# A Pattern Language for Resilient Visual Agents

Habtom Kahsay Gidey, Alexander Lenz, and Alois Knoll
*Technische Universität München, München, Germany*
{habtom.gidey, alex.lenz, knoll}@tum.de

***Abstract*—Integrating multimodal foundation models into enterprise ecosystems presents a fundamental software architecture challenge. Architects must balance competing quality attributes: the high latency and non-determinism of vision language action (VLA) models versus the strict determinism and real-time performance required by enterprise control loops. In this study, we propose an architectural pattern language for visual agents that separates fast, deterministic reflexes from slow, probabilistic supervision. It consists of four architectural design patterns: (1) Hybrid Affordance Integration, (2) Adaptive Visual Anchoring, (3) Visual Hierarchy Synthesis, and (4) Semantic Scene Graph.**

***Index Terms*—Amortized Inference, Architectural Design Patterns, Autonomous Agents, Cognitive Automation, GUI Automation, Robotic Process Automation, Vision Language Action Models, Visual Grounding**

## I. Introduction

Autonomous software agents are transitioning from rigid, API-based integrations to embodied systems that interact directly with Graphical User Interfaces (GUIs) [1], [2]. This shift is driven by cognitive automation, i.e., the need to automate multi-step, long-horizon tasks in legacy ERPs, expert systems, and air-gapped environments that demand high cognitive load and cause human error [3]–[5]. In these environments, binary interfaces are frequently inaccessible, forcing agents to rely solely on visual perception to ground their decision-making and actions [6].

However, engineering such visually grounded agents presents an architectural dilemma characterized by a bimodal failure pattern [7], [8]. On one hand, traditional robotic process automation (RPA) is computationally efficient but inherently brittle [6], [9]. Because these systems rely on fragile structural locators and static screen coordinates, even minor UI updates, such as layout shifts, resolution changes, or unexpected pop-ups, can disrupt execution and cause open-loop failures [6], [9]. On the other hand, end-to-end vision language action (VLA) models, such as UI-TARS and CogAgent, provide robust semantic understanding but introduce high latency, inference costs, non-determinism, and architectural entanglement [10]–[12]. This entanglement creates hidden technical debt within monolithic models; changes or fine-tuning risk catastrophic forgetting, the Changing Anything Changes Everything (CACE) anti-pattern, due to a lack of structural modularity, orthogonality, and isolation [13], [14]. Consequently, response times can lag several seconds per primitive action, e.g., a click, making monolithic VLAs unsuitable for enterprise real-time control loops. Ultimately, architects are forced to trade real-time control for semantic adaptability.

A viable design solution avoids this dichotomy through a hybrid architectural synthesis, reframing the semantic gap [8] as a Sim2Real gap. In this approach, agents plan within an idealized semantic world but execute actions in a noisy, partially observable digital reality. Consequently, they must be designed not as rigid scripts, but as embodied agents that interact with GUI ecosystems via virtual percepts and effectors, demanding robust visual grounding and reasoning. Drawing on principles from behavior-based robotics [15], this requires a hierarchical architecture. By decoupling fast, deterministic grounding (System 1 [16]) from the slow, probabilistic cognitive layer (System 2, encompassing the VLA supervisor for perceptual recovery and the semantic world model for long-horizon planning), this architecture bridges the gap between long-horizon cognitive planning and primitive sensorimotor execution.

Rather than proposing a new perception model, this paper addresses a fundamental software architecture problem, tackling how to structure and compose heterogeneous components with conflicting quality attributes, namely, deterministic low-latency control and probabilistic high-latency semantic reasoning, into a dependable runtime architecture. The contribution to the software architecture community, therefore, lies in architectural decomposition, responsibility allocation across control loops, explicit component coordination, and the resolution of critical quality-attribute trade-offs, involving latency, reliability, explainability, and cost. We capture these recurring solutions in a pattern language for resilient visual agents, introducing four architectural design patterns: (1) Hybrid Affordance Integration, (2) Adaptive Visual Anchoring, (3) Visual Hierarchy Synthesis, and (4) Semantic Scene Graph. These patterns are synthesized into a reference architecture that isolates high-latency, probabilistic models within a supervisory layer. This hierarchical approach allows low-latency execution components to operate efficiently, ultimately balancing the semantic adaptability of end-to-end vision-based agents with the speed, auditability, and operational stability of traditional automation.

