

Hierarchical Task Networks and Goal Oriented Action Planning for Modern Agentic Systems

Executive summary

Hierarchical Task Network (HTN) planning and Goal-Oriented Action Planning (GOAP) are two distinct but related approaches to deliberative action selection and multi-step behavior synthesis. GOAP, as used in game AI, is primarily an adaptation of STRIPS-style classical planning: it represents world states as sets of propositional facts, represents actions with preconditions/effects and costs, and uses search (commonly A* or related techniques) to find a low-cost action sequence to reach a desired goal condition. ¹

HTN planning is a hierarchical refinement paradigm: it represents “what to do” initially as abstract tasks and repeatedly decomposes them into more concrete subtasks via methods until primitive (directly executable) actions remain, subject to ordering and other constraints. This introduces an explicit engineering layer for hierarchical operational knowledge; in many practical settings this constrains the plan space and can reduce search relative to unconstrained goal-state planning, while also creating distinct correctness/verification questions and potential expressivity/decidability issues depending on the HTN fragment. ²

The most practically relevant differences for software architecture work are: (a) whether the plan space is *goal-defined* (GOAP) versus *procedure-constrained* (HTN); (b) whether hierarchical control knowledge is explicit and mandatory (HTN) versus optional (GOAP, via heuristics/costs); (c) how execution monitoring and repair are handled, where HTN planning has a mature literature on plan repair/replanning with task hierarchy constraints and on integrating acting/planning over shared hierarchical operational models. ³

In modern non-game applications, HTN planning appears in domains such as web service composition, human-assistance systems that plan/repair/execute/explain, and robotics task planning integrated with acting and (in some lines) with motion planning. ⁴

In agentic execution platforms (tool-using agents, long-horizon workflows, coding agents), both HTN and GOAP require explicit treatment of (i) partial observability and stale state, (ii) nondeterministic outcomes and expensive actions, (iii) side effects and idempotency, (iv) persistence of execution traces and artifact lineage. The relevant mature bodies of work are (a) planning under uncertainty (e.g., FOND and contingent planning) and its hierarchical extensions, and (b) workflow/durable execution patterns (event histories, idempotency, sagas, provenance standards). ⁵

Terminology and concept glossary

This glossary is scoped to the specific deliverables requested, and uses established formal planning conventions.

World state (state)

A representation of the environment at a time point used for planning and execution. In STRIPS-style planning, a state is often represented as a set (conjunction) of ground atoms/literals (facts) taken as true; actions transform one state into another via add/delete (or more general) effects. ⁶

Action / operator (primitive action)

A directly executable step with: - **Preconditions:** conditions that must hold in the current state for the action to be applicable. - **Effects:** how the action changes the state (classically as add/delete effects; often extended with conditional effects, numeric/temporal effects, etc., in richer languages). ⁷

Plan (classical plan)

A sequence or partial order of actions that, when executed from an initial state (according to the action semantics), results in a state satisfying a goal condition. ⁸

Goal condition

A logical condition over states that must be satisfied by a plan's final state (or by a policy's terminal states in nondeterministic settings). ⁹

Execution trace

A record of executed actions (and often observed states/observations) with timestamps/outcomes; used for monitoring, debugging, and repair. In workflow systems, analogous constructs include event histories used for recovery and replay. ¹⁰

HTN-specific terms**Task**

An activity to be accomplished. Tasks are usually represented syntactically like predicates/atoms with arguments (parameters), similar to actions, but differ semantically depending on whether they are primitive or compound. ¹¹

Primitive task

A task that corresponds to a directly executable action/operator (with state-transition semantics). ¹²

Compound (abstract) task

A task that is not directly executable and must be refined/decomposed into subtasks using a method. ¹¹

Task network

A (typically partially ordered) set of task occurrences plus constraints (e.g., ordering constraints). Many HTN formalisms work with partially ordered task networks; some restrict to totally ordered networks for tractability or tooling reasons. ¹³

Method

A decomposition rule that can replace a compound task (or a task occurrence in a task network) with a sub-network of tasks, optionally guarded by method preconditions/constraints. ¹¹

Decomposition (refinement)

The operation of applying a method to a compound task, thereby introducing its subtasks and constraints into the current task network (and removing or expanding the refined task). ¹⁴

HTN plan (solution concept)

A plan is typically a sequence (or a partially ordered set) of primitive tasks/actions that can be obtained via repeated decomposition/refinement of the initial task network and that is executable from the initial state. Exact solution definitions vary across HTN variants and restrictions, and directly affect decidability/complexity. ¹⁵

Replanning / repair in HTN

Replanning is generating a new plan from an updated state; HTN repair must also account for hierarchical constraints and for already executed actions/tasks. Work in HTN plan repair notes that naive discarding/replanning can be nontrivial because the solution criteria constrain how execution history fits the remaining task network. ¹⁶

GOAP-specific terms

GOAP world state representation

In GOAP as described for game AI, the current world state is represented as a conjunction (set) of literals/facts (or equivalently as assignments to variables describing the world). ¹⁷

Goal selection

A mechanism that chooses which goal to pursue given current world state and competing objectives. In GOAP game AI, goals are often prioritized or weighted (e.g., “insistence”/priority) and then the planner generates a plan for the selected goal. ¹⁸

Action modeling in GOAP

Actions are modeled with preconditions and effects (often STRIPS-like), plus a cost used for least-cost planning. Some GOAP implementations also incorporate “procedural” preconditions/effects for runtime practicality. ¹⁹

Heuristic choice

GOAP planning frequently uses A* or related search methods where the heuristic estimates remaining cost-to-go, analogous to pathfinding. This is a transfer of standard heuristic search theory to the action-planning graph. ²⁰

World state progression

Applying an action to a state yields a successor state via the action's effects (classically: remove negative effects, add positive effects). ²¹

Execution loop (plan-act-monitor-replan)

A standard GOAP control loop in games is: sense or update world state; select a goal; plan a sequence of actions; execute stepwise; replan when the plan becomes invalid or higher-priority goals arise. The motivating examples in game practice explicitly emphasize replanning when actions fail (e.g., door blocked ⇒ replan to kick door ⇒ replan to alternate entry). ²²

Historical timeline

This timeline prioritizes milestones that shaped today's engineering practice and tool ecosystems.

1970s

- STRIPS introduced a planning framework based on searching sequences of operators to transform an initial world model into one that satisfies a goal condition; it formalized operator preconditions and effects as central constructs. ²³
- *A established a formal basis for heuristic minimum-cost path search with evaluation functions of the form $f(n)=g(n)+h(n)$; GOAP's common use of A-like search for action sequences is an application of this framework to the action space rather than navigation graphs.* ²⁴

1990s

- HTN planning complexity/expressivity results formalized that the complexity of HTN plan existence varies widely depending on restrictions on task networks and methods; this period also includes formal syntax/semantics and sound/complete procedures for specific HTN fragments (e.g., UMCP). ²⁵
- Graphplan introduced planning graphs and a planning procedure that returns a shortest partial-order plan or reports no plan; it influenced later heuristic planning and planning-graph-based heuristics. ²⁶
- PDDL appeared to standardize planning domain/problem descriptions for competitions and comparative evaluation, later extended (e.g., PDDL2.1) to represent temporal/numeric properties. ²⁷

Early 2000s

- SHOP2 described an HTN planner that generates plan steps in the order they will be executed, enabling the planner to track the current state during planning and supporting expressive temporal/metric domains. ²⁸
- HTN planning was applied to semantic web service composition (composition of OWL-S services), illustrating non-game applications where hierarchical decomposition aligns with structured processes. ²⁹

Mid 2000s

- GOAP became a widely cited game AI interpretation of planning, exemplified by the "Three States and a Plan" approach for the video game F.E.A.R. ³⁰, emphasizing decoupling goals and actions, layering behaviors, and dynamic problem solving via replanning with a STRIPS-like representation and A*-style search. ³¹

2010s

- Hierarchical planning integrated with robotics task-and-motion planning, exemplified by "Hierarchical Task and Motion Planning in the Now," emphasizing aggressive hierarchical commitment to reduce search and limit long-horizon projection, and by subsequent work on integrating acting/planning/learning in hierarchical operational models. ³²
- Task insertion HTN (TIHTN) formalized a hybrid of classical planning and HTN planning by allowing insertion of actions outside the method hierarchy to address incomplete hierarchies and increase flexibility. ³³
- Behavior Trees expanded from game AI to robotics and AI as a modular control structure for task switching, with formal analysis tools and extensions to stochastic outcomes. ³⁴

