Data Distribution Service as an alternative to CORBA Notification Service for the Alma Common Software

Jorge A. Avarias Alfaro
(ALMA UTFSM group/NRAO)
javarias@inf.utfsm.cl

Presented by Gianluca Chiozzi (ESO)
Contents

• ALMA and ACS
• ACS Data Channel
• Data Distribution Service
• The prototype
• Tests
• Conclusion
Introduction
ALMA Common Software

ACS is a software infrastructure for the development of distributed systems based on the Component/Container paradigm.

- end-to-end: from data reduction to control applications
- common application framework and design patterns, not just libraries
- well tested software that avoids duplication
- make upgrades and maintenance reasonable
- common development environment and tools
- Open source (LGPL)
- Used by several projects
ACS provides the basic services needed for object oriented distributed computing. Among these:

- Transparent remote object invocation
- Object deployment and location based on a container/component model
- Distributed error and alarm handling
- Distributed logging
- Distributed events
ACS Data/Notification Channel

- Publisher/Subscriber mechanism.
- Asynchronous communication.
- Publisher doesn’t know about Subscribers.
- Subscriber doesn’t know about Publishers.
- Many to many communication.
- Based on CORBA Notify Service.
CORBA Notify Service

- Publisher/Subscriber implementation with CORBA.
  - Pull model (Pull Subscriber and Pull Consumer).
  - Push model (Push Subscriber and Push Consumer).
- A broker (Notify Service) is responsible to deliver messages.
- Message Filtering:
  - Channels.
  - Structured messages.
ACS Data/Notification Channel

- Based on CORBA Notify Service.
- Handles structured messages.
  - Data type is defined in IDL file.
- Support set of Quality of Service properties.
- Uses CORBA Naming Services to make visible the channel.
- Provides:
  - SimpleSupplier class
  - SimpleConsumer class
The Problems with NS

- CORBA Notify Service always embeds the message as Any CORBA data type:
  - Marshalling uses a lot of CPU time.
  - Length of the message requires a lot of network bandwidth.
- CORBA Notify Service doesn’t scale very well.
  - Centralized delivery.
- ACS Notification Channel lacks of late joining subscriber feature.
An alternative: Data Distribution Service

- OMG Open Standard (as well as CORBA)
- High performance Publisher/Subscriber specification (focused on Real-Time)
  - Peer to Peer
  - Resource Management through QoS policies.
- Entities:
  - Publisher
    - Data Writer
  - Subscriber
    - Data Reader
  - Topic
Why DDS?

- Real data centric publisher/subscriber. Not just events!
- Data Distribution Service can do the same as CORBA Notify Channel.
- But... it could offer more features:
  - Better performance
  - Multicast transport support
  - Can be configured to support *late joining subscriber* feature.
- Would be desirable to provide these features in ACS: *DDS for ACS.*
Objectives of the prototype

- To compare Data Distribution Service and CORBA Notify Service, trying to find a common abstraction, considering the functionality offered by the ACS Notification Channel API.
- Implement an ACS Notification Channel alternative based on OpenDDS (in C++), trying to maintain the actual ACS Notification Channel API.
- Set up a test suite including some measurement of throughput or other performance indicators.
Data Distribution Service for ACS

- DDS Notification Messaging API
- DDS specification only implements push model.
- Maps:
  - `SimpleSupplier` in `DDSPublisher`
  - `SimpleConsumer` in `DDSSSubscriber`
- Offers a very similar API.
- Hide DDS complexity:
  - Initialization of entities of DDS.
  - Transport selection and initialization of OpenDDS.
VERIFICATION
Tests

1. Scalability test
   - 1 Publisher.
   - 1, 10, 20, 30, 50, 60, 75, 100 Subscribers.
   - 1000 messages at 10 Hz.

2. Slow Consumer test
   - 1 Publisher.
   - Several Subscriber and one Slow Subscriber.

3. Throughput test
   - 1 Publisher.
   - 1 Subscriber.
   - Message frequency: as fast as possible.
Test Configuration

- ACS 8 Release Candidate running on Scientific Linux 5.2.
- ACE version 5.6.5, provided as part of ACS 8 RC.
- TAO version 1.6.5, provided as part of ACS 8 RC.
- OpenDDS version 1.1
Scalability test results