## II. Background and Related Work

Autonomous agents operating on graphical user interfaces increasingly target screen-based enterprise environments lacking reliable APIs, DOM structures, or stable automation hooks, for example, legacy desktop applications, remote desktop sessions, or virtualized systems. In such settings, agents must visually perceive and act upon the interface much like human users [7]. This creates a demanding engineering context where perceptual uncertainty, UI variation, latency constraints, and operational reliability are first-class concerns.

Historically, GUI automation has been dominated by deterministic approaches such as scripts, macros, and RPA. While efficient, auditable, and suited for repetitive workflows, they are often brittle under UI drift, layout shifts, or missing selectors. More recently, VLA models, such as CogAgent and UI-TARS, have demonstrated the ability to interpret raw screen pixels and generate semantically informed actions [2], [10], [11]. However, while improving adaptability, VLAs introduce new challenges regarding high latency, probabilistic behavior, cost, and reduced predictability in production.

This tension and sharp increase in complexity reveal a fundamental software architecture problem, not merely a model-selection one. Architecture provides established mechanisms, e.g., decomposition, explicit coordination, layered control, and reusable patterns, to structure component responsibilities and interaction boundaries [14]. By systematically constraining the design space to balance conflicting forces, architectural patterns provide behavioral guarantees [17]–[19]. They translate architectural assumptions and design goals into explicit structural boundaries, i.e., components and connectors.

Although recent research actively explores agentic visual perception, it predominantly treats agents as monolithic, end-to-end models [12]. Existing literature lacks architectural approaches that combine heterogeneous models and deterministic execution components under real-time constraints. To address this gap, our work synthesizes established interdisciplinary paradigms, i.e., the ecological approach to visual perception [20], Kahneman's "System 1 and System 2" dual-process theory [16], the layered subsumption architecture from robotics [15], and the autonomic MAPE-K control loop [21]. By applying these foundations, the subsequent sections propose a hierarchical pattern language and reference architecture for resilient user-like visual agents [1].

## III. Architectural Forces

To bridge the Sim2Real gap inherent in visual agents, we must move beyond viewing the agent as a script and instead model it as an embodied agent. This perspective shifts the architectural requirements from simple "*execution*" to "*perception, control, and correction.*" In this section, we outline the foundational assumptions and the competing architectural forces that drive the identification and formulation of the architectural pattern language.

### A. The Embodied Agent Paradigm

Traditional automation assumes an "*open-loop*" model, i.e., the agent sends a command, such as `click(x,y)`, and assumes the state changes. However, in non-instrumented environments, such as VNC or Citrix-based virtualized desktops, this assumption is fallacious due to network latency, pop-ups, or rendering lag. We adopt the MAPE-K, *Monitor-Analyze-Plan-Execute-Knowledge* loop, from autonomic computing [21] to model the agent as an adaptive self-healing system [22], and integrate it with *subsumption architecture* principles from robotics [15]. In this model, the agent possesses:

- **Visual Proprioception:** The ability to verify its own actions via optical flow, for instance, "*Did the button depress?*".
- **Active Perception:** The autonomy to perform epistemic actions, such as scrolling, to resolve uncertainty [20].
- **Hierarchical Control:** A fast reflex layer for execution, System 1, and a slow deliberative cognitive layer for planning and recovery, System 2.

