Enterprise Messaging the JMS Way

Few Words About The Speaker

- Marian Edu, working in IT industry since 1997 in areas wagging from system administration, business application development, database design, application and solution architecture to business intelligence and enterprise data warehouse implementation.

- Independent consultant
  - Consulting Services for Progress OpenEdge
  - Business Intelligence, Data Integration
  - Framework Development & Tooling
  - marian.edu@ganimede.ro

Enterprise Integration

- Distributed computing environments
- System Integration
- Application Integration
- Data/Information Integration
- Business Integration
- Enterprise Integration

Application Integration

- Remote Procedure Call (RPC) – 1976 RFC 707
- Common Object Request Broker Architecture (OMG)
- Distributed Component Object Model (Microsoft)
- Web Services
- Message Oriented Middleware (MOM)
- Enterprise Service Bus

Enterprise Messaging

- A form of loosely coupled distributed communication (exchange of messages between software components)
- Replace tightly coupled communication (TCP sockets, CORBA or RMI) by the introduction of an intermediary component.
- Allows software components to communicate 'indirectly' with each other.
- Lift the need for message senders to have precise knowledge of their receivers

Message Oriented Middleware

- Introduced to respond the applications integration needs
- Removes the need for application-to-application adapters
- Adds asynchronous call support
- Message queue (persistence, routing, transformation)
- Lack of standards, each vendor with its own implementation, programming interface API

Java Message Service

- Java API specification is a messaging standard that allows application components to create, send, receive, and read messages
- Part of the Java 2 Platform, Enterprise Edition
- Developed under the Java Community Process
- It allows the communication between different components of a distributed application to be loosely coupled, reliable, and asynchronous.
- Version
  - JMS 1.0 - June 2001
  - JMS 1.1 - March 2002

OpenEdge Message Service

- Simply a 'translation' of JMS for OpenEdge ABL
- Still an Interface (mostly)
- Program to the Interface
- Vendor Independent
- Wrapper over 'default' SonicMQ Adapter API

Message Service Interface

- Message
- Queue (Point-to-Point)
- Topic (Publish-Subscribe)
- Connection Factory
- Session

Message Service Providers

- Sonic MQ
- IBM WebSphere MQ
- Oracle Enterprise Messaging Service
- Tibco Enterprise Messaging Service
- Storm MQ

- Open Source
  - Apache Active MQ
  - Rabbit MQ
  - HornetQ (JBOSS)

OpenEdge Providers?

- Sonic MQ Adapter
- REST
- Wire Level Protocol
  - Stomp
  - Advance Message Queuing Protocol – AMQP
  - OpenWire
  - XMPP (Jabber)

OpenEdge Message Service

- Flusso Stomp Adapter (OEHive)
- JMS Interface (javax.jms)
- ABL Implementation
- Provider Specific Implementation
  - SonicMQ Adapter
  - Stomp

What Does it Change

- Application code use JMS vs. SonicMQ Adapter API
- Isolate the messaging component within application
- Open the application for other providers
- Object Oriented vs. Persistent Procedures
- Strong Typing

Point-to-Point Session (Queue)

SonicMQ Adapter API

```
RUN jms/pptsession.p PERSISTENT SET sessionH
                          ("-SMQConnect").
RUN beginSession in sessionH.
```

JMS

```
qconn=QueueConnectionFactory:createQueueConnection().
qsess=qconn:createQueueSession().
qconn:start().
```

Message Types

SonicMQ Adapter Specific

- XML Message
- TempTable Message
- DataSet Message
- Multipart Message

JMS

- Object Message

SDK

- Lang: Exception, Array, BitWise, Primitive Wrappers
- Utils: Collection, List, Set, HashMap
- IO: Input/Output Streams, Buffered, Data Stream
- Net: Socket, Socket Input/Output Streams

Dive Into Code

WHY USE AN INTERFACE

- It isolates the messaging component within Application
- Makes your Application provider independent
- Supports plug-able providers
- It does not have to fully implement JMS
- It does not even have to be JMS

Questions?

