Component Based DDS using C++11

R2DDS (Ruby to DDS)

RTI London Connext Conference 2014

Johnny Willemsen
CTO Remedy IT
jwillemsen@remedy.nl
Remedy IT

- Remedy IT is specialized in communication middleware and component technologies
- Strong focus on open standards based solutions
- Actively involved in the Object Management Group, leading several OMG open standardization efforts
- Our customers are in various domains including telecom, aerospace and defense, transportation, industrial automation
What we do

- Develop implementations of OMG open standards
  - Open source; TAO, CIAO, R2CORBA
  - Commercial; TAOX11, AXCIOMA
- Deliver services related to OMG standards including CORBA, CCM, DDS
- Deliver services for specific implementations, including RTI Connext DDS
- Develop open standards as part of the OMG
Component Based DDS

DDS is a great technology but

• It provides a messaging protocol, not a complete architecture
• Provides a lot of freedom to the developer which can lead to misuse
• Lots of things are left to the application like deployment and threading model design
• Vendor portability can be a challenge
• No vendor support for the IDL to C++11 language mapping
CBDDS combines eleven OMG standards into a comprehensive software suite

- IDL4, IDL2C++11
- CORBA
- DDS, DDS X-Types, DDS Security, RPC4DDS
- LwCCM, DDS4CCM, AMI4CCM
- D&C
CBDDS Principles

- Interoperable Open Architecture (IOA)
- Component Based Architecture (CBA)
- Service Oriented Architecture (SOA)
- Event Driven Architecture (EDA)
- Model Driven Architecture (MDA)
What is a Component?

- Independent revisable unit of software with well defined interfaces called “ports”
- Able to be packaged as an independent deployable set of files
- Smallest decomposable unit that defines standard ports is a “monolithic component”
- A “component assembly” is an aggregation of monolithic components or other component assemblies
Why Component Based Development?

- Modularity
- Reuse
- Interoperability
- Extensibility
- Scalability
- Reduced Complexity
- Faster and Cheaper Development
- Quality and Reliability
- Deployability
Interaction Patterns

- CBDDS uses so-called interaction patterns to define the interaction between user components.
- CBDDS defines request/reply, state, and event interaction patterns.
- An interaction pattern is realized at deployment time using a specific communication middleware, legacy system, or hardware.
Support for synchronous and asynchronous invocations

Delivered with a function style API

Defined in IDL using operations with arguments and an optional return value

The application code that uses this interaction pattern is unaware of how the interaction pattern is realized
Event Interaction Pattern

The event interaction pattern defines extended ports for the following roles:

- Basic many-to-many publish subscribe messaging
- Event distribution with optional user defined data
State Interaction Pattern

The state interaction pattern defines extended ports for the following roles:

- Distributed state management and access
- Distributed database functionality with eventual consistency
Deployment

- CBDDS also includes deployment tooling
- Supports single process, single node, and distributed deployments
- Applications can be deployed using a deployment plan
- Which DDS QoS is used is a deployment decision, not hardcoded into the business logic
- Various options to define a deployment plan
IDL to C++11 is a formal OMG standard that greatly simplifies the development of IDL based applications.

- Reuse as much as possible from the C++11 standard
  - Standard basic types
  - Uses STL containers like std::string, std::vector, std::array
  - Uses C++11 move semantics to provide a safe and fast API
- Standardized IDL::traits<T> to simplify template programming
- Automatic reference counting by using std::shared_ptr and std::weak_ptr semantics
- No new/delete and no plain C++ pointers!
CBDDS with C++11 is implemented as part of our AXCIOMA product

Component related glue code is generated by our Ruby based IDL compiler

- No dependency on remote CORBA support, all local interfaces

C++11 representation of the user defined types, including all types that are used for DDS communication

But, no DDS vendor supports the IDL to C++11 language mapping out of the box
C++11 to C++ Conversion Framework