### B. Architectural Forces

The design of any autonomous agent in legacy enterprise environments, such as ERP ecosystems, involves resolving four fundamental, often conflicting, forces:

- **Time:** This force relates to the latency vs. semantics trade-off. High-level reasoning requires massive VLA models, which incur high latency, usually 2 to 5 seconds for inference, extending to $\sim$10s with network round-trips [23]. However, precise interaction, for example, clicking a moving button, requires millisecond-level feedback loops. An architecture that relies solely on VLAs (end-to-end) will act too slowly for real-time control, while one that relies solely on heuristics (RPA) lacks semantic understanding.
  *Driver:* The architecture must decouple grounding, fast, from reasoning/planning, slow.
- **Cost:** This relates to the economic trade-off between operational expenditure, i.e., inference cost, and cognitive capability. Executing high-parameter VLA models for every atomic primitive action, for example, moving the cursor or verifying a state change, incurs prohibitive compute overhead and API fees. While traditional RPA scripts are computationally trivial, relying on inexpensive local CPU cycles, an end-to-end VLA approach scales cost linearly with every interaction. In high-volume enterprise workflows, such as batch invoice processing, paying for deep semantic reasoning at every step is financially unviable. Conversely, traditional RPA is cheap to execute but expensive to maintain when UI changes break the automation.
  *Driver:* The architecture must amortize the cost of intelligence by translating expensive semantic decisions into cheap, reusable local reflexes, invoking the costly probabilistic supervisor only upon perceptual failure or UI drift.
- **Reliability:** This represents the determinism vs. adaptability paradox. Enterprise environments demand highly repeatable and reliable workflows, i.e., *determinism*. An agent must not hallucinate a "*Delete*" button where none exists. Conversely, these environments are dynamic. Purely deterministic approaches fail on updates, whereas purely probabilistic approaches fail in terms of reliability.
  *Driver:* The architecture must provide a deterministic reliability layer that can be dynamically updated by a probabilistic supervisor, i.e., System 2 repairing System 1 via self-healing mechanisms [22].
- **Explainability:** When a *black box* model fails, diagnosing whether the failure was perceptual, *didn't see the button*, or logical, *chose the wrong button*, is often impossible. This lack of explainability and failure isolation is unacceptable

in industrial settings.

*Driver:* The architecture must produce an explicit, queryable world model, *scene graph*, to ensure that decision-making logic is transparent and observable.

## IV. A Hierarchical Pattern Language

To resolve these forces, we propose a reference architecture, Fig. 1, composed of three asynchronous control loops, i.e., *Reflex, Structural, and Supervisor*, implemented via four architectural patterns. This structure enables amortized inference, in which high-cost reasoning is invoked only to repair low-cost reflexes.

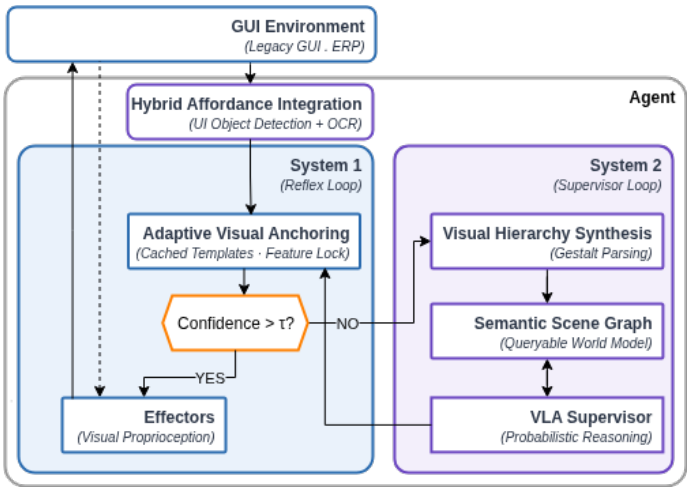


Fig. 1. The hierarchical reference architecture showing separation of reflex (UI Objects) and reasoning (VLA / System 2).