Thank-you!

Enterprise Messaging the JMS Way

Few Words About The Speaker

- Marian Edu, working in IT industry since 1997 in areas wagging from system administration, business application development, database design, application and solution architecture to business intelligence and enterprise data warehouse implementation.

- Independent consultant
  - Consulting Services for Progress OpenEdge
  - Business Intelligence, Data Integration
  - Framework Development & Tooling
  - marian.edu@ganimede.ro

Enterprise Integration

- Distributed computing environments
- System Integration
- Application Integration
- Data/Information Integration
- Business Integration
- Enterprise Integration

Aims to connect and combines people, processes, systems, and technologies to ensure that the right people and the right processes have the right information and the right resources at the right time (Brosey et al. 2001 - Grand Challenges of Enterprise Integration)

Challenges: Customer Responsive Enterprises and Totally Connected Enterprises

Early days of computing in manufacturing industry - Computer Integrated Manufacturing (CIM) is about operations integration.

Vertical Integration: the process of integrating subsystems according to their functionality (silos)

Star Integration:integration of the systems where each system is interconnected to each of the remaining subsystems (Spaghetti)

Horizontal Integration: a specialized subsystem is dedicated to communication between other subsystems (ESB).

Application Integration

- Remote Procedure Call (RPC) – 1976 RFC 707
- Common Object Request Broker Architecture (OMG)
- Distributed Component Object Model (Microsoft)
- Web Services
- Message Oriented Middleware (MOM)
- Enterprise Service Bus

RPC – RFC 707 1976, Sun RPC (UNIX), Java RMI, JSON-RPC

CORBA (1.0 late 1991)→ language/platform neutral RPC, intermediate object request broker (ORB), any system, any language, IDL (interface definition language) → WSDL. Wire level protocol GIOP (general inter orb protocol). Freedom of technology, strong data-typing, compression.

DCOM Microsoft version of Corba, implementation for Linux/Java (Wine/Samba/JInterop) → .Net

Web Services: SOAP → XML-RPC/HTTP, WSDL (web service description language), moving to REST (representational state transfer)

MQ – Unix, Bea (Tuxedo) → Oracle (AT&T, 1983 - LMOS Telco mainframe), high availability & scalability, data dependent routing (route depending on message content), transaction coordination
IBM MQ Series Integrator → WebSphere Message Broker, Microsoft MQ

ESB – SOA: MQ + Web Services (message oriented services)

Synchronous(RPC, request/response) vs. Asynchronous(MQ) aspect of inter-application communication
Procedural Programming vs Inversion of Control (don't call us we'll call you)

Enterprise Messaging

- A form of loosely coupled distributed communication (exchange of messages between software components)
- Replace tightly coupled communication (TCP sockets, CORBA or RMI) by the introduction of an intermediary component.
- Allows software components to communicate 'indirectly' with each other.
- Lift the need for message senders to have precise knowledge of their receivers

MOM is all sending and receiving messages between distributed systems

The middleware consist of a distributed communications layer that insulates the application developer from the details of the various platforms, operating system and network protocols used for application integration.

This is where the shift take place compared with the RPC one, MOM are essentially asynchronous (even if messages can be grouped in transactions to mimic the response-request metaphor).

Problem was, each vendor with its own implementation: application programming interface, management tools, message transport protocol – vendor lock-in.

