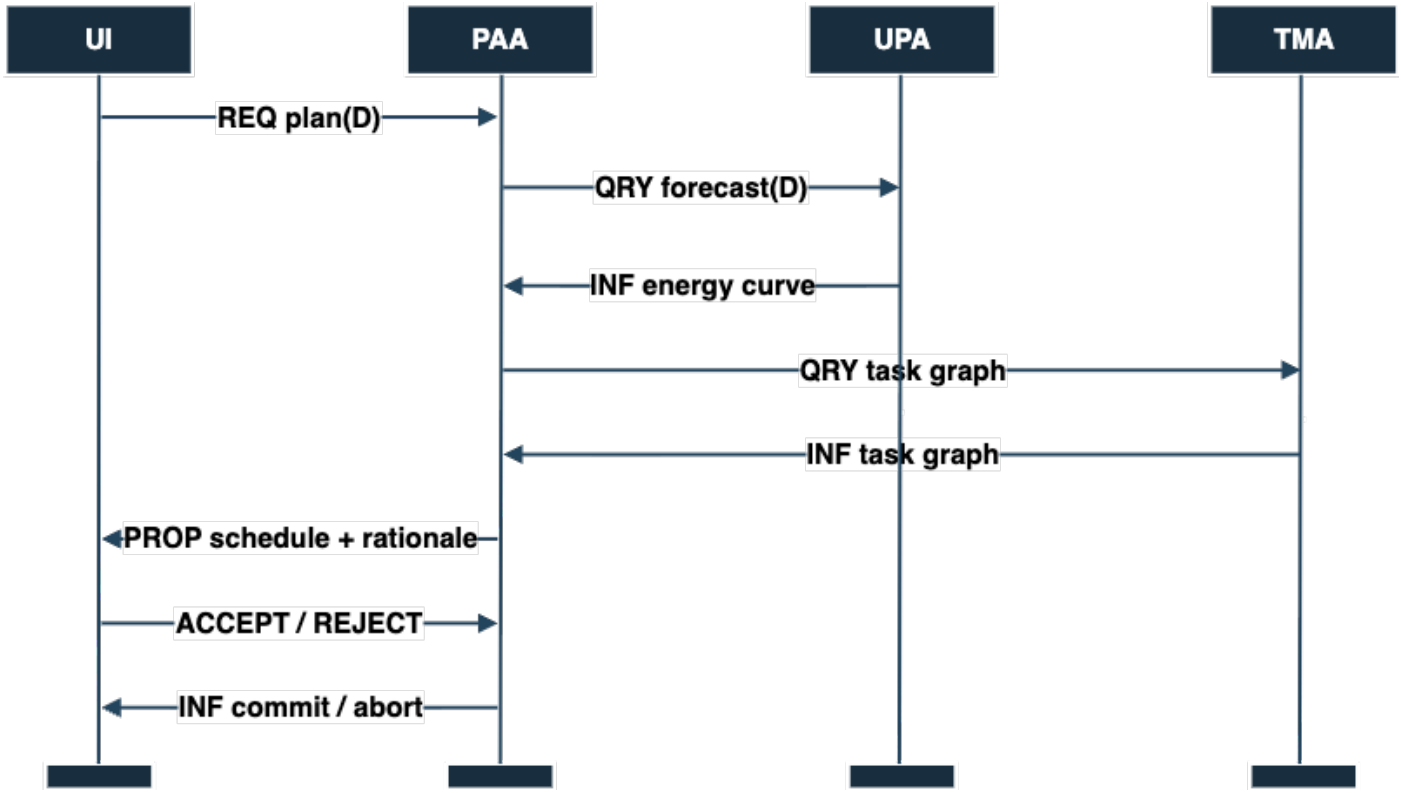

Figure 5.1: Plan request protocol.

