A Software Library Model for the Internet of Things

Ian McCormack* ● Dr. Jonathan Aldrich** ● Dr. Robert Iannucci** ● Kyle Liang**

Primary Goal

A library system for ultra-large-scale IoT systems
- that uses minimal storage,
- is compatible with dataflow,
- and supports **concurrent, disjoint versions.**

What should be considered?

The IoT is aimed at **ultra-large scale networks** [1] that include devices with storage capacities ranging from terabytes to **as little 100 kb!** [2]

Scale, heterogeneity, and constraints **increase opportunities for breaking changes.**

Mutable state requires **expensive synchronization** [3] in parallel environments.

Existing library systems (PiP and NPM) are **monolithic**, and **poorly support disjoint versions** of the same library [4-5].

How should users interact?

Library solutions must be **storage efficient and immutable,** but also must address the issues of IoT developers.

Library Users:

- Updates might be functional but **exceed resource limits.**

Library Maintainers:

- There is l**ess incentive** to support specific deployment scenarios.

Users should be able to **mix and match disjointly versioned components** to adapt to breaking changes by balancing compability with updated functionality.

System Design Principles

A library is **a versioned collection of imports, exports, and global constants.** These limits **prevent mutable state** from being hidden.

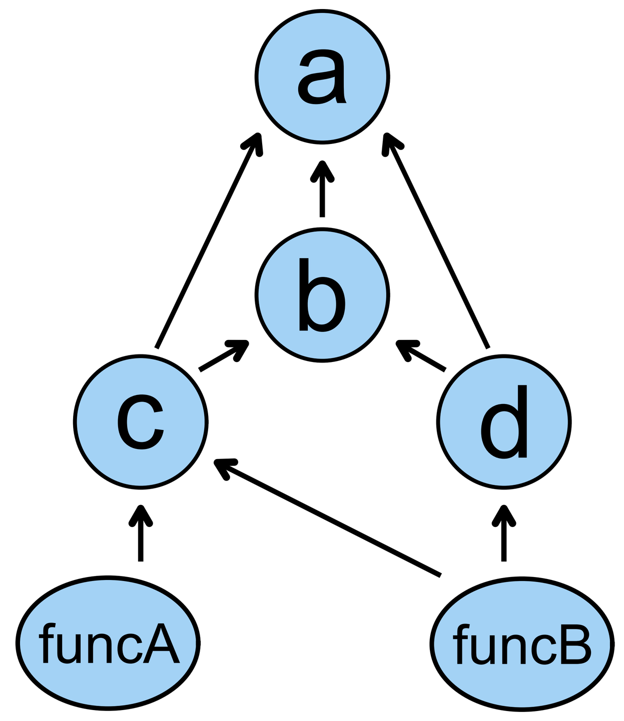
Library definitions are stored as a **set of data dependence graphs** [6-7], where each node contains an identifier, its code snippet, and a set of parent nodes.

Library functions are **imported individually at specific versions** in a program. This queries a repository where the graph is stored. The graph is traversed and code snippets are recorded and sent back.

```
library test@1.0;

const int a = 50;
const int b = 5*a;
const int c = 4*a + 3*b;
const int d = 3*a + 2*b;

export funcA() { return c };
export funcB() { return c * d };
```



A sample library definition and its graph.

This representation **increases storage** efficiency by allowing individual functions to be extracted from a library via traversal.

Library components **can be mixed and matched** at multiple versions by performing traversals on different versions of the graph.

References

[1] L. Northrop, et al. 2006. Ultra-large-scale systems: The software challenge of the future. Technical Report. Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst.

[2] C. Bormann, M. Ersue, and A. Keranen. 2014. Terminology for Constrained-Node Networks. https://tools.ietf.org/html/rfc7228#section-2.1.

[3] V. Gajinov et al. "Supporting stateful tasks in a dataflow graph," 2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT), Minneapolis, MN, 2012, pp. 435-436.

[4] 2020. Virtual Environments and Packages. https://docs.python.org/3/tutorial/venv.html. [Online; accessed 20-July-2020].

[5] 2020. npm-install.https://docs.npmjs.com/cli/install. [Online; accessed20-July-2020].

[6] D. J. Kuck et al. 1981.Dependence Graphs and Compiler Optimizations. In Proceedings of the 8th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '81). Association for Computing Machinery, NewYork, NY, USA, 207–218.https://doi.org/10.1145/567532.567555[12]

[7] K. Pingali et al. 1991. Dependence Flow Graphs: An Algebraic Approach to Program Dependencies. In Proceedings of the 18th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages(POPL '91). Association for Computing Machinery, New York, NY, USA,67–78.https://doi.org/10.1145/99583.99595[

* University of Wisconsin-Eau Claire
** Carnegie Mellon University

Carnegie Mellon University
School of Computer Science

institute for SOFTWARE RESEARCH