Dynamic resource configuration and control for an autonomous robotic vehicle

Abhishek Kulkarni, Bryce Himebaugh and Steven D Johnson

School of Informatics and Computer Science, Indiana University, Bloomington, 47401
{adkulkar, bhimebau, sjohnson}@cs.indiana.edu

October 23, 2009
Outline

- ERTS robotic vehicle
  - Mission
  - Architecture
- ERTS software requisites
- SyncFS
- Cart component model
- Work in Progress
  - platform heterogeneity
  - reactive programming in limbo
  - 9p on embedded network protocols
ERTS Mission

▶ Lessons learned from participation in the Darpa Grand Challenge development effort
▶ Developed for and by the participants of introductory course on **Embedded and Real-time Systems**
▶ Explore embedded system design through the control of an autonomous vehicle
▶ Platform for local experimentation, collaborative research and instruction
ERTS robotic vehicle

- EZGO® golf cart augmented with controls
- Sensors
  - GPS, compass, IMUs, vision, joystick, laser, IR, ...
- Actuators
  - VCS, steering control, obstacle avoidance, voltage throttle, ...
- Onboard LAN of ARM Linux computer nodes (CNODEs)
- Networked navigation system
Overview of the cart’s electronics
Steering system overview

Cart’s steering system architecture
ERTS software requisites

- flexible, modular and composable
- horizontal and vertical configurability
- light-weight with sufficient abstractions
- sensor network architecture
- platform heterogeneity
- implicit/explicit time synchronization
Synchronous reactive systems

Reactive systems are computer systems that continuously react to their environment at a speed determined by this environment.

- **ERTS** is essentially a reactive system
- timing behaviour can be formalized trivially
- easier to model, design and verify
- signals change only at the clock edge
- no data races and hazards
Reactive components

- control unit
  - usually implemented as a state machine

- data unit
  - processes, stores or exchanges data

- example
  - in a navigation system: GPS, IMUs, compass . . .
ERTS software ecosystem

- **Operating System**
  - Linux, RT-Linux, QNX RTOS, Plan 9, Inferno...

- **CartFS**
  - components obeying synchronous access conventions

- **SyncFS**
  - synchronous file server

- Experiments
SyncFS Overview

- synchronous file server
- modeled after a globally clocked D flip-flop
- defers writes/stats to a simulated "clock edge"
- MRSW model (multiple readers single writer)
- RAM based, w/ double buffering
- system-wide write commits
facilitates buffer-based inter-component communication
- blocking stat as the synchronization element
- implicit CLK component
- exports elapsed ticks through the clock file
- 800 lines of C code, uses npfs
Component Framework (CartFS)

- obeys synchronous access conventions defined by SyncFS
- components communicate with the device or with each other
- exposes files command, status, ...
- adds itself to the global SyncFS namespace explicitly
- components write only to their own status file
- uses JSON for exchanging structured data

Control Loop Code

while True:
    wait_for_clock()
    read_files()
    has_requirements?()
    process()
    write_files()
Component access flow

- **COMPONENT A**
- **COMPONENT B**
- **COMPONENT C**

Clock (CLK) tracks the timing of read and write operations across components.
Component Architecture

device -> DRIVER -> data image

MONITOR

file image

COMPONENT

CLK

file image

file image

file image

f
CartFS overview

- uniform component directory structure
- \textit{status} (\texttt{s}) - output for the component
- \textit{configuration} (\texttt{c}) - contains path to the input channels
- \textit{doc} (\texttt{d}) - contains description of each of the status and configuration variables
  - \textit{log} (\texttt{log}) - contains diagnostic data
- open close operations minimized to reduce load on \texttt{SyncFS}
- use seek to return to the beginning of a file
Component interface

```
$ cat /tmp/cartfs/config/config_s
{'clock': ['../clock', 'clock', null],
'percent_throttle': ['../jdriver/jdriver_s', 'percent_throttle', 0]}

$ cat /tmp/cartfs/compass/compass_s
{'clock': 1423, 'enable': 'True', 'heading': 124.00}

$ cat /tmp/cartfs/jdriver/jdriver_c
{'joystick_throttle': -0.69999999999999996, 'joystick_steering': 0.0,
'direction': 'Forward', 'enable': 'True', 'clock': 1538}

$ cat /tmp/cartfs/jdriver/jdriver_c_d
{'enable': 'True/False — stops reads on jdriver device',
'clock': 'The clock value on which this data was written.'}

$ cat /tmp/cartfs/jdriver/jdriver_s_d
{'enable': 'True/False — stops reads on jdriver device',
'clock': 'The clock value on which this data was written.'}
```
9P on Windows

- in user space
- allows interacting components to be written in Windows
- facilitates collaborative research
- makes you unhappy 😞

```c
device = CreateFile(DOKAN_GLOBAL_DEVICE_NAME,
    GENERIC_READ | GENERIC_WRITE,
    FILE_SHARE_READ | FILE_SHARE_WRITE,
    NULL,
    OPEN_EXISTING,
    0,
    NULL
);
```
Work in Progress

- Rewrite the component framework in Limbo
  - leverage typed channels, ADTs
  - time as a first-class notion

- Port the existing simulator to Inferno
  - graphics is a pleasure to work with
Distributed clock synchronization
  - implement IEEE 1588

Support 9P over embedded network protocols
  - EtherCAT, Ethernet Powerline, CAN bus
Finally...

Questions?