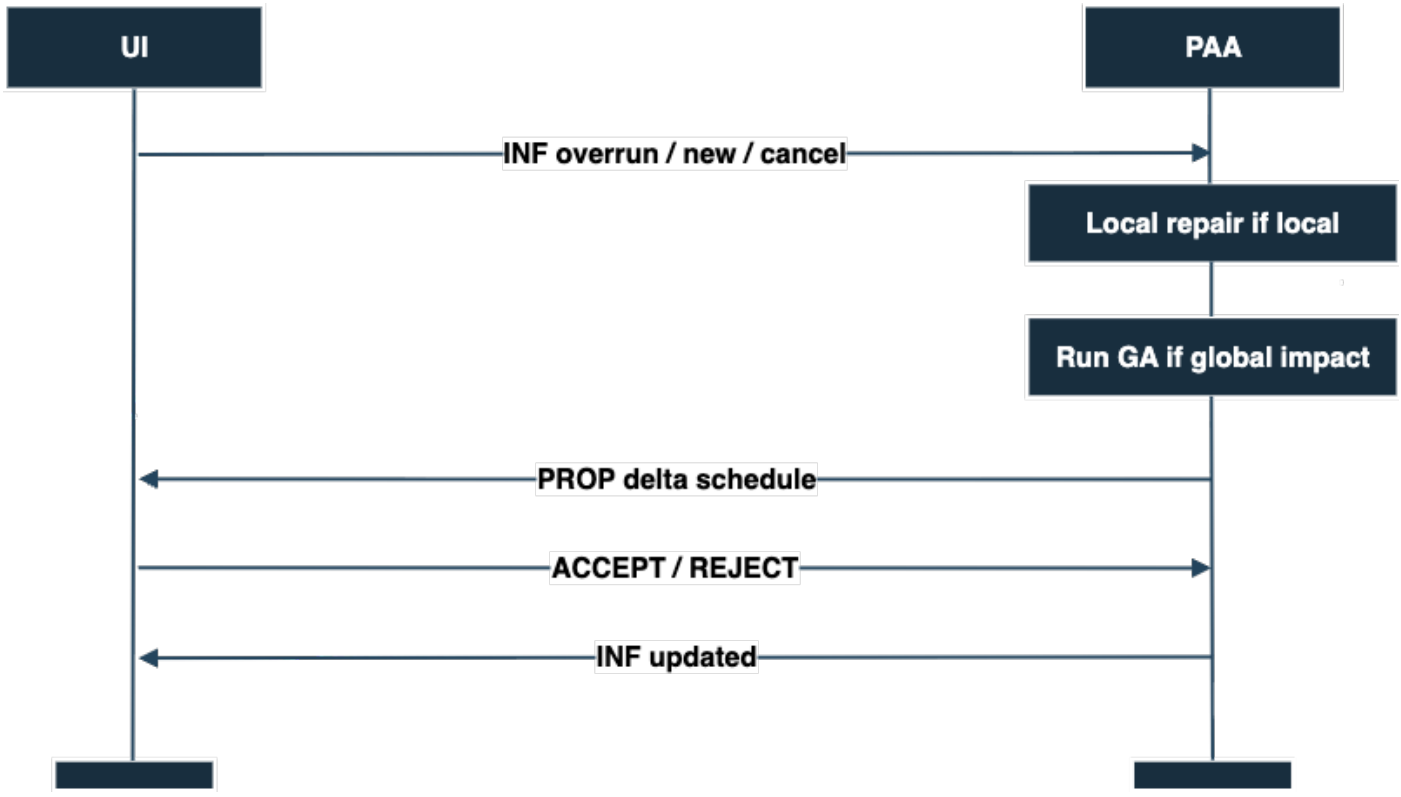

Figure 5.2: Disruption handling protocol.

