Static Component Configuration Support for Real-Time Platforms

Chris Gill,

Venkita Subramonian, and Liang-Jui Shen

Dept. of Computer Science and Engineering
Washington University
St. Louis, MO 63130

{cdgill,venkita,ls1}@cse.wustl.edu

Nanbor Wang

Tech-X Corporation
5561 Arapahoe Ave., Suite A
Boulder, CO 80303

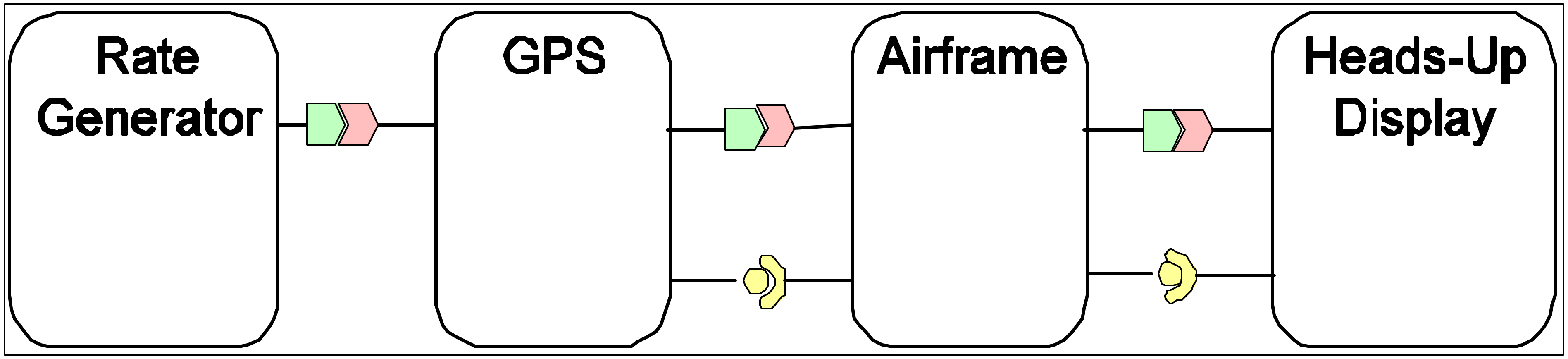
nanbor@txcorp.com

This work was supported in part by the DARPA PCES program, contracts F33615-00-C-3048 and F33615-03-C-4111

