Real Time Innovations

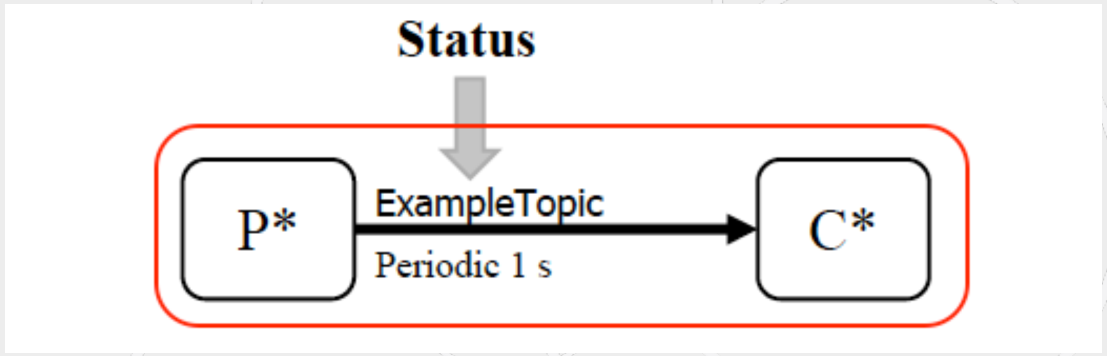
Tactical Microgrid Example Code Walk-thru

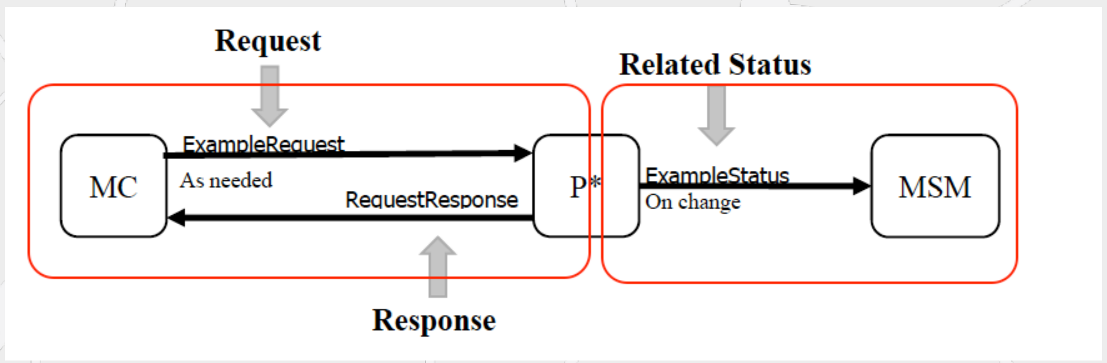
April 1, 2021

RTI makes the world run better

Components Involved

- Tactical Microgrid Spec (TMS)
  - Data Patterns
    - Publish Once (Durable)
    - Periodic
    - Request / Response
    - On-Change
    - Custom (Not implemented)
  - Topics
- Distributed Data Service (DDS)
- Customer Legacy System
  - Gateway Module / Interface between TMS/DDS and Legacy Communications





Issues

- TMS Spec is complex in draft form, and hard to get any examples to help confirm understanding
- Code Complexity to provide separation of concerns
- Lumping Complexities: TMS and application code structure complexity (to isolate legacy adaptation) may be lumped in as DDS complexities
- DDS Experience – recommend RTI Virtual Instructor Led Training (24 hour custom training) to get familiar with DDS

Example Design Approach

- Use System Designer
  - Existing model well defined
- Use XML APP Create and Dynamic Data
  - Easy to stand up DDS Participants, Pubs, Subs, Readers, Writers (one line of code)
- Separate out ("templatize") where customer adapter code goes

Implementation

- C++ Language binding
  - MacOS, Visual Studio Code
- Thread based model
  - Thread per topic*
    - separates out separate concerns
  - Note: Threads have little interaction
    - No Locks, Mutex's, synchronization between them
    - Rules to manage Request Correlation to Responses
      - Note clear from spec if one request is outstanding at time, or what one is to do with a 'not OK' Response

* Threads per Writer Event Threads Optional

Project Directory Structure

- Directories Under Project Directory tms
  - – tmsApp (example Application)
  - – tmsAppTest-SimMSM
    - MSM simulation used to unit test tmsApp
- .git – git management directory
- .DS_Store (Mac OS file)
- .gitignore (git file to instruct git to ignore built files)

tmsApp – Directory Structure

- Model (tms Data Model Directory) Contains:
  - Tms supplied idl and xml files
    - tms.idl, tms.xml, tms-topics.idl, tms-topics.xml
  - System Designer Project file ‘tmsTestApp.rtisdproj’
  - tmsTestApp.xml file
    - The xml file we’ll use to to ’standup’ all the DDS entities (Participant, Publisher, Subscriber, Readers, Writers and topics)
- cpp – holds source files
- objs – holds the output of the make/build
  - This directory is always removed and recreated with make clean/make (all)
  - It is and its contents are in .gitignor and not checked in the git

tmsApp Directory level files

- makefile – builds tmsApp
- USER_QOS_PROFILES.xml (copy of ./model/tmsTestApp.xml)
  - USER_QOS_PROFILES.xml is the default xml file DDS uses if no URL is specified.
  - We can avoid the copy by setting an env var to direct the application to the non-default URL of “Project Directory path/tmsApp/model/tmsTestApp.xml”
- tms.code-workspace and .vscode directory
  - Visual Studio Code workspace and configuration directories

tmsApp cpp files

- tmsTestExample.h
  - Created by rtiddscode gen from the tmsTestApp.xml file and used by the .cxx files to properly reference tms_ specific data structures and names
- tmsCommon.h
  - my common header file for .cxx files during compilation
- tmsTestExampleApp.cxx
  - This is the application C++ top level code

tmsApp cpp files (con't)

