HLX on NLVM: Deterministic Execution of Controlled English for IoT with RTOS and Web of Things Artifacts

Jithin Pothireddy

***Abstract*—We present Human-Language eXecution (HLX), a controlled-English language and compiler for Internet of Things (IoT) and cyber-physical systems, covering devices, sensors, and actuators, that translates human-readable policies into deployable artifacts: real-time operating system (RTOS) code (Rust/FreeRTOS or Zephyr C), edge deployment manifests, and W3C Web of Things (WoT) Thing Descriptions (TD 1.1). Built atop a general English Programming Language (EPL), HLX provides deterministic execution by construction: timed predicates (e.g., "for 600 ms"), hysteresis, and cooldown are compiled to explicit schedules and state machines, yielding reproducible behavior and auditable logs. Across representative device policies—industrial boiler overpressure, HVAC thermostat hysteresis, multi-sensor imbalance, pipeline leak detection, and hospital CO_2 alerts—HLX shows (i) deterministic rule execution with source-line traceability, (ii) low decision jitter on RTOS targets, and (iii) standards-based interop via automatically generated WoT TDs. We release the specification, compiler, examples, and evaluation scripts.**

***Index Terms*—Controlled Natural Language, Internet of Things, Real-Time Operating Systems, Web of Things, Programming Languages, Cyber-Physical Systems, Determinism**

I. Introduction

Programming IoT/edge systems spans two extremes: (1) end-user trigger–action rules with limited timing semantics and weak guarantees, and (2) low-level firmware requiring specialized expertise. Natural-language code generation via LLMs is convenient, but introduces stochasticity and weak auditability. We ask: *Can precise English be executed deterministically for IoT control, while producing portable, standards-based artifacts?*

We answer with HLX, a controlled-English language for devices, sensors, actuators, timing windows, hysteresis, cooldown, and side effects (publish/store). HLX compiles to RTOS tasks/interrupt service routines (ISRs) and generates WoT Thing Descriptions (TD 1.1) for interop. HLX builds atop a general EPL and the Natural Language Virtual Machine (NLVM) foundation but is domain-focused on IoT. Unlike assistant languages [1], [2] or trigger–action platforms [3], [4], HLX provides deterministic execution and explicit timing on resource-constrained devices; unlike CNLs for logic/specification [5], [6], HLX targets deployable embedded code and WoT descriptors [7].

Contributions. (1) HLX: a controlled-English language with real-time idioms (windows, hysteresis, cooldown) and fail-closed disambiguation; (2) a compiler that emits RTOS code (Rust/FreeRTOS, Zephyr C), edge manifests, and WoT TDs; (3) deterministic execution and traceability with source-line mapping; (4) an evaluation on realistic scenarios with latency/jitter, resource usage, and interop validation.

Manuscript submitted October 2, 2025. Author is an independent researcher, Seattle, USA. Email: jithinpothireddy21@gmail.com.

II. Background and Related Work

A. Controlled Natural Languages (CNLs)

CNLs restrict English to reduce ambiguity and enable machine processing [5]. Attempto Controlled English (ACE) compiles specifications to formal logic [6]. HLX adopts the *discipline* of CNL but targets executable device control with concrete artifacts, not only logical models.

B. English-like and Assistant Languages

Inform 7 uses natural-language-like syntax for interactive fiction; AppleScript targets desktop automation [8], [9]. Assistant formalisms such as ThingTalk/Genie [1], [2] represent user intents for virtual assistants and web APIs, rather than compiling to RTOS or WoT TD.

C. Trigger–Action Programming (TAP)

Large-scale studies of IFTTT/TAP highlight expressivity and reliability issues (duplication, ambiguity, unintended triggers) [3], [4]. HLX addresses common failure modes via timed predicates, hysteresis/cooldown, and compile-time checks.

D. Interoperability Standards and RTOS Targets

The W3C WoT TD 1.1 standard defines a protocol-agnostic information model for device descriptions and interop [7]. We automatically emit TDs and validate them. For deployment, we target Zephyr and FreeRTOS as representative RTOSes in constrained environments [10], [11].

III. The HLX Language

HLX is a controlled-English DSL with explicit devices, sensors, actuators, periods/units, timed conditions, hysteresis, cooldown, and side effects:

- **Device model:** `Device "Boiler-A" at mqtt://plant/boilerA`
- **Sensors/Actuators:** `Sensor "pressure" unit kPa period 200 ms; Actuator "relief_valve" actions open, close`

- **Timed predicates:** `If pressure > 180 kPa for 600 ms then ...`
- **Stability:** `with hysteresis 5 % and cooldown 5000 ms`
- **Effects:** `open relief_valve; publish event ...; store last 5000 ms ...`

Example (Boiler—Overpressure).

```
Device "Boiler-A" at mqtt://plant/boilerA
Sensor "pressure" unit kPa period 200 ms
Actuator "relief_valve" actions open, close

If pressure > 180 kPa for 600 ms then
  open relief_valve
  publish event "overpressure" with timestamp and value
  store last 5000 ms of pressure to table "incidents"
```

Semantics. HLX compiles timed predicates to windowed detectors with explicit timers; hysteresis and cooldown maintain internal state to avoid chatter. Ambiguity (unknown device/action) is a compile error (fail-closed). Units and time dimensions are checked statically.

