

Statecharts as an IoT/WSN Programming Solution

SeminnarIT 2025 - Generatiivinen tekoäly nyt ja tulevaisuudessa

Xinyu Tan

September 12, 2025



Kokkola University Consortium Chydenius



UNIVERSITY OF JYVÄSKYLÄ



1. Constrained Nodes and Challenges
2. Existing IoT Programming Approaches
3. Statechart Programming Approach
4. Evaluating Statechart Programming Approach
5. Summery



Key Challenge (RFC7228)

... The tight limits on power, memory, and processing resources (of constrained nodes) lead to hard upper bounds on state, code space, and processing cycles, making optimization of energy and network bandwidth usage a dominating consideration in all design requirements. [1]

Class 0

Data: \ll 10 KB

Code: \ll 100 KB

- No direct internet access
- Use low-power WSN protocols (e.g., IEEE 802.15.4)

Class 1

Data: \sim 10 KB

Code: \sim 100 KB

- Requires lightweight protocols (e.g., CoAP)
- Cannot support full stacks (e.g., HTTP)

Class 2

Data: \sim 50 KB

Code: \sim 250 KB

- Can run standard IP protocols
- Still highly limited vs. PCs



Programming Challenges of Constrained IoT Devices

Device & Network Constraints

- **Compatibility:** Limited CPU, memory, and power restrict the use of standard OS and protocols.
- **Interoperability:** Handling the complexity of diverse devices, platforms, and communication standards.
- **Scalability:** Managing and configuring potentially thousands of devices in a network.

Development & Maintenance Lifecycle

- **Reconfigurability:** Efficiently updating software for bug fixes, security patches, or new features after deployment.
- **Abstraction:** Hiding underlying hardware and network complexity with high-level programming interfaces (APIs).

Developer & Team Experience

- **Visualization:** Using graphical tools to simplify prototyping and understanding of system logic.
- **Collaboration:** Enabling seamless teamwork between developers from different domains (e.g., hardware, application).



Outline

1. Constrained Nodes and Challenges
2. Existing IoT Programming Approaches
3. Statechart Programming Approach
4. Evaluating Statechart Programming Approach
5. Summery



Core Idea

Provides fundamental programming abstractions and direct control over hardware, tailored to the unique constraints of sensor nodes.

Pros

- Efficient resource management
- High flexibility and performance
- Fine-grained system control

Cons

- Requires decent embedded programming expertise
- Relies on imperative languages (e.g., C, nesC)
- Difficult for high-level design and abstraction

Examples: TinyOS [2], Contiki [3], RIOT [4], Zephyr [5], [StateOS](#) [6]



Scripting Programming Approaches

Core Idea

Uses high-level languages (Python, JavaScript, Lua) to control a pre-existing system, making IoT development accessible to a broader audience.

Pro

- Simpler syntax and interactive debugging
- Comprehensive libraries
- Lowers the barrier for non-expert programmers

Cons

- Performance overhead from runtime interpretation.
- May negatively impact event responsiveness.
- Memory consumption is a significant concern.

Examples: MicroPython [7], Espruino [8] (JavaScript), NodeMCU [9] (Lua)



Model-Based Approach

Core Idea

Uses formal models (UML, XML, etc.) to design system behavior before implementation, separating design from platform-specific code.

Pros

- Separates application design from implementation.
- Allows for rapid conceptualization, simulation, and troubleshooting before coding.
- Can utilize existing tools for code generation

Cons

- Often fails to effectively address resource constraints.
- Lacks support for the dynamic reconfigurability needed in WSNs.

Examples: DSML4TinyOS [10], Paulon et al. [11], AI-based MDE approach [12]



Core Idea

Uses graphical elements (blocks, nodes) to create applications, lowering the barrier for non-expert developers.

