Developing NaradaBrokering Applications

Outline

- Primer on events, synopsis, profiles and templates
- Developing a simple application
  - Specifying different subscription formats and available transports
  - Utilizing different transports
- Exploiting available Quality-of-Service capabilities
  - Compression/Decompression of payloads
  - Building a Reliable Delivery application

Outline – (II)

- Managing Replays
- Exactly once delivery of clients
- Fragmentation & Coalescing of Events

- Writing JMS applications in NaradaBrokering
  - Simple applications
  - Applications requiring reliable delivery
- An Audio/Video conferencing application
- Advanced applications
  - GridFTP and NaradaBrokering
  - Shared SVG application
  - Shared Microsoft application

NaradaBrokering Overview

- Open source project. http://www.naradabrokering.org
- Based on a network of cooperating broker nodes
  - Cluster based architecture allows system to scale in size
- Provides a variety of services
  - Reliable, ordered and exactly once delivery.
  - Compression and fragmentation of large payloads.
  - Support for multiple subscription types
- Used in the context of A/V applications and to enhanced Grid apps such as Grid-FTP
- Provides support for variety of transports: TCP, UDP, HTTP, SSL, Multicast and parallel TCP streams.
- JMS compliant. Will provide WS-Notification support.
- Includes bridge to GT3. April 2004 release.
- Support for Web Services being incorporated.

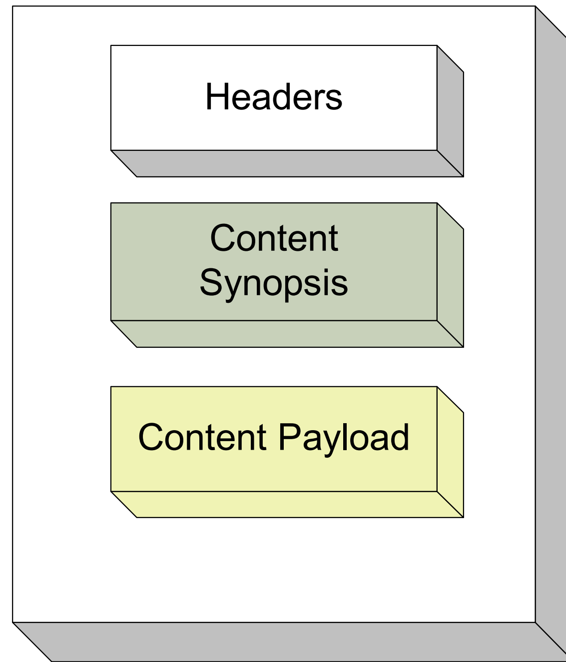
Current NaradaBrokering Features

Multiple transport support In publish-subscribe Paradigm with different Protocols on each link	Transport protocols supported include TCP, Parallel TCP streams, UDP, Multicast, SSL, HTTP and HTTPS. Communications through authenticating proxies/firewalls & NATs. Network QoS based Routing
Subscription Formats	Subscription can be Strings, Integers, **XPath** queries, **Regular Expressions**, **SQL** and tag=value pairs.
Reliable delivery	**Robust** and **exactly-once delivery** of messages in presence of failures
Ordered delivery	**Producer Order** and **Total Order** over a message type **Time Ordered** delivery using Grid-wide **NTP based absolute time**
Recovery and Replay	**Recovery from failures** and disconnects. **Replay** of events/messages at any time.
Security	**Message-level WS-Security** compatible **security**
Message Payload options	Compression and Decompression of payloads Fragmentation a nd Coalescing of payloads
Messaging Related Compliance	Java Message Service (**JMS**) 1.0.2b compliant Support for routing P2P **JXTA** interactions.
Grid Application Support	NaradaBrokering enhanced **Grid-FTP**. Bridge to the **Globus TK3**.
Web Service reliability	Prototype implementation of **WS-ReliableMessaging**