2020s

- HDDL proposed a shared input language for hierarchical planning (an extension of PDDL) to enable

domain-independent HTN planning comparisons and integration, and hierarchical planning tracks emerged in the International Planning Competition using HDDL. ³⁵

- Tooling matured around hierarchical planning model authoring/validation (e.g., HDDL Parser as an IDE-integrated language server). ³⁶

- Work on HTN plan verification and plan repair expanded, including translations/compilations and parsing-based approaches for verification in (notably) totally ordered HTN fragments. ³⁷

- Planning under nondeterminism and partial observability continued to mature (FOND, contingent planning) and hierarchical analogues appeared (e.g., “strong solutions for FOND HTN” and probabilistic hierarchical goal networks). ³⁸

- Emerging research interleaves hierarchical symbolic planning with LLM-generated decompositions (e.g., ChatHTN), with explicit claims of soundness while using LLMs as approximate decomposition generators when methods are missing. ³⁹

HTN deep dive

Rigorous conceptual model

A common formal view of HTN planning defines a planning problem by: - an initial world state s_0 , - an initial task network (often containing at least one initial abstract task), - a set of primitive actions/operators with state-transition semantics, - a set of methods that decompose abstract tasks into networks of subtasks with constraints. ⁴⁰

Operationally, HTN planning is a refinement process: 1. Start from the initial task network. 2. Choose a task occurrence to refine. 3. If it is primitive: check applicability in the current state and commit it to the plan (depending on planner style). 4. If it is compound: choose an applicable method and replace/refine the task occurrence with its subtasks and constraints. 5. Continue until only primitive tasks remain and the resulting set/sequence is executable (respecting ordering constraints). ⁴¹

This operational structure is shared across many planners but parameterized by: - task network ordering (totally ordered vs partially ordered), - search strategy (forward progression vs plan-space refinement vs compilations), - whether and how states are propagated during refinement (e.g., SHOP2’s “planning in the execution order” property). ⁴²

Precise definition set (requested deliverable)

The following definitions use the standard HTN framing in formal treatments and surveys.

State

A set of facts (literals) describing the world. Primitive actions define state transition semantics via preconditions/effects. ⁴³

Task

A symbolic description of an activity. Tasks can be parameterized and appear as nodes in task networks. ⁴⁴

Primitive task

A task corresponding to an action/operator that can directly execute in the environment (subject to preconditions). ⁴⁵

Compound task

A task that is not directly executable and must be decomposed using a method. ⁴⁶

Method

A rule of the form “to achieve compound task t , if method constraints hold, replace t with subtask network N ,” where N includes task nodes and constraints (ordering, variable bindings, and often state constraints). ⁴⁷

Decomposition (refinement) step

Applying a method to expand a compound task into its subtask network, adding constraints and replacing the expanded task occurrence in the current network. ⁴⁴

Precondition / effect (in HTN)

- For primitive tasks: preconditions/effects are defined as in STRIPS-style action models. ⁴⁸
- For compound tasks/methods: formalisms vary; some include method preconditions and state constraints that must hold for a decomposition to be applicable (and surveys treat these as significant semantic parameters). ⁴⁹

Plan

A sequence or partially ordered set of primitive tasks/actions that is obtainable by repeated decomposition from the initial task network and is executable from the initial state, respecting both action applicability and task network constraints. ⁵⁰

Execution trace

A history including (at minimum) executed primitive actions and observed outcomes/states; used for monitoring, diagnosis, and repair. In HTN-specific plan repair literature, the distinction between executed prefix and remaining hierarchical structure is treated as a central issue. ⁵¹

Replanning and plan repair

- **Replanning:** compute a new plan from a new state/problem instance.
- **Plan repair:** modify an existing plan/hierarchy to accommodate execution failures or model inaccuracies. HTN plan repair work states that discarding and replanning is not “easily possible” in general HTN settings because hierarchical solution criteria require already executed actions to be taken into account, motivating repair approaches and transformations. ⁵²

Representational consequences and algorithmic tradeoffs

Expressiveness and complexity

Formal results show HTN planning complexity varies with restrictions and can be high; the AAAI-era complexity work explicitly studies how task-network conditions influence complexity. ⁵³

Hybrid variants (e.g., task insertion) are analyzed separately, including tight bounds for TIHTN. ⁵⁴

Search space shaping

HTN methods encode permissible decompositions, which can function as “search control knowledge” (explicitly recognized in work discussing SHOP2’s performance dependence on hand-designed control knowledge) and can reduce the need to discover structure that is otherwise implicit in goal-state planning. ⁵⁵

Planning style variants in HTN systems

A survey of hierarchical planning emphasizes that “one abstract idea” has multiple concrete realizations, and modern frameworks (e.g., PANDA) explicitly integrate multiple solving techniques (progression search, plan-space search, compilations) and preprocessing steps like hierarchical reachability/grounding. ⁵⁶

Standardized languages and benchmarks

HDDL was proposed to address the lack of a common input language in hierarchical planning, with explicit motivation around domain-independent planners and system comparability/integration. ⁵⁷
HTN competition tracks use HDDL as input. ⁵⁸

Major variants and extensions of HTN planning

The following list is restricted to variants explicitly formalized and/or repeatedly referenced in hierarchical planning literature/tooling.

Totally ordered vs partially ordered HTN

Totally ordered fragments are often treated as prominent in benchmarks and tooling; plan verification for totally ordered HTN has connections to parsing/context-free grammar membership, and dedicated verification approaches exploit this structure. ⁵⁹

Task Insertion HTN (TIHTN) and extensions

TIHTN allows insertion of actions outside the decomposition hierarchy, hybridizing classical planning with HTN planning and addressing incomplete hierarchies. ³³
Further extensions incorporate constraints (e.g., TIHTNS adds state constraints). ⁶⁰

Hybrid planning (HTN + causal links / POCL ideas)

Work on complexity of “hybrid planning” treats the combination of HTN-style decomposition with partial-order causal link structures as a distinct formalism and analyzes tight bounds across subclasses. ⁶¹

Goal-hierarchy formalisms related to HTN

Hierarchical Goal Network (HGN) planning decomposes goals rather than tasks and is explicitly positioned as bridging classical planning and hierarchical planning, partly to make classical heuristics easier to incorporate. ⁶²
Goal-Task Network (GTN) approaches unify goal and task decomposition with task sharing. ⁶³

Translations/compilations between HTN and classical planning

The HTN-to-PDDL translation line shows restricted HTN classes can be compiled into classical planning encodings and that “small and incomplete” HTN knowledge can improve classical planner performance when translated. ⁶⁴

More recent work continues translating HTN problems to classical planning and evaluating competitiveness.

⁶⁵

Plan repair and verification as first-class problems

HTN plan repair via model transformation aims to enable use of off-the-shelf HTN planners for repair by transforming repair into planning. ⁶⁶

HTN plan verification has dedicated lines of work including parsing-based verification and compiling verification into planning. ⁶⁷

Temporal/numeric extensions to hierarchical languages

Work proposing “HDDL 2.1” argues HDDL lacks temporal and numeric constraints needed for some “real world applications,” and proposes extending HDDL by drawing from PDDL2.1 and ANML concepts. ⁶⁸

Hierarchical planning under nondeterminism and uncertainty

Recent work claims the first approach to finding strong solutions for fully observable nondeterministic (FOND) HTN problems, using relaxations/compilations to deterministic HTN to reuse deterministic grounders/heuristics. ⁶⁹

Canonical non-game applications (examples)

Web service composition

HTN planning (using SHOP2) was applied to automatic composition of OWL-S web services; the paper argues HTN is suitable for service composition and presents implementation details. ⁷⁰

Interactive assistance with planning, repair, execution, explanation

A system for assisting home theater assembly combines planning and interaction components to generate, execute, repair, present, and explain plans, and reports an empirical evaluation. ⁷¹

Robotics planning/acting integration

Hierarchical planning integrated with task and motion planning (“planning in the now”) emphasizes top-down hierarchical commitments to reduce search and lessen long-range projection; other robotics work describes integrating planning and acting with dispatching and execution monitoring. ⁷²

Worked example A (HTN): document automation workflow

This example is intentionally “workflow-engine shaped”: the primitives correspond to external tools/services; the compound tasks represent structured process knowledge.