OMG Real-time and Embedded Systems Workshop

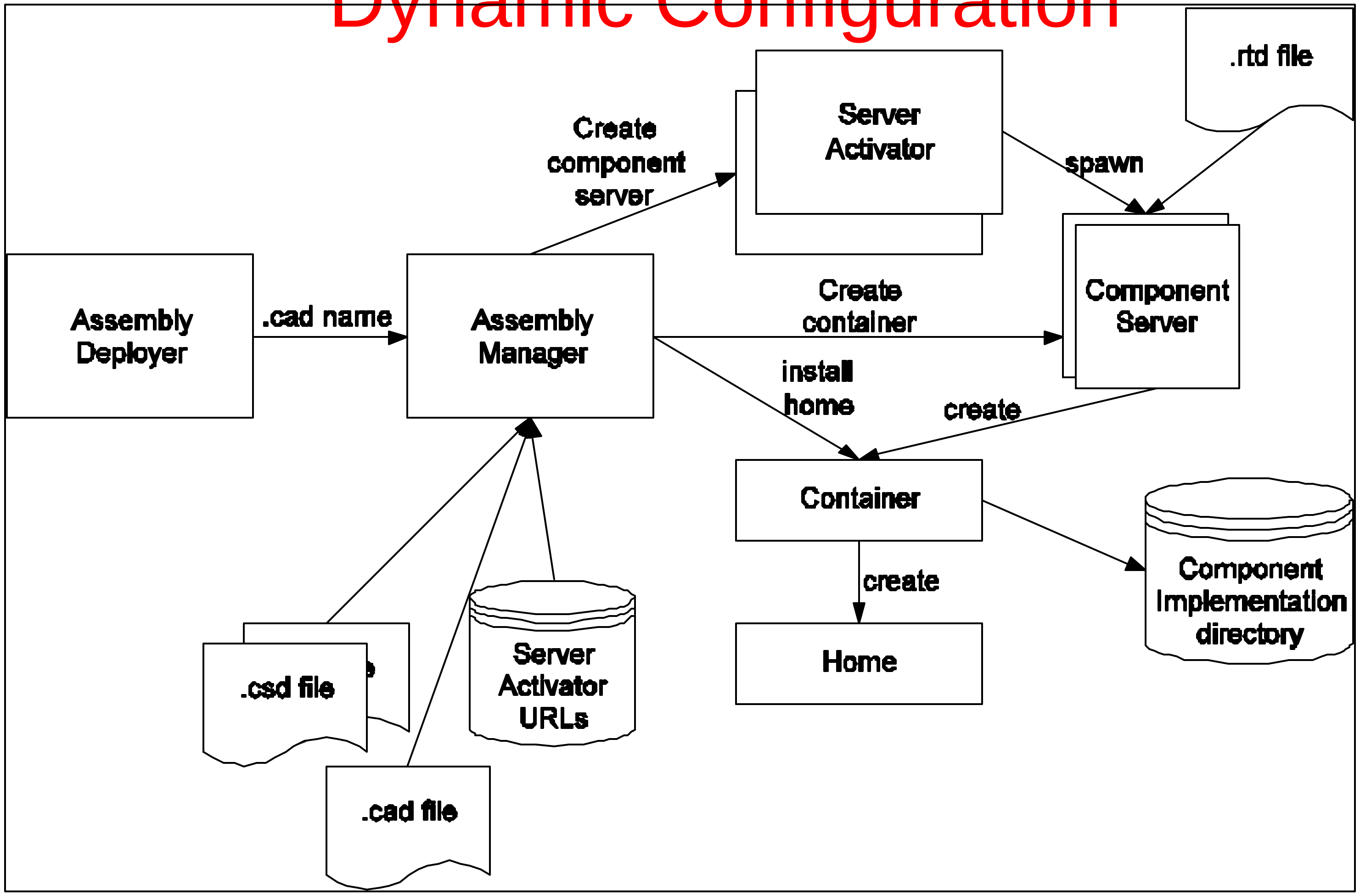
Reston, VA, USA July 13, 2004

Motivating Example Application



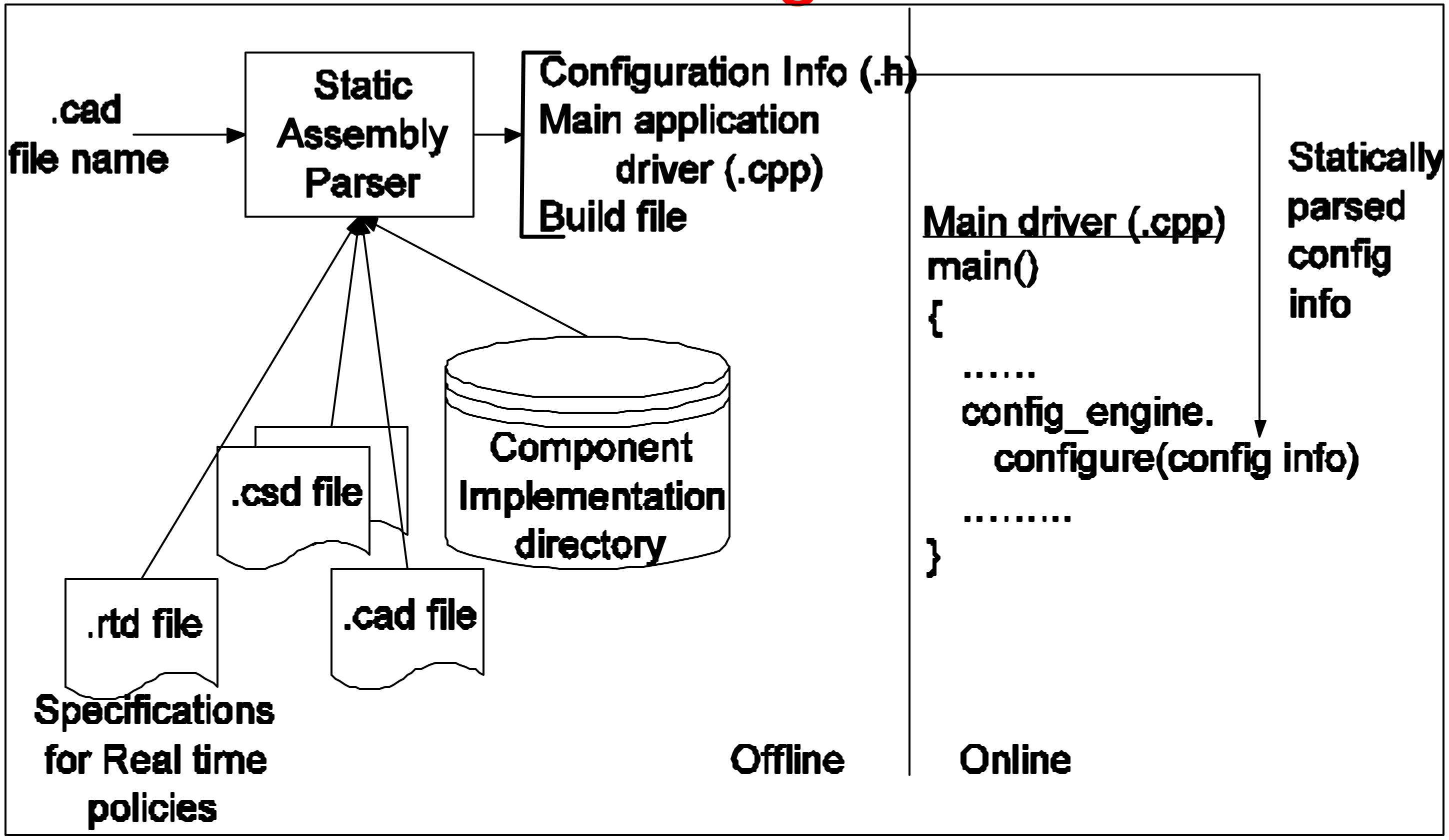
- Simple CCM application from avionics mission computing
- Representative of other distributed real-time applications
- Components are connected flexibly via ports
- In other work we've looked at configuring real-time aspects
  - E.g., RT-CORBA policies, thread pools from within CCM
- In this work we look at real-time bounds on system start-up
  - Affects how we view assembly and deployment stages

Dynamic Configuration



- Classical assembly and deployment approach using DLLS, XML parsing
- Problems: no DLLs or SO libraries on VxWorks, time to load, parse