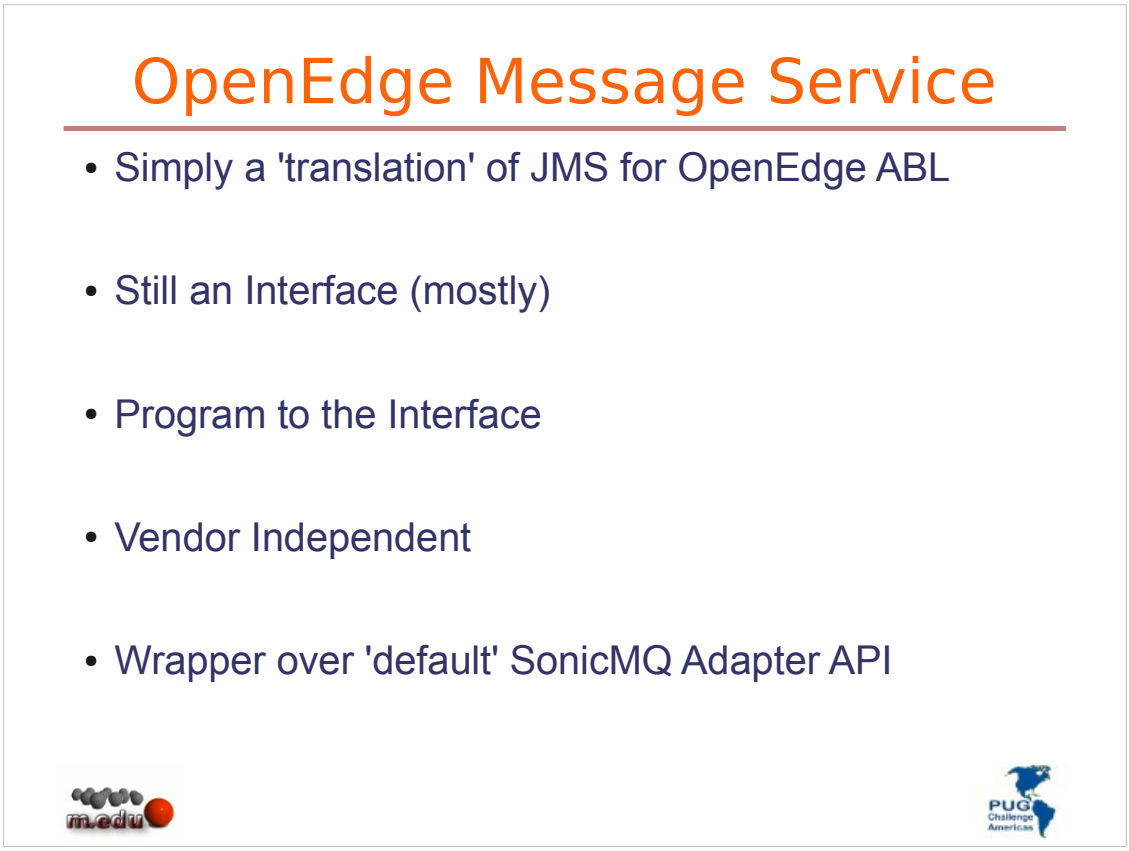
Java Message Service

- Java API specification is a messaging standard that allows application components to create, send, receive, and read messages
- Part of the Java 2 Platform, Enterprise Edition
- Developed under the Java Community Process
- It allows the communication between different components of a distributed application to be loosely coupled, reliable, and asynchronous.
- Version
  - JMS 1.0 - June 2001
  - JMS 1.1 - March 2002

It's a messaging standard, API level interface definition for MOM (MQ) providers.

The standard is implemented by most MOM vendors and aims to hide the particular MOM API implementations; however, JMS does not define the format of the messages that are exchanged, so JMS systems are not interoperable (with each other).

The main reason was to be able to have vendor independent applications – the same application can work with different messaging provider with no need to re-write the whole part that makes use of messaging functionality.



The need for a vendor independent messaging API in Open Edge is not any different from what they experienced in Java.

Sonic MQ – the default messaging provider, not that integrated as one would expect, still the default choice for messaging in ABL world.

While SonicMQ fully implements JMS specifications, the ABL implementation tries to follow but not that close given the non-OO approach taken.

Having an interface in place makes the application truly vendor independent and by that opens more markets for the ISV: when cost is a problem, when a messaging provider is already in place.