- Based on the user defined IDL types, an implied IDL definition is generated by RIDLC
  - This implied IDL definition is passed to rtiddsgen
- RIDLC generates a set of conversion traits to convert between the RTI C++ type definition and the C++11 type definition
  - By using C++11 move semantics this is in most cases just a move of memory, no copy!
- C++11 representation of the DDS entities uses the RTI C++ representation and the conversion traits
- At the moment RTI supports IDL to C++11 the conversion traits will expand to nothing!
// Sender component class which publishes one sample to DDS
class Sender_i : public IDL::traits<Sender>::base_type
{
public:
   // Register an instance to DDS
   virtual void configuration_complete () override {
      IDL::traits<Shapes::ShapeType_conn::Writer>::ref_type writer =
         context_--\>get_connection_info_write_data();
      instance_handle_ = writer--\>register_instance (square_);
   }

   // Write one sample to DDS
   virtual void ccm_activate () override {
       IDL::traits<Shapes::ShapeType_conn::Writer>::ref_type writer =
          context_--\>get_connection_info_write_data();
       writer--\>write_one (square_, instance_handle_);
   }
private:
   DDS::InstanceHandle_t instance_handle_; // Use C++11 uniform initialization to initialize the member
   ShapeType square {"GREEN", 10, 10, 1};
/ Receiver component which receives the samples from DDS
class Receiver_i : public IDL::traits<Receiver>::base_type
{
public:
virtual void configuration_complete () override {}
    // We want sample by sample
    virtual void ccm_activate () override {
        IDL::traits<CCM_DDS::DataListenerControl>::ref_type lc =
            context_->get_connection_info_data_control();
        lc->mode (CCM_DDS::ListenerMode::ONE_BY_ONE);
    }
private:
    IDL::traits<Shapes::CCM_Sender_Context>::ref_type context_;
    IDL::traits<Shapes::ShapeType_conn::CCM_Listener>::ref_type
    data_listener_;
};
// Receive the sample from DDS and just dump it to the console
class info_out_i: public
    IDL::traits<Shapes::ShapeType_conn::CCM_Listener>::base_type
{
public:
    // Sample has been received by DDS
    virtual void on_one_data (const ShapeType& shape,
        CCM_DDS::ReadInfo&) override {
        std::cout << “Received ” << shape << std::endl;
    }
private:
    IDL::traits<Shapes::CCM_Sender_Context>::ref_type context_;
Ruby to DDS
Ruby to DDS

- Ruby is a powerful scripting language
- Integrated part of the Ruby on Rails framework for developing web based applications
- R2CORBA makes it possible to use and provide CORBA functionality using Ruby
- R2DDS is a prototype developed by Remedy IT to use DDS from Ruby
- Push data from DDS to the web browser using Ruby on Rails
- Currently generates C++ code based on the IDL type definition, could potentially use Dynamic Data in the future
// Create all needed DDS entities (without QoS for simplicity)
dfp = DDS.DomainParticipantFactory_init()
dp = dfp.create_participant()
topic = dp.create_topic()
pub = dp.create_publisher()
sub = dp.create_subscriber()
dw = pub.create_datawriter(topic)
dr = sub.create_datareader(topic, listener)

// Create an ORANGE shape and publish it 10 times with increasing size and position
shape = ShapeType.new("ORANGE", 10, 10, 10)
$i = 1
while $i <= 10  do
dw.write(shape)
shape.shape_size = $i * 10
shape.x = $i * 10
shape.y = $i * 10
$i=$i+1
sleep(1)
end
// Define a new Listener that prints the received shape to the console
class ShapeListener < DDS::DataReaderListener
  def initialize()
    end

  def on_data_available(reader)
    shape = ShapeType.new()
    reader.read (shape);
    puts "Read sample #{shape.color()} #{shape.x()} #{shape.y()} #{shape.shapesize()}";
  end
end

// Create an instance and create a new DDS datareader with this listener
listener = ShapeListener.new()
sub = sub.create_datareader(topic, listener)
Questions?
Want to know more?

- Let us meet here at the RTI London Connext Conference 2014!
- See our website at http://www.remedy.nl
- Check our slideshare presentations at http://www.slideshare.net/RemedyIT
- Email me at jwillemsen@remedy.nl
- Call us at +31-10-5220139
- Follow us on Twitter @RemedyIT