The FAIR Timing System

Wesley W. Terpstra
Mathias Kreider

CSCO-TG

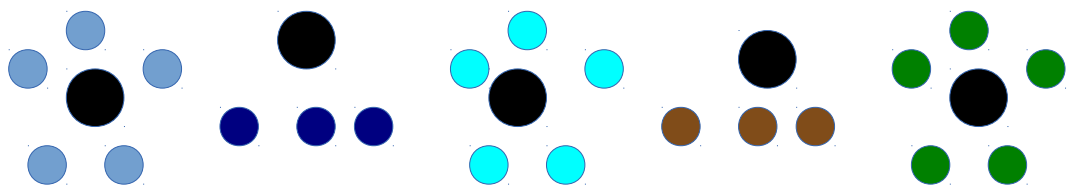
Outline

- How has the architecture changed
- What consequences for design / critical costs
- How CSCO-TG meets these constraints
- Using the system: addressing MID vs. GID
- Using the system: Atomic commands

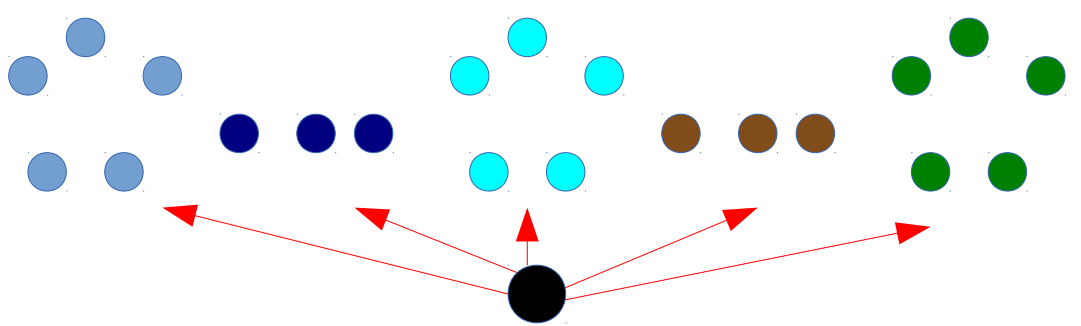
- Discussion at end + whenever you like

Arch. Change 1: Central Control

Before: one timing/control system per “machine”



Future: one timing/control system for entire facility



Arch. Change 1: Central Control

Before: one timing/control system per “machine”

- Worked well when there were few machines (3)
- Inter-machine events were tricky; coordination needed

Future: one timing/control system for entire facility

- Coordination of nodes is easier
- The data master will be very complicated

Arch. Change 2: Time at all nodes

Before: only the MIL master has the time

- Does not scale to large distances
- Every time-sensitive action must be announced to “dumb” nodes

Future: every node has the time

- The alarm “execute now” is moved from master to node
- Nodes (front-end controllers) execute on a schedule
- At some point, nodes are committed to execution / run autonomously

The Most Expensive Resources

Complexity of the Data Master

- One data master controls entire facility → very complicated!
- Facility is much larger → must keep DM as simple as possible

Real-time bandwidth between DM and nodes

- It is essential that commands reach nodes on time
- Cannot afford to send “just any” traffic → only stuff that matters!
- Build for a large safety margin assuming worst-case traffic

Timing System Design Goals

Reduce: Complexity of the Data Master

- ECA accepts high-level commands → complexity from DM to node
- Matching rules (MIL+steroids) → use same command for many nodes

Reduce: Real-time bandwidth between DM and nodes

- Multiple actions at one node can result from one command
- Multiple nodes can respond to one command
- There are no “machine” boundaries between nodes

→ Leverage ECA to reduce # of commands (complexity+bandwidth)