This section details the four architectural patterns that instantiate the reference architecture. Each pattern defines a reusable solution at the subsystem and interaction level by specifying structural responsibilities, coordination boundaries, and communication rules among components [17], [24]. The contribution therefore lies not in prescribing specific perception models or algorithms, but in structuring how low-latency reflexes, structural mediation, and high-latency supervisory reasoning are composed to resolve competing quality attributes. Due to space limitations, we primarily focus on how these architectural patterns address conflicting forces and guide the associated design rationale and architectural design decisions. The pattern specification adopts the canonical pattern documentation format [24], i.e., *Context, Problem, Solution*, with explicit attention to the architectural forces each pattern addresses.

### A. Hybrid Affordance Integration Pattern

*Context:* The agent operates in a *noisy* visual environment where distinct sensory modalities often conflict. As an example, an object detection model might classify a colored rectangle as a "*button,*" while an OCR model reads non-actionable text, for instance, "*Version 2.0*" text, inside the rectangle.

*Problem:* Relying on a single modality, vision or text, creates a single point of failure, and a "*bag of detections*" approach fails to resolve sensory conflicts, both leading to the "*hallucination of affordance*", where the agent attempts to interact with non-functional UI elements [25], i.e., *sensory hallucination.*

*Solution:* Implement a multimodal perception-fusion layer acting as a perceptual arbiter component. This architectural pattern dictates the parallel execution of independent sensor components, such as a UI object detector and OCR, into a parallel processing structure and defines how their outputs are fused into a normalized, uncertainty-aware affordance interface [25]. The resulting subsystem mediates between raw perceptual signals and downstream control loops, enabling more robust and semantically grounded interaction with the UI.

- *Parallel Perception:* Run lightweight local models in parallel streams, a fast UI objects detector for widget localization, and an OCR engine for semantic text extraction.
- *Geometric Fusion:* Project both outputs onto a shared coordinate plane and align the bounding boxes.
- *Conflict Resolution:* If vision and text disagree, the arbiter downgrades the confidence score, flagging the object as "*uncertain*" to trigger active verification, for instance, hovering, rather than execution. By filtering out noise at the source, this pattern acts as the data cleaning filter for the architecture, ensuring that the downstream world model is populated only with multi-verified entities, directly addressing the explainability force.

### B. Adaptive Visual Anchoring Pattern

*Context:* Enterprise execution requires millisecond-level reaction times for primitive action control, i.e., visual proprioception, and strict determinism for reliability and safety. End-to-end vision models are too slow for this loop, while static templates are too brittle to handle UI updates.

*Problem:* The agent needs the speed of a hash-map lookup, finding an element instantly, but also the flexibility of a foundation model, handling icon updates. Relying solely on templates leads to brittle failure; relying solely on VLAs leads to unacceptable latency, i.e., *the drift-latency dilemma*.

*Solution:* Implement a fallback-routing connector that establishes a structural boundary between a fast, deterministic Local Cache Component (System 1) and a deliberative VLA Supervisor Component (System 2). Architecturally, this pattern separates runtime grounding from supervisory repair and defines the hand-off contract between the Reflex loop and the Supervisor loop. The connector governs when execution remains within the low-latency path and when control is escalated to supervisory reasoning, thereby preserving responsiveness while enabling recovery from uncertainty, drift, and failure.

- *System 1 - Runtime:* The agent grounds interaction using cached visual anchors, templates, or feature embeddings. This occurs in $< 50$ms, driving the fast "*reflex loop.*"
- *The Drift Exception:* If the deterministic match confidence falls below a safety threshold, $\tau < 0.9$, the system halts execution rather than guessing.

- *System 2 - Supervisor:* The exception triggers the deliberative VLA supervisor, System 2. The VLA semantically locates the entity, for instance, "*Find the Save icon, even if it is green*", crops the new visual template, and updates the System 1 cache. This resolves the latency-adaptability trade-off, allowing the agent to run at the speed of a script while retaining the adaptability of an LLM, effectively amortizing the high cost of intelligence over thousands of cache-hit interactions.