Static Configuration



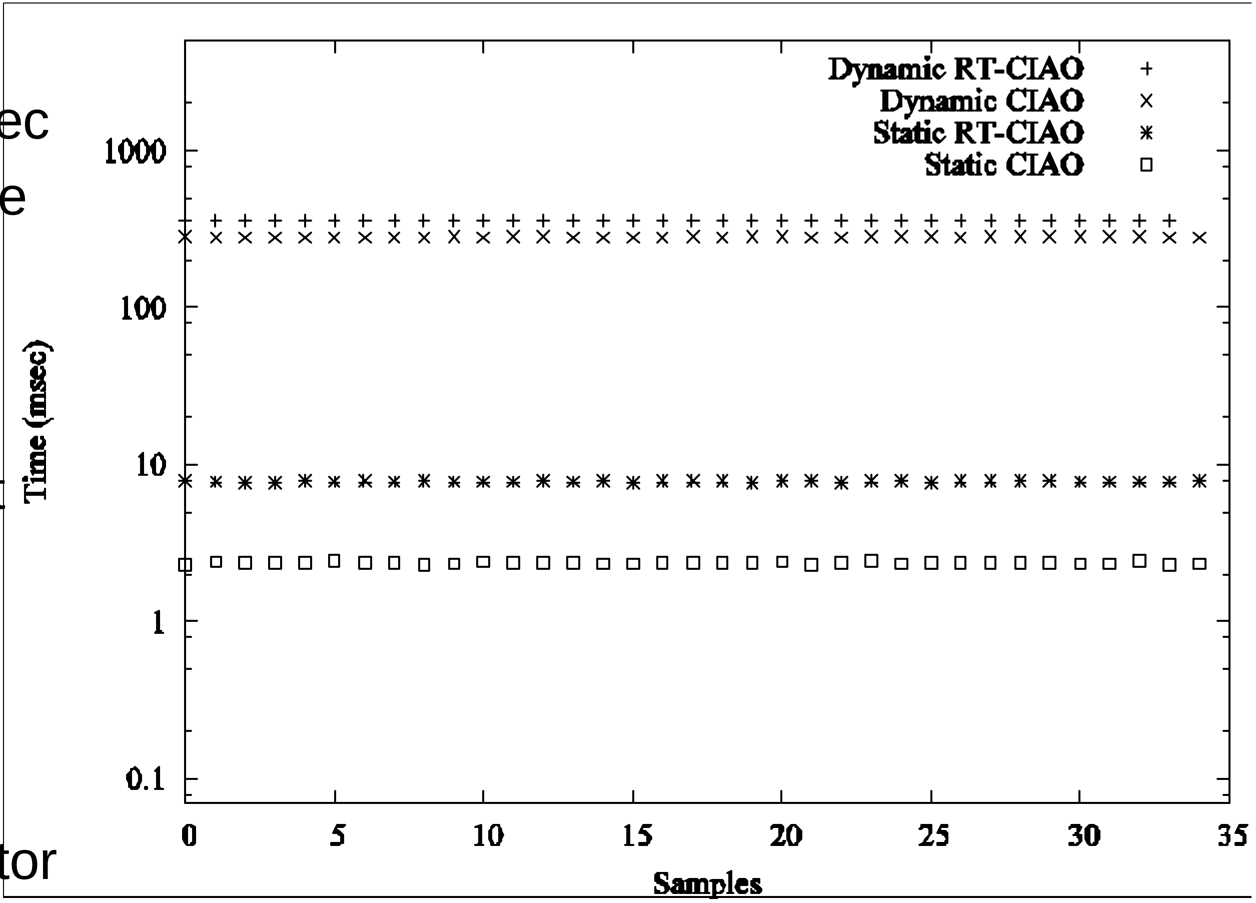
- To fit within stringent system initialization bounds, need to rethink lifecycle
  - Move as much off-line as possible while preserving config flexibility
  - Use static linking to “load” implementations, drivers to configure

Experiments

- Used example application with dynamic vs. static approaches
  - Dynamic composition: on-line DLLs and XML parsing
  - Static composition: off-line parsing, on-line init drivers
- Compare performance of static & dynamic configuration
- Identify most important sources of performance difference
- Tests were run on a single machine
  - Pentium IV 2.5GHz CPU, 500MB RAM, 512KB Cache
  - OS was Linux (with KURT 2.4.18 patches applied)
    - Supports DLLs for dynamic configuration approach
    - Offers good predictability for performance comparisons
  - Experiments used CIAO 0.4.1 / TAO 1.4.1 / ACE 5.4.1

