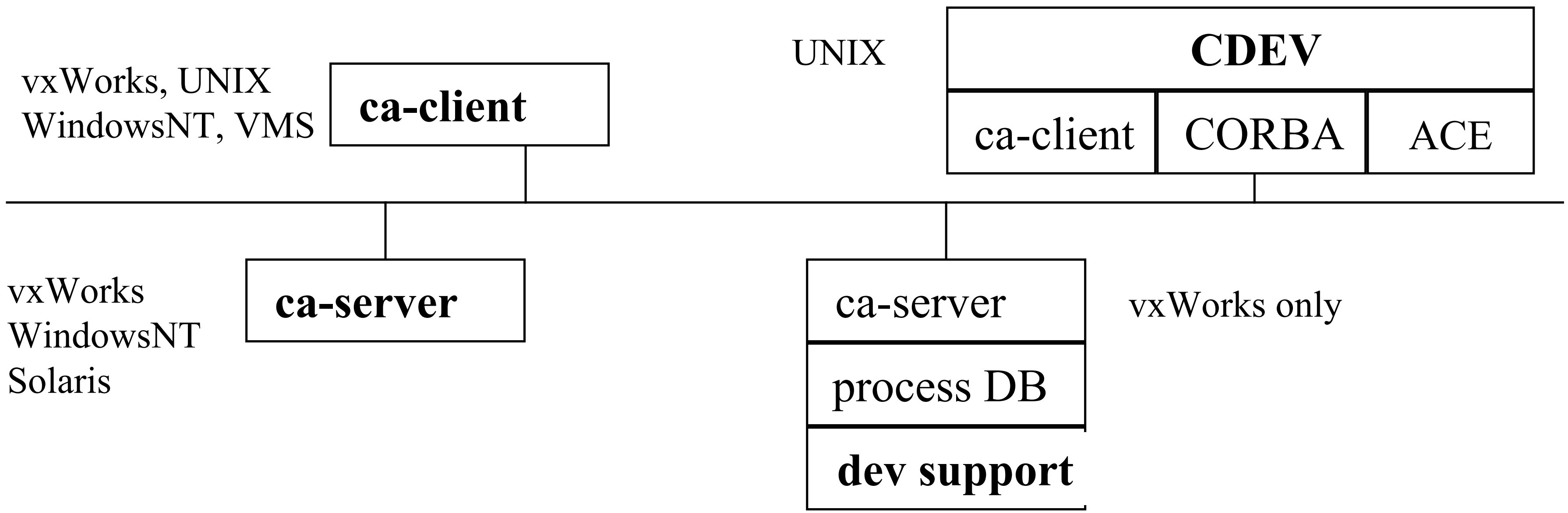
Writing a Channel Access Client in EPICS

Bob Dalesio, August 11, 1999

Outline

- Channel Access in the EPICS Architecture
- Channel Access Client Overview
- Channel Access Clients
  - synchronous client
  - composite data structures
  - buffering for efficiency
  - asynchronous connection handling
  - asynchronous monitoring