Pros

- Intuitive and user-friendly
- Enables rapid prototyping
- Advanced solutions support team collaboration

Cons

- Can struggle with complex designs
- Advanced approaches require significant hardware resources

Examples: Node-RED [13], Ardublock [14], SmartBlock [15]



Outline

1. Constrained Nodes and Challenges
2. Existing IoT Programming Approaches
- 3. Statechart Programming Approach**
4. Evaluating Statechart Programming Approach
5. Summery



Relevant Publications

- I. Hakala and X. Tan, “A statecharts-based approach for wsn application development,” *Journal of Sensor and Actuator Networks*, vol. 9, no. 4, p. 45, 2020
- X. Tan and I. Hakala, “Stateos: A memory-efficient hybrid operating system for iot devices,” *IEEE Internet of Things Journal*, vol. 10, pp. 9523–9533, June 2023
- X. Tan and I. Hakala, “Event-driven performance evaluation of statecharts and micropython on esp32-c3 platforms,” in *21st EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, (Oslo, Norway), EAI, 2024
- X. Tan, K. Illka, and I. Hakala, “Performance evaluation of scripting languages on iot programming solutions,” in *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications 2025*, (Istanbul, Türkiye), 2025 (**Just Accepted**)
- and other under-reviewed submissions...



Why Statecharts for IoT Programming?

Core Concept

Statecharts are a visual formalism that extends finite state machines (FSMs), making them ideal for modeling the complex, concurrent, and event-driven behavior inherent in IoT applications.

Technical Advantages

- **Visual & Intuitive:** Represents complex logic with clear, understandable diagrams.
- **Naturally Event-Driven:** Seamlessly models system responses to sensor inputs, timers, and network events.
- **Handles Complexity:** Manages intricate systems through hierarchy (states within states) and concurrency.

Development & Design Benefits

- **High-Level Abstraction:** Separates application logic from low-level, platform-specific hardware code.
- **Fosters Collaboration:** Visual models are easy for multidisciplinary teams to discuss, evaluate, and verify.
- **Enables Formal Analysis:** Allows for system verification and troubleshooting before implementation, leading to more robust designs.

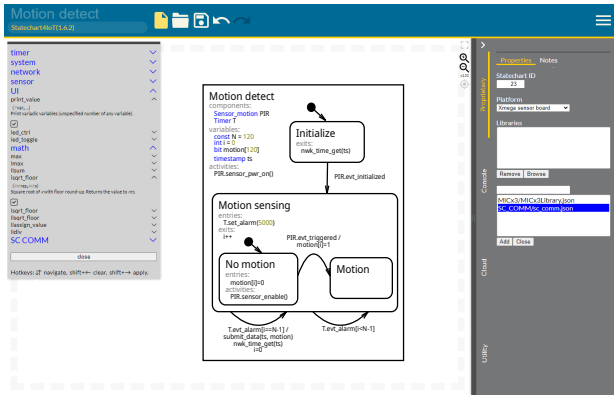


Statechart4IoT Framework

Heterogeneous sensor tasks

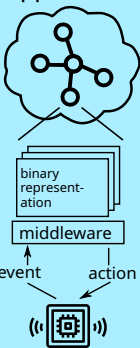


Design a visual **statechart** representation of a sensor application



Transmit the compressed binary representation of the statechart to the sensor device

Sensor application





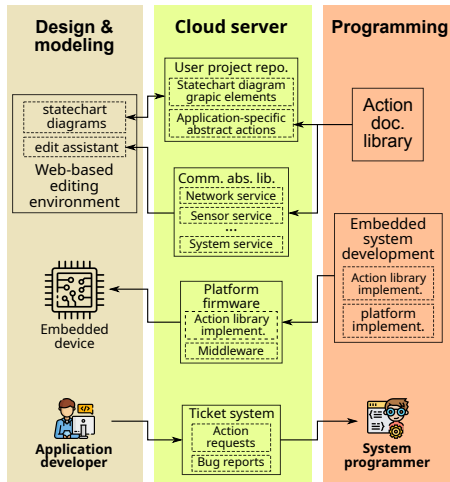
Working with Statechart4IoT

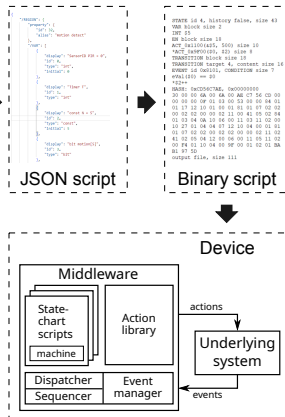
Application Developer

- **Focus:** High-level application logic and behavior.
- **Tools:** Uses the Web-based visual editor to design and model statecharts.
- **Process:** Downloads firmware and deploys statechart scripts to devices.
- **Interaction:** Submits requests for new functionality through a ticket system.