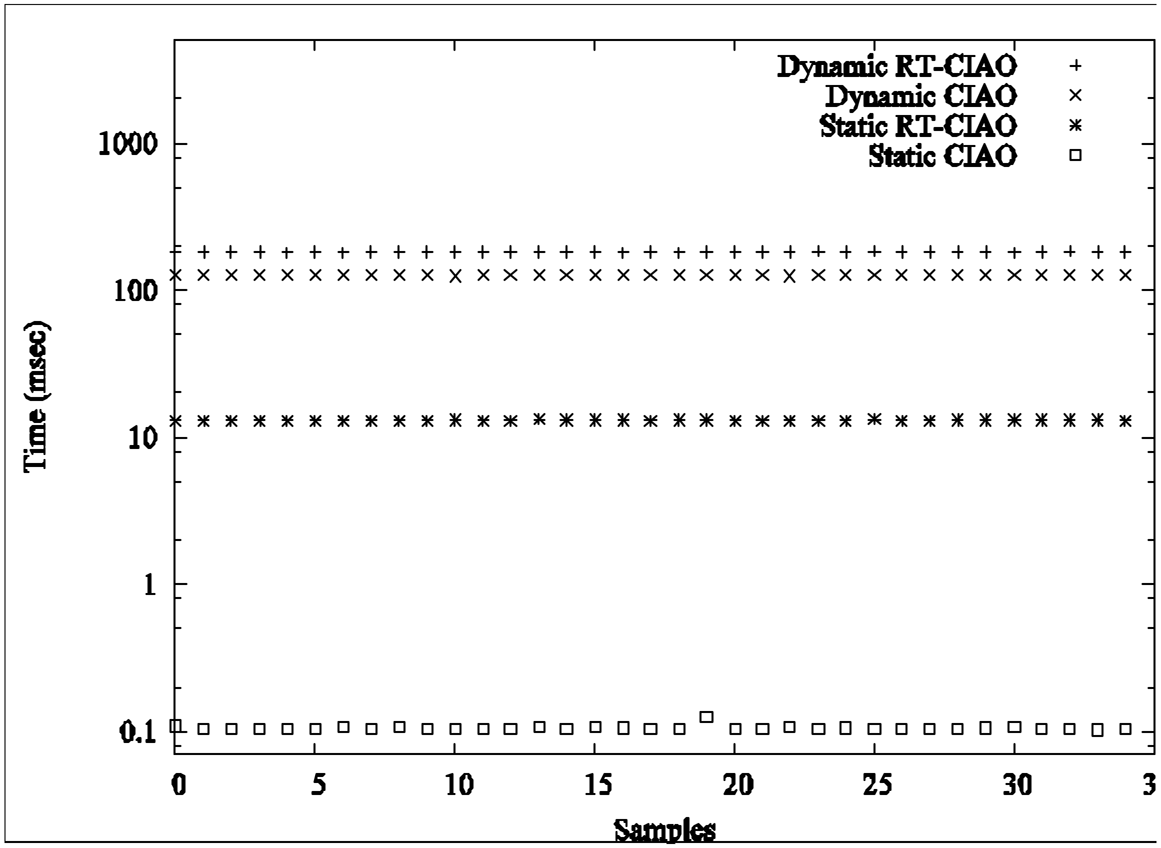
Time for Assembly

- Log scale used for plots
- Without RT features
  - msec vs. 100s of msec
  - 2 orders of magnitude
- With RT features
  - Fixed *additive* overhead
  - Greater relative hit at lower orders of mag.
- Difference attributed to
  - Loading DLLs
  - XML parsing on-line
- Most significant contributor to performance difference