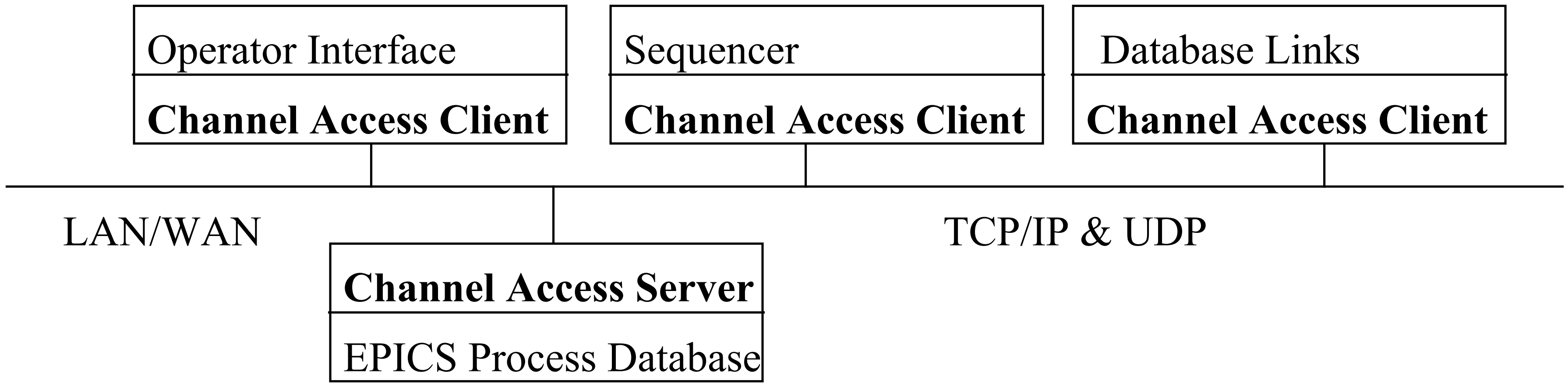
Distributed Software Architecture



Many tools are available in the EPICS tool-kit

- EPICS tools are connected via the Channel Access client/server libraries
- Server Interfaces:
    - Process Database
    - Gateway (CA-Client - GDD Library - Portable server on Solaris)
- Client Interfaces
    - Process Database Links
    - Sequential Control Language
    - Data Visualization Packages
    - Data Analysis Packages
    - Modeling and Automation Packages

Channel Access Client/Server Libraries



Client: Provides read/write connections to any subsystem on the network with a channel access server

Server: Provides read/write connections to information in this node to any client on the network through channel access client calls

Services: Dynamic Channel Location, Get, Put, Monitor
Access Control, Connection Monitoring, Automatic Reconnect
Conversion to client types, Composite Data Structures

Platforms: UNIX, vxWorks, VMS, Windows NT

Channel Access

Operator Interface	Sequencer	Database Links
Channel Access Client	**Channel Access Client**	**Channel Access Client**

TCP/IP & UDP — LAN/WAN

Channel Access Server
EPICS Process Database

Performance:
68040 over 10 Mbit Ethernet

Gets

Propagation Delay	2 mS
Throughput	7.8K /sec

Puts

Propagation Delay	1 mS
Throughput	17K /sec

Monitors

Propagation Delay	Dependent
Throughput	10K / sec

(Typically 10% channels have monitors)
(memory use in IOC - 2 Meg / 60 connections)
(30% network load for 10K monitors / second)

Increase bandwidth with Routers, Bridges, Higher speed networks and EPICS gateway

Simple Channel Access Client

```
#include <cadef.h>
main ( int argc, char **argv){
    dbr_double_t data;
    chid mychid;

    ca_task_initialize();                  /* ca initialization */
                                           /* find the specified channel */
    ca_search_and_connect(argv[1],&mychid,NULL,NULL);
    ca_pend_io(5.0);                       /* synchronous completion of search */
                                           /* get the value */
    ca_get(DBR_DOUBLE,mychid,(void *) data);
    ca_pend_io(5.0);                       /* synchronous completion of get */
}
```

Channel Access ‘PUTS’

- ca_put

  a request is placed in the local queue - program control returns immediately
  in the server - the new value is a cached put
- ca_put_callback

  a request is placed in the local queue - program control returns immediately
  client is notified when the put and all related record processing is complete
  in the server - these new values are queued
- ca_sg_put

  a request is placed in the local queue
  after all of the ca_sg_puts are queued - ca_sg_block is issued
  program control waits until all puts and related record processing completes
  client is notified when each put and all related record processing is complete
  in the server - these new values are queued

Data Type Conversions in Channel Access

DBR

_STRING, _DOUBLE, _FLOAT, _LONG, _CHAR, _ENUM

Data type conversions are performed in the server

Endian and floating point conversions are done in the client

Polite clients requests data in native type and perform necessary conversion on the client side

Composite Data Structures

Requests can be made for data related to the value field:

- Float, Short, Int Long, Char….
  - status, severity, time stamp, alarm limits, display limits, control limits
- String
  - status, severity, time stamp, max string length
- Enumerated
  - status, severity, time stamp, max string length, choices, #choices

Example Request:

```
struct dbr_ctrl_enum        data;

ca_get(DBR_CTRL_ENUM,mychid,(void *)&data);
```

Accessing Composite Data Structures

Many fields are fetched from the data store in one access:

```
struct dbr_ctrl_float        data;
struct dbr_ctrl_float        pdata = &data;
ca_get(DBR_CTRL_FLOAT,mychid,(void *)pdata);
printf("%d %d\n",pdata->status, pdata->severity);
printf("%d %d\n",pdata->stamp.secPastEpoch, pdata->stamp.nsec);
printf("%f %f\n",pdata->high_display_limit,pdata->low_display_limit);
printf("%f %f\n",pdata->high_warning_limit,pdata->low_warning_limit);
printf("%f %f\n",pdata->high_alarm_limit,pdata->low_alarm_limit);
printf("%f %f\n",pdata->high_control_limit,pdata->low_control_limit);
printf("%f %s\n",pdata->value, pdata->units);
```

*Refer to db_access.h for structures...

