Web-Enabled DDS
Accessing Real-Time DDS Data From Web-Based Clients

Gerardo Pardo-Castellote, Ph.D.

The Real-Time Middleware Experts

Fabrizio Bertocci
Principal Engineer, Real-Time Innovations
Fabrizio.bertocci@rti.com

Andrea Iannitti
Software Engineer, Real-Time Innovations
andrea.iannitti@rti.com

Gerardo Pardo-Castellote, Ph.D.
Co-chair OMG DDS SIG
CTO, Real-Time Innovations
gerardo.pardo@rti.com
Outline

- Motivation & Use-Cases
- Part I - Service Model
  - Why a new model
  - Service Objects
  - Service Behavior
  - Implementation
- Part II – Service Access API
  - WS-DDS
  - REST-DDS
  - DDS-Eventing
- Demo
- Conclusion
Use cases

- Access DDS without loading/linking libraries
  - JavaScript, Flash, Apache, AppServers…
  - Light-weight adaptor (thin client, minimal memory)
  - Any OS, No OS… Universal access from any web-enabled device

- Access DDS from any Language (with Web Support)
  - Scripting Languages
  - Pure Native environments .NET, BPEL

- Disconnected/Stateless clients / fire & forget
  - CGI scripts
  - Command-line tools
  - Tools like Nagios, …

- Access from Web-enabled Environments
  - ESBs
  - EAI
  - .NET tools for WS…
Desired Solution: **Web Enabled DDS**

A service that exposes DDS Global Data over Web Protocols: Applications can interact with DDS directly over the Web

No need for bridges or special bindings for scripting languages.
Define Web-Enabled DDS as a PIM including
- A Service Object Model that
  - Exposes the DDS Object Model to Web Clients
  - Defines how Web Clients access DDS Capabilities
  - Provides a Session Model supporting disconnected web clients
  - Provides an Access Control Model for web-clients accessing the service

Define several equivalent PSMs allowing access to the gateway
- Each PSM as a separate compliance point:
  - REST/HTTP, WSDL/SOAP, WS-Eventing/SOAP, RSS, XMPP
Examples:

- Access DDS data from a web browser:

  Web Browser
  (Flash, JS, Java…)

  Internet
  DDS-WS

  DDS

  PyShapes
  (Python)

  HTTP
Examples:

- Disconnected/Stateless clients

```
#!/bin/sh
set session=`ddsws_login MyUserName MyPassword`
set pub=`ddsws_create_publication Foo Bar...
ddsws_pub_write $pub MySample
```
Demo: DDS as a Web Service

- **Web Enabled Protocols**
  - HTTP
  - SOAP
  - DDS-RTPS

- **Hi-Performance Embedded Real-Time**
  - DDS Web Service
  - Generic Web Client (.NET / Java)
  - IONA Web Client
  - IONA Artix Transports

- **Start**
  - DDS Web Service
  - Global Data Space RTI, DDS, Real-Time
Examples:

Air-traffic control & airport schedule information

DDS

DDS-WS
Examples:

Remote System Monitoring
Implementation Details

- Define a Service Model WebEnabled DDS
  - What operations it supports
  - What is the access control
  - Service Object Behavior

- Map Service operations to Web-Enabled protocols
  - WS-DDS/SOAP
  - REST-DDS/HTTP
  - RSS
  - XMPP
  - WS-Eventing/SOAP
  - …
Outline

- Motivation & Use-Cases
- **Part I - Service Model**
  - Why a new model
  - Service Objects
  - Service Behavior
  - Implementation
- **Part II – Service Access API**
  - WS-DDS
  - REST-DDS
  - DDS-Eventing
- Demo
- Conclusion
Why not use DDS as the Service Model?

- **Simplify/Reduce API**
  - Expected use-case is strongly “managed” access to the data
    - Topics, Types, QoS, …

- DDS Listeners do not map easily to Web Client/Server

- DDS has no access control model.
  - Need that for remote access

- DDS objects do not survive across sessions
  - In fact, no concept of session in DDS
Service Model

- Single Object

- Concepts: User, Session, Topic, Subscription, Publication, NotificationEndpoint

- Behavior
  - Access Control
  - Session Management
  - Topics
  - Subscription, Publication
  - NotificationEndpoint
### WSDDSService

