RADLER

May 2017

AFM'17

Leonard Gerard, SRI

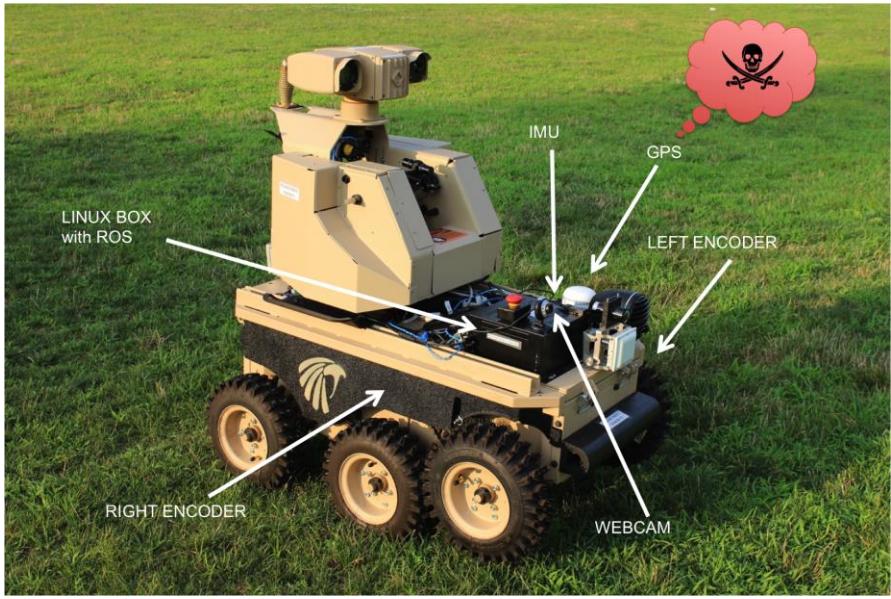
What, Why, Who

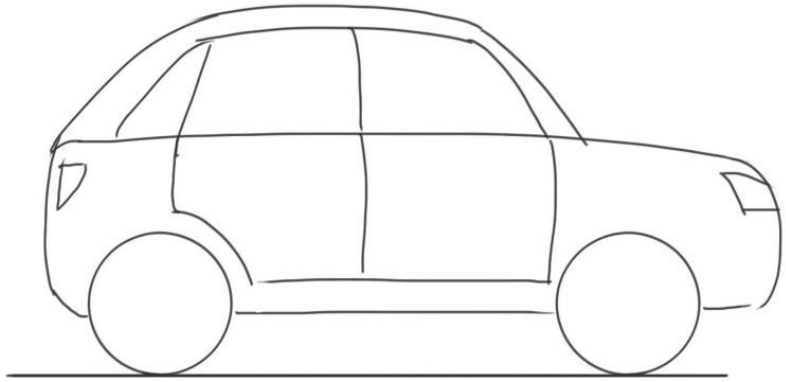
- Cyber-physical system description language
- Automatic code generation and system deployment
- Provides a computation model with time properties
  - Computations
  - Communications
- High-assurance systems
  - Safety
  - Security
- Compositional proof of complex systems
- Started at SRI for HACMS

HACMS Story

- Autonomous ground vehicle
- Safety / high assurance
  - Algorithmic
    - Obstacle detection
  - Implementation
  - Logic and compositional
    - Obstacle -> avoidance
  - System
  - Hardware, etc

- Car hacking
  - No safety without security





Goals

- Specification and integration of high assurance components
- User code is
  - System agnostic
  - Communication agnostic
- Magic creation of a system from components
- Timing requirements are expressed at the logical level, checked for constancy with the physical mapping and monitored at runtime

High Level View of Radler

- Description of a static system
  - Logical Level: formal semantics of the system
  - Physical Level: mapping and constraints
- Compilation
  - Glue code generation: ensure the semantics
  - Build and deployment: ensure the mapping
- Interface with other tools
  - PVS: check constraints and properties
  - ETB: assurance case
  - Hybrid SAL: timing properties and semantics

Logical Level

- Node (minimal description)
  - A periodic processing unit
  - User provided state and step function
  - Communication with messages
    - Publish
    - Subscribe
    - Channel maximum latency
- Topics
  - A topic name
  - Message type with fields
  - Initial value of the mailboxes

Computational Model

- Through nodes and topics, the logical description establishes a static network of computation units communicating by mailbox messages
- Restriction to single publisher per topic ensure a simple node to node communication network
- Nodes are given a fixed period
- Each period, the node step function is called with the latest received messages and its output is published

Physical Architecture

- A plant
  - Describes the mapping logical->physical
  - Information to generate glue / system code
  - Fixes the graph
    - We can check consistencies and properties
    - We can compute the runtime flags
  - Not necessary for testing things with ROS

User Provided Code (c++ version)

- State and step function
  - Gathered in a class
  - Constructor is the state initialization
  - Destructor is the state deinitialization
  - A method step
- Generated header
  - Defines the input and output structured types according to the description
  - Provides access to the radl helper macros and library

Flags

- Metadata attached to each message and mailbox
- Operational flags (expected)
  - Stale: value is outdated
    - Mailbox: no new message arrived since the previous step
    - Message: set by message producer
- Failure flags (something bad is happening)
  - Timeout: channel latencies are exceeded
    - Mailbox: no new message arrived while it should have
    - Message: set by message producer
- Propagation
  - The output flags are initialized to the OR of the input ones
  - The step function can modify the output flags

Introspection

- In the user code, access to radl file values
  - For example, from the step function:
    - use the node period from step function
    - Use the default value of a topic field
- Useful sharable constants should be written in the RADL description

Generation process (high level)

- A .radl file is a RADL module
  - May depend on other RADL modules (not circular)
  - Reference auxiliary files like:
    - C file with the step function
    - User libraries
- Compilation generate
  - A RADL object module (.radlo)
  - A catkin package ready to be compiled
  - Both are used to compile dependent modules

Code Generation

- A process per node
  - Initialization then simulation loop at fixed freq
    - Read input mailboxes
    - Compute: call the user step function
    - Write output mailboxes
- Communication is handled by
  - ROS backend
  - HACMS flavor backend

HACMS (not released)

- Hypervisors
  - Certikos from Yale University
  - Lynxsecure
- Shared memory
  - Between process
  - Between virtual machines
- Bare metal process on Certikos

- High Assurance

Community oriented with ROS

- ROS
  - www.ros.org
  - De facto standard middleware for robotics
    - Large library corpus
    - Large sensors and actuators drivers
  - Provides communication over tcp
  - No hard guarantees on timing

ROS backend

- Follows ROS naming conventions
- Maps nodes and topics to ROS ones
- A Plant
  - By default ROS is dynamic
    - A way to make it static is Secure ROS (demo after)
  - Periods and latencies are resolved in launch files
  - Each machine has a launch file

Conclusion

- radler.csl.sri.com
- You can play with it on github
- Even contribute (GPLv3)
- Current work in progress
  - Re-implement a simple backend without ROS
    - ZeroMQ / nanomsg are our current target
  - Secure ROS