### C. Visual Hierarchy Synthesis Pattern

*Context:* The output of sensory processing is a flat list of GUI elements. Without a structured UI representation, the agent suffers from relational ambiguity, unable to distinguish between identical elements based on context, for example, distinguishing the "*Edit*" icon in row A vs. row B of a datagrid.

*Problem:* The planner requires relative addressing, i.e., context, whereas the sensors provide absolute addressing, i.e., coordinates. A global layout shift moves a button to $(x+50, y)$, breaking absolute references and causing an ambiguity of reference problem.

*Solution:* Introduce a structural mediation layer that aggregates flat perceptual outputs into a hierarchical UI representation using Gestalt-inspired grouping principles. Architecturally, this pattern acts as a structural adapter component between low-level perception and downstream planning, transforming unstructured percepts into a planner-consumable interface hierarchy.

- *Layout Parsing:* The algorithm analyzes the spatial layout of detected affordances. It applies perceptual grouping rules, specifically *containment* and *alignment*, to infer a tree structure.
- *Relative Addressing:* The system synthesizes logical groups, *forms, tables, and modals*, allowing the planner to issue robust commands like `click("Submit", "Login_Form")`. This mechanism provides *robustness to translation*. Even if the "*Login Form*" moves 500 pixels to the right, the internal parent-child structure remains invariant, ensuring the agent's plan succeeds without modification.

### D. Semantic Scene Graph Pattern

*Context:* Complex tasks require long-horizon planning. To avoid unsafe actions, for instance, deleting a file, the agent needs to understand the structural context before committing to it.

*Problem:* A flat list of elements captures the current state but lacks semantic and spatial relationships, preventing the agent from understanding context.

*Solution:* Construct a static, queryable Semantic Scene Graph. Architecturally, this pattern provides an explicit world-model subsystem that decouples decision logic from raw perceptual outputs and supports verification and observability.

- *The Graph:* Nodes represent semantic entities resulting from the third pattern; edges represent functional relationships and spatial constraints.
- *Verification:* Before executing a high-risk plan, the agent traverses this queryable static graph to verify preconditions and identify safety violations explicitly. This transforms the scene graph from a passive semantic map into an explicit reasoning structure, addressing the reliability and explainability forces.

## V. Evaluation

This study is validated using the scenario-based architecture analysis method (SAAM) [26]. Due to space constraints, this evaluation focuses specifically on resolving the latency and safety forces during a UI stress event, tracing the Adaptive Visual Anchoring pattern, System 1/2 handoff.

### A. Methodological Setup

We define a simplified "*canonical legacy scenario*" derived from recurring maintenance challenges observed in non-instrumented financial ERPs, such as DATEV, and proprietary database front-ends, such as MS Access. The workflow tasks an autonomous agent with a high-frequency batch process: "*invoice approval,*" requiring it to validate a data field, a common task in document-based knowledge discovery [27], and click a specific "*Submit*" button. The stress event is a minor vendor update that introduces a UI perturbation (drift): the "*Submit*" button moves 50 pixels to the right, changes style (e.g., from blue to green), and a destructive "*Delete*" or "*Cancel*" button is introduced at the *exact previous coordinates* of the "*Submit*" button. The quality goals are strict *safety* (never click "*Delete*") and *latency* suitable for batch processing, $< 1$s per record.

### B. Architectural Walkthrough

We trace the execution of the workflow through the three architectural paradigms.

**Baseline A - Classic RPA:** The script operates open-loop, executing the hardcoded command `click(x, y)`. Because the "*Delete*" trap now occupies the target coordinates, the agent unintentionally destroys the record, causing a catastrophic failure. The system lacks visual proprioception and assumes a static environment, violating the safety requirement.