<table>
<thead>
<tr>
<th>Method</th>
<th>Parameters</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>+Login</td>
<td>user_name : String, password : String</td>
<td>long long</td>
</tr>
<tr>
<td>+Logout</td>
<td>session_id : long</td>
<td>int</td>
</tr>
<tr>
<td>+CreateSubscription</td>
<td>session_id : long, domain_id : int, topic_name : String, type_name : String, type_schema : String, qos_profile : String, cf_exp : String, data_encoding : int</td>
<td></td>
</tr>
<tr>
<td>+GetSubscriptions</td>
<td>session_id : long, domain_id : long</td>
<td>SubscriptionsSeq</td>
</tr>
<tr>
<td>+RemoveSubscription</td>
<td>session_id : long, subscription_id : long</td>
<td>int</td>
</tr>
<tr>
<td>+CreatePublication</td>
<td>session_id : long, domain_id : int, topic_name : String, type_name : String, type_schema : String, qos_profile : String</td>
<td>long</td>
</tr>
<tr>
<td>+GetPublications</td>
<td>session_id : long, domain_id : long</td>
<td>PublicationsSeq</td>
</tr>
<tr>
<td>+RemovePublication</td>
<td>session_id : long, publication_id : long</td>
<td>int</td>
</tr>
<tr>
<td>+Read</td>
<td>session_id : long, subscription_id : long, timeout : int</td>
<td>SamplesSeq</td>
</tr>
<tr>
<td>+Write</td>
<td>session_id : long, publication_id : long, sample : String, data_encoding : int</td>
<td></td>
</tr>
<tr>
<td>+AddNotificationEndpoint</td>
<td>session_id : long, subscription_id : long, notif_style : NotificationStyle, data_encoding : int, port_number : long, ip_address : String</td>
<td>int</td>
</tr>
<tr>
<td>+GetNotificationEndpoint</td>
<td>session_id : long</td>
<td>NotificationEndpointInfoSeq</td>
</tr>
<tr>
<td>+RemoveNotificationEndpoint</td>
<td>session_id : long, notif_endpoint_id : long</td>
<td>int</td>
</tr>
</tbody>
</table>

---

**SubscriptionInfo**

- id : long
- domainId : long
- topic_name : String
- qos_profile : String
- status : long

**PublicationInfo**

- id : long
- domainId : long
- topic_name : String
- qos_profile : String
- status : long

---

**Only one object ➔ Trivial to use!!**
<table>
<thead>
<tr>
<th>Authentication</th>
<th>Reading</th>
<th>Writing</th>
<th>Notification</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Login</td>
<td>● CreateSubscription</td>
<td>● CreatePublication</td>
<td>● AddNotificationEndpoint</td>
</tr>
<tr>
<td>● Logout</td>
<td>● RemoveSubscription</td>
<td>● RemovePublication</td>
<td>● RemoveNotificationEndpoint</td>
</tr>
<tr>
<td></td>
<td>● GetSubscriptions</td>
<td>● GetPublications</td>
<td>● GetNotificationEndpoints</td>
</tr>
<tr>
<td></td>
<td>● Read</td>
<td>● Write</td>
<td>● Notify</td>
</tr>
</tbody>
</table>
Authentication & Access control

- **Authentication:**
  - IP Address and/or user name & password are provided for authentication
  - If login is successful, a session token is produced with a time-to-live that is updated on every Service call.
  - Token is used in all the subsequent calls.