System Programmer

- **Focus:** Low-level, platform-specific implementation.
- **Tools:** Uses traditional embedded development environments.
- **Process:** Implements and documents middleware and action libraries.
- **Interaction:** Responds to tickets by creating new actions and updating the firmware.







Execution model

Statecharts are not compiled directly to machine code. Instead, they are executed by a **middleware** that runs on the device.

- The middleware translates the statecharts into event-triggered actions implemented in the underlying system.
- This extra software layer introduces performance overhead.

Performance Consideration

Given that IoT nodes have limited CPU, memory, and power:

How significant is the performance impact of this middleware-based execution?



Outline

1. Constrained Nodes and Challenges
2. Existing IoT Programming Approaches
3. Statechart Programming Approach
- 4. Evaluating Statechart Programming Approach**
5. Summery



Key Performance Factors

- Middleware design ← **Subject**
- Underlying (operating) system
- Host microcontroller

Evaluation Methods

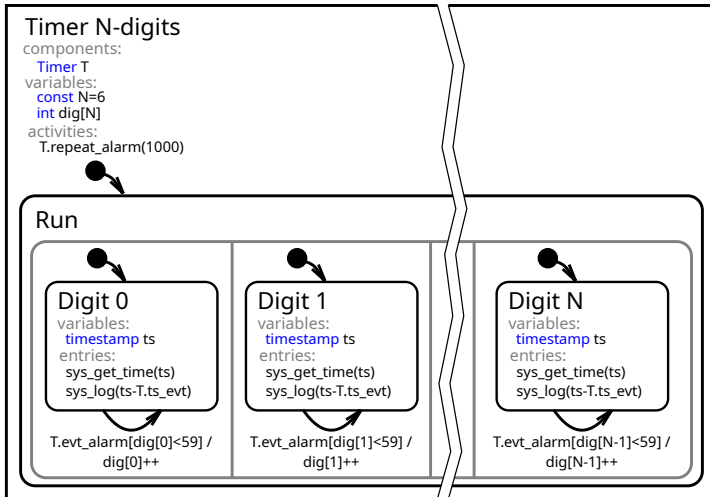
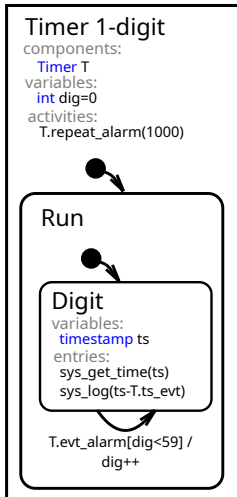
- Evaluate across platforms of different constraint classes.
- Compare upon OSes with different scheduling priorities.
- Benchmark against interpreter-based (scripting) solutions.

Evaluation Dimensions:

- **Event Responsiveness:**
How quickly can it react to internal and external events?
- **Memory Consumption:**
How much data memory does it use?
- **Scalability :**
How much the overhead increases with increasing concurrent components.