6. PAA Algorithms & Behaviour

6.1 Planning/Replanning Flowchart

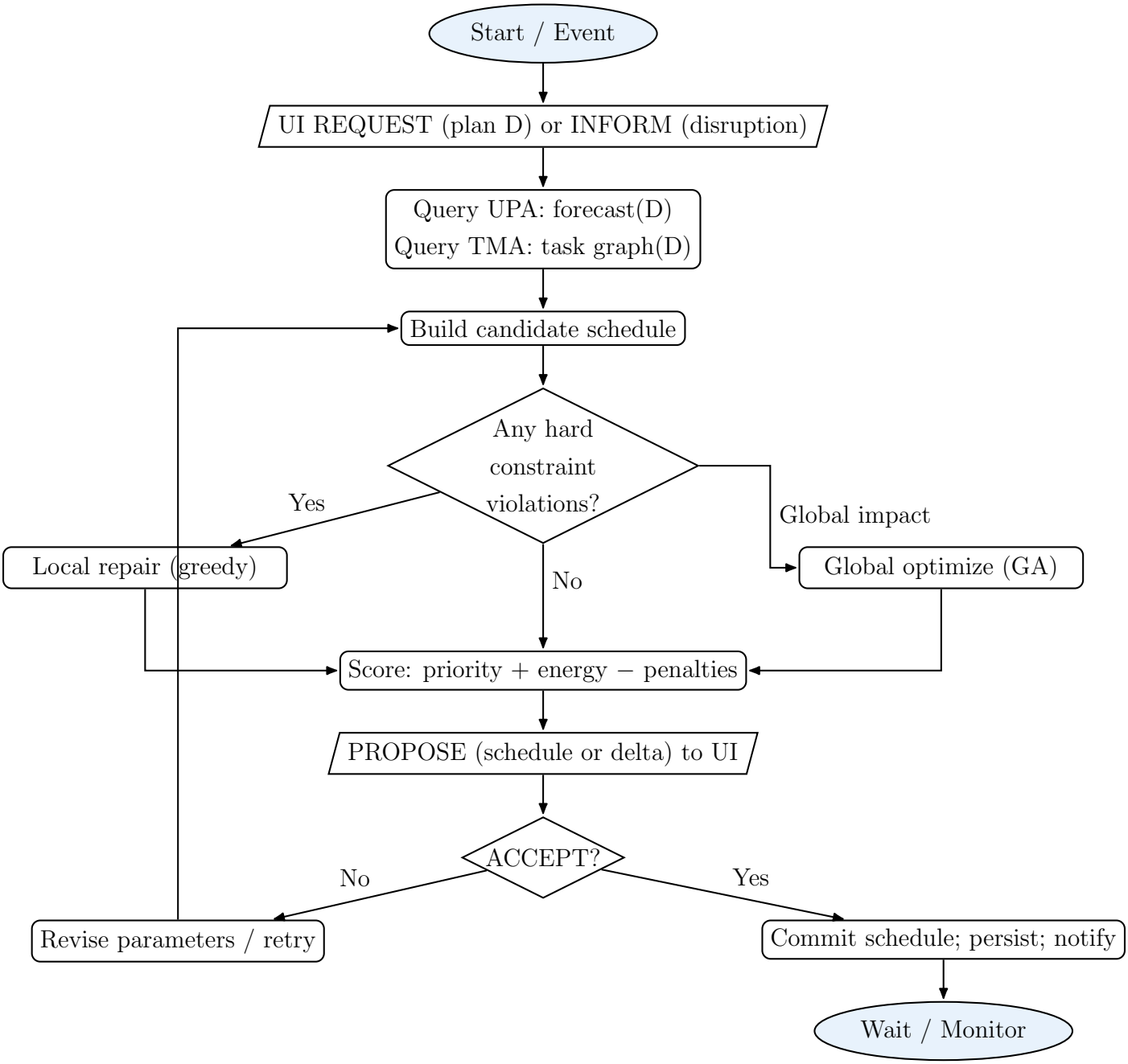

Figure 6.1: PAA planning and re-planning flow.