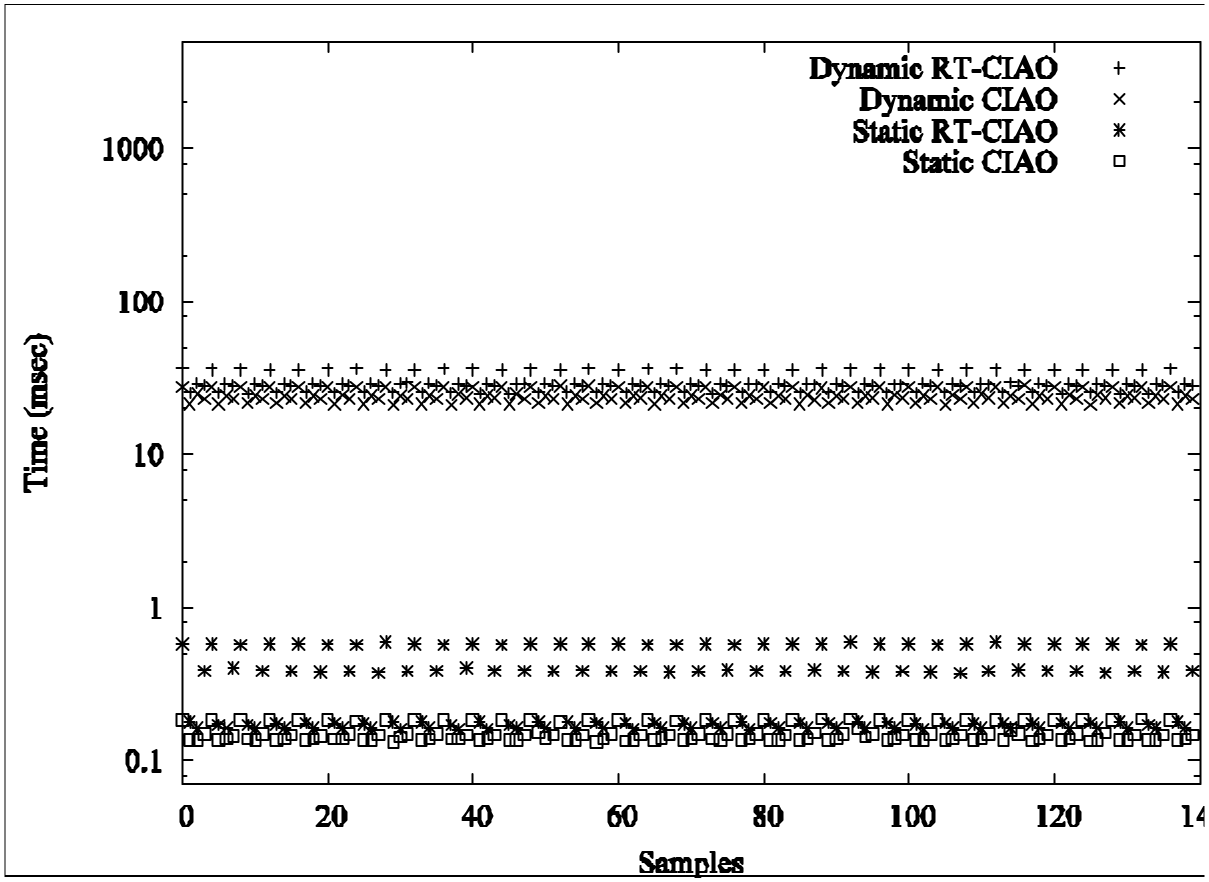
Component Server Creation Time

- 2nd largest contributor to performance differences
- 100 vs. 10 msec
- Most of the time spent on static component server was hooking into RT CORBA features