Domain primitives (primitive tasks/actions)

Let the state be a set of facts; primitive tasks include:

- `FetchDoc(id)`
Preconditions: `DocExists(id)`
Effects: `DocFetched(id)`
- `ExtractClaims(id)`
Preconditions: `DocFetched(id)`
Effects: `ClaimsExtracted(id)`

- `CiteSources(claims)`
Preconditions: `ClaimsExtracted(id)`
Effects: `CitationsAttached(id)`
- `RenderPDF(id)`
Preconditions: `CitationsAttached(id)`
Effects: `PdfRendered(id)`

These correspond to a STRIPS-like action semantics. ⁴³

Compound task

`ProduceCitedReport(id)`

Methods (decomposition alternatives)

- M1: `ProduceCitedReport(id)`

Preconditions/guards: `DocType(id, "web")`

Decomposes into:

`FetchDoc(id) < ExtractClaims(id) < CiteSources(id) < RenderPDF(id)`

- M2: `ProduceCitedReport(id)`

Preconditions/guards: `DocType(id, "local")`

Decomposes into:

`LoadLocalDoc(id) < ExtractClaims(id) < CiteSources(id) < RenderPDF(id)`

This illustrates HTN's "method library" as the central authoring artifact. The meaning of the plan is not merely "achieves a goal predicate," but "is reachable via permitted decompositions for `ProduceCitedReport`."

⁴⁶

Execution trace and repair points

If `CiteSources` fails due to insufficient data (a model mismatch between extraction and citation), HTN repair work frames repair relative to the remaining hierarchy and executed prefix, and discusses transforming repair problems so off-the-shelf planners can be used. ⁷³

GOAP deep dive

Rigorous conceptual model

GOAP in widely cited game AI practice is a planning architecture that: - maintains a representation of current world state as a set of facts/variables, - defines goals as desired world-state conditions (and typically priorities), - defines actions with preconditions/effects and action costs, - uses search to sequence actions that achieve the selected goal from the current state, - executes stepwise and replans when the current plan becomes invalid or better goals arise. ⁷⁴

A direct statement in the "Three States and a Plan" document describes world state as a conjunction of literals / assignment to variables. ⁷⁵

That same source explicitly frames A* as the “old friend” used after introducing action costs to guide search toward a lowest-cost sequence of actions to satisfy a goal. ⁷⁶

A separate practitioner-oriented planning chapter describes GOAP as implementing practical planning with “actions as C++ classes,” “plans as paths in a space of states,” and “search as path planning in a space of states,” with many implementations applying actions backward from goal to initial state, and notes forward search variants. ⁷⁷

Precise definition set (requested deliverable)

Goal (in GOAP)

A desired condition over world state (often a partial assignment). In game GOAP slides and writeups, goals are frequently expressed as desired world-state variables/facts. ⁷⁸

Goal selection

A method that chooses a goal based on current world state and priority/utility-like values; the Orkin account explicitly discusses goals competing and falling back to lower-priority goals when a plan cannot be formed. ⁷⁹

Action modeling

An action a is modeled by: - preconditions $\text{Pre}(a)$ (facts that must hold to apply the action), - effects $\text{Eff}(a)$ defining how the world state changes, - cost $\text{Cost}(a)$ used to compute plan cost. ⁸⁰

Heuristic choice

GOAP commonly uses heuristic graph search (A or variants). A theory provides conditions under which the heuristic yields optimal minimum-cost paths given admissibility; GOAP transfers this to the action graph. ²⁰

World state progression

Applying an action updates the state using its effects, often described in STRIPS-like add/delete terms or equivalent set operations. ⁸¹

Plan

A totally ordered sequence of actions (common in game GOAP implementations) representing a path in the state space from current state to a state satisfying the goal; a practitioner chapter explicitly calls a GOAP-like plan “a totally ordered set of actions.” ⁸²

Execution loop

A plan is executed action-by-action. When an action fails or when new information invalidates assumptions, the system replans. The Orkin scenario explicitly describes repeated replanning after failures (door won't open $\Rightarrow$ kick $\Rightarrow$ alternate entry). ²²

How GOAP differs from “pure STRIPS” in game practice

Orkin explicitly describes GOAP as inspired by STRIPS planning while making changes for real-time practicality, including adding cost per action and adding procedural preconditions/effects; the document also claims elimination of add/delete lists as an implementation difference in that specific system. ⁸³

The practitioner chapter emphasizes implementation and optimization details that are not central in academic planning papers but are architecturally relevant: representing actions in data files for designer iteration, choosing search procedures (e.g., forward BFS vs backward regression), and optimizing memory/time via compact predicate encoding and data structures. ⁸²

Major variants and extensions of GOAP

Unlike HTN, GOAP is not a single standardized formalism with a unified competition language. The “variant set” is therefore best described as recurring engineering patterns documented in game AI practice and hybrids.

Forward vs backward search

The GOAP-like planning chapter describes backward application from the goal state as a common pattern and mentions forward-search variants that appear easier to debug. ⁸⁴

Representation optimizations

Compact bitset encodings for state predicates, predicate indexing, and parameter unification optimizations are described as main levers to keep planning within real-time budgets. ⁸²

GOAP + reactive execution hybrids

Academic game AI work explicitly positions planners (including GOAP and HTN) as suited for longer-term planning and Behavior Trees for reactive acting, motivating hybrid approaches. ⁸⁵

Utility-driven goal selection

Game utility theory materials formalize scoring-based decision selection among options (utility calculations, considerations, etc.). Such systems are frequently used to pick goals or action “intent” that is then realized by planners like GOAP. ⁸⁶

Multi-agent GOAP

There are specialized studies and systems for multi-agent cooperation using GOAP-like approaches (typically focusing on coordination and performance tradeoffs). ⁸⁷

Worked example B (GOAP): tactical behavior selection with replanning

This example abstracts Orkin’s described “layering behaviors” and “dynamic problem solving” claims into a minimal GOAP model.

World state facts (subset)

HasCover, EnemyVisible, InMeleeRange, DoorBlocked, HasWindowEntry, Alive, Reloaded.

Goals (prioritized)

- G1: Survive (desired: Alive=true and optionally HasCover=true)
- G2: KillEnemy (desired: EnemyDead=true)

Orkin describes layering goals such as Cover and KillEnemy and emphasizes that the system discovers dependencies at runtime through preconditions/effects. ⁸⁸

Actions

- GotoCover

Preconditions: HasCoverLocation=true

Effects: HasCover=true

Cost: low/moderate

- AttackFromCover

Preconditions: HasCover=true $\wedge$ EnemyVisible=true $\wedge$ Reloaded=true

Effects: EnemyDead=true (simplified deterministic effect for illustration)

Cost: moderate

- MeleeAttack

Preconditions: InMeleeRange=true

Effects: EnemyDead=true

Cost: moderate

- OpenDoor

Preconditions: DoorClosed=true

Effects: DoorOpen=true

Cost: low

- KickDoor

Preconditions: DoorClosed=true

Effects: DoorOpen=true

Cost: higher

- EnterViaWindow

Preconditions: HasWindowEntry=true

Effects: InsideRoom=true

Cost: higher

Orkin's text includes the "door blocked $\Rightarrow$ kick $\Rightarrow$ window" replanning narrative, indicating execution updates working memory/world state and invokes replanning when actions fail. ²²

Execution monitoring

Upon OpenDoor failure, set DoorBlocked=true in working memory and trigger replanning; the narrative explicitly describes this as the source of dynamic problem solving. ²²

Comparative analysis

Representation, planning flow, search behavior, monitoring, authoring

Representation - HTN: primary representation is a task hierarchy plus decomposition methods that generate constrained task networks; primitive actions still require state semantics. ⁴⁶

- GOAP: primary representation is a factored world state plus actions with preconditions/effects/costs and goals as desired world-states; planners search for a minimal-cost action sequence. ⁷⁴

Planning flow - HTN: top-down refinement from abstract tasks to primitive actions via method selection; planning can interleave with acting, but the hierarchy constrains what “counts” as a solution. ⁸⁹
 - GOAP: pick a goal, then run a state-space search (often framed as pathfinding in state space) to find a plan; execute and replan as needed. ⁹⁰

Search behavior - HTN: search is over decompositions (method choices, task orderings, variable bindings), and may be structured as progression search, plan-space search, compilations, or SAT/BDD approaches; its branching is often dominated by method choice and applicable decompositions. ⁹¹
 - GOAP: search is over states/actions like classical planning; branch factor is the number of applicable actions; planning cost is sensitive to state representation and action set size; practice-focused material discusses explicit optimization for time and memory budgets. ⁹²

Execution monitoring and replanning - HTN: repair and replanning are complicated by hierarchical constraints and executed prefixes; there is explicit HTN plan repair literature and explicit integration of acting and planning over hierarchical operational models. ³
 - GOAP: game-practice descriptions explicitly trigger replanning when failures occur or new information arrives; this is typically framed as updating working memory/world state and rerunning the planner. ²²

Authoring style - HTN: authoring centers on method libraries and task decomposition structure (procedural knowledge encoding). HDDL standardization work and hierarchical planning surveys treat model engineering as a core cost/benefit axis. ⁹³
 - GOAP: authoring centers on action definitions and goal predicates; Orkin emphasizes decoupling goals and actions to avoid “embedded FSM per goal” complexity, and practitioner chapters discuss representing actions as data files to enable non-programmer iteration. ⁹⁴

Comparison matrix (requested deliverable)

The table entries are phrased as structural properties or as claims explicitly grounded in cited sources.

