



cAER

Luca Longinotti <luca.longinotti@inilabs.com>

Updated 2015-11-04

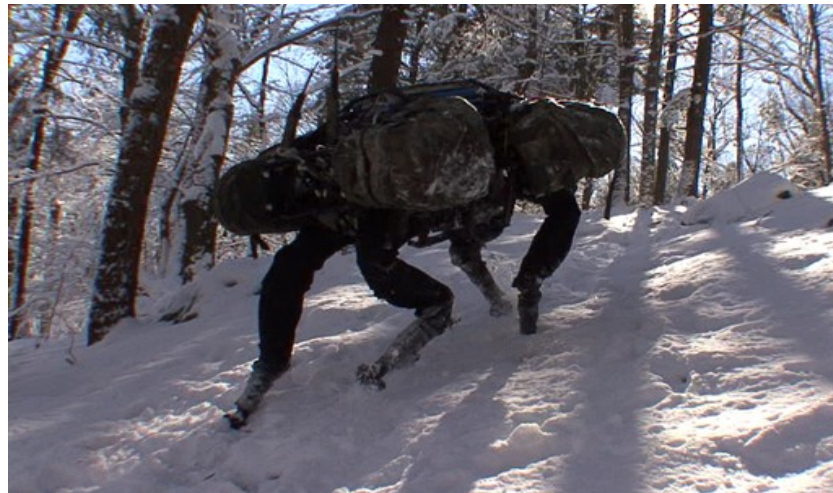


Table of Contents

- Introduction
- Architecture
- Usage
- Performance
- Demo
- (Implementation Challenges)
- Summary

cAER: what is it?

- AER: Address-Event-Representation
- New generation of efficient, low-power sensors
- Event-based data processing framework
(get, **process**, output data)
- Targets embedded systems: low power applications, robotics





cAER: why C?

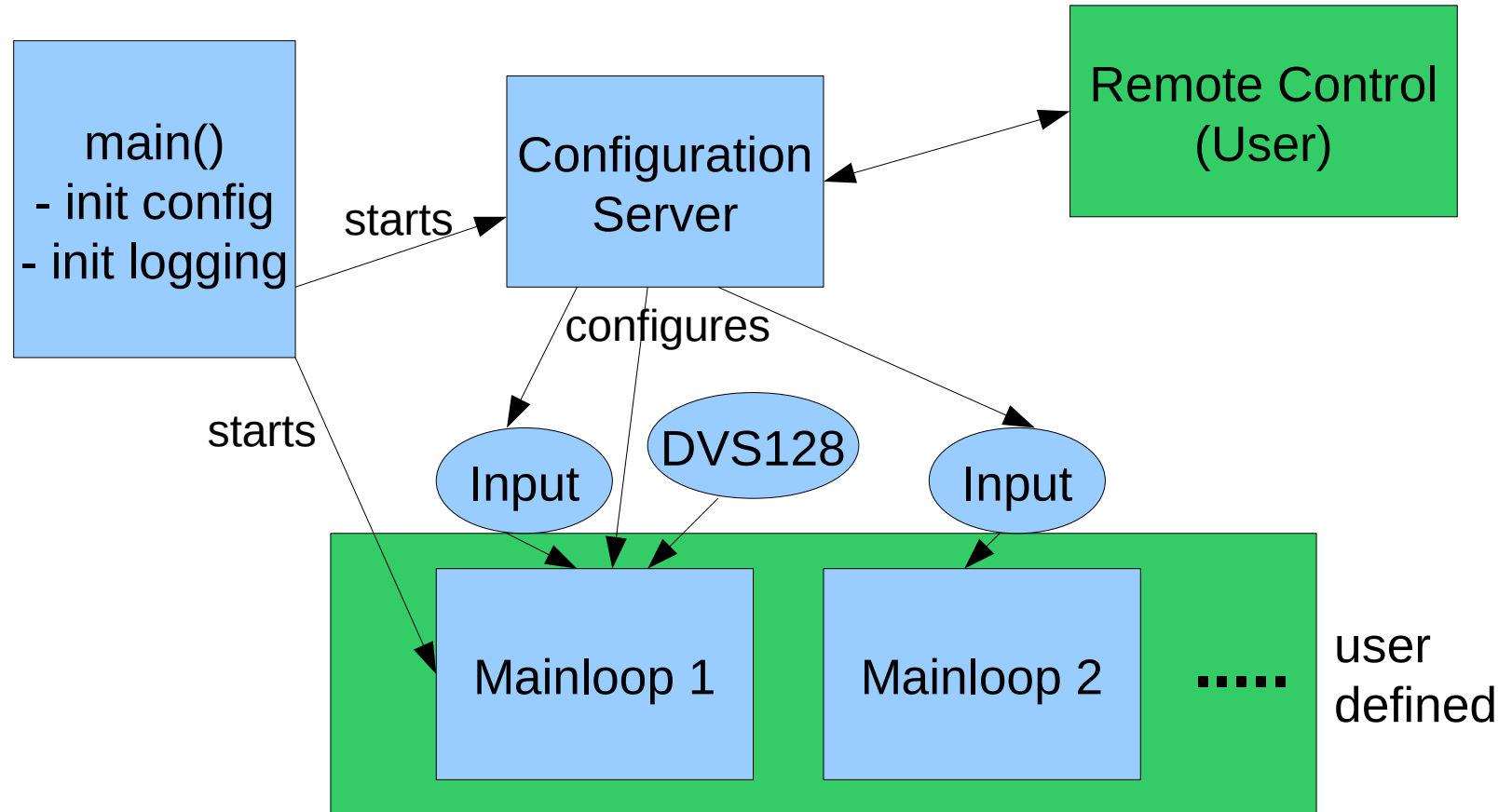
- jAER → Java AER framework (focus on ease of use, GUI)
- Goal: portability
 - Systems with no JVM (until recently: ARM)
- Goal: efficiency
 - Systems where JVM overhead too big
- Goal: complex applications
 - Access to libraries/APIs (OpenCV, OpenCL, parallelization, ...)
 - Access to hardware (servos, gyros, ...)
- Solution: the C language
 - Most widely supported, great tools, low-level