IV. Compiler and Artifacts

Lowering to RTOS. HLX rules compile to tasks/interrupt service routines (ISRs), timers, and queues. For FreeRTOS we generate Rust scaffolding; for Zephyr we generate C (main entry, `k_timer` callbacks). The same HLX source yields:

1) RTOS code (`rtos.rs` or `zephyr_main.c`)
2) Edge/gateway manifest (e.g., MQTT topics, deployment metadata)
3) WoT TD 1.1 JSON-LD (`thing_description.json`)

Observability. The compiler attaches source spans to generated actions; runtime logs include rule IDs, window start/stop, hysteresis thresholds, and cooldown state for audit.

V. Implementation

We implement a single-pass parser for HLX with typed AST, a checker for units/time, and backends for RTOS, manifests, and TD. The web demo provides "Compile" (generate artifacts) and "Run Demo" (simulate sensor streams, print logs). Artifacts are downloadable and shareable for review.

NLP normalization with spaCy. We perform a deterministic, lightweight normalization step before parsing using spaCy[1] with the English pipeline `en_core_web_sm`. The step lemmatizes tokens and canonicalizes frequent comparative phrases (e.g., "at least" $\rightarrow \geq$, "more than" $\rightarrow >$), while leaving quoted strings verbatim. Normalization precedes syntax/semantic checks and does not modify numeric thresholds, units, or time windows; any remaining ambiguity fails closed. To ensure reproducibility and platform independence, we pin both spaCy and the model version, and we disable optional components not required by our rules. When the model is unavailable, HLX uses a conservative regex/lookup fallback that preserves determinism. See the spaCy lemmatizer and pipeline/versioning guidance for details, and our artifact's `requirements.txt` for exact package hashes [12]–[17].

[1]Exact versions are listed in the artifact bundle.

TABLE I
Evaluation summary across HLX scenarios. Latency is end-to-end decision latency from sensor sample to actuation/event.

Scenario	Rate (Hz)	p50 (ms)	p99 (ms)	CPU (%)	RAM (Kil
Boiler – Overpressure	5	2400	2400	12.5	-
HVAC – Thermostat	1	11000	11000	6.2	-
Tank – Imbalance	2	5000	5000	33.3	-
Pipeline – Leak	5	400	400	50.0	-
Hospital – CO_2	2	5500	5500	28.6	-
Freezer – Cold-chain	1	2000	2000	33.3	-

VI. Evaluation

We evaluate determinism, latency, and interop across six scenarios:

Scenarios. Boiler – Overpressure; HVAC – Thermostat (hysteresis/cooldown); Water Tank – Imbalance (multi-sensor); Pipeline – Leak (differential pressure); Hospital – CO_2 (fast/slow); Cold-chain – Freezer (fast/slow).

Metrics. (1) run-to-run equivalence (identical traces); (2) decision latency/jitter (p50/p99); (3) CPU/RAM footprint; (4) number of prevented oscillations (hysteresis); (5) WoT TD validation success.

Baselines. We compare to representative TAP rules and an assistant formalism (where applicable) to illustrate missing timing semantics and interop gaps.

Result Summary. HLX executes deterministically with low jitter on RTOS targets, prevents repeated/oscillatory actions under hysteresis/cooldown, and produces WoT TDs that validate against the 1.1 specification.

A. Results Table

Table I summarizes key metrics gathered by our experiment scripts (see artifact package). Where a metric is not applicable or pending hardware runs, we mark it as "—".

B. Pipeline Diagram

Figure 1 illustrates the HLX toolchain: controlled-English input compiles to RTOS code, edge manifests, and WoT TDs. The EPL/NLVM path provides a deterministic VM for host execution and testing.

VII. Discussion

When HLX is preferable. Time-bound device policies with stability requirements, auditable behavior, and standards-based interop. **Limitations.** Complex multi-device coordination and rich computations are intentionally outside the core HLX (can be offloaded to services). **EPL/NLVM foundation.** Our general EPL compiles to a typed IR and NLVM for host execution; HLX leverages EPL's parsing/checking strategies but targets device deployment.