- **Access control:**
  - Individual users can be allowed or denied to:
    - Join a domainId
    - Create a topic
    - Subscribe or Publish a Topic
    - Add NotificationEndpoint
package Database [ Authentication ]

```
<<table>>
User
<<key>>-user_name : String
-password : String
-max_sessions_count : int
-inactivity_timeout : long
-session_duration_limit : int
-defaultEncodingStyle : int
```

```
<<table>>
Role
<<key>>-role_name : String

ROLE_TOPIC_RIGHTS
1..*

权利 : bitmask

ROLE_DOMAIN_RIGHTS
1..*

权利 : bitmask
```

```
<<table>>
Topic
<<key>>-topic_name : String
-domain_id : int
```

```
<<table>>
Domain
<<key>>-domain_id : int
```

```
<<table>>
IP_Address
<<key>>-IP_Address : String
```

```
USER_IPADDRESS
1..*
```

```
USER_ROLE
0..
```

```
ROLE_TOPIC_RIGHTS
1..*
```

```
ROLE_DOMAIN_RIGHTS
1..*
```

```
domain_id
```

```
```
```
```
```
Sessions Management

- Client applications can open and close DDSWebSvc sessions using Login and Logout respectively.
- Only a specific number of sessions can be kept open for a specific user.
  - This parameter is configurable per user.
- DDS Entities belong & exclusive to a user.
  - Therefore all the sessions belonging to the same user can interact with the same set of entities, even those created within a different session.
  - Methods for getting the entities are provided.
- Sessions expire after a timeout if not renewed.
  - The session gets renewed just calling any operation.
- Session expiration does not imply DDS Entity destruction.
  - Configurable Inactivity Timeout \([0, \infty[^\)
DDSWebSvc does not include API for creating DDS DomainParticipants, Publishers, Subscribers and Topics.

For simplicity these Entities are created automatically the first time you call CreateSubscription or CreatePublication within any <userName, domainId> context.

Since the interactions among Web Services are stateless, DDSWebSvc Server keeps the state of the DDS environment. Entities persist across sessions.

Entities are reclaimed when the user becomes inactive (no sessions open anymore) and a specific timeout expires.

QoS policies can be set by means of QoSProfiles.
DDSWebSvc additions beyond DDS model

- Authentication Access Control
- MultiSessions Access System
- Partially Defined Entities Management System
- Four Delivery Modes
  - Reading
    - Polling or Server Push
  - Notification
    - Push or Pull
- Recoverability
  - A WSDDS server has its own state that must be recovered in case of crashes.
- Data Type Format Transformation (temporary)
  - DynamicData Object $\leftrightarrow$ XML
  - DynamicData Object $\leftrightarrow$ JSON
Data Type Transformation

- Data is delivered and received in a type-neutral form through JSON or XML
- Conversion from XML Dynamic Data and vice versa is done automatically by WSDDS

Sample (CDR)
- 143321
- John Doe
- 1234
- 1345.90

Type definition
```c
struct Account {
    long id;
    string name;
    long pin;
    double balance;
};
```

WS-DDS

JSON
```json
{
    #id: 143321,
    name:"John Doe",
    pin: 1234,
    balance: 1345.90
}
```

XML
```xml
<sample>
    <id key='true' type='long'>143321</id>
    <name type='string'>John Doe</name>
    <pin type='long'>1234</pin>
    <balance type='double'>1345.90</balance>
</sample>
```
Behavior: DDSEntity Creation

- Scenario 1: An application tries to create entities with a “partial” definition or their Topics
- Scenario 2: An application tries to create an entity on a topic. The topic doesn’t exist and the user is not allowed to create it
- In these two cases, entities are created under a “Pending” status
- DDSWebSvc includes some algorithms that dynamically resolve pending entities
- Entities Discovering (Built-in types) plays a fundamental role
Behavior: Read Data Non-Blocking

- Client periodically call read() to get the available samples.
- All the received samples are then returned.
- Read() is not blocking.
Behavior: Read Data - Blocking

- Client call `read()`
- If no samples are available, the calls block without sending back the HTTP response.
- When samples are received, WS-DDS completes the request
- The client specifies the timeout
The client implements a SOAP server implementing a Notification Callback mechanism.

Notification Endpoints can be added to specific subscriptions, the client passes its IP and port number.

WS-DDS will notify the client of new samples through the Notification SOAP server.

Two delivery modes: Push and Pull.
Service Configuration

How?
- XML file like other RTI Services?
- Admin User Tool?
  - Database interaction
  - Offline vs Online (Requires more logic)
  - Interfaced by
    - Command Line application
    - Web Application
  - Should be harmonious to other RTI Services

Things to configure
- Service parameters
- Per User parameters
  - Sessions per User, Multithreaded notification, Session lifetime, Inactivity Timeout…etc.
Motivation & Use-Cases

Part I - Service Model
- Why a new model
- Service Objects
- Service Behavior
- Implementation

Part II – Service Access API
- WS-DDS
- REST-DDS
- DDS-Eventing

Demo