Message Service Interface

- Message
- Queue (Point-to-Point)
- Topic (Publish-Subscribe)
- Connection Factory
- Session

Basic concepts of messaging service API, it's all about messages. There are several types of messages defined in JMS:
Text – pass text only data load
Map – key/values
Byte – binary message
Object – serialized object (Java)

Two messaging domains
- P2P: queue, senders/receivers, messages kept in queue till consumed or expire, each one consumer
- PUB/SUB: topics, publishers/subscribers, messages kept till all subscribers get it, one message more consumers

Access to messaging services are done by establishing connections through abstract connection factories (Queue/Topic). Those are administrative objects, configurable by administrators (part of infrastructure). There is no naming policy enforced but it's recommended that those can be placed in the JNDI name space (java naming and directory interface).This looks a bit strange as when making a connection we expect to provide all connection information (it is like that when connection is made using SonicMQ adapter), this is different from the approach taken in JDBC interface where while no naming convention is enforced but still the driver manager finds the suitable driver given the connection url.

Session is a single-threaded context used to produce and consume messages. Given the single-thread model of Progress client most of the reasons for which sessions exists in Java world does not really apply. Each connection can have multiple sessions, on each session groups of producers and consumers work can be combined in atomic units (transactions) – this means that a session can block till a consumer receives the expected response (mimic the request/response), only the session thread will block.

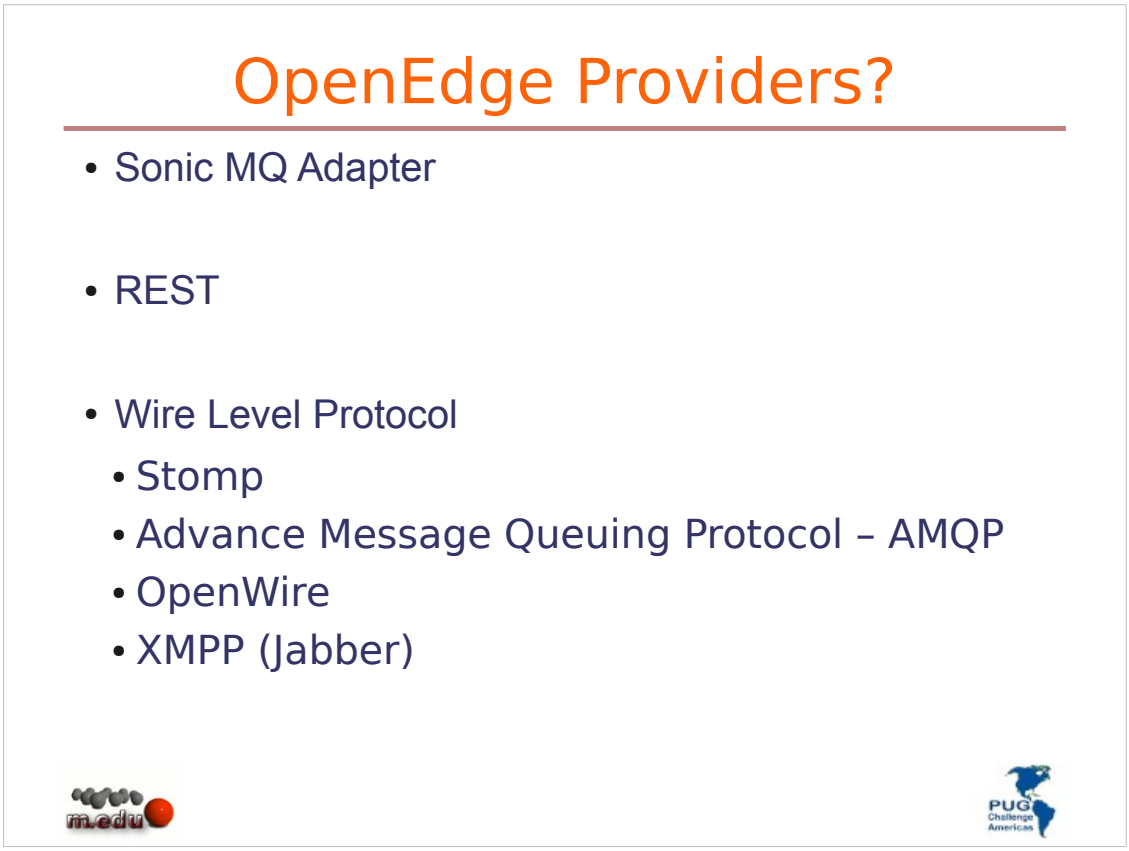
There are a great number of providers in the area of enterprise massaging, both commercially and open-source.