Making Efficient Use of Synchronous Channel Access Calls

Buffer up requests before flushing the buffer and waiting for the result.

```
…
    ca_search(chan_nam1,&chid1);
    ca_search(chan_nam2,&chid2);
    ca_search(chan_nam3,&chid3);
    ca_pend_io(1.0);
    ca_get(DBR_DOUBLE, chid1,(void *)&data1)
    ca_get(DBR_DOUBLE,chid2,(void *)&data2);
    ca_get(DBR_DOUBLE,chid3,(void *)&data3);
    ca_pend_io(1.0);
```

Asynchronous Name Resolution

```
dbConnectionHandler(
struct connection_handler_args          arg)
{
    if (ca_state(arg.chid) != cs_conn)    ….this channel is newly disconnected
    else                                  ….this channel is newly connected
}

main()
...
    ca_search_and_connect(name,&chid1,dbConnectionHandler,(void *)NULL);
    ca_pend_event(.001);      /* in this case - this is only a buffer flush */
}
```

Asynchronous Data Notification

```
dbConnectionHandler(
struct connection_handler_args          arg)
{
    if (ca_state(arg.chid) != cs_conn)   ….this channel is newly disconnected{
            return;
    else{                                ….this channel is newly connected

    ca_add_array_event(dbf_type_to_DBR_CTRL(ca_field_type(arg.chid)),
            ca_element_count(arg.chid), arg.chid,
            caEventHandler,0,0.0,0.0,0.0,(evid *)NULL);
}
```

Asynchronous Data Notification - 2

```
caEventHandler(
struct event_handler_args       arg)
{
    switch (arg.type){
    case(DBR_STS_STRING):
    case(DBR_STS_SHORT):
    case(DBR_STS_FLOAT):
    case(DBR_STS_ENUM):
    case(DBR_STS_CHAR):
    case(DBR_STS_LONG):
    case(DBR_STS_DOUBLE):
    default:
    }
}
```

Channel Access Clients with X/Motif
Using the File Descriptor Manager (fdmgr)

- Mainline code must:
  - call channel access file descriptor registration: ca_add_fd_registration
  - call fdmgr_add_timeout for periodic event that allows CA to run
  - get the X file descriptor and register it with the fdmgr
  - fdmgr_pend in an infinite loop to service CA,X, and Motif events
- ca_add_fd_registration routine adds new fd's to the fdmgr
- fdmgr time-out routine issues a ca_pend_event(.001) and reissues time-out
- ca event handlers are called from the fdmgr events
- X event handler is called from fdmgr events

Error Checking

- Error codes and error related macros are in caerr.h
- SEVCHK will exit on errors it deems irrecoverable
- ECA_NORMAL means the exchange was initiated successfully
- SEVCHK exit behavior can be replaced with your own exception handler
  ca_add_exception_event(…..)
- example:
  status = ca_array_put(data_type,channel_id,pvalue);
  SEVCHK(status,"additional info in error message");

Caching vs. Queuing

- An event handler can either take it's actions in the event handler -
  - queuing
  - all data is handled
  - degradation mode is longer delays
- Place data into an intermediate buffer and have an alternate thread handle the data
  - caching
  - data can be overwritten
  - degradation mode is intermediate data is discarded
- note that buffer in IOC will overwrite the last monitor on the queue when a buffer overflows in the IOC

Channel Access Notes

- ca_repeater needs to be run once on each workstation

- in the database,
  - a deadband of 0 posts a monitor on any change
  - a deadband of -1 posts monitors on every scan

- read cadef.h, caerr.h and db_access.h before writing a channel access client

- it is most efficient to use native data types and handle data conversions in the client program

Channel Access Environment Variables

Evironment Variable Name	**Range**	**Default**
EPICS_CA_ADDR_LIST	{n.n.n.n n.n.n.n n.n.n.n …}	None
EPICS_CA_AUTO_ADDR_LIST	{YES,NO}	YES
EPICS_CA_CONN_TMO	$r > 0.1$ seconds	30.0
EPICS_CA_BEACON_PERIOD	$r > 0.1$ seconds	15.0
EPICS_CA_REPEATER_PORT	$i > 5000$	5065
EPICS_CA_SERVER_PORT	$i > 5000$	5064
EPICS_TS_MIN_WEST	$-720 <= i <= 720$	360