Behavior of DDS and NC - 1 Publisher - 1 Subscriber

Arrival time [μs]

Message #
Scalability test results

Message Arrival Time Average -- 1 Publisher - n Subscribers

Arrival Time [µs] vs. Number of Subscribers

Graph showing the relationship between the number of subscribers and the average message arrival time for DDS and NC protocols.
## Scalability test results: Resource consumption

<table>
<thead>
<tr>
<th>Program</th>
<th>Memory Usage (MiB)</th>
<th>Number of Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDS Subscriber</td>
<td>4,8</td>
<td>12</td>
</tr>
<tr>
<td>NC Consumer</td>
<td>3,4</td>
<td>2</td>
</tr>
<tr>
<td>DDS Publisher (Container)</td>
<td>55,4</td>
<td>216</td>
</tr>
<tr>
<td>NC Supplier (Container)</td>
<td>3,9</td>
<td>7</td>
</tr>
<tr>
<td>DCPSInfoRepo</td>
<td>73</td>
<td>206</td>
</tr>
<tr>
<td>NotifyService</td>
<td>68,9</td>
<td>102</td>
</tr>
</tbody>
</table>
Throughput test

- Container crashes for memory exhaustion queuing undelivered messages with both implementations.

- Some values:

<table>
<thead>
<tr>
<th></th>
<th>DDS</th>
<th>NS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Received</td>
<td>~80.000</td>
<td>~230.000</td>
</tr>
<tr>
<td>Maximum Latency [uS]</td>
<td>~2.500.000</td>
<td>~35.000.000</td>
</tr>
</tbody>
</table>

- The crash can be avoided in DDS
  - Set correctly the QoS properties (History and Resources).
CONCLUSIONS
Conclusion

- ACS has now a prototype DDS replacement for the CORBA Notify Service.
- We have performed a comparison between the two implementations:
  + Good performance and scalability
  + Good QoS options and granularity
    - A lot of memory
    - A lot of threads
    - Interoperability and portability still limited
- The prototype is used for the E-ELT evaluation of ACS.
- Future Work
  - Get from prototype to “production quality”
  - Port all other ACS Services now using NS to DDS.
  - Test and compare with RTI DDS and OpenSplice
  - Test Multicast protocols.
QUESTIONS?

For more details:
http://www.eso.org/projects/alma/develop/acsc

Next ACS workshop:
UTFSM Valparaiso, Chile
November 2009

This work was supported
by the ALMA-Conicyt Fund #31060008
EXTRA SLIDES
Data Distribution Service

 DDS Peer (Producer)

 Application

 DataWriter

 Publisher

 DomainParticipant

 DDS Peer (Consumer)

 Application

 DataReader

 Subscriber

 DomainParticipant

 Peer-to-Peer Interaction

 Data-centric

 Matching Topic, Incompatible QoS
 Communication is not established

 Requested QoS

 Incompatible with Offered QoS

 Offered QoS

 Requested QoS

 DDS Domain (Global Data Space)

 Communication Not established
ACS Notification Channel features

• Hide as much as possible the CORBA complexity.
• Have three levels of message filtering
  – Event Domain
  – Event Type
  – Channel Name
• Offers two simple classes:
  – SimpleSupplier, Publisher class
    • void publishData(T data)
  – SimpleConsumer, Subscriber class
    • void consumerReady()
DDS Conceptual Mapping

Event domain
Domain = 411

Topic can be used as Channel for DDS

Data Type “a” Topic
Data type “b” Topic
Data type “c” Topic

Topics only support one data type
DDS Example Deployment

DDSPublisher <DataType1>

DDSPublisher <DataType2>

Tao Pluggable Transport (SimpleTCP)

Data Distribution Service

DCPSInfoRepo

DDSSubscriber <DataType1>

DDSSubscriber <DataType2>
Solution Proposal: Class model
module DDS_SIMPLE_EXAMPLE{
  const string CHANNEL_NAME = "simple_example";