6.2 PAA Statechart

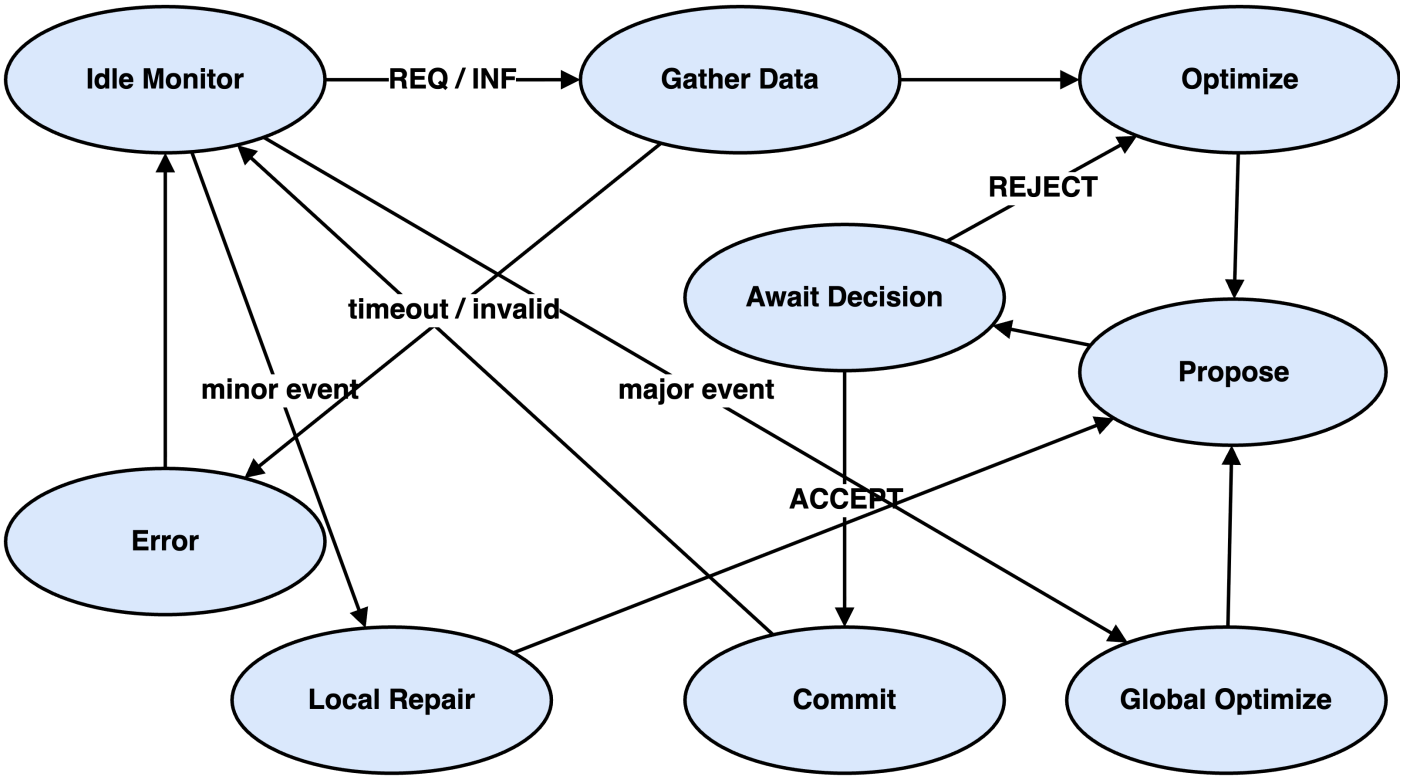

Figure 6.2: PAA behavioural statechart

6.3 Agent Workflow (End-to-End)

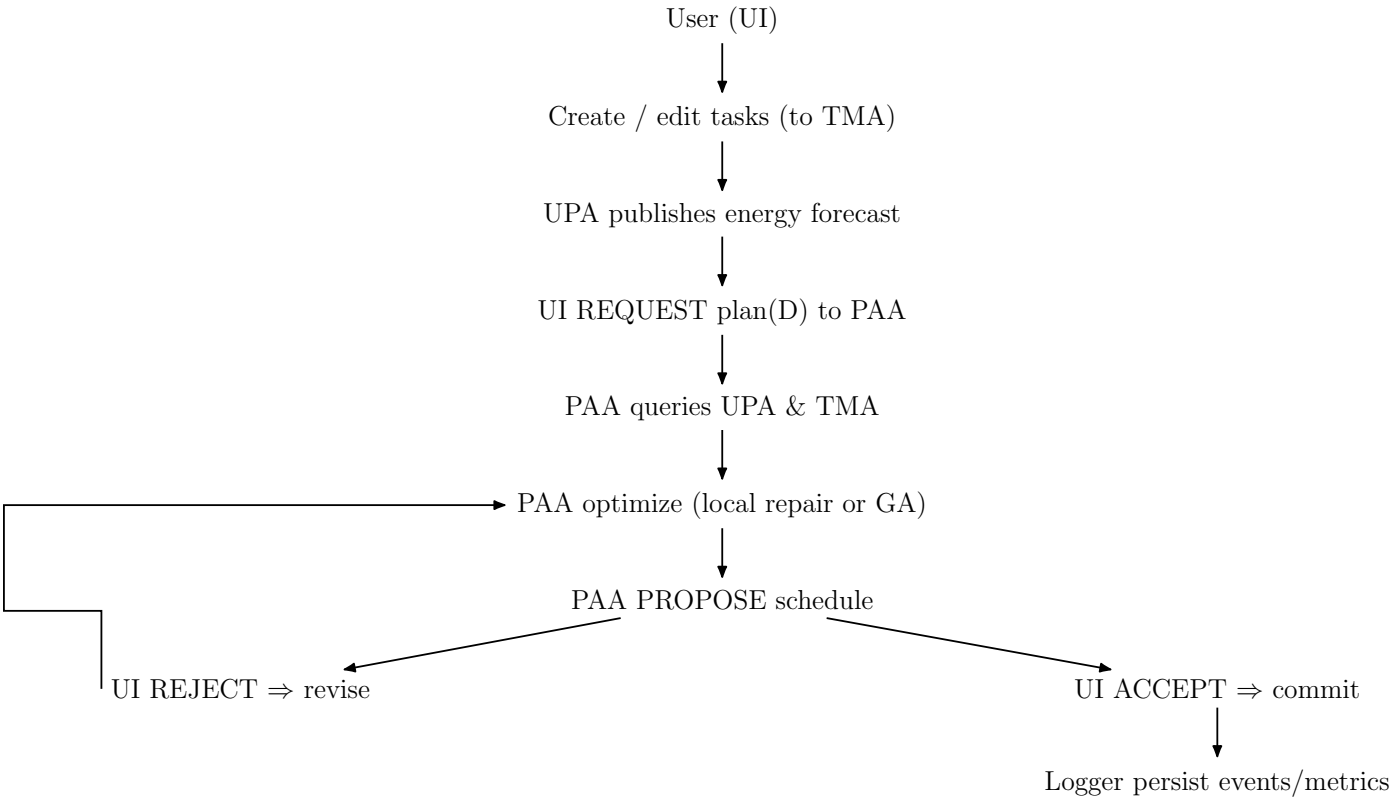

Figure 6.3: Agent workflow from task intake to plan commit.

7. Services Model

7.1 Table 7.1: Services Model

Service	Provide by	Inputs	Processing / Algorithm	Outputs / KPIs / Failures
Generate Plan	PAA	task_graph (ver), energy_forecast, availability window, weights	Modified GA optimizing: priority fit + energy alignment − penalties (violations, transitions); constraint manager prunes infeasible solutions; explainer collects top-3 reasons per placement	*Out:* schedule + rationale. *KPIs:* latency $\leq$ 2s; violations=0. *Fail:* invalid DAG, timeout $\Rightarrow$ fallback template.
RePlan Fast	PAA	disruption_event (overrun/new/-cancel), current schedule	Greedy local repair in impact neighborhood; minimize churn objective; if no feasible local fix, escalate to GA	*Out:* delta schedule. *KPIs:* p50 $\leq$ 150ms (p95 $\leq$ 500ms); churn $\leq \theta$. *Fail:* escalate to GA; notify UI.
Global Optimize	PAA	task_graph, energy_forecast, current locks/commits	GA with capped population & early stopping; seeded by previous best; strong penalties for hard-constraint violation	*Out:* full schedule. *KPIs:* convergence under cap; violations=0. *Fail:* timeout $\Rightarrow$ last-best + buffers.
Validate Constraints	TMA	task CRUD payload; dependency edits	Cycle detection; field validation; recompute critical path; version the DAG	*Out:* new task_graph (version); diagnostics. *KPIs:* time-to-graph. *Fail:* reject with reasons.
Commit Schedule	PAA	accepted proposal (schedule/delta), locks	Persist schedule; lock blocks; emit commit events to Logger	*Out:* committed plan; audit trail. *KPIs:* atomic commit; durability. *Fail:* rollback and notify UI.

Table 7.1: Services Model for MADPS (Part 1).