Beside the big fish like Sonic, IBM, Oracle that you might find in any large companies there are a number of interesting options like that StormMQ which is cloud-based and offered as monthly subscription.

RabbitMQ is developed by a division of VMWare while the major open-source Java application server providers have their own MQ product: Apache and Jboss, not that PSC also offer commercial support for Apache MQ (FUSE Source).

While all implements the JMS interface, most of them are really more than just message queues (MQ). With the advent of SOA the old 'message queues' providers were re-discovered and used as the message back-bone of the ESB (Enterprise Service Bus), mainly for adding asynchronous processing for RPC (especially XML-RPC, aka web-services).

Well, many will argue that there is more in this big-sounding term ESB... yes, it helps defining service interfaces and act as a service broker by handling all the service discovery and integration job that used to be in charge of the client but at its core it's still a message oriented middleware. Eventually, in the SOA world the loosely-coupled characteristic of MOM is seen as a drawback, a service need to be self-describing and it can't simply process any kind of messages, those need to have a predefined structure as well.



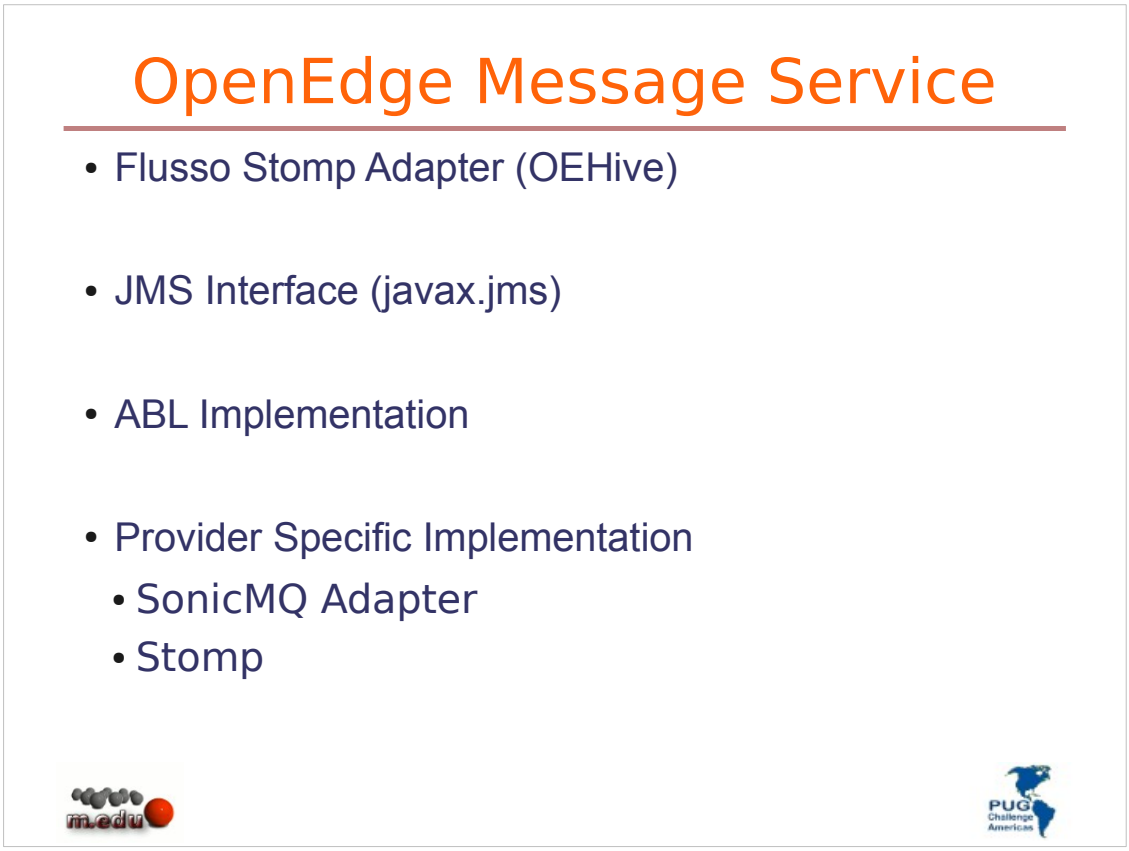
While all message service providers does implement the JMS Java interface as well as providing client API for a number of other languages like C/C++, .Net, popular scripting languages (Perl, Python, PHP)... some even for Cobol, there is really only one option available for Progress ABL – the SonicMQ Adapter.