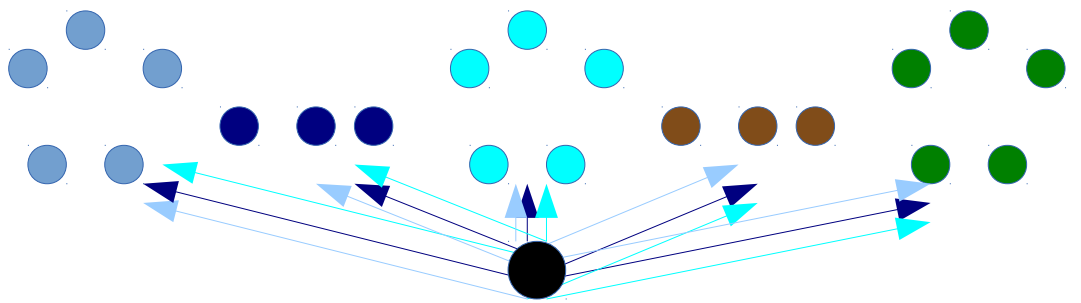
Addressing: MID vs. Group ID (GID)

“Machines” don't exist anymore

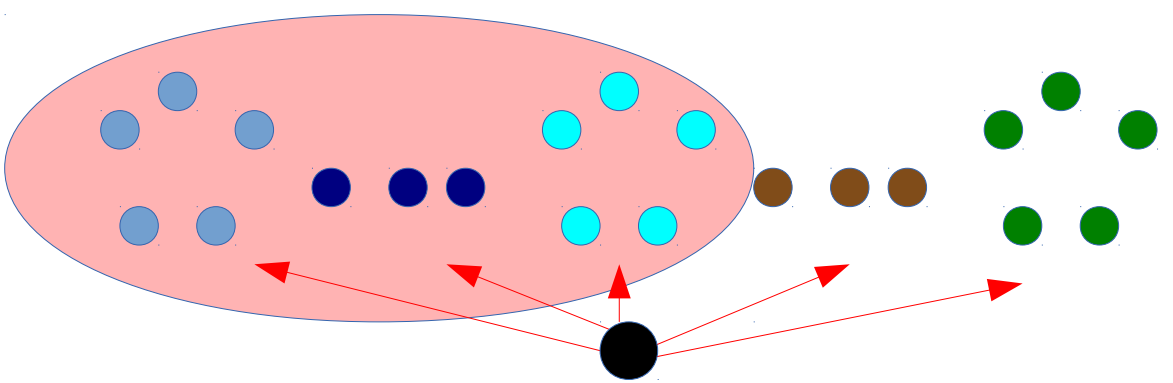
- There is one control system → segregation is only a convenience
- Broadcast: every node already receives every command
  - The decision to act on a command is governed by rules
  - No need to dupliate a command that affects two “machines”
- Organize addressing by groups of nodes
  - Example groups: SIS, Unilac, Unilac→SIS, SIS100→SFRSTarg

Addressing: MID vs. Group ID (GID)

Before: Send command to each machine separately

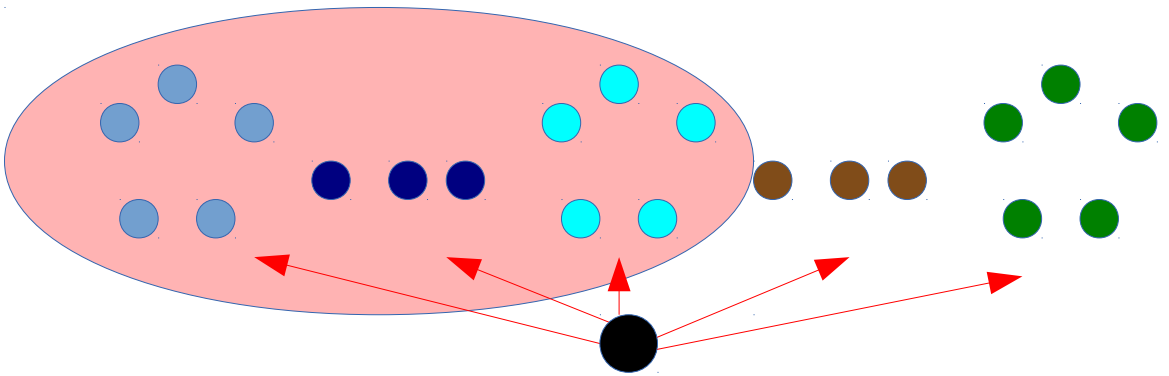


Future: Send command to node group



Less commands = less traffic + simpler to debug

Addressing: MID vs. Group ID (GID)



How to set this up?