Statecharts for Evaluation



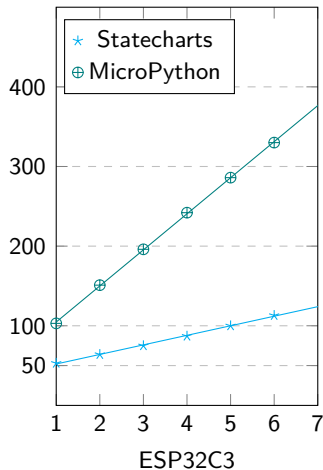
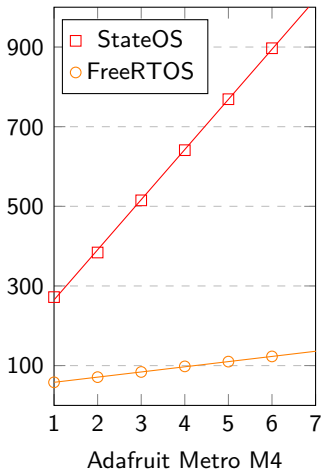
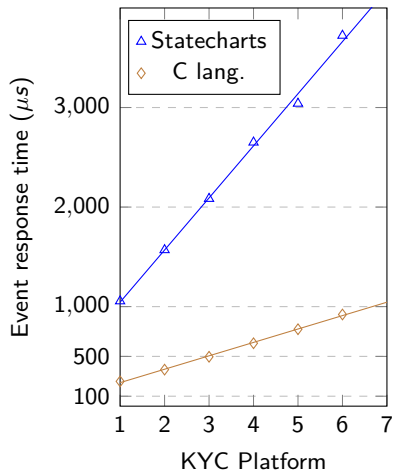


Experiment Configuration & Results in *Timer 1-digit*

Test platform	Constrained class	Main clock freq.	Language	Underlying OS	Res. in <i>Timer 1-digit</i>		
					Response time (μS)	Memory usage (bytes)	Code size (bytes)
KYC	C1	32MHz	Statechart	StateOS	1055.15	316	104
			C language	StateOS	133.8	38	648
Adafruit Metro M4	C2	120MHz	Statechart	StateOS	260.2	496	104
			Statechart	FreeRTOS	58.1	1040	104
ESP32C3	C2	160MHz	Statechart	FreeRTOS	52.7	3294	104
			MicroPython	FreeRTOS	102.4	4832	932

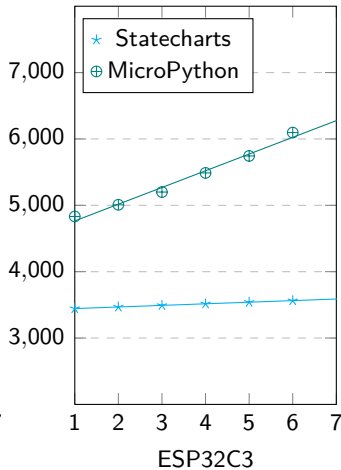
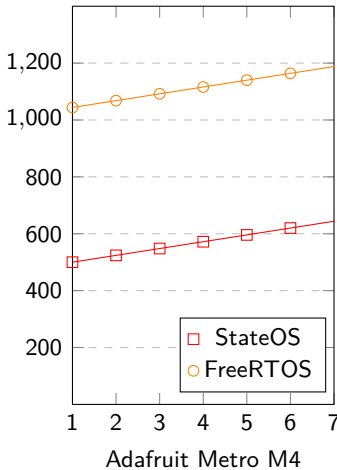
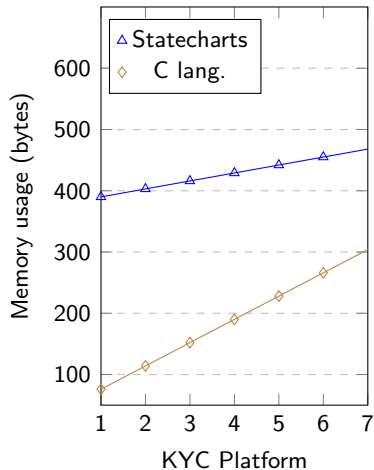


Results on *Timer N-digits 1*





Results on *Timer N-digits 2*





Outline

1. Constrained Nodes and Challenges
2. Existing IoT Programming Approaches
3. Statechart Programming Approach
4. Evaluating Statechart Programming Approach
5. Summery



How Statechart4IoT Addresses IoT Programming Challenges

Device & Network Solutions

- **Compatibility:** Uses an optimized middleware design and compact binary scripts for a minimal footprint.
- **Interoperability:** Employs abstract action libraries to hide platform-specific hardware differences.
- **Scalability:** Supports concurrent statecharts and energy-efficient remote script distribution.

Development & Lifecycle Solutions

- **Reconfigurability:** Enables post-deployment reconfiguration of statecharts without system reboots.
- **Abstraction:** Separates high-level application logic from low-level implementation.

