An XML-Based Approach to the Configuration and Deployment of DDS Applications

Javier Sanchez-Monedero (javism at correo.ugr.es)
Javier Povedano Molina, Jose M. Lopez-Vega & Juan M. Lopez-Soler
Signal Theory, Telematics and Communications Dept.
University of Granada

Workshop on Distributed Object Computing for Real-time and Embedded Systems
July 15, 2008, Washington, DC, USA
The problem:

The development of a flexible framework for fast deploying of DDS scenarios
1. Motivation of the problem
2. Architectural overview
3. Modules
   • The Static Model
   • The Dynamic Model
   • The Commander
4. Scenario Deployment Example
5. Features and benefits summary
6. Future work
1. Motivation of the problem and goals

- Research group purpose
  - Leverage Spain's “Planet Lab”: PASITO
    - This is an inter-university project
    - WAN network, 15 sites in the start up
    - Support infrastructure for researches in networks and massive computing
1. Motivation of the problem and goals

- Distributed test scenarios issues and goals:
  - Stress Testing Scenario:
    “It involves testing beyond normal operational capacity, often to a breaking point, in order to observe the results”
  - DDS Stress Testing Scenario tasks:
    - Involves the creation of DDS applications with a high number of entities distributed across multiple nodes
    - Requires a way to establish relationship between these entities and associate test behaviors
    - Coordinate and synchronize the Life-cycle of large number of DDS entities

- We need fast ways to prototype DDS systems!
1. Motivation of the problem and goals

Why use **XML** to prototype DDS systems?

- Open, industry accepted standard
- Ideal for setting up applications and representing knowledge
- Supported by all common languages (C, C++, Java, C#...)
- Easily editable and transformable:
  - By humans: text-based format
  - By XML specific tools or IDE such as Eclipse
- Valid for describing DDS Entities parameters and DDS Entities relationship
1. Motivation of the problem
2. Architectural overview
3. Modules
   • The Static Model
   • The Dynamic Model
   • The Commander
4. Scenario Deployment Example
5. Features and benefits summary
6. Future work
2. Architectural Overview

How to describe the DDS test scenario?

- **Static Model**
  - XML description of the DDS entities

- **Dynamic Model**
  - Programmatic description of behavior (typically write and read) & listeners

Network node executing a DDS application

![Diagram showing DDS Application, Static Model, Dynamic Model, and DDS Global Data Space](image-url)
2. Architectural Overview

How to **synchronize** the nodes?

- Command Master node
- Command Slave nodes

![Diagram showing DDS Global Data Space and communication between Command Master and Command Slaves](image.png)
1. Motivation of the problem
2. Architectural overview
3. Modules
   • The Static Model
   • The Dynamic Model
   • The Commander
4. Scenario Deployment Example
5. Features and benefits summary
6. Future work
3. Modules

Main modules of the framework

The **Static Model**:  
- Provides a way of describing DDS entities and the relationships between them  
- Information to set up the DDS Entities  
- Location of the DDS Entities  
- Allows the massive deployment of applications across the network nodes

The **Dynamic Model**:  
- Provides a way of writing a programmatic description of the activities of the entities

The **Commander**  
- Deals with nodes and entities synchronization
3.1 Modules: the Static Model

Static Model components:

- The XML Parser is built with Expat libraries
- Document Type Definition (DTD), XML Schema (XSD) and Document Object Model (DOM)
- The DOM is composed of a specific objects set
- These objects are the DDS XML Entities:
  - Manage information for creating DDS entities and set their behavior
    - Stores DDS Entities data information: domain id, topic name...
    - A QoS profile name
      - We store a set of XML QoS profiles such as default, LowLatency, MaximizeThroughput...
    - Number of DDS entities to create
    - Behavior reference from the Dynamic Model
    - Listener reference and mask to apply
    - Thread's state information
3.1 Modules: the Static Model

**Static Model** components:
- DOM composed of **DDS XML Entity Objects**

DOM & browsing API

```
get_first_publisher()
```

```
get_xml_subscriber()
```

```
get_xml_topic()
```

```
get_next_datareader()
```
3.1 Modules: the Static Model

Static description of the scenario: XML description of a DDS Application (QoS and Dynamic Model tags are omitted here):

```
<dds>
  <deployment>
    <node>
      <address>10.10.1.100</address>
      <application>hello</application>
    </node>
  </deployment>
  <application name="hello">
    <participant name="participant1">
      <domain_id>0</domain_id>
      <topic name="hello_world">
        <dds_name>Hello world</dds_name>
        <type>String</type>
      </topic>
      <publisher name="publisher1">
        <datawriter name="writer1a">
          <datawriter_topic>hello_world</datawriter_topic>
        </datawriter>
      </publisher>
    </participant>
    ...
  </application>
  ...
  <subscriber name="subscriber1">
    <instances>2</instances>
    <datareader name="reader1">
      <datareader_topic>hello_world</datareader_topic>
      <instances>3</instances>
    </datareader>
    ...
  </subscriber>
</dds>
```

Deployment of the scenario based on roles, addresses
3.1 Modules: the Static Model

