DARPA-Influenced AI & Cognitive Languages/ Frameworks

Rosette(Racket-based solverlanguage)

Rosette is a*solver-aided programminglanguage* builtas an extension of Racket (a Lisp dialect). It provides language constructs for program synthesis and verification by compiling code to SMT solver constraints [1]. In practice, Rosette is **homoiconic** (inheriting Racket's code-as-data) and supports metaprogramming and symbolic execution. These features enable a form of *runtime introspection* (via Racket's meta-environment) and easy encoding of constraints, but Rosette itself does not maintain a persistent "cognitive" memory or agent identity – it's a formal-methods tool rather than an autonomous agent platform. Compared to Jue, Rosette shares homoiconicity and introspective capabilities, but lacks Jue's built-in persistent memory substrate or self-modifying agent behaviors. Rosette has been used in DARPA's BRASS program to synthesize and verify program components [2] [1]. Its focus on correctness and formal specification (rather than learning) means it contributes to AI safety by enabling formally correct program generation, but it doesnot directly address agentic autonomy or safety in deployed AI.

SCENIC (Scenario-generation DSL)

SCENIC is a **probabilistic programming language (DSL)** for describing and generating multi-agent, spatial–temporal scenarios in simulated environments[3][4]. Its design goal is to model "worlds" with distributions over object positions, trajectories, and behaviors (e.g. traffic simulations, robotics scenarios). SCENIC is implemented as an embedded DSL (hosted in Python) that lets users specify declarative spatial and temporal constraints plus randomness[3]. Key characteristics include explicit *probabilistic modeling* of agent interactions and events, composability of scenario fragments, and integration with simulators for autonomous systems testing. Unlike Jue, SCENIC is **not homoiconic** or self-modifying – it's a domain- specific description language rather than a general-purpose programmatic agent. It does not itself run as an agentic identity or persist knowledge over time, but focuses on generating data for training/testing. In comparison to Jue, SCENIC lacks introspective or agentic features; it is purely declarative. However, it is relevant to DARPA's AI safety ethos because it helps uncover failure modes in perception and planning by systematically sampling rare or edge-case situations. SCENIC was developed and used in DARPA's BRASS/ DyAdEm projects for autonomous vehicle and robotic scenario generation[3][4], illustrating how DARPA leverages DSLs to improve robustnessofAI systems.

RADLER (CPS Architecture Language)

Radler is an **architecture-definition DSL andframework** for distributed, real-time cyber-physical systems [5] [6]. Its RADL language lets developers specify both *logical* and *physical* aspects of a multi-node system: periodic execution rates, communication topics, data types, and resource mappings[5]. Radler's design goals are verifiable, certified builds of safety-critical CPS. It uses a publish/subscribe model with guaranteed delivery, and includes formal-verification tools (e.g. contract checks, timing analysis) to ensure properties like timing safety and type correctness[5]. In contrast to Jue, Radler is not a self-modifying or introspective agent; it is a static specification language plus runtime for guaranteed scheduling. Radler is *not* homoiconic and does not embody “agentic identity” or a learning substrate. Instead, Radler provides a fully verifiable engineering stack for real-time systems. Its relevance to AI and DARPA experimentation lies in enabling provably correct system designs: DARPA’s Assured Autonomy program supported Radler development for unmanned systems, highlighting how formal DSLs underpin trustworthy autonomous platforms[5][6]. Compared to Jue, Radler sacrifices run-time flexibility for static guarantees.

MetaFAST (Proteus adaptive programming language)

MetaFAST is the name of a **new adaptive programming language** developed under DARPA’s BRASS program by Rice University (Proteus project)[7]. It is based on Apple’s Swift and introduces novel abstractions for *dynamic, resource-aware application configuration*. The language lets developers declare high-level “intent” (constrained optimization objectives) while leaving low-level configuration (data layouts, kernel choices) to the runtime. In MetaFAST, parts of the program configuration are either controlled manually or optimized autonomously with feedback loops. Technically, MetaFAST supports on-the-fly reconfiguration of software/hardware mappings and formal specification of relaxed memory models[7]. Unlike Jue, MetaFAST is not homoiconic or agentic, and it does not provide a persistent knowledge substrate. It is a statically-typed compiled language (swift-based), so it lacks Jue’s reflexive self-modification. However, MetaFAST does embody *dynamic reconfiguration* of running systems via feedback control, which is a form of online adaptation (though not full self-programming). Its relevance to AI safety comes from the ability to enforce performance and correctness constraints (e.g. maintaining stability under resource changes) automatically[7]. DARPA’s interest was in making deployed systems (e.g. embedded vision, signal processing) resilient to hardware/environment change, which aligns with the safe/autonomous operation goals that Jue also targets.