Developer Experience Solutions

- **Visualization:** Graphic diagrams of statecharts provide a visual programming approach.
- **Collaboration:** Establishes a clear workflow separating the roles of Application Developer and System Programmer.



Key Takeaway

Statecharts are a viable IoT programming approach that presents a promising design, modeling, and programming solution for developing applications on resource-constrained IoT devices.

It empowers Developers by:

- Fostering collaboration in teamwork.
- Offering an intuitive, visual design process.
- Separating application logic from hardware complexity.





It respects the Device by:

- Minimal script footprints and memory-efficient middleware.
- Enabling dynamic, on-the-fly reconfigurability.
- Proving viable performance and memory efficiency.

Q & A Thank you!








Bibliography I

-  C. Bormann, M. Ersue, and A. Keränen, “Terminology for Constrained-Node Networks.” RFC 7228, May 2014.
-  P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, *et al.*, “Tinyos: An operating system for sensor networks,” in *Ambient intelligence*, pp. 115–148, Berlin, Heidelberg: Springer, 2005.
-  A. Dunkels, B. Gronvall, and T. Voigt, “Contiki-a lightweight and flexible operating system for tiny networked sensors,” in *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, LCN-04, (ampa, FL, USA), pp. 455–462, IEEE, IEEE (Comput. Soc.), 2004.
-  E. Baccelli, O. Hahm, M. Gunes, M. Wahlisch, and T. C. Schmidt, “Riot os: Towards an os for the internet of things,” in *2013 IEEE conference on computer communications workshops (INFOCOM WKSHPS)*, pp. 79–80, IEEE, 2013.



Bibliography II

-  Wind River Systems, “Zephyr.” <https://zephyrproject.org/>, 2016.
Accessed: August 28, 2025.
-  X. Tan and I. Hakala, “Stateos: A memory-efficient hybrid operating system for iot devices,” *IEEE Internet of Things Journal*, vol. 10, pp. 9523–9533, June 2023.
-  C. Bell, *MicroPython for the Internet of Things*.
Springer, 2017.
-  W. Gordon, “Espruino.” <https://www.espruino.com/>, 2012.
Accessed: August 28, 2025.
-  NodeMCU team, “Nodemcu.” <https://www.nodemcu.com/>.
Accessed: August 28, 2025.







Bibliography III

-  H. Marah, G. Kardas, and M. Challenger, “Model-driven round-trip engineering for tinyos-based wsn applications,” *Journal of Computer Languages*, vol. 65, p. 101051, Aug. 2021.
-  L. B. Becker, F. P. Basso, A. A. Fröhlich, and A. Paulon, “Model-driven development of wsn applications,” in *2013 III Brazilian Symposium on Computing Systems Engineering*, (Niteroi, Brazil), pp. 161–166, IEEE, IEEE, Nov. 2013.
-  N. Fredj, Y. Hadj Kacem, S. Khriji, O. Kanoun, S. Hamdi, and M. Abid, “Ai-based model driven approach for adaptive wireless sensor networks design,” *International Journal of Information Technology*, vol. 15, pp. 1871–1883, Mar. 2023.
-  OpenJS Foundation & Contributors, “Node-red.” <https://nodered.org/>, 2021. Accessed: August 28, 2025.



Bibliography IV

-  Ardublock Team, “Ardublock.” <http://blog.ardublock.com/>, 2021.
Accessed: August 28, 2025.
-  N. Bak, B.-M. Chang, and K. Choi, “Smart block: A visual programming environment for smartthings,” in *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2, pp. 32–37, IEEE, 2018.
-  I. Hakala and X. Tan, “A statecharts-based approach for wsn application development,” *Journal of Sensor and Actuator Networks*, vol. 9, no. 4, p. 45, 2020.
-  X. Tan and I. Hakala, “Event-driven performance evaluation of statecharts and micropython on esp32-c3 platforms,” in *21st EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, (Oslo, Norway), EAI, 2024.



Bibliography V

-  X. Tan, K. Illka, and I. Hakala, "Performance evaluation of scripting languages on iot programming solutions," in *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications 2025*, (Istanbul, Türkiye), 2025.