Axis	HTN (typical properties)	GOAP (typical properties)
Expressiveness	Can be more expressive than classical STRIPS planning; complexity depends strongly on restrictions; rich control knowledge encoded via methods. ⁹⁵	Aligns with classical planning (STRIPS-like) with costs; expressiveness depends on action language used and whether procedural effects are allowed in practice. ⁹⁶
Determinism assumptions	Baseline HTN planning formalism is deterministic in action effects; extensions exist for nondeterminism (e.g., FOND HTN). ⁶⁹	Baseline GOAP implementations in games typically assume deterministic effects or rely on replanning when reality differs; richer nondeterministic semantics generally move toward FOND/MDP-style planning. ⁹⁷

Axis	HTN (typical properties)	GOAP (typical properties)
Authoring burden	Requires explicit task hierarchy and methods; domain modeling tooling supports validation (HDDL and IDE validation tool). ⁹⁸	Requires action library with preconditions/ effects and goal definitions; practitioner guidance emphasizes data-driven action authoring and optimization for iteration. ⁹⁴
Runtime cost	Highly dependent on method branching and planner style; domain-independent heuristics and preprocessing exist in modern frameworks. ⁹⁹	Sensitive to action set size and state representation; practitioner sources describe time/memory optimization as central to practical GOAP. ¹⁰⁰
Explainability	Plan trace includes decomposition structure, enabling “why” explanations tied to chosen methods; explicit explain/repair work exists in assistance systems. ¹⁰¹	Plans are action sequences with costs; explainability typically comes from goal choice plus action chain; Orkin emphasizes “decoupled goals/actions” and working memory as shared context. ¹⁰²
Adaptability	Methods constrain allowed behavior; flexibility can be increased via TIHTN/ task insertion and via repair/resume planning variants. ¹⁰³	Adaptation via replanning is a core practice narrative; action set/goal set modularity supports behavior variation by composition. ¹⁰⁴
Partial failure handling	Dedicated HTN plan repair literature and comparisons; repair is shaped by definitions of what it means to “repair” under hierarchical constraints. ¹⁰⁵	Common pattern is detect failure → update world state → replan; this is not unique to GOAP but appears explicitly in GOAP game accounts. ²²
Concurrency support	Partially ordered HTN supports concurrency at the representation level; temporal/numeric extensions are under active discussion (HDDL 2.1 proposals). ¹⁰⁶	Most GOAP implementations treat plans as totally ordered sequences; concurrency is usually handled by separate systems (schedulers/BTs) unless extending to temporal planning. ¹⁰⁷
Observability	Hierarchical planning research includes plan verification, debugging, and tooling around model validation; execution monitoring is emphasized in acting/planning integration. ¹⁰⁸	Observability is implementation-defined; some game practice includes working memory shared between goals/actions and emphasizes debugging via decoupled structure, but lacks a standardized observability formalism. ¹⁰⁹
Fit for long-horizon agents	Designed to encode long-horizon structure via hierarchy; robotics and assistance systems explicitly use hierarchical models for extended tasks and propose interleaving acting/planning. ¹¹⁰	Designed for “practical planning” under budgets; long-horizon support scales with action modeling and search constraints; typically used for short-to-medium horizon sequences in games, with hybrids for reactivity. ¹¹¹

Relationship to adjacent planning and execution paradigms

This section frames HTN/GOAP relative to commonly used adjacent paradigms requested.

Classical planning and STRIPS-style action systems

GOAP's core modeling choices (state as set of facts, actions with preconditions/effects, planning as finding a sequence of actions to reach a goal) align with the STRIPS tradition and classical planning formulations. ⁶
As documented in game practice, GOAP is explicitly described as differing from STRIPS in certain implementation details for real-time use (e.g., action costs, procedural conditions). ¹¹²

HTN planning in the modern planning literature is often presented as “classical planning plus hierarchical decomposition knowledge,” where primitive actions still rely on STRIPS-like semantics but the solution space is constrained by allowable decompositions and task-network constraints. ¹¹³

PDDL ecosystems and planning competitions

PDDL was created to standardize planning problem descriptions for competitions and has evolved via competition-driven requirements (e.g., PDDL2.1 for temporal/numeric domains). ²⁷

HDDL extends PDDL for hierarchical planning problems and is used as a common language for hierarchical planning tracks. ¹¹⁴

The existence of dedicated model validation tooling for HDDL indicates the hierarchical planning ecosystem treats authoring correctness as a first-class engineering issue. ³⁶

Behavior Trees, utility systems, and state machines

Behavior Trees are a hierarchical control structure for switching between tasks/actions and are characterized (in robotics/game literature) as modular and reactive; formal analysis tools and stochastic extensions exist. ³⁴

Statecharts extend finite state machines with hierarchy and concurrency and were introduced as a formalism for complex reactive systems. ¹¹⁵

Utility theory approaches score options and select actions/goals via computed utilities; game industry materials document “utility theory” and design patterns for utility-based decision making. ¹¹⁶

Hybrid architectures combining planners (GOAP/STRIPS/HTN) with Behavior Trees are explicitly motivated in game AI research as a way to achieve both long-term deliberation and reactive execution. ¹¹⁷

Workflow DAGs, BPMN-style workflows, and rule engines

Workflow DAG systems represent workflows as directed acyclic graphs of tasks with dependencies (e.g., Airflow's description of DAGs as the workflow model). ¹¹⁸

BPMN standardizes graphical notation and semantics for business process modeling, including constructs for processes and collaborations. ¹¹⁹

Rule engines use pattern matching over working memory to trigger rule firings; the Rete algorithm is a canonical pattern-match algorithm for production systems. ¹²⁰

Structural relationship to HTN/GOAP - DAG workflows: primarily encode a fixed partial order (often static at runtime). GOAP/HTN are plan generators that can *synthesize* sequences/structures rather than merely execute a predefined graph, though HTN methods can be seen as generating constrained partial orders.

- BPMN: serves as a modeling/communication specification; execution semantics depend on engines. HTN resembles a *generative* hierarchical process model; GOAP resembles a *goal-seeking* plan generator.
- Rule engines: provide reactive inference/control based on working memory; GOAP and HTN can also rely on working memory/state, but they produce multi-step plans/policies rather than single-step rule firings.

Practical systems design implications, failure modes, evaluation, and bibliography

This section consolidates: practical implications, failure modes and mitigation patterns, evaluation criteria and decision matrix, recommended heuristics, open questions, and annotated bibliography. The framing is “useful for later architecture synthesis.”

Translation of game-oriented explanations into software architecture language (requested deliverable)

Game GOAP writeups use terms such as “working memory,” “goal/action decoupling,” and “replanning.” These map to standard software architecture constructs as follows:

- **World state / working memory** → a shared mutable state store (in-memory snapshot), or a derived projection from an event log; Orkin explicitly contrasts black-box goal FSMs with shared working memory accessible to goals/actions.
- **Actions** → idempotent (or explicitly compensated) side-effecting operations: API calls, tool invocations, file edits, database writes. Practical planning advice that models actions as code units (e.g., C++ classes) corresponds to encapsulating tool calls behind typed interfaces.
- **Goal selection** → a policy layer that chooses a target condition, which can be driven by priorities, utility scoring, SLA constraints, or deadlines.
- **Plan** → an executable trace (sequence) of operations; in durable execution systems, execution is recorded as event history (commands/events), enabling replay and recovery.
- **Replanning** → re-running synthesis against updated state/projections; in environments with nondeterministic tools, replanning is analogous to retry+branch logic governed by updated observations.

HTN-specific translation: - **Methods** → reusable process fragments / “procedures” with preconditions; comparable to workflow templates, runbooks, or typed subroutines, but explicitly available to a planning algorithm for refinement.