Service	Provided by	Inputs	Processing / Algorithm	Outputs / KPIs / Failures
Learn Preferences	UPA	events, feedback, completion times	Online updates to preference weights; time-series update to energy curve; monotonicity constraints enforced	*Out:* updated weights/curve. *KPIs:* MAE trend ↓. *Fail:* revert to last stable model.
Publish Forecast	UPA	request (day) or scheduled refresh	Build normalized energy curve for day; include confidence	*Out:* energy_curve[0..1], confidence. *KPIs:* freshness; MAE. *Fail:* default curve.

Table 7.2: Services Model for MADPS (Part 2).

8. Data Specification

8.1 Data Dictionary

Entity	Fields	Key
Task	id, title, deadline, effort, priority, status, depends_on (nullable)	pk:id
ScheduleBlock	id, task_id, start, end, confidence	pk:id, fk:task_id
Preference	id, key, value (float), updated_at	pk:id
Event	id, ts, type, payload(json)	pk:id

Table 8.1: Data dictionary for key MADPS entities.

8.2 E–R Sketch

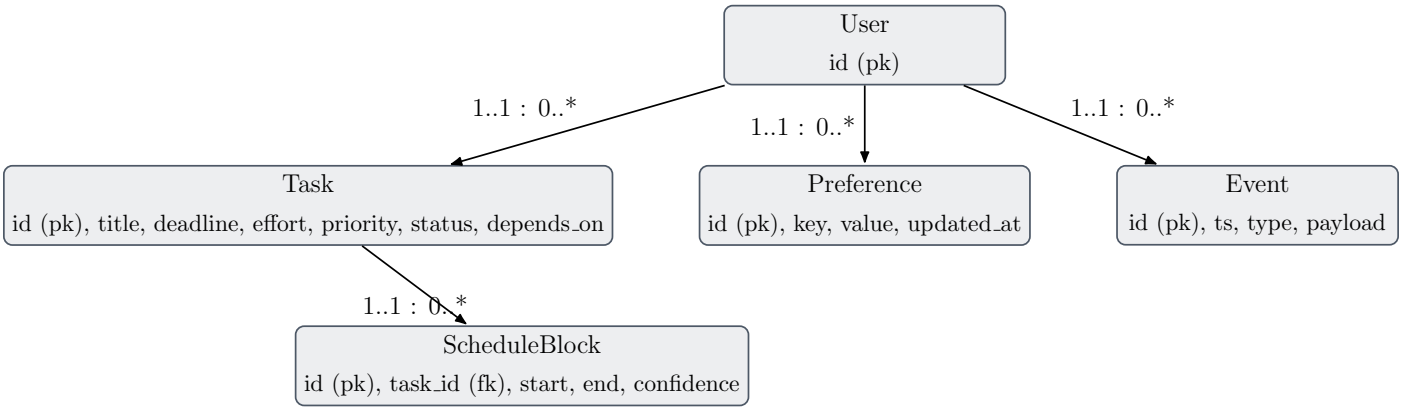

Figure 8.1: E–R sketch for MADPS storage.

9. Acquaintance Model & Summary

9.1 Acquaintance Model

The acquaintance model defines the communication topology of MADPS, specifying which agents are aware of and interact with one another. Unlike architectural diagrams that show physical or logical component relationships, the acquaintance model captures the social structure of the multi-agent system, the pattern of direct communication pathways that enable collaborative problem-solving. This model is essential for understanding information flow, identifying potential communication bottlenecks, and reasoning about system-level properties such as fault propagation and scalability.

In MADPS, the acquaintance structure is deliberately designed to balance flexibility with simplicity. The UI serves as the primary user-facing gateway, maintaining bidirectional communication channels with all three core agents. Specifically, the UI exchanges plan lifecycle messages with the PAA, covering plan requests, proposals, acceptances, and disruption notifications. It maintains a separate channel with the TMA for task CRUD operations, enabling users to create, modify, and delete tasks independently of the planning process. The UI also communicates with the UPA to transmit user feedback, including explicit thumbs-up/down ratings and implicit completion signals. This star-like topology with the UI at the center ensures that all user-initiated actions and system responses flow through a single, well-defined interface point, simplifying session management and user experience consistency.