SPIRAL / HCO (DSL for mathematical kernels)

SPIRAL is a **domain-specific meta-programming system** for numeric computation, funded partly by DARPA[8]. It provides a layered specification language for mathematical transforms (FFT, linear algebra, etc.), which it compiles into highly optimized C/FPGA code. At its core is the **Hybrid Control Operator (HCO) language**, a DSL for succinctly expressing linear algebra operations and loop nests[8]. The SPIRAL approach is to capture the *algorithmic specification* (e.g. matrix multiply) in a high-level DSL and automatically regenerate efficient code when the hardware or requirements change[8]. This gives “write-once, run-anywhere” agility: at runtime or deployment, SPIRAL re-optimizes to new architectures. SPIRAL/HCO is *not* homoiconic or introspective like Jue; it is batch-oriented and specialized to numeric kernels. There is no notion of agent identity or persistent cognition. But SPIRAL does maintain an internal graph of rewrite rules and transformations (a kind of computation graph) to generate code. In comparison with Jue, SPIRAL focuses on optimizing fixed computation rather than adaptive cognition. Its relevance is in building future-proof software: under BRASS, SPIRAL demonstrated dynamic adaptation of deployed DSP code to new hardware[8]. This “future-proof kernel generation” aligns with DARPA’s emphasis on resilient systems, though SPIRAL is more about performance portability than AI safety per se.

PWND² Language (Hidden-Network DSL)

In late 2024 DARPA announced PWND², a program to develop a new domain-specific language (DSL) for designing "hidden" or covert networks with formal guarantees[9]. The goal is to give a rigorous, mathematically precise way to describe and verify properties of anonymizing communication systems (like Tor or steganographic overlays). Although still in conception, the envisioned PWND² DSL would allow analysts to define *weird network* architectures and automatically analyze trade-offs between performance and security[9]. This language would be *formal* (with well-defined semantics) rather than a general- purpose coding language. It differs sharply from Jue: PWND²'s DSL is about network configurations, not agents. It likely won't have introspection or self-modifying code, but instead would compile to analysis models. However, it does embody a kind of *distributed graph* concept (since networks are graphs of nodes) with guarantees. The relevance to AI is indirect; it's about secure communication for autonomy. But as a DARPA experimentation, it shows how DARPA uses DSLs and formal methods for robust systems. Compared to Jue's cognitive substrate, the PWND² DSL addresses trust and privacy of [9] networked communications, another facetof system assurance.

SOTER & Breach (Runtime Assurance/Monitoring frameworks)

SeveralDARPA projects produced **runtime frameworks** for ensuringsafety of adaptive systems. For example, SRI's **SOTER** framework (from the DyAdEm program) provides *runtime assurance* for cyber-physical systems[6]. SOTER sits alongside a running system to monitor its behavior and enforce safety properties (via contracts or monitors). Similarly, the **Breach** toolbox is a runtime monitoring and falsification tool for temporal logic properties[4]. Breach can continuously check Signal-Temporal-Logic specifications against the system's outputs. These frameworks are not programming languages per se, but they embody DARPA's use of formal methods at run-time. Neither SOTER nor Breach has features like introspection or homoiconicity in the Jue sense; they are verification/monitoring systems. However, they provide *automatic, formal checking* of system behavior, which contributes directly to AI safety (by catching anomalies or unsafe states on-the-fly). In Jue terms, they offer assurance as the agent runs, whereas Jue aims to *embed* introspection within the agent itself. Both SOTER and Breach reflect the DARPA ethos of integrating provable safety into autonomous software. [4]

SSR (Pattern-based Code Update DSL)

DARPA'sBRASSprogramalso yielded **SSR(Software Search and Replace)**, a code-transformation toolkit for automated migration[10]. SSR uses a *pattern-based DSL* for edit rules that specify how API calls should be changed. In practice, developers write SSR rules (with type annotations) to match old APIs and generate new code automatically[10]. This DSL approach to code updates parallels Jue's idea of self-modification, except SSR is static and domain-specific (it doesn't adapt at runtime). SSR does embody a form of self- alteration: it can rewrite millions of lines across legacy code safely. Compared to Jue, SSR's "self-modification" is offline and rule-driven, not an autonomous agent rewriting itself. Nevertheless, SSR reflects DARPA's interest in resilient software evolution. While SSR's safety focus is on software correctness during migration (not AI per se), the underlying idea of formalized, automated code change is relevant to building robust, updatable AI systems[10].

Sources: Descriptions drawn from DARPA program materials (BRASS, DyAdEm, PWND², etc.) and project

sites. Rosette[1] [2] and Scenic[3] [4] are from published language documentation; Radler[5] [6], MetaFAST[7], SPIRAL[8] and SSR/SOTER/Breach[4][10] are from DARPA program summaries. Each entry highlights how the project's language/framework goals, features and DARPA context relate to Jue's ambitions (introspection, homoiconicity, persistent knowledge, etc.) and to broader AI safety/assurance.

---

[1] Rosette: About
https://emina.github.io/rosette/

[2] [4] [6] [7] [8] [10] GitHub - darpa-brass/brass: Summary of the BRASS program
https://github.com/darpa-brass/brass

[3] The Scenic Programming Language
https://scenic-lang.org/

[5] RADLER | Automated Rapid Certification Of Software (ARCOS)
https://arcos-tools.org/index.php/tools/radler

[9] A New Kind of Hidden Networking Science | DARPA
https://www.darpa.mil/news/2024/pwnd2