Features

- XML configuration
- Run-time configuration
- Logging (debug messages)
- Daemon mode (background process)
- **Modular structure**
- User configurable data processing loops
- Asynchronous inputs (processing runs only if data available)



Architecture



Structure

- Base: configuration management, logging, mainloop and module execution (framework)
- Events: event definitions (stand-alone headers)
→ now provided by libcaer
- Modules: data in/process/out (user-supplied)
- Utilities: caer-ctl, udpststat, tcpststat, unixststat

 base ~~events~~ ext modules utils



Event Types

- Special Event (time-stamp handling, external input)
- Polarity Event (DVS128, DAVIS)
- IMU6/IMU9 Event (Gyros)
- Frame Event (conventional APS image, DAVIS)
- Ear Event (Cochlea)
- Sample Event (ADCs, Cochlea)

Modules

- Inputs: DVS128, DAVIS FX2/FX3
- Processing: BackgroundActivityFilter
- Outputs: File, UDP (unicast), TCP (server), TCP (client), Unix sockets, Visualizer





Usage: installation

- Dependencies:
 - C compiler (C11 compliant)
 - POSIX environment (Portable Operating System Interface)
 - Cmake ≥ 2.6 (build management)
 - Mini-XML ≥ 2.7 (XML configuration parsing)
 - Libcaer $\geq 1.0.0$ (DVS128/DAVIS device access)
- Simple installation procedure:
 - 1) Get sources, install dependencies.
 - 2) cmake .
 - 3) Edit main.c
 - 4) make

Usage: main.c setup

```
int main(int argc, char *argv[]) {  
    caerConfigInit("caer-config.xml", argc, argv);  
    caerLogInit();  
    caerDaemonize(); // Optional  
    caerConfigServerStart(); // Optional (together with stop)  
    struct caer_mainloop_definition mainLoops[2] =  
        { { 1, &mainloop_1 }, { 2, &mainloop_2 } };  
    caerMainloopRun(&mainLoops, 2);  
    caerConfigServerStop(); // Optional (together with start)  
    return (EXIT_SUCCESS);  
}
```



Usage: mainloop definiton

```
static bool mainloop_1(void) {  
    caerEventPacketContainer container = caerInputDVS128(1);  
    // Typed EventPackets contain events of a certain type.  
    caerPolarityEventPacket dvs128_polarity =  
        caerEventPacketContainerGetEventPacket(container,  
        POLARITY_EVENT);  
    // Output to network via UDP.  
    caerOutputNetUDP(2, 1, dvs128_polarity);  
    return (true); // If false is returned, processing of this loop stops.  
}
```

Performance

- Tested on Raspberry Pi Model B
(700 MHz ARMv6 processor, 512 MB RAM)
- BAFilter (mean time to process an event): ~1000ns (~700 cycles)
- Binary size: ~100 KB
- Processor usage:
 - 1 DVS/1 loop: ~12%
 - 2 DVS/2 loops: ~26%
- Memory usage:
 - 1 DVS/1 loop: ~658 KB
 - 2 DVS/2 loops: ~816 KB





DEMO!


- 2 DVS, 2 loops (one each)
- BackgroundActivityFilter enabled
- Output via TCP and UDP
- Command-line programs



Challenges: inter-thread communication

- Lightweight notifications between threads
 - Config server → signal configuration change
 - Input modules → signal new data available
- Usual methods inappropriate:
 - Sockets, message queues, semaphores (syscalls!)
 - Shared memory (implicit with threads!)
 - Pthread barriers/conditions (solve another problem!)
- Solution: update a simple integer!
 - Synchronization? Locks? Expensive!
 - Atomic operations! (thanks to C11 standard)

Challenges: command line interface

- Usable command line interface requires auto-completion (TAB press)

- Several libraries to read user input and do completion:
 - GNU Readline (GPLv2 license, annoying for commercial usage)
 - Editline (completely broken, bad documentation)
 - Linenoise (minimal, simple implementation)
- Parse string, request possibilities via network, display them, execute command



Summary

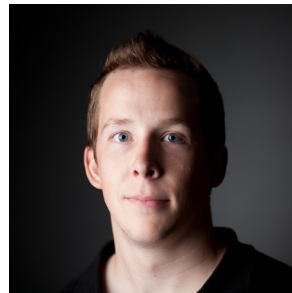
- Working, usable framework for event-based processing
- Supports currently available devices and standards

Future work

- Support for new devices
- Write other useful filters
- Authentication and encryption of communication
-

Thanks!

Tobi Delbruck



Christian Brändli



Any questions?