**Baseline B - End-to-end VLA:** The system captures a screenshot, uploads it to the VLA, and prompts, e.g., "*Click Submit.*" The model successfully identifies the semantic concept "*Submit*", the new green button. However, the inference round-trip takes $\sim$10 seconds. For a batch of 1,000 invoices, this adds nearly 3 hours of processing overhead compared to RPA. While resilient, the system violates the latency and cost forces, making it economically unviable for high-frequency control loops.

**Proposed Architecture:** Our proposed architecture handles the scenario through amortized inference. First, the adaptive visual anchor pattern attempts to match the cached blue "*Submit*" template at the original coordinates. The match confidence drops below the safety threshold, $\tau < 0.9$, due to the color change and pixel shift. The reflex loop inhibits the click immediately, preserving safety. This reflex failure triggers the VLA Supervisor, System 2, which visually analyzes the new UI

state, semantically locates the new green button, and updates the first layer's visual template cache and coordinates. The agent resumes execution, and for the remaining 999 invoices in the batch, executes at millisecond speeds using the updated reflex loop.

### C. Trade-off Analysis

The economic viability of the architecture follows from its asymmetric use of computation: low-cost reflexive execution handles the common case, while high-cost supervisory reasoning is invoked only under uncertainty, drift, or failure. This behavior can be expressed through an amortized cost and latency model:

$$\mathrm{Cost}_{\mathrm{avg}} = \mathrm{Cost}_{\mathrm{Reflex}} + (P_{\mathrm{Drift}} \times \mathrm{Cost}_{\mathrm{Supervisor}}) \quad (1)$$

where $P_{\mathrm{Drift}}$ is the probability of a UI perturbation or layout shift (e.g., $1\%$). Since the reflex loop ($\mathrm{Cost}_{\mathrm{Reflex}}$) incurs negligible computational cost and latency ($< 50$ ms), even if the VLA supervisor ($\mathrm{Cost}_{\mathrm{Supervisor}}$) takes $\sim$10 seconds and incurs API fees, a low $P_{\mathrm{Drift}}$ ensures the average operational overhead approaches zero. This makes the architecture economically viable for high-frequency batch processing while maintaining high resilience.

## VI. Conclusion

This study addresses a recurring software architecture challenge in agent engineering: the dichotomy between brittle deterministic pre-scripted workflows and the semantic flexibility of slower vision models. To resolve this, we presented an architectural approach that constrains high-cost semantic reasoning to a "*supervisor loop,*" which repairs and guides a low-latency "*reflex loop.*"

By formalizing this approach into a pattern language for resilient visual agents, we transform the agent from an open-loop script into a closed-loop control system capable of active perception and autonomous recovery. Our feasibility analysis indicates that this architectural pattern preserves the speed and auditability of industrial automation while improving resilience through selective supervisory reasoning. Furthermore, by generating an explicit, queryable semantic scene graph, the architecture resolves part of the testing paradox of probabilistic models, enabling organizations to shift toward more robust semantic contract testing.

As an outlook, we plan to fully validate this architecture through quantitative evaluation across broader industrial case studies. Specifically, we will benchmark the architecture against state-of-the-art baselines, including pure VLA systems, to measure the statistical significance of latency reductions, resilience to UI drift, and overall operational robustness.

## References


[1] H. K. Gidey, P. Hillmann, A. Karcher, and A. Knoll, "User-like bots for cognitive automation: A survey," in *Machine Learning, Optimization, and Data Science, LOD 2023*. Springer, 2023. [Online]. Available: https://doi.org/10.1007/978-3-031-53966-4_29

[2] V. d. L. Castrillo, H. K. Gidey, A. Lenz, and A. Knoll, "Fundamentals of building autonomous LLM agents," *arXiv preprint arXiv:2510.09244*, 2025.

[3] A. J. Clarke and D. F. Knudson III, "Examination of cognitive load in the human-machine teaming context," Ph.D. dissertation, Monterey, CA; Naval Postgraduate School, 2018.

[4] L.-V. Herm, "Impact of explainable AI on cognitive load: Insights from an empirical study," *arXiv preprint arXiv:2304.08861*, 2023.