The PAA occupies a central position in the agent-to-agent communication topology. As the primary orchestrator of planning activities, the PAA maintains bidirectional channels with both the UPA and TMA. When generating or revising schedules, the PAA queries the UPA for energy forecasts that inform task placement decisions, ensuring alignment with the user's predicted productivity rhythms. The PAA also queries the TMA for the current task graph, including dependency structures, effort estimates, and priorities. These queries follow a synchronous request-response pattern with defined timeouts, ensuring that the PAA can proceed even if a dependent agent experiences transient failures. Conversely, both the UPA and TMA can proactively notify the PAA of significant updates, such as substantially revised energy models or critical changes to task dependencies, that may warrant schedule reconsideration. This bidirectional communication enables both pull-based coordination (PAA requesting data) and push-based notification (supporting agents alerting the PAA to relevant changes).

A universal communication pathway exists from all agents to the Logger/Monitor component. This unidirectional channel carries events, metrics, and diagnostic information that support system observability, performance analysis, and debugging. Unlike the interactive channels described above, logging communication is asynchronous and fire-and-forget, ensuring that logging overhead does not impact

critical path latency. The Logger maintains append-only storage with configurable retention policies, enabling post-hoc analysis of system behavior and user interaction patterns.

The acquaintance model deliberately avoids direct communication between the UPA and TMA. These two agents operate in largely independent domains, user modeling and task structure, respectively and have no direct operational dependencies. All coordination between these domains is mediated by the PAA, which integrates their outputs during planning. This design choice reduces coupling and simplifies the mental model of system behavior, at the cost of introducing the PAA as a potential single point of failure. Future iterations might explore peer-to-peer channels for specific scenarios, but the current design prioritizes simplicity and clear separation of concerns.

9.2 Design Summary and Validation

This GAIA-based design specification for MADPS provides a comprehensive blueprint for implementing an intelligent, adaptive, multi-agent daily planning system. The design process began with the articulation of three high-level goals: maximizing priority satisfaction under constraints, minimizing replanning latency and schedule churn, and personalizing schedules through learning. These goals were systematically decomposed into agent roles, with each role assigned specific responsibilities, permissions, and knowledge bases formalized through liveness and safety properties. The Role Model ensures that every system capability is clearly attributed to an agent, enabling focused implementation and testing.

The Interaction Model formalizes the social dimension of the multi-agent system, specifying four primary protocols, Plan Request, Disruption Handling, Feedback Loop, and Task Graph Update, that govern agent communication. Each protocol is defined through FIPA-compliant performatives, message sequences captured in UML sequence diagrams, and explicit pre- and post-conditions that characterize correct execution. This formalization transforms informal collaboration patterns into verifiable contracts, reducing ambiguity and enabling systematic testing of inter-agent interactions.

The PAA, as the system's most complex agent, receives detailed behavioral specification through multiple complementary models. The planning/replanning flowchart captures the control flow of schedule generation, illustrating decision points where the PAA chooses between local repair and global optimization strategies. The statechart provides a state-based view of PAA behavior, showing how the agent transitions between idle monitoring, data gathering, optimization, proposal, and commitment states in response to external events and internal conditions. Together, these models provide both algorithmic detail and state-based reasoning, supporting implementation in languages ranging from procedural (following the flowchart) to event-driven (following the statechart).

The Services Model bridges the gap between role-level abstractions and concrete algorithms. For each service, GeneratePlan, RePlanFast, GlobalOptimize, ValidateConstraints, CommitSchedule, LearnPreferences, and PublishForecast, the model specifies inputs, processing logic, outputs, key performance indicators, and failure modes. This level of detail enables independent development and testing of services, facilitates performance profiling, and provides a foundation for future optimization. The inclusion of explicit failure modes and fallback strategies reflects the design's emphasis on robustness,

ensuring that the system degrades gracefully rather than failing catastrophically when exceptional conditions arise.

The Data Specification grounds the design in concrete storage structures. The entity-relationship model identifies four core entities, Task, ScheduleBlock, Preference, and Event and their relationships, while the data dictionary defines fields, types, and keys. This specification ensures that all agents share a consistent understanding of data semantics, preventing integration issues that often arise when components make incompatible assumptions about data structure or meaning. The use of versioned task graphs and timestamped events supports both concurrency control and historical analysis.