- tmsCommPatterns.cxx & .h files
  - Holds the definitions for the tms communications patterns and associated thread control blocks (see tmsCommPatterns.h file). These include
    - ReaderEventThread
      - Required Thread used for all readers
    - WriterEventThread
      - Optional thread to monitor writer topic status
    - PeriodicWriterThread
      - Use to send periodic topics
    - OnChangeWriterThread
      - Used when the user code sees a change in Status or variable requiring a tms 'On-Change' topic publication
      - Uses a User Defined DDS Guard Condition to trigger the associated topic writer

tmsCommPatterns – Reader Processing

- Control Block (ReaderTheadInfo) option
  - echoResponse (true/false)
    - Intended for received requests that require writing a tms_TOPIC_REQUEST_RESPONSE
    - Copies the Request SampleId to the RelatedRequestId
    - Sets the Status code to infoBlock tms_REPLY_code and tms_reason (can be changed in individual topic handlers)

tmsCommPatterns – Periodic Writer

- Writes associated topic periodically
  - As initialized (passed in variable to) by the control block c’tor
- Control Block (PeriodicTheadInfo) option
  - enabled (true/false)
    - Disables the both the call to the handler and the write operation

tmsCommPatterns – OnChange Operation

- Waits on your user defined DDSGuardConditon
  - Handed in to the associated info block at creation (of the info block) – part of the c'tor
- Idea is change an internal_variable different from the tms_external_variable to cause 'On Change' publishing
  - Due to an internal device change of state or from a tms_request topic
  - In the main_loop, if the internal_variable != tms_external_varable, set up the topic change the condition variable to true
    - Also take care to either preconfigure the topic data (if static) or first change any of the topic data prior to triggering the condition

tmsCommPattern – Topic Handlers

(tmsCommPatternTopicHndlrs.cxx & .h)

- Allows customization for topics by user
- Currently only envisioned for all Readers and PeriodicWriters
  - Periodic data is set per specific topic
- No perceived use cases for WriterEvents and OnChange Writers
  - WriterEvents publishes nothing and allows you to ‘handle’ status of the writer changing
  - OnChange you set up the data in the context of where you trigger the event through a DDSGuardCondition
  - Easy to add infrastructure for these if use case arises
    - i.e., for some reason you want to handle a the same writer status differently for one topic vs. another.

tmsCommPattern – Topic Handlers (Con’t)

(tmsCommPatternTopicHndlrs.cxx & .h)

- Declare a custom handler in tmsCommPatternTopicHndlrs.h
- Write a custom handler in tmsCommPatternTopicHndlrs.cxx
- Install the handler in the appropriate handler array location in tmsCommPatternTopicHndlrs.cxx
  - reader_handler_ptrs[]
  - Periodic_handler_ptrs[]
- The tmsCommsPattern thread processing does the generic/common processing before and after calling the handler
  - E.g., if on a request receive topic, if EchoResonse enabled, the readerThread will
    - First: Detect data arrival, get the data
    - Second: call the handler (passing a pointer to the info block)
    - Third: If EchoResponse enabled, copy the deviceId and sequenceNumber to the relatedRequestId in requestResponse. Copy the handler set or default status code and status reason to the requestResponse and publish

tmsAppTest-SimMSM – Directory Structure

- Similar to tmsApp Directory Structure but…
  - No model directory
    - Uses the Same model dir as tmsApp
  - Has its own Participant defined in the model
- cpp directory only contains the tmsAppTest-SimMSM.cxx
  - Make file builds and links with the same header files, tmsCommPatters and handler files as in the tmsApp directory
  - tmsCommsPatternTopicHndrs.cxx/.h will hold both tmsApp and tmsAppTest-SimMSM handlers
    - But each instantiates complementary topics and patterns

Current Status

- https://github.com/psmass/tms
- Provides tmsApp and tmsAppTest-SimMSM
  - MSM Sim test code only implemented enough to keep app stimulated
    - E.g., does not even look at device discovery topics
- Six of eight planned topics basically working
  - Between the App and Test App – all planned patterns working

What's left for Example (priority)

- Durable QoS (gets rid of startup order issues)
  - XML via System designer
- Filter for topics ‘For Me’ (XML edits via SD)
  - In a real system don’t want to see every one's requests (other devices and manager to other devices)
  - In a real system don’t want to see other device responses
- Dispose of topics once processed
  - Currently TMS implements SampleID as Key’d but each one is unique (device ID + Unique Sequence ID) – this effectively consumes resources for each Topic
- Test Request from Device / Response from MSM correlation mechanism
- Add SourceTransitionRequest to provide example of request to device and OnChange response
  - (currently you can see this from MSM code)

What’s left clean up

- Put reader and writer handles as well as data topic handles in an array and loop to get/create them
  - Cleans up code, and enables additional readers and writers to be more easily added
- Add a callback from Req/Response correlation mechanism if outstanding request is to be overwritten/missed.