- **DDS QoS Profiles:**
  - A **QoS profile** groups a set of related QoS, usually one per entity.
  - A **QoS Library** is a named set of QoS profiles.

```xml
<qos_library name="MyTestLibrary">
  <writer_qos name="LowLatency">
    <history>
      <kind>DDS_KEEP_ALL_HISTORY_QOS</kind>
    </history>
  </writer_qos>
  <reader_qos name="HighAvailability">
    <latency>
      <deadline>
        <value>10</value>
        <kind>DDS_LOOSE_DEADLINE_QOS</kind>
      </deadline>
    </latency>
  </reader_qos>
</qos_library>
```
3.1 Modules: the Static Model

**Static Model components: the DDS XML Entities**

- We provide a **flexible API** for managing a scenario Application
- The set of DDS XML Entities provides an **API**:
  - wrap the DOM **browsing** API
  - **get/set** accessors methods (for retrieving & modifying the parsed data)
  - **create** & **destroy** DDS Entities instances
    - `xmlEntity.create.dds_entity(index, recursive)`
  - **start** & **stop** the entities behavior
    - `xmlEntity.start_behavior(index, recursive)`

- `index` refers to a DDS instance number inside the DDS XML Entity
- `recursive` allows to call to all the children's create, destroy, start & stop methods
3.1 Modules: the Static Model

Create a subset of DDS Entities:

- xmlParticipant.create_dds_entity(1, no_recursive)
- xmlPublisher.create_dds_entity(1, no_recursive)
- xmlDataReader.create_dds_entity(1)

Create all DDS Entities:

- xmlParticipant.create_dds_entity(1, recursive)
3.2 Modules: the Dynamic Model

- The **Dynamic Model** describes the behavior of the DDS entities that compose the Static Model.

- This description is done by programming a behavior method and configuring entity listeners.
  - Minimum DDS API knowledge is required.

- The **Dynamic Model** module provides:
  - A set of built-in behavior functions, listeners and data type support.
  - API for retrieving the methods and listeners.

- The user needs to include his methods and listeners into the Dynamic Model.
3.2 Modules: the Dynamic Model

How to **connect** the user **Dynamic Model** with the **Static Model**:

- The Dynamic Model is a shared library
- The user should specify where to look for the Dynamic Model

```xml
<datareader name="reader1">
  <datareader_topic>hello_world</datareader_topic>
  <behavior>
    <shared_library>mylibrary.so</shared_library>
    <function_name>write_hello_loop</function_name>
    <user_data>200</user_data> <!-- write period -->
  </behavior>
</datareader>

<datawriter name="writer1">
  <datawriter_topic>hello_world</datawriter_topic>
  <behavior>
    <shared_library>mylibrary.so</shared_library>
    <function_name>write_hello_loop</function_name>
    <user_data>200</user_data> <!-- write period -->
  </behavior>
</datawriter>
```
3.3 Modules: the Commander

- The **coordination** of the scenario is done by using DDS itself.
- It is composed of two DDS based classes:
  - **CommandPublisher**
  - **CommandSubscriber**

- The test framework provides DDS tools to the user:
  - **cmdmaster**: uses the a **CommandPublisher** instance to publish commands
    - Command line tool
    - Simple authentication via built-in phrase pass
    - Command Topic configured with QoS Owned, reliable...
  - **cmdslave**: uses a **CommandSubscriber** instance to get and process commands
    - Each node in the scenario must load this application

- Theses classes uses the Static and Dynamic Models API for deploying the test scenario
3.3 Modules: the Commander

- Current user's **Commands set:**
  - Commands can be applied to concrete entities
  - **Load** of the scenario XML file:
    - By specifying it as a command line argument to the DDS applications for each of the nodes
    - By publishing it to the nodes

<table>
<thead>
<tr>
<th>Command</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>load &lt;xml filename&gt;</td>
<td>load &quot;scenario.xml&quot;</td>
</tr>
<tr>
<td></td>
<td>load &quot;url://.../scenario.xml&quot;</td>
</tr>
<tr>
<td>create &lt;xml_entity_id&gt; &lt;dds_instance_number&gt;</td>
<td>create &quot;mydatawriter&quot; 2</td>
</tr>
<tr>
<td>createall</td>
<td>createall</td>
</tr>
<tr>
<td>destroy &lt;xml_entity_id&gt; &lt;dds_instance_number&gt;</td>
<td>destroy &quot;mydatawriter&quot; 2</td>
</tr>
<tr>
<td>destroyall</td>
<td>destroyall</td>
</tr>
<tr>
<td>start &lt;xml_entity_id&gt; &lt;dds_instance_number&gt;</td>
<td>start &quot;mydatawriter&quot; 2</td>
</tr>
<tr>
<td>startall</td>
<td>startall</td>
</tr>
<tr>
<td>stop &lt;xml_entity_id&gt; &lt;dds_instance_number&gt;</td>
<td>stop &quot;mydatawriter&quot; 2</td>
</tr>
<tr>
<td>stopall</td>
<td>stopall</td>
</tr>
<tr>
<td>terminate</td>
<td>terminate</td>
</tr>
</tbody>
</table>
1. Motivation of the problem
2. Architectural overview
3. Modules
   • The Static Model
   • The Dynamic Model
   • The Commander
4. Scenario Deployment Example
5. Features and benefits summary
6. Future work
The scenario is ready with `cmdmaster` and `cmdslove` running in the proper nodes.