- **Compound tasks** → business-level intents or work packages; primitive tasks correspond to tool calls.
- **Task networks** → partially ordered workflow graphs produced by decomposition; align with DAG execution models when restricted to acyclic partial orders.

How HTN and GOAP change with nondeterministic, model-driven, expensive, or state-revealing actions (requested deliverable)

Nondeterministic action outcomes (fully observable)

Classical deterministic planning (including typical GOAP and baseline HTN) assumes actions have predictable effects. When actions are nondeterministic, the planning problem becomes policy synthesis rather than fixed-sequence synthesis: in FOND planning, a solution is typically a policy that prescribes actions contingent on reached states, and concepts such as “strong” and “strong cyclic” solutions capture guarantees under nondeterminism/fairness assumptions. ¹³²

Hierarchical variants exist: recent work introduces/claims approaches to strong solutions for FOND HTN problems. ⁶⁹

Partial observability and state-revealing actions

Under partial observability, planning is performed in belief space (sets/distributions of possible states) and conditional plans branch on observations/sensing actions; strong planning under partial observability is defined and solved via AND-OR search in belief space in classical literature. ¹³³

For GOAP-like systems in software tools, this corresponds to actions that reveal state only after execution (e.g., call API → learn permissions; run tests → learn failing subset). Without explicit belief modeling, the common engineering response is to replan after observation; this is analogous to online contingent planning methods that select useful sensing actions and interleave planning with execution. ¹³⁴

Expensive actions

When actions are expensive (latency, cost, risk), both HTN and GOAP require cost models that match operational objectives. GOAP already centers cost as a planner input. ¹³⁵

For HTN planning, “acting and planning” integration work frames decision steps as selecting among refinement/acting choices, optionally using a planner for near-optimal advice under a utility function. ¹³⁶

In workflow systems, expensive actions also implicate retry policy and idempotency; safe retries depend on idempotent APIs or deduplication keys. ¹³⁷

Model-driven actions and model mismatch

Both approaches assume a model of action effects. Model mismatch arises when the modeled preconditions/effects diverge from reality; HTN plan repair work treats this as a central cause of execution failure and contrasts repair vs replanning. ¹³⁸

In GOAP game practice, the response is typically to replan using updated working memory. ²²

Telemetry, durable state, and artifact lineage in hierarchical planning (requested deliverable)

Telemetry as state estimation

Planning depends on a state representation; in dynamic environments, state is inferred from telemetry/events. Durable execution systems record event histories and replay workflow code deterministically to reconstruct state; this is a concrete mechanism for maintaining a consistent execution trace over long horizons. ¹³⁹

Durable state as the substrate for replanning/repair

HTN plan repair and acting/planning integration assume access to execution history and a current state estimate; durable execution event histories supply this history. ¹⁴⁰

Artifact lineage

For document and code workflows, outputs (artifacts) must be traceable to inputs, transformations, and responsible agents. PROV-DM defines a provenance data model using entities, activities, and agents, intended for interoperable interchange of provenance records. ¹⁴¹

Extending PROV with workflow structure has been explicitly studied, indicating direct relevance to recorded multi-step processes. ¹⁴²

Interaction with hierarchical planning

Hierarchical planning produces structured execution traces: decomposition choices, method applications, and primitive execution. Capturing these as provenance “activities” linked to produced “entities” (artifacts) yields an audit-friendly lineage that supports debugging and postmortems and can be used by repair algorithms that require knowledge of what has been executed. ¹⁴³

Failure modes and mitigation patterns (requested deliverable)

The failure modes below are framed as categories; mitigation patterns are stated as design constraints or mechanisms and linked to relevant mature sources.

Stale world state - Failure mode: planning is performed on a stale or inconsistent state snapshot; action applicability/effects assumptions fail at runtime.

- Mitigations: - Maintain an explicit execution trace/event history and reconstruct state from it (durable execution/replay). ¹³⁹
- Use execution monitoring and plan repair rather than assuming single-shot planning; HTN plan repair literature treats model inaccuracies and execution failures as a central case. ⁷³
- For partial observability domains, model sensing actions / belief updates when guarantees are needed.

¹⁴⁴

Over-decomposition (HTN-specific risk) - Failure mode: method libraries encode excessive micro-structure, increasing authoring burden and brittleness to change; also increases the surface area for mismatches.

- Mitigations: - Introduce task insertion or hybridization when hierarchies are incomplete or too rigid, acknowledging that this changes the allowed solution space. ¹⁴⁵
- Use tooling and validation for hierarchical models (HDDL + HDDL Parser) to reduce structural errors (e.g., cycles, invalid constructs). ¹⁴⁶

Search blowup - Failure mode: combinatorial explosion from many actions (GOAP) or many method choices/parameterizations (HTN).

- Mitigations: - GOAP: use representation/indexing optimizations (bitsets, predicate indexing), and constrain action set sizes; practitioner materials treat time/memory profiling and optimization as required steps. ⁸²
- HTN: use hierarchical reachability/grounding preprocessing (e.g., task decomposition graphs) and heuristic guidance in modern frameworks. ¹⁴⁷
- Use translations/compilations when advantageous (HTN→PDDL) to leverage classical heuristics/search.

⁶⁴

Hidden side effects - Failure mode: actions have side effects not reflected in the planner’s state model; repeated execution (due to retries) causes duplication or corruption.

- Mitigations: - Enforce idempotent operation design for retried actions (idempotency keys / server-side

deduplication); AWS guidance explicitly frames idempotent APIs as a mitigation for undesirable retry side effects. ¹³⁷

- Use saga-style compensation for long-lived multi-step processes where atomic rollback is not feasible; the saga notion decomposes a long-lived transaction into a sequence of transactions with compensating steps. ¹⁴⁸

Poor authoring ergonomics - Failure mode: models become hard to evolve; errors are hard to detect; coupling to a planner/framework is high.

- Mitigations: - Adopt standardized languages and validation toolchains where possible (HDDL, HDDL Parser). ¹⁴⁹

- For GOAP-like systems, keep actions data-driven to enable iteration without recompilation, as described in practical planning chapters. ⁷⁷

Nondeterministic tools - Failure mode: action outcomes vary; plans fail or partially complete; naive retry loops induce cascades.

- Mitigations: - Formalize outcome uncertainty when guarantees matter (FOND planning / contingent planning); otherwise explicitly adopt replan-on-observation semantics and monitor for loops. ¹⁵⁰

- Use bounded retries and explicit retry policies with idempotency safeguards. ¹³⁷

Partial completion - Failure mode: some actions succeed and commit side effects before later failure; state becomes “in between plans.”

- Mitigations: - Use saga-style compensation and “repair vs replan” strategies; HTN plan repair work explicitly separates repair from restarting planning, and saga work provides the compensating-transaction structure. ¹⁵¹

- Maintain an append-only event history and reconstruct projections; event sourcing empirical studies characterize benefits and challenges (including rebuilding projections and evolution). ¹⁵²

Observability gaps - Failure mode: insufficient visibility into why the planner chose a plan, why execution diverged, or what state was assumed.

- Mitigations: - Record both planning-time artifacts (selected goal/task, decomposition choices, heuristic/cost evaluations where feasible) and execution-time events; workflow systems explicitly record command/event histories for replay. ¹⁵³

- Use provenance standards (PROV) to tie artifacts to generation activities/agents. ¹⁵⁴

Evaluation framework and decision matrix (requested deliverable)

This framework is phrased as evaluation criteria against which HTN, GOAP, or hybrids can be assessed for an agentic execution platform.

Correctness and guarantees - Deterministic correctness: does the planner’s model correspond to actual action semantics? (model mismatch rate; failure rate). ¹⁵⁵

- Nondeterministic guarantees: if needed, is the system solving a policy problem (FOND/contingent), and which guarantee class is targeted (strong vs strong cyclic)? ¹⁵⁶

Modeling and authoring cost - Time to add a new action/tool and validate it; availability of static validation (types, cycles, missing symbols). ¹⁵⁷

- Time to add/modify task decompositions and assess reachability/grounding impact (HTN). ¹⁵⁸

Runtime performance - Median and tail latency for planning; memory footprint; sensitivity to branching factor and action set size; explicit profiling/tracing strategy. 159

Operational resilience - Idempotency, retries, deduplication, compensation (saga). 160
- Ability to resume after crash using durable state/event history. 139

Observability and audit - Presence of explicit execution trace with replay; provenance metadata for artifacts. 161
- Ability to explain plan choices; HTN assistance systems include plan explanation as a delivered feature. 162

Change management - Model evolution impact: how often planning models change; whether old executions must remain reproducible; event-sourced systems report schema evolution as a major challenge and develop tactics (versioning, upcasting, etc.). 163

Decision matrix (condensed)

- Prefer HTN when: operational procedures are constrained (compliance/runbooks), hierarchical structure is stable and reusable, and explanation/repair require mapping failures back to structured intent. 164
- Prefer GOAP when: the domain is best modeled as modular actions with costs and the desired sequences should be discovered by search from current state to goal, with frequent replanning under changing contexts. 90
- Prefer a hybrid when: long-horizon intent requires explicit structure (HTN or goal decomposition) but execution must remain reactive (BT/state-machine layer), matching the planner-vs-BT separation discussed in hybrid game AI work. 165

Recommended design heuristics (requested deliverable)

These heuristics are stated as “if-conditions” rather than prescriptions.