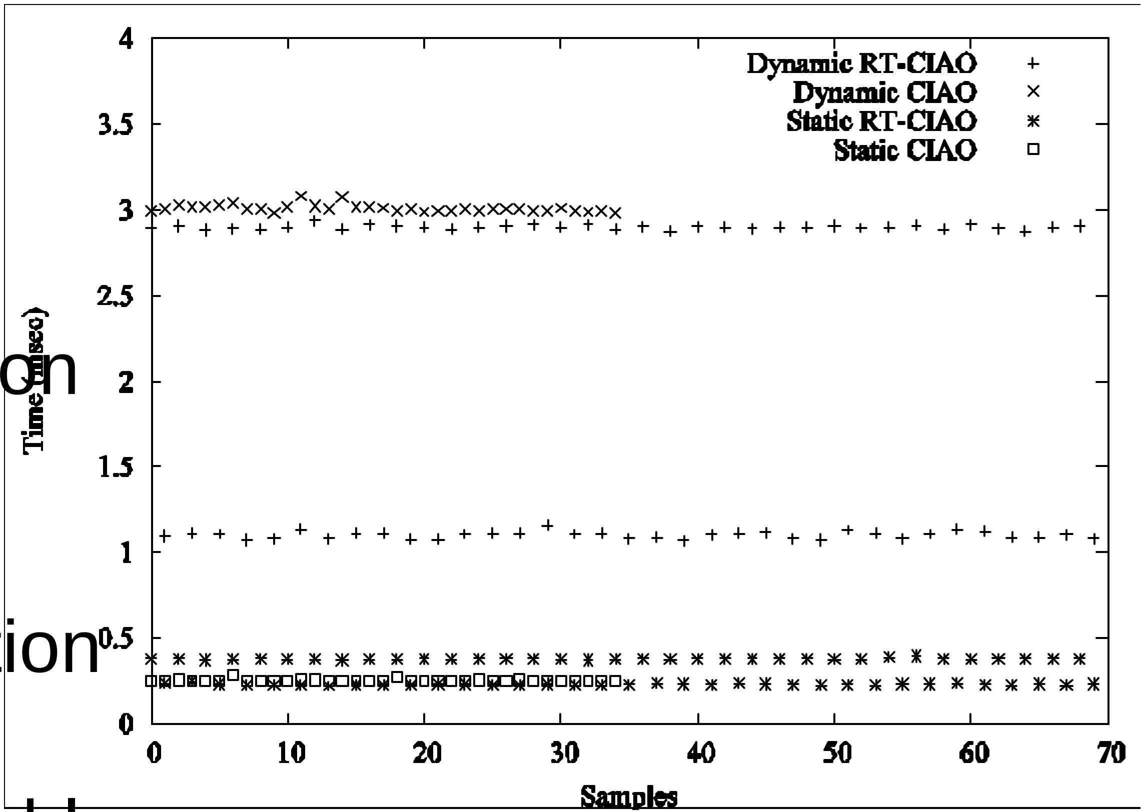
Home Creation Time

- Less expensive than
  - application assembly
  - component server
- Loaded vs. linked homes accounts
  - Dynamic > static
- Real-time features
  - Didn't increase the total time significantly



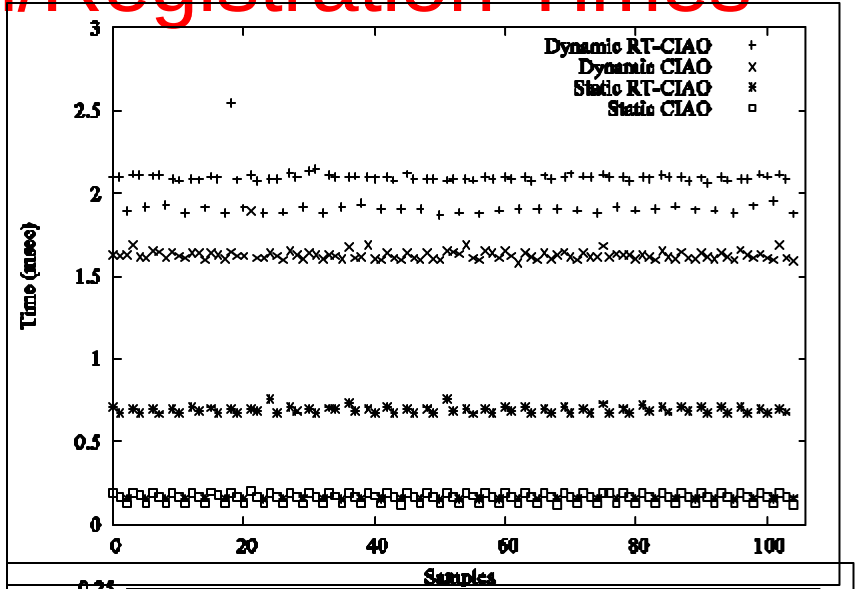
Container Creation Time

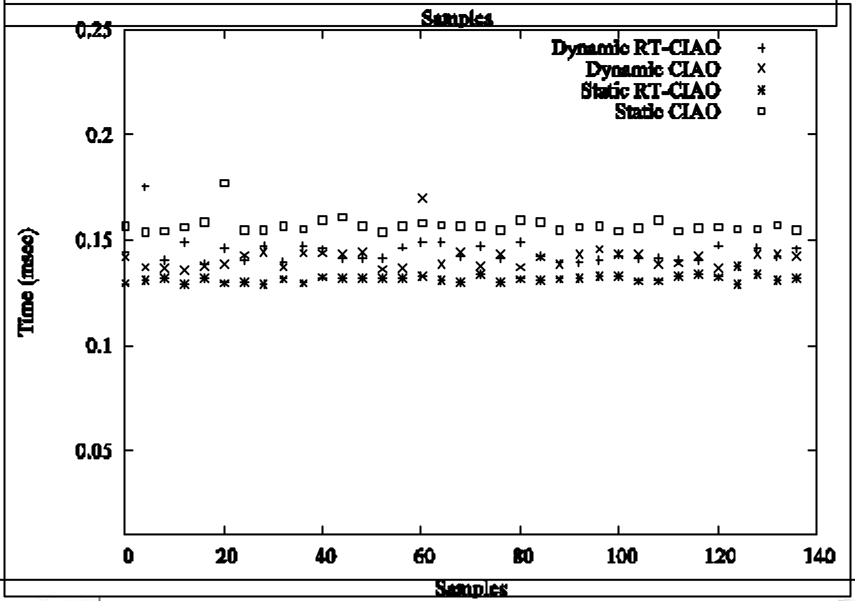
- Two containers for RT versions of tests
  - One with RT features, one without
  - Bimodal distribution
  - Twice as many samples
- Very small contribution to total overhead
  - Relative to assembly, server creation