While this is just fine for those that can go for SonicMQ and don't care about other providers I will say that is still good to keep the options open and try to keep the application code independent of the provider.

There are actually two main options that can be used to communicate with such a messaging provider:
- REST interface (representational state transfer), some providers support this or are adding support for it mainly for AJAX clients
- Wire-Level Protocols, open specifications (each provider has its own protocol just that specifications are not made available)

On the second group there is the already largely adopted STOMP (streaming text oriented messaging protocol) which covers most of the basic messaging needs and a newly emerging binary protocol that might (or not) make it to a 'standard' level – AMQP: Rabbit MQ, Appache QPID, OpenAMQ.

OpenWire is the binary protocol used by Apache ActiveMQ and it's specifications are open while XMPP (Extensible Messaging and Presence Protocol) is an XML based protocol best known as Jabber and used mostly for instant messaging (IM) – Google Talk, Facebook chat.

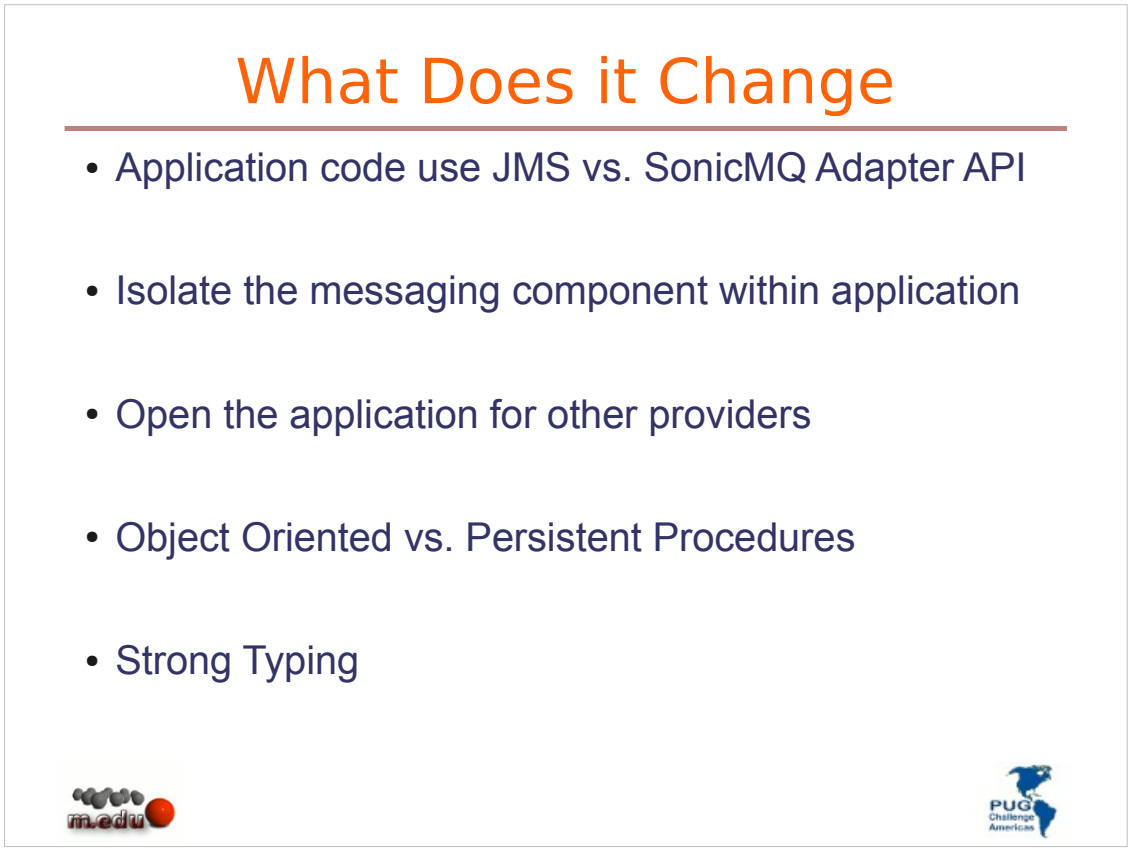


There are probably more than a single implementations out there for STOMP protocol, the only one that was made publicly available (MIT) is the one of Flusso, Richard Kelters published that as a project on OEHive web-site.

No very interested about the messaging services topic at the time but quite fond of Java 'program to the interface' way I've found the project interesting and thought about extending it to fully cover the JMS specs and turn it in a community project of some kind; there was a message exchange at the time on this subject.

Anyway, the project started on sourceforge with complete ABL translation of JMS interface – small deviations due to the restrictions of ABL OO (interface inheritance, static properties in interfaces). Although each provider is expected to fully implement the JMS interface as it see fit, I felt that ABL implementations might take the common part of JMS components as a provider independent implementation (mostly the message variations).

Provider specific implementations were planed for Stomp wire-level protocol (low-level socket data exchange) and for Sonic MQ as a wrapper over the provided adapter API.



Instead of coding the application against the provider specific API, in this case the Sonic MQ adapter API, another level of abstraction is introduced in order to keep the application code provider independent – this is the JMS interface.