- scenario.xml has two DDS applications
  - Application_radar, Application_traffic_control

Control Topic:

- DDS Global Data Space
- CommandPublisher
- Node1 CommandSubscriber
- Node2 CommandSubscriber
- Node3 CommandSubscriber
- control topic data
4. Scenario Deployment Example (II)

- Distribute the XML scenario across the nodes.
- The nodes will automatically select the application based on the deployment tag

Control Topic:
> load "scenario.xml"

```
<deployment>
  <node>
    <address>Node1</address>
    <application>App_radar</application>
  </node>
  <node>
    <address>Node2</address>
    <application>App_radar</application>
  </node>
  <node>
    <address>Node3</address>
    <application>App_traffic_ctr</application>
  </node>
</deployment>
```
Create Participants, Topics, Publishers and Subscribers
Create instance 1 of DataReader "reader_a"

Control Topic:
> create participants with participant's childs
> create "reader_a" 1

An XML-Based Approach to the Configuration and Deployment of DDS Applications
Javier Sanchez-Monedero, Javier Povedano Molina, Jose M. Lopez-Vega & Juan M. Lopez-Soler
Create instance 1 for all DataWriters with name "writer_a"

Up to this point, we can measure, for example, discovery time

Control Topic:
>create “writer_a” 1
4. Scenario Deployment Example (V)

- Start the behavior of some DataWriters
  - Typically, this is writing data

**Control Topic:**
> start "writer_a" 1

![Diagram of DDS Global Data Space with nodes and topics](image)

- **cmdmaster**: CommandPublisher
- **Node1**: App_radar, CommandSubscriber, writer_a:1
- **Node2**: App_radar, CommandSubscriber, writer_a:1
- **Node3**: App_traffic_ctr, CommandSubscriber, reader_a:1

Typically, this is writing data.
Create more entities and start their behavior

**Control Topic:**
- `create "writer_a" 2`
- `start "writer_a" 2`

---

**DDS Global Data Space**

- **cmdmaster**
  - CommandPublisher
  - Node1
    - **App_radar**
      - CommandSubscriber
      - `writer_a:1`
      - `writer_a:2`
  - Node2
    - **App_radar**
      - CommandSubscriber
      - `writer_a:1`
      - `writer_a:2`
  - Node3
    - **App_traffic_ctr**
      - CommandSubscriber
      - `reader_a:1`
4. Scenario Deployment Example (VII)

Stop all the entities and destroy them

Control Topic:
> stopall
> destroyall
Terminate the \textit{cmdslaves}

\begin{itemize}
  \item \textbf{Control Topic:}
  \begin{itemize}
    \item \texttt{>terminate}
  \end{itemize}
\end{itemize}
Presentation Outline

1. Motivation of the problem
2. Architectural overview
3. Modules
   - The Static Model
   - The Dynamic Model
   - The Commander
4. Scenario Deployment Example
5. Features and benefits summary
6. Future work
5. Features and benefits summary (I)

- Division between static and dynamic model allows:
  - Reducing user's source code.

- The flexibility of the framework API and commands design permits:
  - Measuring DDS entities creation and discovery times
  - Performing gradual increase of network load and nodes activity

- The XML description allows:
  - To control if every node can “see” all the other entities in the scenario.
    - Each node can collect discovery information and compare it with the XML scenario description it has
  - To change the DDS entities configuration, number of instances and behavior by modifying a few XML lines in a file
5. Features and benefits summary (II)

- The **Commander** module allows:
  - To distribute the scenario description by using DDS itself:
    - We don't need to worry about XML files synchronization
    - We do not depend on SSH or NFS to synchronize resources
    - We can deploy on a Wide Area Network
    - We can use our test tool for testing any platform supported by a DDS product, even if the platform does not support a file system
  - At the end, we are creating a fast-prototype tool for DDS applications
    - Allows testing any kind of scenarios before starting the real DDS application deployment
      - The user only needs to program some methods for integrating the prototype into his system.
  - We can reuse this platform for synchronize non DDS applications execution in distributed environments
1. Motivation of the problem
2. Architectural overview
3. Modules
   - The Static Model
   - The Dynamic Model
   - The Commander
4. Scenario Deployment Example
5. Features and benefits summary
6. Future work
6. Future work

- Establish a full plugin API for loading the Dynamic Model
- Extend the XML model:
  - Allow XML user types definition
  - Support full DDS features:
    - Network transports
    - Discovery peers
    - ...
- Improve authentication module for Command Topic
- Define a statistics generation Dynamic Model
- Add a GUI scenario visualization tool
- Feedback from the experience in Spanish's planet lab *PASITO*
Thank you very much!

Real-Time Innovations, Inc.
University Program

www.rti.com

tstc.ugr.es

This work has been partially supported by the Spanish Government through MEC (Project TSI2005-08145-C02-02, FEDER funds 70%)