Conclusion
Part II -- Service Access via WS-DDS

From W3C:

“A Web Service is a software application identified by a URI, whose interfaces and bindings are capable of being defined, described and discovered as XML artifacts. A Web Service supports direct interaction with other software agents using XML-based messages exchanged via Internet-based protocols”

Here when W3C says “WebService” they really mean WS-*
Other people talk about “RESTful Web Services” (O’Reilly book)
<message name="Read">
  <part name="subscriptionId" type="xsd:int"/>
  <part name="timeout" type="xsd:int"/>
</message>

<message name="ReadResponse">
  <part name="returnCode" type="xsd:int"/>
  <part name="dataSamplesSeq" type="wsdds:DataSampleSeq"/>
</message>

<portType name="wsddsPortType">
  <operation name="Read">
    <input message="tns:Read"/>
    <output message="tns:ReadResponse"/>
  </operation>
</portType>

<binding name="wsddsBinding" type="tns:wsddsPortType">
  <port name="wsddsPort" binding="tns:wsddsBinding">
    <SOAP:address location="http://localhost:18038"/>
  </port>
</binding>

<service name="wsdds">
  <port name="wsddsPort" binding="tns:wsddsBinding">
    <SOAP:address location="http://localhost:18038"/>
  </port>
</service>

wsdds.wsdl
WS-* – Web Services - SOAP

HTTP Message

SOAP envelope

SOAP header
- header block

SOAP body
- body block

TODO: simples soap message
RESTful is a design methodology valid for many protocols (HTTP, TCP, UDP…)

Resource Oriented Architectures
- Addressability, Statelessness, Connectedness

HTTP rich enough to model resource mgmt
- Any resource is mapped to an URI.
- HTTP ops (GET, POST, PUT, DELETE…) map well to resource ops: get, modify, add and remove “resources”.

Rest/DDS requires a mapping from entities, session, data…to URIs.

HTTP response code helps!

SOAP is completely bypassed
RESTful – Example (Create/Write)

- HTTP Request and Response can pass additional argument using both the body and the header
- Example:
  CreateSubscription (32, Square, ShapeType, ShapeTypeSchema, QoSDefault, “x < 100”, JSON_ENCODING) ->

  PUT www.rti.com/wsdds/Andrea/32/Square/Subscription
  Inside the HTTP Body: <Subscription>
  
  <typeName> ShapeType </typeName>
  <typeSchema> ShapeTypeSchema </typeSchema>
  <qosProfile> QoSDefault </qosProfile>
  <cfExp> "x < 100" </cfExp>
  <encodingStyle> JSON_ENCODING</encodingStyle>

  </Subscription>

- The response would contain the subscriptionId that can be used to read the data.
- The HTTP response code 200 tells me that subscription has been successfully created.
- Resources should be linked! A response could contain links to:
  - All the resources belonging to the same:
    - topic
    - domain,
    - user
RESTful – Example (Read)

- **HTTP GET** with Response containing data in XML or JSON

- **Example:**
  Read (sessionId, subscriptionId, “x < 100”, MAX_SAMPLES, TIMEOUT) →

  Option 1: (High-Performance; full access to all Entities)
  GET [www.rti.com/wsdds/Subscription]? SessionId=SID & SubsId=SUBID &FILTER=X<100 & MaxSamples=MAXSAMPLES & TimeOut=TIMEOUT

  Option 2: (More RESTful. Simpler, Human & Browser friendly)
  GET [www.rti.com/wsdds/DOMID/TOPICNAME/Subscription]? &FILTER=X<100 & MaxSamples=MAXSAMPLES & TimeOut=TIMEOUT

- For Option2 SessionId is in the cookie. Limited to one User per client, one Subscription per User per Topic

- The response would contain the data.

- The HTTP response code 200 returns data (can be empty sequence)

- HTTP retcode 404 means the Subscription was not found
WS – Eventing specification describes a protocol that allows Web Services to subscribe to or accept subscription for event notification messages.

It’s part of the WS - *

Web Services involved
- Subscriber Event Sink
- Event Source
- [Subscription Manager]

Delivery mode
- Push (only supported)
- Pull
WS-Eventing Subscription manager

- Subscription manager is optional
- Event Source may define to be itself the Subscription manager or designate another Web Services
- The topic is the service itself
Outline

- Motivation & Use-Cases
- Part I - Service Model
  - Why a new model
  - Service Objects
  - Service Behavior
  - Implementation
- Part II – Service Access API
  - WS-DDS
  - REST-DDS
  - DDS-Eventing
- Demo
- Conclusion
Outline

- Motivation & Use-Cases
- Part I - Service Model
  - Why a new model
  - Service Objects
  - Service Behavior
  - Implementation
- Part II – Service Access API
  - WS-DDS
  - REST-DDS
  - DDS-Eventing
- Demo
- Conclusion
Conclusion

- Web-Enabled DDS would greatly extend the class of applications that can access the DDS global data space and benefit from its performance and scalability
  - Web client applications, including JavaScript
  - Thin clients with no DDS code loaded
  - Scripting languages (Perl, PHP, Python, Ruby, …)
- RTI has already developed a prototype showing the power of this approach
- WS-* is lacking good pub-sub standards, this approach could fulfill that gap
  - The OMG is already working on a Web-Enabled DDS RFP. This is a great opportunity to formalize this approach