The Acquaintance Model completes the design by making explicit the communication topology. By documenting which agents interact and for what purposes, this model enables reasoning about information flow, fault propagation, and potential performance bottlenecks. The deliberate choice of a hub-and-spoke topology for user-facing interactions, combined with PAA-centered agent-to-agent coordination, balances flexibility with simplicity.

9.3 Design Validation Against Goals

The design can be validated against the original goals through direct mapping. Goal G1 (maximize priority satisfaction under constraints) is addressed through the PAA's GeneratePlan service, which employs a modified genetic algorithm with explicit scoring of priority fit, energy alignment, and constraint penalties. The constraint manager component within the PAA ensures that no hard constraints are violated in committed schedules, satisfying the safety requirement. Goal G2 (minimize replanning latency and churn) is addressed through the PAA's two-tier replanning strategy. The RePlanFast service provides sub-second responses to minor disruptions through greedy local repair, while the GlobalOptimize service handles major disruptions when necessary. The explicit churn threshold ensures that replanning respects user tolerance for schedule instability. Goal G3 (personalize through learning) is addressed through the UPA's LearnPreferences and PublishForecast services, which continuously update models based on completion data and explicit feedback. The integration of these forecasts into the PAA's scoring function closes the loop, ensuring that learned preferences influence future schedules.

Performance requirements are similarly traceable to specific design decisions. The two-second target for initial planning is addressed through algorithmic choices in GeneratePlan: capped population size, early stopping conditions, and constraint-based pruning all reduce computational load without sacrificing solution quality. The sub-second replanning targets are met through the local repair strategy, which restricts search to the impact neighborhood of a disruption. The zero-violations requirement is enforced through safety properties in the PAA role specification and validation logic in the constraint manager. The week-over-week improvement in energy alignment is enabled by the UPA's online learning algorithms, which incorporate exponential smoothing and monotonicity constraints to ensure model stability while allowing adaptation.

9.4 Future Extensions and Scalability

While the current design targets single-user scenarios with moderate task volumes (up to 40 tasks per day), the architecture is extensible in several dimensions. Multi-user coordination could be supported by instantiating separate agent sets per user and introducing a new Coordination Agent responsible for resolving conflicts over shared resources or collaborative tasks. External calendar integration could be achieved by extending the TMA with connectors to Google Calendar, Outlook, or other services, treating external events as immutable tasks in the dependency graph. Multi-day planning could be supported by extending the PAA's horizon and introducing explicit carryover logic for incomplete tasks. Machine learning enhancements could replace the current rule-based energy forecasting in the UPA with more sophisticated time-series models such as LSTM networks or Prophet, potentially improving prediction accuracy for users with complex patterns.

Scalability to larger task sets would require algorithmic optimization rather than architectural change. The genetic algorithm in GeneratePlan could be replaced with more efficient approaches such as constraint programming solvers or hybrid methods that combine local search with systematic search. Incremental recomputation techniques could reduce the cost of replanning by caching intermediate results and updating only affected portions of the schedule. Parallelization of constraint checking and scoring could leverage multi-core processors to improve throughput. These optimizations are compatible with the current design and could be introduced without disrupting inter-agent protocols or data structures.

9.5 Conclusion

This GAIA design specification provides a complete, formal, and traceable blueprint for the Multi-Agent Daily Planning System. By systematically decomposing goals into roles, formalizing interactions through protocols, specifying services with algorithmic detail, and grounding the design in concrete data structures, the specification bridges the gap between high-level vision and an implementable system. The emphasis on liveness and safety properties, explicit failure modes, and performance criteria ensures that the design is not merely descriptive but prescriptive, providing clear guidance for implementation and validation. The use of multiple complementary models, roles, interactions, services, data, and acquaintances reflects the multifaceted nature of agent-based systems, where behavior emerges from the interplay of autonomous components communicating through well-defined protocols. The resulting design balances competing demands: constraint satisfaction versus optimization, responsiveness versus solution quality, autonomy versus coordination, and simplicity versus extensibility. MADPS stands as a comprehensive example of how the GAIA methodology can be applied to produce rigorous, implementable specifications for intelligent multi-agent systems.