- If the domain requires that only certain procedural decompositions are acceptable (policy/compliance/safety), use HTN-style explicit method constraints to define the admissible plan space, and treat classical goals as insufficient to capture admissibility. 166
- If the domain is naturally described as a set of reusable tools/actions with well-defined preconditions/effects and a cost model, and acceptable solutions are any that reach the goal state, use GOAP/classical planning search as the primary synthesis mechanism. 6
- If the environment is dynamic and long-running, implement explicit plan monitoring with replan/repair triggers, and persist execution traces in a replayable history to make monitoring and recovery operationally reliable. 167
- If tools are nondeterministic or state-revealing, decide explicitly whether the platform requires policy-level guarantees (FOND/contingent) or accepts replan-on-observation behavior; the former implies policy synthesis and the latter implies robust state refresh and idempotent retries. 168
- If actions have irreversible side effects, require idempotency or compensations at the action interface; sagas provide a decomposition and compensation pattern for long-lived transactions. 169
- If hierarchical models are used, adopt shared languages and validation tooling to reduce model engineering errors and to support multi-planner portability assumptions. 149

Worked example C (hybrid): coding agent task with deterministic workflow substrate and model-driven steps

This example demonstrates why a hybrid is frequently superior for modern tool-using agents that must mix deterministic steps (build/test) with model-driven steps (issue interpretation).

Problem

“Fix repository bug described in an issue; produce a verified patch.”

This problem class is used as an evaluation target in SWE-bench, which frames real-world software engineering as a challenging sustained testbed for language-model-based agents. ¹⁷⁰

Hybrid architecture (conceptual)

- High-level intent decomposition (HTN-like): `FixIssue(issue_id)` decomposes into `Reproduce`, `Localize`, `Patch`, `Validate`, `Submit`.
- Within `Localize` and `Patch`, use GOAP-like planning to select sequences of tool actions (search, open file, edit, run unit tests) that satisfy subgoals such as “failing test identified” or “all tests pass.”
- Execute via a deterministic workflow runtime with durable event history (for crash recovery and reproducibility). ¹⁷¹

GOAP-like action set (subset)

- `SearchRepo(query)` → effects: `SearchResultsAvailable`
- `OpenFile(path)` → effects: `FileOpen(path)`
- `EditFile(path, diff)` → effects: `FileModified(path)`
- `RunTests()` → effects: `TestResult(pass|fail)` (nondeterministic/time-varying in practice due to environment)
- `ApplyFix()` → effects: `IssueResolved` (validated by tests)

This aligns with GOAP’s “plans as paths in state space” concept; state predicates are derived from tool outputs rather than physical sensors. ¹⁷²

Execution trace and lineage

Store each tool invocation and output as an event; store produced artifacts (patches, logs) with provenance links (entity/activity/agent) per PROV concepts. ¹⁶¹

Failure handling

- If `RunTests()` fails after `EditFile`, the workflow is partially complete; use compensation (revert) or branch to `Localize` again; retried calls must be idempotent or deduplicated. ¹⁷³

Open questions and unresolved tensions (requested deliverable)

Learning and maintaining hierarchical models

Hierarchical planning workshops explicitly list automated learning/synthesis of hierarchical models and use of generative AI/LLMs for hierarchical planning/modeling as current topics, indicating this remains an open research area rather than settled engineering practice. ¹⁷⁴

LLM integration with hierarchical planning

ChatHTN and related work (including extensions/learning work built on ChatHTN) propose interleaving symbolic HTN planning with LLM-generated decompositions when methods are missing, with stated soundness properties for produced plans. This is an emerging line that shifts some authoring burden from manual method design to LLM-assisted decomposition while retaining symbolic validation. ¹⁷⁵

Semantic gap between world models and execution reality

Both GOAP and HTN abstractions rely on modeled preconditions/effects; plan repair work, acting/planning integration, and provenance/event-history practices address symptoms (repair, monitoring, traceability) but do not eliminate the mismatch problem. ¹⁷⁶

Guarantees under nondeterminism

FOND and partial observability planning provide formal solution concepts (strong/strong cyclic, conditional plans) but integrating these guarantees into practical multi-tool agent stacks introduces cost and modeling complexity; hierarchical extensions are an active research area rather than standardized practice. ¹⁷⁷

Concurrency semantics

HTN naturally represents partial orders; GOAP implementations are typically sequential. Extending hierarchical planning languages with explicit temporal/numeric constraints is actively discussed, indicating a gap between baseline HDDL and operational concurrency needs in some domains. ¹⁷⁸

Annotated bibliography (requested deliverable)

Entries are grouped by theme and each annotation states the specific contribution relevant to HTN/GOAP system architecture.

Foundations: classical planning and search - Automated Planning ¹⁷⁹ (theory/practice text referenced widely in planning curricula and later work): establishes classical planning models, algorithms, and extensions; serves as a general reference context for both GOAP (as classical planning adaptation) and HTN (as a distinct formalism). ¹⁸⁰

- STRIPS original paper: defines operators, preconditions/effects, and goal-based plan search framework. ²³
- A* original formulation: defines heuristic minimum-cost path search foundations used by GOAP-style planning. ¹⁸¹
- Graphplan: introduces planning graphs and shortest partial-order plan generation. ²⁶
- PDDL manual + PDDL2.1: define standardized planning domain/problem languages and temporal/numeric semantics. ²⁷

HTN core and ecosystem - HTN planning complexity/expressivity: formal complexity results for HTN planning under varying restrictions; foundational for understanding theoretical limits and fragment selection. ⁵³

- UMCP sound/complete procedure for HTNs: formal syntax/semantics and algorithmic foundations for specific HTN fragments. ¹⁸²
- SHOP2: HTN planner that plans in execution order and supports temporal/metric domains; influential in HTN practice narratives. ¹⁸³
- Hierarchical planning survey: taxonomy of hierarchical planning realizations; useful for system designers to enumerate variant semantics and solver approaches. ¹⁸⁴

- PANDA framework: modern domain-independent hierarchical planning framework integrating preprocessing and multiple solving approaches. ¹⁸⁵
- HDDL language proposal: motivates and defines a common hierarchical planning input language based on PDDL for comparability/integration. ⁵⁷
- HDDL Parser demo/tooling: IDE-integrated validation tooling for HDDL models; indicates maturation of model authoring infrastructure. ³⁶
- TIHTN and bounds: formalizes task insertion as hybrid planning and analyzes complexity. ³³
- HTN plan repair (model transformation) and comparative plan repair analyses: formalize repair problems and compare algorithm definitions/behaviors. ¹⁸⁶
- HTN verification: compilation-based verification and parsing-based approaches for totally ordered fragments. ¹⁸⁷
- HTN with nondeterminism: strong solutions for FOND HTN and related probabilistic hierarchical work. ¹⁸⁸

GOAP and game AI planning practice - Orkin, “Three States and a Plan”: describes GOAP for F.E.A.R. and presents the decoupling goals/actions, layering behaviors, and replanning narratives; includes explicit world-state-as-literals framing and A*-style search framing for action planning. ¹⁸⁹

- Practical GOAP optimization chapter: documents GOAP-like planning as “actions as C++ classes,” “plans as paths in states,” backward vs forward search choices, and implementation-level optimization concerns. ¹⁹⁰
- Hybrid planner + Behavior Trees paper (game context): motivates mixing planners (including GOAP/HTN) for long-term deliberation with Behavior Trees for reactive execution. ¹⁹¹

Planning, acting, and workflow-oriented reliability - Acting/planning integration with hierarchical operational models: defines an integrated acting and planning system using the same hierarchical operational models and a reactive acting engine, relevant to runtime execution monitoring and decision steps. ¹³⁶

