A DISTRIBUTED INFRASTRUCTURE
FOR THE SYNCHRONIZED ACQUISITION
OF SENSOR DATA

-ICS 214b class project-
Winter 2006

Chi Fai Chan
Hojjat Jafarpour
Shengyue Ji
Kuan Sung Lee
Daniel Massaguer
Rares Vernica
{chancf, hjafarpo, shengyue, klee10, dmassagu, rvernica}@uci.edu

Project advisors: Utz Westermann and Prof. Sharad Mehrotra
The Place
RESPONSHERE

The Place

Sensing and communication infrastructure
- Temp. sensors
- Outdoor cameras
- Wireless
- Indoor cameras
- Bluetooth
- Cellular
- Ethernet
- Powerline
- PDA
- RFID
- Cellphone
- Your devices
- Acoustic sensors
- Evacpack
- Mesh routers
- Multi-Gas sensor
- Sun SPOT
- Aut. car

The Paradigm

Dynamic community input

Design Drill
- Customer 1
- Customer 2
- Customer 3

Deploy Drill
- Updated to new drill
- Actuation of Technology Solutions

Command Center
- Storage
- Visualization
- Servers
- Data sets
- Facilitate
- Search
- Maps
Computing, visualization, and datasets

- RAID storage server
- multi-tile visualization display
- 8-32-bit-processor IBM server
- 8-64-bit-processor Sun server
- Datasets: drill, 911 calls, 9-11, CAD, GIS, people counter logs, LDC TDT4, disasters, KDD, UCI facilities

http://rescue-ibm.calit2.uci.edu/datasets/
APPLICATIONS

The infrastructure will allow to perform:
- People counting
- People tracking
- Event detection (hazard, security policy violation, etc)
- etc

Which enables applications such as:
- Testing IT solutions for emergency response in the context of a drill
- Surveillance (e.g. Video and/or RFID surveillance)
  - Coffee room control
- Augmenting a drill simulation
- SAMI
- Quasar
- Media broker (SAMI, VIEWS)
- Aut. sensing platform
- etc
Collect data from sensing infrastructure

Data is unsynchronized

Data is multi-modal

Event detection by multi-modal processing
A DISTRIBUTED INFRASTRUCTURE FOR THE SYNCHRONIZED ACQUISITION OF SENSOR DATA
Multi-model sensing

Flexibility to support different applications

Simple real-time content analysis:
  reliability (nodes failing)
  synchronization
  abstraction from physical nuisances
• Controls the node's functionality
• Function:
  – Creating different modules
  – Connecting them to each other
  – Starting up the node
• Configuration
  – File
  – Network
• Sample configuration file content:

Server Config

- Port
  - 8089
- Server Channel
  - channel_1
  - channel_2

Client Config

- addChannel
  - channelName
  - channel_1
  - serverName
  - localhost
  - serverPort
  - 8089
- addChannel
  - channelName
  - channel_2
  - serverName
  - 127.0.0.1
  - serverPort
  - 8089
COMMUNICATION BETWEEN NODES

Data wrapping

Reason for wrapping:
- Network traffic modes: data stream, data diagram…
- Data formats: mpeg, asf…

By wrapping we provide a generic data sending/receiving interface for different kinds of usage.

Also we hide the networking details of transferring data from the above layer.
COMMUNICATION BETWEEN NODES

Data wrapping

• We are using:
  – TCP protocol (to ignore data lost in network)
  – Data packet mode (to provide support for data diagram, also compatible with data stream)
    *this also enables the controlling of traffic loads.
  – HTTP protocol (to take advantage of existing protocol)
COMMUNICATION BETWEEN NODES

Data wrapping

Header

- First line
- Attribute
- Attribute
- End of header

Content

[64 bytes of binary content]

Header

- POST / HTTP/1.1
- Content-Length: 64
- Content-Type: video/mpeg
- Connection: Keep-Alive
- Channel: channel_1
- Timestamp: 433632
- Anything: aaaa

Content
COMMUNICATION BETWEEN NODES

Communication method

Node 1

Agent

receive

send

Server (In)

Client (Out)

pushing

Node 2

Agent

receive

send

Server (In)

Client (Out)

connect

post

ok

post

ok

…
COMMUNICATION BETWEEN NODES

Channels in the network

Node A → Channel 1 → Node B
Node A → Channel 2 → Node C
Node B → Channel 4 → Node D
Node D → Channel 6 → Node E
Node C → Channel 3 → Node D
Node C → Channel 5 → Node E
Mobile agent middleware for injecting arbitrary operators/agents (Java)

Discussion: Operator generates agents—data wrapped by agent (e.g. Access control)

Code Injection:
- Agent migration:
  - (semi) strong
  - (semi) weak
- Remote injection
- Local injection
OUTBOX

<table>
<thead>
<tr>
<th>Channel Name</th>
<th>Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>chan_1</td>
<td></td>
</tr>
<tr>
<td>chan_2</td>
<td></td>
</tr>
<tr>
<td>chan_x</td>
<td></td>
</tr>
<tr>
<td>chan_y</td>
<td></td>
</tr>
</tbody>
</table>

Buffer (Message Queue)

Thread creation process

LOG

AGENT

package1 (to chan_1)

packagex (to chan_x)

Some other node..
OUTBOX
Functions provided in the outbox

- outbox(client*);
- int startUp(char*, char*, int);
- int send(packages*);
- int getStatus();
- static void *run(void *arg);
- void getMessage();
- void printChannel();
- messageQueue
  - packagesInQueue
LOG MODULE
Functionality

• Stores all the messages send by Outbox on external storage
• Retrieves messages from the external storage (during node recovery)
• Can be adapted for Inbox
One log file for all the *channels*
Each message has a *timestamp* (unique and always increasing)
*Index* on timestamp – efficient retrieval of messages based on timestamp
Allows retrieval of:
- Message with Timestamp greater or equal to a specified Timestamp
- Next Message
- Newest Message
DEMO