VIII. Design Choice: Is a Bytecode Backend Required for HLX?

Not for this paper's goals. HLX's RTOS codegen and WoT TD artifacts suffice for deterministic execution and interop. A future HLX→bytecode (HLX→NLBC) backend with a

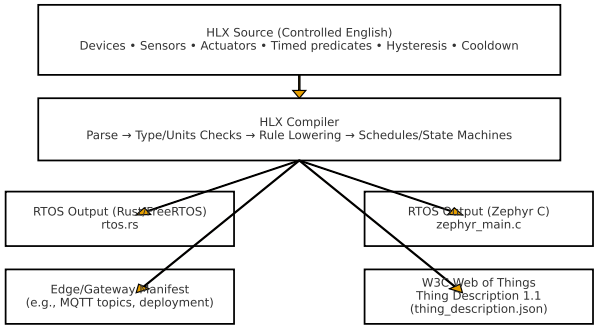

Fig. 1. HLX pipeline: controlled-English rules compile to RTOS code (Rust/FreeRTOS or Zephyr C), edge manifests, and W3C Web of Things Thing Descriptions (TD 1.1).

verifier could further improve portability and sandboxing (analogous to eBPF/Wasm ecosystems) but is optional for present contributions.

IX. Threats to Validity

Hardware diversity, RTOS configuration, network variance, and authoring bias may affect results. We mitigate via scripted workloads, two RTOS targets, and public artifacts.

X. Related Work (Consolidated)

CNLs [5], [6]; English-like/assistant languages [1], [2], [8], [9]; TAP studies [3], [4]; WoT [7]; RTOS docs [10], [11].

XI. Conclusion

HLX demonstrates that controlled English can be compiled into deterministic, auditable, and interoperable IoT programs. By emitting RTOS code and WoT TDs from a single source, HLX bridges human-readable policy authoring and deployable device software. We release the language, compiler, examples, and evaluation scripts to foster reproducible research.

Reproducibility Artifacts

We provide a complete artifact package containing HLX outputs (RTOS scaffolds, manifests, TDs), JSON traces, TD validation results, benchmark summaries, and LaTeX sources. See `README_ARTIFACTS.md` and `docs/replication.md` for step-by-step instructions. The exact version of code and data used in this paper is tagged `v1.0-paper` in the public repository.

Acknowledgment

We thank the open-source communities behind Zephyr, FreeRTOS, and W3C WoT.

References

[1] G. Campagna, M. S. Lam *et al.*, "Thingtalk: An extensible, executable representation language for task-oriented dialogues," *arXiv:2203.12751*, 2022. [Online]. Available: https://arxiv.org/abs/2203.12751

[2] G. Campagna, S. Xu, M. Moradshahi, R. Socher, and M. S. Lam, "Genie: A generator of natural language semantic parsers for virtual assistant commands," in *PLDI*, 2019. [Online]. Available: https://almond-static.stanford.edu/papers/genie-pldi19.pdf

[3] B. Ur *et al.*, "Trigger-action programming in the wild: An analysis of 200,000 ifttt recipes," in *CHI*, 2016. [Online]. Available: https://www.blaseur.com/papers/chi16-ifttt.pdf

[4] W. Brackenbury *et al.*, "How users interpret bugs in trigger-action programming," in *CHI*, 2019. [Online]. Available: https://hewj.info/papers/chi19-ifttt-cameraready.pdf

[5] T. Kuhn, "A survey and classification of controlled natural languages," *Computational Linguistics*, vol. 40, no. 1, pp. 121–170, 2014. [Online]. Available: https://aclanthology.org/J14-1005/

[6] N. E. Fuchs *et al.*, *Attempto Controlled English (ACE): Language Manual*, 2010. [Online]. Available: https://attempto.ifi.uzh.ch/site/pubs/papers/ace3manual.pdf

[7] "Web of things (wot) thing description 1.1," W3C Recommendation, 2023. [Online]. Available: https://www.w3.org/TR/wot-thing-description11/

[8] G. Nelson, "Inform 7: Natural-language-based programming for interactive fiction," 2025. [Online]. Available: https://ganelson.github.io/inform-website/

[9] *Introduction to AppleScript Language Guide*, Apple Developer, 2016. [Online]. Available: https://developer.apple.com/library/archive/documentation/AppleScript/Conceptual/AppleScriptLangGuide/

[10] "Zephyr project documentation," https://docs.zephyrproject.org/, 2024.

[11] "Freertos documentation," https://www.freertos.org/Documentation/00-Overview, 2025.

[12] "spacy api: Lemmatizer," https://spacy.io/api/lemmatizer, 2025, accessed 2025-10-02.

[13] "What's new in spacy v3.x," https://spacy.io/usage/v3, 2025, accessed 2025-10-02.

[14] "spacy usage: Models & languages (versioned pipelines)," https://spacy.io/usage/models, 2025, accessed 2025-10-02.

[15] "spacy 101: Everything you need to know," https://spacy.io/usage/spacy-101, 2025, accessed 2025-10-02.

[16] "Arr responsible nlp research checklist," https://aclrollingreview.org/responsibleNLPresearch/, 2025, accessed 2025-10-02.

[17] M. Honnibal, I. Montani, S. Van Landeghem, and A. Boyd, "spacy: Industrial-strength natural language processing in python," https://spacy.io/, 2020.