- FOND planning and probabilistic planning languages: define nondeterministic/probabilistic planning representations and solution concepts (policies, strong cyclic). ¹⁹²
- Saga pattern: long-lived transaction decomposition into subtransactions and compensations; relevant to partial completion and rollback/compensation in multi-step execution. ¹⁴⁸
- Idempotent APIs for safe retries: architecture guidance on mitigating retry side effects by idempotent operations. ¹³⁷
- PROV-DM and workflow provenance extensions: provenance representation for entities/activities/agents and extensions capturing workflow structure. ¹⁹³
- Durable execution workflows with event histories: durable workflow execution semantics with deterministic replay and event history records. ¹³⁹

Emerging adaptations to LLM agent systems - ReAct: interleaves reasoning traces and tool actions for LLM agents; relevant to plan-act interleaving and exception handling. ¹⁹⁴

- Reflexion: maintains reflective text memory across trials to improve agent behavior; relevant to learning from execution feedback rather than static planning models. ¹⁹⁵
- ChatHTN: interleaves symbolic HTN planning and LLM-generated decompositions when methods are missing, with stated soundness; represents a concrete emerging bridge between HTN authoring burden and LLM-generated structure. ¹⁹⁶
- SWE-bench: an evaluation framework for repository-level software engineering tasks that highlights long-horizon, tool-driven complexity. ¹⁷⁰

Implications For Deterministic Workflow Engines

- Systems should borrow from HTN when: acceptable executions must conform to explicit procedural constraints (runbooks, compliance, safety envelopes), and when hierarchical explanation/repair is required; hierarchical planning and plan repair literature specifically address repair under hierarchical constraints and execution-history dependence. ¹⁹⁷
- Systems should borrow from GOAP when: behavior is best specified as a modular action library with preconditions/effects/costs and goal conditions, and the system should dynamically synthesize short-to-medium action sequences through search and replanning; GOAP game practice is explicitly framed as action planning via A*-style search over world states. ⁹⁰
- Systems should avoid both (as primary mechanisms) when: the dominant need is executing already-specified dependency graphs (static DAG workflows) or reactive rule firing (rule engines) rather than synthesizing new multi-step plans; in such cases DAG/BPMN/statecharts/rule systems provide direct execution semantics without the modeling overhead of planning. ¹⁹⁸

¹ ⁶ ⁷ ⁸ ⁹ ²³ ⁴³ ⁴⁸ ⁸¹ ⁹⁶ [PDF] STRIPS: A New Approach to the Application of Theorem Proving to ...

https://ai.stanford.edu/~nilsson/OnlinePubs-Nils/PublishedPapers/strips.pdf?utm_source=chatgpt.com

² ¹² ¹³ ¹⁵ ²⁵ ⁴⁵ ⁵⁰ ⁵³ ⁹⁵ HTN Planning: Complexity and Expressivity* Kutluhan Erol ...

https://euro.ecom.cmu.edu/program/courses/tcr854/2001/readings/Hendler_Nau_AAAI-94.pdf?utm_source=chatgpt.com

³ ¹⁶ ⁵¹ ⁶⁶ ⁷³ ¹³⁸ ¹⁴⁰ ¹⁴³ ¹⁵¹ ¹⁵⁵ ¹⁷⁶ ¹⁸⁶ <https://bercher.net/publications/2020/Hoeller2020HTNRepair.pdf>

<https://bercher.net/publications/2020/Hoeller2020HTNRepair.pdf>

⁴ ²⁹ ⁷⁰ HTN Planning for Web Service Composition Using SHOP2

https://www.cs.umd.edu/~nau/papers/sirin2004htn.pdf?utm_source=chatgpt.com

⁵ <https://www.haz.ca/papers/muise-dmap15-mapasfond.pdf>

<https://www.haz.ca/papers/muise-dmap15-mapasfond.pdf>

¹⁰ ¹³⁹ ¹⁵³ ¹⁶¹ <https://docs.temporal.io/workflows>

<https://docs.temporal.io/workflows>

¹¹ ¹⁴ ⁴⁰ ⁴¹ ⁴⁴ ⁴⁶ ⁴⁷ ⁸⁹ ¹¹³ ¹²⁹ ¹³⁰ ¹³¹ ¹⁶⁶ ¹⁸² ¹⁹⁷ <https://www.cs.nmsu.edu/~tson/classes/spring04-579/umcp-erol-hendler-nau.pdf>

<https://www.cs.nmsu.edu/~tson/classes/spring04-579/umcp-erol-hendler-nau.pdf>

¹⁷ ¹⁸ ¹⁹ ²² ³¹ ⁷⁴ ⁷⁵ ⁷⁶ ⁷⁹ ⁸⁰ ⁸³ ⁸⁸ ⁹⁰ ⁹⁴ ⁹⁷ ¹⁰² ¹⁰⁴ ¹⁰⁹ ¹¹² ¹²⁴ ¹²⁶ ¹²⁸ ¹³⁵ ¹⁶⁷ ¹⁸⁹ <https://www.gamedevs.org/uploads/three-states-plan-ai-of-fear.pdf>

<https://www.gamedevs.org/uploads/three-states-plan-ai-of-fear.pdf>

²⁰ ²⁴ ¹⁸¹ https://people.stfx.ca/jdelamer/courses/csci-564/_downloads/b2220c66675ddde471ca1795147b8e86/A_Formal_Basis_for_the_Heuristic_Determination_of_Minimum_Cost_Paths.pdf

[A_Formal_Basis_for_the_Heuristic_Determination_of_Minimum_Cost_Paths.pdf](https://people.stfx.ca/jdelamer/courses/csci-564/_downloads/b2220c66675ddde471ca1795147b8e86/A_Formal_Basis_for_the_Heuristic_Determination_of_Minimum_Cost_Paths.pdf)

https://people.stfx.ca/jdelamer/courses/csci-564/_downloads/b2220c66675ddde471ca1795147b8e86/A_Formal_Basis_for_the_Heuristic_Determination_of_Minimum_Cost_Paths.pdf

[A_Formal_Basis_for_the_Heuristic_Determination_of_Minimum_Cost_Paths.pdf](https://people.stfx.ca/jdelamer/courses/csci-564/_downloads/b2220c66675ddde471ca1795147b8e86/A_Formal_Basis_for_the_Heuristic_Determination_of_Minimum_Cost_Paths.pdf)

21 77 82 84 92 100 107 111 125 127 159 172 190 [https://www.gameaiapro.com/GameAIPro2/](https://www.gameaiapro.com/GameAIPro2/GameAIPro2_Chapter13_Optimizing_Practical_Planning_for_Game_AI.pdf)
[GameAIPro2_Chapter13_Optimizing_Practical_Planning_for_Game_AI.pdf](https://www.gameaiapro.com/GameAIPro2/GameAIPro2_Chapter13_Optimizing_Practical_Planning_for_Game_AI.pdf)
https://www.gameaiapro.com/GameAIPro2/GameAIPro2_Chapter13_Optimizing_Practical_Planning_for_Game_AI.pdf

26 <https://home.ttic.edu/~avrim/Papers/graphplan.pdf>
<https://home.ttic.edu/~avrim/Papers/graphplan.pdf>

27 PDDL | The Planning Domain Definition Language
https://www.cs.cmu.edu/~mmv/planning/readings/98aips-PDDL.pdf?utm_source=chatgpt.com

28 SHOP2: An HTN Planning System
https://www.jair.org/index.php/jair/article/download/10362/24790/19158?utm_source=chatgpt.com

30 136 171 <https://www.sciencedirect.com/science/article/pii/S0004370221000746>
<https://www.sciencedirect.com/science/article/pii/S0004370221000746>

32 72 110 <https://people.csail.mit.edu/lpk/papers/hpnICRA11Final.pdf>
<https://people.csail.mit.edu/lpk/papers/hpnICRA11Final.pdf>

33 54 103 145 179 Tight Bounds for HTN Planning with Task Insertion
https://www.ijcai.org/Proceedings/15/Papers/215.pdf?utm_source=chatgpt.com

34 Behavior Trees in Robotics and AI: An Introduction
https://arxiv.org/abs/1709.00084?utm_source=chatgpt.com

35 57 93 98 114 149 [https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.090/Publikationen/2020/](https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.090/Publikationen/2020/Hoeller2020HDDL.pdf)
[Hoeller2020HDDL.pdf](https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.090/Publikationen/2020/Hoeller2020HDDL.pdf)
https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.090/Publikationen/2020/Hoeller2020HDDL.pdf