#pragma DCPS_DATA_TYPE
"DDS_SIMPLE_EXAMPLE::simpleMessage"
struct simpleMessage{
  unsigned long seqnum;
};
Publisher

DDS

void SimpleExampleDDSImpl::sendMessage()
{
    pub_p = new acsPublisher(
        DDS_SIMPLE_EXAMPLE::CHANNEL_NAME, component, [NC/DDS]);

    DDS_SIMPLE_EXAMPLE::simpleMessage m;
    m.seqnum=1;

    PUBLISH_DATA(pub_p, DDS_SIMPLE_EXAMPLE::simpleMessage, m);

    sleep(1);
    pub_p->disconnect();
    delete pub_p;
}
void handlerFunction(DDS_SIMPLE_EXAMPLE::simpleMessage m, void *other)
{
    std::cout << "Arrived message" << std::endl;
}

void main()
{
...
    ddsnc::DDSSubscriber *sub_p=0;

    ACS_NEW_DDS_SUBSCRIBER(sub_p,
                          DDS_SIMPLE_EXAMPLE::simpleMessage,
                          DDS_SIMPLE_EXAMPLE::CHANNEL_NAME,
                          &handlerFunction, (void *)0);

    sub_p->consumerReady();

    ACE_Time_Value tv(100);
    client.run(tv);

    sub_p->disconnect();
    delete sub_p;
}
Slow Consumer test results

- In ACS NC Notify Service memory usage raises
  - Reliable QoS policy retains the messages not delivered.

- DDS have the same behavior
  - But it can be solved setting correctly the QoS properties (History and Resources)

- In DDS implementation the “normal” subscribers don’t have delays.
OpenDDS programming
ACS NC Structured Message

Definition of a Structured Event

EventHeader header

FixedEventHeader fixed_header

EventType event_type

- string domain_name = const string defined in an IDL file
- string type_name = const string defined in an IDL file
- string event_name = "" (default, but can be overridden in setData(...) method)

OptionalHeaderFields variable_header(PropertySeq)

0 to n instances of Property structures

Property

- string name = (optionally) user-defined
- any value = (optionally) user-defined

FilterableEventBody filterable_data (PropertySeq)

0 to n instances of Property structures

Property

- string name = const string defined in an IDL file
- IDL #include "some IDL file"
- string any value = user-defined (IDL struct, sequence, simple CORBA type, etc.)

- any remainder_of_body = (optionally) user-defined
Usage of ACS Notification Channel

High-level Usage of ALMA Events

- Astronomer Consuming ALMA Events as They Occur
- CORBA Notification Service Running On an Unsecured Server
- Telescope Calibration (C++)
- Pipeline processes (Python)
- Observation Scheduler (Java)
- CAN bus devices (C++)
- ALMA prototype antenna at the ATP
- Secured Host for CORBA Notification and Naming Service
- ALMA Operator (Java)
Technical Details

- OpenDDS availability as part of ACS.
- Compilation of OpenDDS with ACS.
- Modifications done to ACS Makefile
  - OpenDDS processing.
    - dcps_ts.pl
    - Generated extra classes for type support.
Threads Management
Problems, Solutions and Alternatives

• DDS CORBA Compatibility
• Participants parameters required by OpenDDS
  – DCPSInfoRepo Location
  – Transport Configuration
  – Configuration File
• Factories
  – Participant
  – Transport
Known Problems

• Threads and Container
  – The participant factory and the transport factory are threads that cannot be stopped.

• Resource consumption (threads)
  – For each participant in the connection, OpenDDS creates a new thread to establish the communication between others participants.
  – Requires a lot of memory in comparison of NC.

• Occasionally container crash
  – When a subscriber finish after the component is deactivated
Acknowledgements

• ESO and the ALMA-Conicyt Fund #31060008
• Heiko Sommer for his dedication and help guiding this work.
• Matías Mora and Rodrigo Tobar for their support.
• Thesis advisors: Horst von Brand and Javier Cañas.
• My family, my friends and the Computer Systems Research Group.