Even if no other provider is expected to be used currently other than the default isolating the messaging component from the rest of the application by using a interface can be considered a good practice. When needed different provider specific implementation can be plugged-in with no changes required on the application code.

Using object oriented implementation is a better choice over the persistent procedures one given better encapsulation and also providing strong typing, most probably PSC will come up with an OO variant in the future – I just hope they'll start with the interface :)

Point-to-Point Session (Queue)

SonicMQ Adapter API

```
RUN jms/pptsession.p PERSISTENT SET sessionH
                     ("-SMQConnect").
RUN beginSession in sessionH.
```

JMS

```
qconn=QueueConnectionFactory:createQueueConnection().
qsess=qconn:createQueueSession().
qconn:start().
```

Because the Sonic MQ Adapter API is modeled after the JMS API - to some extent, most visible changes are the one caused by switching from persistent-procedures to object oriented.

Message Types

SonicMQ Adapter Specific

- XML Message
- TempTable Message
- DataSet Message
- Multipart Message

JMS

- Object Message

Sonic MQ Adapter introduced a number of 'specific' message types, XML is a well formatted XML pay-load on a common Text message – temp-table and dataset just build on that write/read-xml.

Multipart Message in a Byte Message that contains multiple 'parts', each section can be text, binary data or a full message.

There is no support for Object Message as defined by JMS interface (Java Object serialization). A serializable object can easily travel over the wire as an Object Message and proven the 'same' object exists on the ABL side this can be instantiated from the serialization data received – given limited support for reflection in ABL this can't be implemented in a generic way (just yet). Some thought about object serialization can be found on the AutoEdgeFactory project. One approach on object serialization is to use code annotations but this require a code pre-processor in place which might be easily integrated in OA Architect.

The JMS interface defined in Java EE builds on top of existing 'commons' – the Java SDK, software development kit.

Standard objects like collections, maps, input/output data streams were not available in OO ABL... there goes the most considerable part of the development effort of this JMS project so far.

Dive Into Code

WHY USE AN INTERFACE

- It isolates the messaging component within Application
- Makes your Application provider independent
- Supports plug-able providers
- It does not have to fully implement JMS
- It does not even have to be JMS

Questions?

Thank-you!