Component Creation/Registration Times

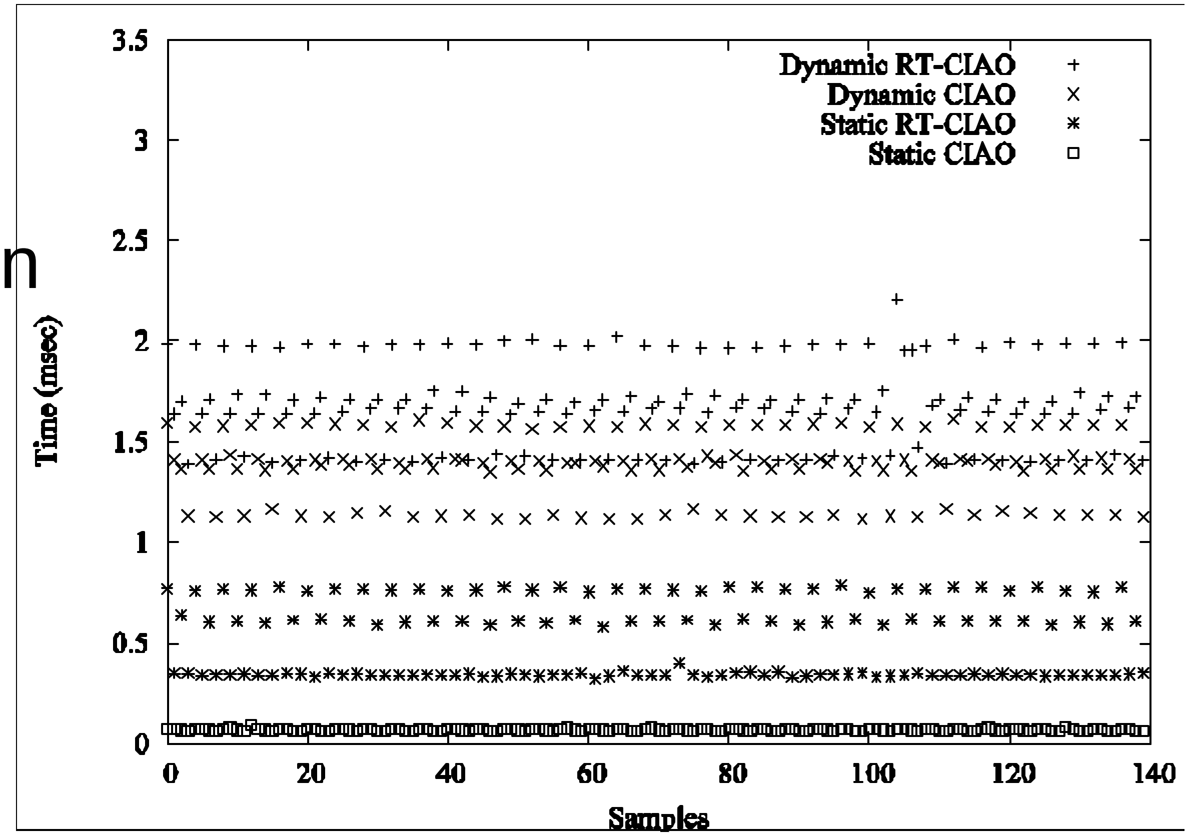
- Also small contribution to total overhead
- Interesting inversion: static non-RT CIAO takes longest
  - “In the noise” where cost of registration mechanisms ~same
  - Tracked this down to variations in IOR creation times





Connection Establishment Time

- Again a minor contributor to total overhead
- Bimodal distribution within data sets
  - Differences in port types among test program components



Concluding Remarks

- Static configuration improves performance
  - At a cost of less run-time flexibility
  - Does not depend on DLLs: works on VxWorks
- Major differences due to DLL loading, XML parsing
  - Concentrated in application assembly, server creation
- Static configuration still parses XML, off-line
  - Specification driven generation of on-line drivers
  - Gets many of the most important benefits
  - Without paying for the most expensive mechanisms