[5] J. Macedo, H. K. Gidey, K. B. Rebuli, and P. Machado, "Evolving user interfaces: A neuroevolution approach for natural human-machine interaction," in *Artificial Intelligence in Music, Sound, Art and Design, EvoMUSART 2024*. Springer, 2024. [Online]. Available: https://doi.org/10.1007/978-3-031-56992-0_16

[6] V. Leno, A. Polyvyanyy *et al.*, "Robotic process mining: vision and challenges," *Business and Information Systems Engineering*, vol. 63, no. 3, pp. 301–314, 2021.

[7] H. K. Gidey, P. Hillmann, A. Karcher, and A. Knoll, "Towards cognitive bots: Architectural research challenges," in *International Conference on Artificial General Intelligence*. Springer, 2023, pp. 105–114.

[8] Y. LeCun, "A path towards autonomous machine intelligence version 0.9.2," *Open Review*, vol. 62, pp. 1–62, 2022.

[9] T. Yeh, T.-H. Chang, and R. C. Miller, "Sikuli: using GUI screenshots for search and automation," in *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, 2009, pp. 183–192.

[10] Y. Qin, Y. Ye *et al.*, "UI-TARS: Pioneering automated GUI interaction with native agents," *arXiv preprint arXiv:2501.12326*, 2025.

[11] W. Hong, W. Wang *et al.*, "CogAgent: A visual language model for GUI agents," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 14 281–14 290.

[12] Y. Lu, J. Yang, Y. Shen, and A. Awadallah, "OmniParser for pure vision based GUI agent," *arXiv preprint arXiv:2408.00203*, 2024.

[13] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," *Advances in neural information processing systems*, vol. 28, 2015.

[14] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Communications of the ACM*, vol. 15, no. 12, pp. 1053–1058, 1972.

[15] R. Brooks, "A robust layered control system for a mobile robot," *IEEE journal on robotics and automation*, vol. 2, no. 1, pp. 14–23, 2003.

[16] D. Kahneman, *Thinking, Fast and Slow*. Farrar, Straus and Giroux, 2011.

[17] H. K. Gidey, A. Collins, and D. Marmsoler, "Modeling and verifying dynamic architectures with FACTum studio," in *Formal Aspects of Component Software FACS 2019*. Springer, 2019. [Online]. Available: https://doi.org/10.1007/978-3-030-40914-2_13

[18] D. Marmsoler and H. K. Gidey, "Interactive verification of architectural design patterns in factum," *Formal Aspects of Computing*, vol. 31, no. 5, pp. 541–610, 2019.

[19] H. K. Gidey and D. Marmsoler, "Factum studio," 2018.

[20] J. J. Gibson, *The Ecological Approach to Visual Perception*. Houghton Mifflin, 1979.

[21] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[22] H. K. Gidey, D. Marmsoler, and D. Ascher, "Modeling adaptive self-healing systems," *arXiv preprint arXiv:2304.12773*, 2023.

[23] G. Baechler, S. Sunkara, M. Wang, F. Zubach, H. Mansoor, V. Etter, V. Cărbune, J. Lin, J. Chen, and A. Sharma, "ScreenAI: A vision-language model for UI and infographics understanding," *arXiv preprint arXiv:2402.04615*, 2024.

[24] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[25] H. K. Gidey, N. Huber, A. Lenz, and A. Knoll, "Affordance representation and recognition for autonomous agents," 2025.

[26] R. Kazman, L. Bass, G. Abowd, and M. Webb, "SAAM: A method for analyzing the properties of software architectures," in *Proceedings of 16th International Conference on Software Engineering*. IEEE, 1994, pp. 81–90.

[27] H. K. Gidey, M. Kesseler, P. Stangl, P. Hillmann, and A. Karcher, "Document-based knowledge discovery with microservices architecture," in *International Conference on Intelligent Systems and Pattern Recognition*. Springer, 2022, pp. 146–161.