36 146 157 https://icaps25.icaps-conference.org/program/demos-pdfs/ICAPS25-Demo_paper_2.pdf
https://icaps25.icaps-conference.org/program/demos-pdfs/ICAPS25-Demo_paper_2.pdf

37 108 187 <https://bercher.net/publications/2022/Hoeller2022VerificationViaCompilation.pdf>
<https://bercher.net/publications/2022/Hoeller2022VerificationViaCompilation.pdf>

38 132 150 156 168 177 192 <https://arxiv.org/pdf/2312.11675>
<https://arxiv.org/pdf/2312.11675>

39 175 196 <https://arxiv.org/abs/2505.11814>
<https://arxiv.org/abs/2505.11814>

42 183 SHOP2: An HTN Planning System
https://arxiv.org/pdf/1106.4869?utm_source=chatgpt.com

49 59 67 <https://staff.fnwi.uva.nl/g.behnke/papers/Lin2022CYK.pdf>
<https://staff.fnwi.uva.nl/g.behnke/papers/Lin2022CYK.pdf>

52 [https://bibbase.org/network/publication/hller-bercher-behnke-biundo-](https://bibbase.org/network/publication/hller-bercher-behnke-biundo-htnplanrepairviamodeltransformation-2020)
[htnplanrepairviamodeltransformation-2020](https://bibbase.org/network/publication/hller-bercher-behnke-biundo-htnplanrepairviamodeltransformation-2020)
<https://bibbase.org/network/publication/hller-bercher-behnke-biundo-htnplanrepairviamodeltransformation-2020>

55 An Admissible HTN Planning Heuristic
https://www.ijcai.org/proceedings/2017/0068.pdf?utm_source=chatgpt.com

56 91 184 A Survey on Hierarchical Planning – One Abstract Idea, Many ...
https://bercher.net/publications/2019/Bercher2019HierarchicalPlanningSurvey.pdf?utm_source=chatgpt.com

58 <https://ipc2023-htn.github.io/>

<https://ipc2023-htn.github.io/>

60 Hierarchical Task Network Planning with Task Insertion ...

https://www.ijcai.org/proceedings/2017/0623.pdf?utm_source=chatgpt.com

61 Tight Bounds for Hybrid Planning

https://www.ijcai.org/proceedings/2022/0638.pdf?utm_source=chatgpt.com

62 Relating Task and Goal Decomposition with Task Sharing

https://www.ijcai.org/Proceedings/16/Papers/429.pdf?utm_source=chatgpt.com

63 [PDF] Hierarchical Planning: Relating Task and Goal ...

https://www.semanticscholar.org/paper/Hierarchical-Planning%3A-Relating-Task-and-Goal-with-Alford-Shivashankar/3faa3ccdb00db722010c4968bde6d83758eaa814?utm_source=chatgpt.com

64 <https://www.cs.umd.edu/~nau/papers/alford2009translating.pdf>

<https://www.cs.umd.edu/~nau/papers/alford2009translating.pdf>

65 <https://cdn.aaai.org/ojs/15958/15958-40-19451-1-2-20210517.pdf>

<https://cdn.aaai.org/ojs/15958/15958-40-19451-1-2-20210517.pdf>

68 178 HDDL 2.1: Towards Defining a Formalism and a Semantics for Temporal HTN Planning

https://arxiv.org/abs/2306.07353?utm_source=chatgpt.com

69 188 Hierarchical Goal Networks for Probabilistic Planning

https://icaps25.icaps-conference.org/files/HPlan/HPlan2025_paper_1.pdf?utm_source=chatgpt.com

71 101 162 164 https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.090/Publikationen/2014/Bercher14PlanRepairExecuteExplain.pdf

https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.090/Publikationen/2014/Bercher14PlanRepairExecuteExplain.pdf

78 https://media.gdcvault.com/gdc2015/presentations/Higley_Peter_Goal-Oriented_Action_Planning.pdf

https://media.gdcvault.com/gdc2015/presentations/Higley_Peter_Goal-Oriented_Action_Planning.pdf

85 117 165 191 <https://cdn.aaai.org/ojs/13044/13044-52-16561-1-2-20201228.pdf>

<https://cdn.aaai.org/ojs/13044/13044-52-16561-1-2-20201228.pdf>

86 https://www.gameai.pro/GameAIPro/GameAIPro_Chapter09_An_Introduction_to_Utility_Theory.pdf

https://www.gameai.pro/GameAIPro/GameAIPro_Chapter09_An_Introduction_to_Utility_Theory.pdf

87 <https://ojs.aaai.org/index.php/AIIDE/article/download/18909/18674/22675>

<https://ojs.aaai.org/index.php/AIIDE/article/download/18909/18674/22675>

99 185 The PANDA Framework for Hierarchical Planning | KI

https://link.springer.com/article/10.1007/s13218-020-00699-y?utm_source=chatgpt.com

105 <https://www.cs.umd.edu/~nau/papers/goldman2024comparative.pdf>

<https://www.cs.umd.edu/~nau/papers/goldman2024comparative.pdf>

106 HDDL 2.1: Towards Defining an HTN Formalism with Time

https://arxiv.org/abs/2206.01822?utm_source=chatgpt.com

115 <https://www.state-machine.com/doc/Harel87.pdf>

<https://www.state-machine.com/doc/Harel87.pdf>

116 https://media.gdcvault.com/gdc10/slides/MarkDill_ImprovingAIUtilityTheory.pdf
https://media.gdcvault.com/gdc10/slides/MarkDill_ImprovingAIUtilityTheory.pdf

118 121 198 <https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/overview.html>
<https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/overview.html>

119 122 <https://www.omg.org/spec/BPMN/2.0/PDF/>
<https://www.omg.org/spec/BPMN/2.0/PDF/>

120 https://www.csl.sri.com/users/mwfong/public_html/Technical/RETE%20Match%20Algorithm%20-%20Forgy%20OCR.pdf
https://www.csl.sri.com/users/mwfong/public_html/Technical/RETE%20Match%20Algorithm%20-%20Forgy%20OCR.pdf

123 <https://www.sciencedirect.com/science/article/pii/S0004370282900200>
<https://www.sciencedirect.com/science/article/pii/S0004370282900200>

133 144 <https://www.sciencedirect.com/science/article/pii/S0004370206000075>
<https://www.sciencedirect.com/science/article/pii/S0004370206000075>

134 <https://mers-papers.csail.mit.edu/Conference/2014/maliah2014icaps/7913-37000-1-PB.pdf>
<https://mers-papers.csail.mit.edu/Conference/2014/maliah2014icaps/7913-37000-1-PB.pdf>

137 160 <https://aws.amazon.com/builders-library/making-retries-safe-with-idempotent-APIs/>
<https://aws.amazon.com/builders-library/making-retries-safe-with-idempotent-APIs/>

141 154 193 <https://www.w3.org/TR/prov-dm/>
<https://www.w3.org/TR/prov-dm/>

142 <https://www.usenix.org/system/files/conference/tapp13/tapp13-final3.pdf>
<https://www.usenix.org/system/files/conference/tapp13/tapp13-final3.pdf>

147 158 On Succinct Groundings of HTN Planning Problems
https://cdn.aaai.org/ojs/6529/6529-13-9754-1-10-20200518.pdf?utm_source=chatgpt.com

148 169 173 <https://www.cs.cornell.edu/andru/cs711/2002fa/reading/sagas.pdf>
<https://www.cs.cornell.edu/andru/cs711/2002fa/reading/sagas.pdf>

152 163 <https://www.sciencedirect.com/science/article/pii/S0164121221000674>
<https://www.sciencedirect.com/science/article/pii/S0164121221000674>

170 https://proceedings.iclr.cc/paper_files/paper/2024/file/edac78c3e300629acfe6cbe9ca88fb84-Paper-Conference.pdf
https://proceedings.iclr.cc/paper_files/paper/2024/file/edac78c3e300629acfe6cbe9ca88fb84-Paper-Conference.pdf

174 <https://icaps25.icaps-conference.org/program/workshops/hplan/>
<https://icaps25.icaps-conference.org/program/workshops/hplan/>

180 <https://www.sciencedirect.com/book/9781558608566/automated-planning>
<https://www.sciencedirect.com/book/9781558608566/automated-planning>

194 <https://arxiv.org/pdf/2210.03629>
<https://arxiv.org/pdf/2210.03629>

195 <https://openreview.net/pdf?id=vAElhFckW6>
<https://openreview.net/pdf?id=vAElhFckW6>