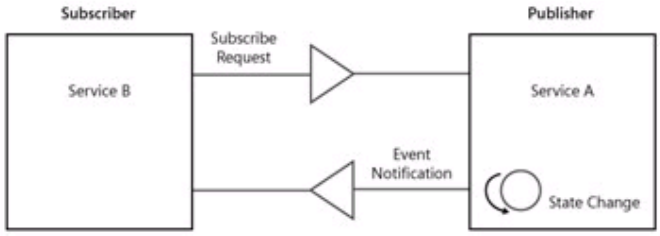
Introduction to Game Programming and Robotics

Unit # 7

Subscription Handling

- DSSP allows services to receive event notifications from other services.
- With a subscription involving two services (e.g., Service A and Service B).
- Service A could provide the state and serve as the publisher. Service B, the subscriber, requests to be notified whenever the state for service A changes.
- You will need to add subscription code to both services.
- Service A will need to support subscriptions, and Service B will need to initiate the subscribe request. Service B will then receive an event notification when the state for Service A changes



Subscription Manager

- MSRS provides a system service named SubscriptionManager.
- The SubscriptionManager service is responsible for forwarding notifications to the appropriate subscribers and maintaining a list of those subscribers.
  - using submgr = Microsoft.Dss.Services.SubscriptionManager;
- The SubscriptionManager service functions as a partner service for each service that needs to support subscriptions.

Subscription Manager (Cont'd)

- The SubscriptionManager service is declared using the *Partner* attribute. For example, the following code is used to declare the SubscriptionManager as a partner for the publisher service:
  - [Partner("SubMgr", Contract = submgr.Contract.Identifier, CreationPolicy = PartnerCreationPolicy.CreateAlways)]
  - private submgr.SubscriptionManagerPort _submgrPort = new submgr.SubscriptionManagerPort();

Subscribe Class

- For a service to allow subscriptions, it must provide a subscribe message that supports SubscribeRequestType and SubscribeResponseType, such as the following:
  - public class Subscribe : Subscribe<SubscribeRequestType, PortSet<SubscribeResponseType, Fault>> { }

Subscription Service

- Once the message has been created, it must be added to the PortSet for the main operations port.
- There will also need to be a service handler for the subscribe operation.
- This handler is responsible for adding subscribers to the list using the SubscriptionService. Assuming that your handler accepts a parameter named sub, the *DsspServiceBase.SubscribeHelper* method can be placed in the SubscribeHandler:
  - SubscribeHelper(_submgrPort, sub.Body, sub.ResponsePort);

Notification to Subscribers

- The last thing the publisher needs to do is send notifications to subscribers whenever the state has changed.
- Code to initiate a send notification must be added anywhere the state is updated.
- For example, the following code could be added to a ReplaceHandler:
  - base.SendNotification(_submgrPort, replace);

Operations that Generate Notifications

- The operations that do generate events are as follows:
  - Delete
  - Drop
  - Insert
  - Replace
  - Update
  - Upsert

Configuring the Subscriber

- After the publisher service is configured, you will need to add code to the subscriber service.
- The first thing to do is reference the proxy assembly file for the publisher service.
- This is because services communicate with each other through their proxy files.
- The proxy assembly is referenced by right-clicking References from Solution Explorer and selecting Add Reference.
- You will then browse to the bin subdirectory for the MSRS installation and locate the proxy.dll file for the publisher service.

Configuring the Subscriber (Cont'd)

- To access the publisher class easily, you may add a namespace alias declaration to the subscriber's implementation class.
- You will need to add a Partner declaration, similar to the one created in the subscriber service. This will allow the subscriber service to access the *Subscribe* operation that was added to the publisher service.
- This is what allows the subscriber to receive notifications when the state for the publisher service changes.
- The last thing to do is to add code to the subscriber service that calls the Subscribe operation.

Tutorial # 2

- This tutorial shows how to use a contact sensor to start and stop a motor.
- The tutorial is based on Tutorial # 1.

Tutorial # 2 – Step 1

- Add a reference to Robotics.Common.Proxy to your project.
- Add following lines
  - using motor = Microsoft.Robotics.Services.Motor.Proxy;
  - using bumper = Microsoft.Robotics.Services.ContactSensor.Proxy;

Tutorial # 2 – Step 1 (Cont'd)

- Create partnership with the Motor service
  - [Partner("motor", Contract = motor.Contract.Identifier, CreationPolicy = PartnerCreationPolicy.UseExisting)]
    private motor.MotorOperations _motorPort = new motor.MotorOperations();
- Create partnership with the Bumper service
  - [Partner("bumper", Contract = bumper.Contract.Identifier, CreationPolicy = PartnerCreationPolicy.UseExisting)]
    private bumper.ContactSensorArrayOperations _bumperPort = new bumper.ContactSensorArrayOperations();

Tutorial # 2 – Step 2

- Add MotorOn variable to service state:

```
[DataContract]
public class RoboticsTutorial2State
{
    [DataMember]
    public bool MotorOn;
}
```

- Add manifest info to the project, subscribe to bumper service and add a bumper handler as was done in Tutorial # 1.

Tutorial # 2 – Step 3

```
/// <param name="notification">Update notification</param>
private void BumperHandler(bumper.Update notification)
{
    string message;
    if (!notification.Body.Pressed) return;
   _state.MotorOn = !_state.MotorOn;
   // Create a motor request
   motor.SetMotorPowerRequest motorRequest = new motor.SetMotorPowerRequest();
    if (_state.MotorOn)
    {
        motorRequest.TargetPower = 1.0;
        message = "Motor On";
    }
    else
    {
        motorRequest.TargetPower = 0.0;
       message = "Motor Off";
    } // Send the motor request
    _motorPort.SetMotorPower(motorRequest);
    // Show the motor status on the console screen
   LogInfo(LogGroups.Console, message);
}
```

Tutorial # 2 – Step 3 (Cont'd)

- The previous code does the following:
  - Check whether the bumper is pressed. If the event that occurred was a bumper release, the **BumperHandler** method returns.
  - If there was no bumper release, the **BumperHandler** method toggles the state of the motor, turning it on or off.
  - Depending on the new motor state value, the **BumperHandler** method then creates a **SetMotorPower** request message to set the motor power to either 0% or 100%.
  - The **BumperHandler** sends the **SetMotorPower** message.
  - and then displays the action that was taken on the console window.