- Nodes watch for commands and commands
- Nodes watch for commands and commands
- Nodes watch for commands and commands

→ ECA already supports this.

→ DM already supports this.

Addressing: MID vs. Group ID (GID)

Think in terms of node groups!

- “Machines” are node groups
- Nodes can be members of multiple groups
- Node groups can overlap

- Groups → less messages
- Groups → a simpler data master

Atomic commands

Commands cause actions. Several.

- In the old system, only the master had time
  - Thus: the master had to generate each action on time

- Now, all nodes have time
  - Nodes react to commands that affect them
  - Nodes add fine delay to compensate local configuration
  - Nodes can take multiple actions (eg: toggle pulse on then off)

- Don't fight the design!
  - Not a question of IF commands cause several actions
  - Question: WHICH actions should be grouped into WHAT commands

Atomic commands

Actions should be grouped by atomicity

- If two actions always happen together, they belong together
  - Atoms are inseparable
  - Inseperable actions = atomic command
- If two actions might NOT occur together → two actions
  - Example: prepare transfer, execute transfer
  - Prepare = ramp up all transfer magnets, charge kickers
  - Execute = trigger both kickers with appropriate offset
- Happens together = belongs together

Atomic commands

Offsets are your friend, not enemy

- Grouping actions = less commands = simpler DM = easier to debug
- But, what if two actions are separated in time?
  - In old system: only DM has time → must be two commands
  - In new system: CAN use offsets to combine the actions
- No one is forced to do this
  - You MAY reduce the command count this way
  - Respect atomicity! Don't combine actions unless an atom!
  - No downside: commands=atomic → delayed actions safe

Summary

New requirements → new design → rethink approach

- FAIR is bigger and more complex
- We are already commited to: central DM + distributed alarm model

- The only question is how we use the system we have
  - Do we treat it like MILv2?
  - Do we use it to its full potential?

- Timing group wants:
  - Keep data master as simple as possible (critical and complex!)
  - Keep real-time bandwidth low (finite resource, future uses unclear)

?
?
?
?
?
?
?

? Questions

?
?
?
?
?
?
?
?

The FAIR Timing System Concept

Concerns:

- **“It is simpler when all actions come from one place”**
- **“These 'long' actions can't be stopped”**
- **“No way to tell what's going with delays and rules”**
- **“Intelligent Endpoints are hell to debug”**

The FAIR Timing System Concept

Concerns:

- **“It is simpler when all actions come from one place”**

  - First: They still do. Only the DM can causes actions.

  - Fine delay would still be unknown to central master
  - Complexity moved to DM
  - Introduced more causes for errors

The FAIR Timing System Concept

Concerns:

- **“These 'long' actions can't be stopped”**
  - There is always a point of no return
  - Just a question of time
  - We give finer control

The FAIR Timing System Concept

Concerns:

- **“No way to tell what's going on with delays and rules”**
  - Unified distributed system via EtherBone and Wishbone
  - All local rules and delays can be read by DM
  - Some conflicts can be detected at setup time

The FAIR Timing System Concept

Concerns:

- **“Intelligent Endpoints are hell to debug”**
  - It never was easy. Impossible to see everything at runtime.
  - Post mortem debug
  - New system simpler:
    - Log EVERYTHING → state, sent, received and done
    - Gathering information easy → unified protocols
    - Post mortem correlation easy → absolute timestamps