Distributed Sensors, Instrument, and Software

In Situ Instruments Workshop

Jason J Hyon, Kenneth Hurst
Jet Propulsion Laboratory,
California Institute of Technology
June 13, 2002
Realtime Scenario for Data Processing, Management, and Archiving

Instrument Data via TCP/IP

Navigation, Command & Search

Search & Retrieve

Data I/O

Process 1

Process 2

Realtime Display

Visualization

Computation and modeling

Seismic

GPS

Instrument

Raw Data

Science Data + Metadata

Data Archive

Data Distribution
Distributed sensors measure spatial patterns

Remote Sensing
InSAR
GPS
Gravity
Magnetics

Earthquakes/Volcanic eruptions/Landslides
Subsidence/Erosion
Plate Boundaries
Convection

Spaceborne Observations
Field Measurements

Laboratory Experiments

Time Scales (sec.)

Crystals/Atoms/Molecules

Remote Sensing
InSAR
GPS
Gravity
Magnetics

Earthquakes/Volcanic eruptions/Landslides
Subsidence/Erosion
Plate Boundaries
Convection

Spaceborne Observations
Field Measurements

Laboratory Experiments

Time Scales (sec.)

Crystals/Atoms/Molecules
NEPTUNE – sensors on the Juan de Fuca plate
NEPTUNE – node detail

- Backbone
- Branching unit
- 2n spur cable to science node, 2½ water depths
- Network module
- Instrument module
- Long "extension cord" up to 100 km
- Short "extension cord" up to 1 km
- Sensor
- Sensor
GPS networks

- Southern California Integrated GPS Network (SCIGN) is made of about 250 stations.
- Hundreds of stations in the global network.
- Automated analysis picks out earthquakes.
- Shows other problems with data (e.g., ground water pumping).

Courtesy R. Granat and M. Hefflin
Strain rates from GPS

Strain rates determined from GPS measurements in California. 
*Courtesy Steve Ward*
SCSN seismometer network
Space-based sensor

In Situ Instruments Workshop

Earth Observing System

Origins Program

Sun-Earth Connections

Structure & Evolution of the Universe
In Situ Instruments Workshop

Space-based sensor application

Isolation & Integrity

- High Noise
- Low Resolution Need
- Single Spacecraft

Signal Separation

Rate & Predictability

- High Rate
- Low Predictability
- Single Spacecraft

Signal Space Coverage

Signal Combination
Software Needs for Types of In Situ Sensors

- Integrated Payload
  - Stationary (On Board data management & processing)
  - Mobile (Location)

- Distributed Sensors
  - Wired (Network control)
  - Wireless (Communication, silicon micro-machining, LPE)
  - Mobile (Routing, Adaptive algorithms)
  - Combination of above (Clustering)
Communication Challenges Introduced by Mobile Instruments

- Location Tracking
- Routing of data between receivers and senders
- Data volume – processed vs raw
- Efficient dissemination of information – Communication strategy
- Fault tolerance
- Data security
Distributed Communication Examples

- WINS (Wireless Integrated Network Sensors) – UCLA
  - Integrated solution for low power communication environment

- LEACH (Low Energy Adaptive Clustering Hierarchy) – MIT
  - Efficient strategy for communication with data negotiation and resource-adaptive algorithms

- Architecture for Distributed Sensor Network – Lyndell St. Ville, U of Glasgow
  - An approach to a solution
Realtime Data Management and Instrument Control

- Instrument Interfaces
- Data Processing
- Data Transport
- Data Management
- Autonomy - feature extraction, optical navigation, network control (WDM, BS)
- Fault tolerance
- System Architecture
The Evolution of COTS

Historically, mission-critical apps were built directly atop hardware & OS
  • Tedious, error-prone, & costly over lifecycles

Standards-based COTS middleware helps:
  • Manage end-to-end resources
  • Leverage HW/SW technology advances
  • Evolve to new environments & requirements

The domain-specific services layer is where system integrators can provide the most value & derive the most benefits

Key R&D challenges include:
  • Layered QoS specification & enforcement
  • Separating policies & mechanisms across layers
  • Time/space optimizations for middleware & apps
  • Multi-level global resource mgmt. & optimization
  • High confidence
  • Stable & robust adaptive systems

Prior R&D efforts have address some, but by no means all, of these issues
Overview of the ACE Framework

www.cs.wustl.edu/~schmidt/ACE.html

Features
- Open-source
- 200,000+ lines of C++
- 30+ person-years of effort
- Ported to Win32, UNIX, & RTOSs
  - e.g., VxWorks, pSoS, LynxOS, Chorus, QNX

Large open-source user community
- www.cs.wustl.edu/~schmidt/ACE-users.html

Commercial support by Riverace
- www.riverace.com/
Key Capabilities Provided by ACE

Service Access & Control

Application Layer

- API
- Operation request
- In args
- Out args + return value

Middleware Layer

- API
- Middleware

OS Layer

- API
- OS Kernel & Protocols

Event Handling

PEER₁, Client Initiator

- Process event
- 1: send request event
- 4: recv completion event

PEER₂, Service Provider

- Event Handlers
- Process event
- 2: recv indication event
- 3: send response event

Concurrency

Thread₁ executes critical section

Synchronization

// Do something
lock.acquire();
// Begin critical section
operation₁();
... 
operationₙ();
lock.release();
// End critical section
// Do something else

Thread₂ blocks until Thread₁ leaves the critical section
Example of Applying Patterns & Frameworks:

Real-time CORBA & The ACE ORB (TAO)

www.cs.wustl.edu/~schmidt/TAO.html

Features

- Open-source
- 500+ classes & 500,000+ lines of C++
- ACE/patterns-based
- 30+ person-years of effort
- Ported to UNIX, Win32, MVS, & many RT & embedded OSs
  - e.g., VxWorks, LynxOS, Chorus, QNX

- Large open-source user community
  - www.cs.wustl.edu/~schmidt/TAO-users.html

- Commercial support by OCI
  - www.theaceorb.com/
Tutorial Example 3: Applying Patterns to Real-time CORBA

Patterns are used throughout The ACE ORB (TAO) Real-time CORBA implementation to codify expert knowledge & to generate the ORB's software architecture by capturing recurring structures & dynamics & resolving common design forces.
In Situ Instruments Workshop

Concluding Remarks

• Researchers & developers of distributed applications face common challenges
  • e.g., connection management, service initialization, error handling, flow & congestion control, event demuxing, distribution, concurrency control, fault tolerance synchronization, scheduling, & persistence
• Patterns, frameworks, & components help to resolve these challenges
• These techniques can yield efficient, scalable, predictable, & flexible middleware & applications

Jjh/kh20