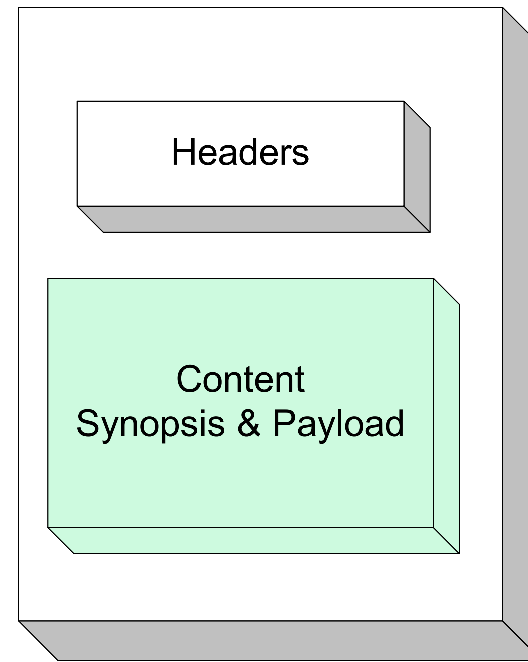
Primer (I)

- An event comprises of *headers*, *content descriptors* and the *payload* encapsulating the content.
- An event's headers provide information pertaining to
  - the type, unique identification, timestamps, dissemination traces and other QoS related information pertaining to the event.
- The content descriptors for an event describe information pertaining to the encapsulated content.
  - The content descriptors and the values these content descriptors take collectively comprise the event's ***content synopsis***.

Primer – (II)



(a)



(b)

- Complexity of content description can cause the demarcation between synopsis and the content to blur
  - Here they end up being indistinguishable from each other.

Primer – (III)

- The set of *headers and content descriptors* constitute the *template* of an event.
- Events containing identical sets of headers and content descriptors are said to be conforming to the same template.
  - Values the content descriptors take and the content payloads itself may be entirely different for events conforming to the same template.

Primer (IV)

- Entities have multiple *profiles* each of which signifies an interest in events conforming to a certain template.
- Interest is in the form of constraints.
  - Constraint also referred to as a ***subscription***.
- Entities specify constraints on the content descriptors and the values some or all of these descriptors might take.
- Constraint complexity can vary from simple strings to <tag, value> pairs to XPath queries to Regular expressions.

Starting the broker

- In the **bin** directory of the NaradaBrokering installation please update the **NB_HOME** variable.
  - Note that that the location of the installation directory does not have a trailing slash "/".
  - For Windows, update **startBroker.bat**.
    - Please also include the **%NB_HOME%\dll** in your path variable.
  - For UNIX users, modify the **startbr.sh** file
- Double click the **startBroker.bat** file or run **./startbr.sh**
  - Note that you need to download **jms.jar** (Version 1.0.2b) and **jmf.jar**. Move them into the **NB_HOME/lib** directory.

Developing applications

- Entities need to specify an identifier
  - Currently this is an integer value. We are proposing to replace this by UUIDs.
- Next control the configuration of the client. See the **`$NB_HOME/config/ServiceConfiguration.txt`** for a sample configuration file.
  - File is used to set up and control parameters needed by various services.
  - Defaults used if correct file not specified.
- Initialize roles of producer and consumer

Sample Service Configuration file (I)

```
FragmentationDirectory=D:/TempFiles/tmpFiles/fragment

#This specifies the location of the coalescing directory
CoalescingDirectory=D:/TempFiles/tmpFiles/coalesce

#Specifies location of stratum-1 time servers.
NTP_Servers =
   129.6.15.28,129.6.15.29,132.163.4.101,132.163.4.102,132.163
   .4.103,192.43.244.18

## This is the time interval (milliseconds) between runs of
## the NTP synchronization NTP_Interval=2000
NTP_Debug=OFF

#Time Ordered Buffering related parameters
TOB_MaximumTotalBufferSize=2500000
TOB_MaximumNumberOfBufferEntries=10000

#In milliseconds#
TOB_MaximumBufferEntryDuration=50000
TOB_BufferReleaseFactor=0.8
```

Sample Service Configuration file (II)

```
#These pertain to Reliable Delivery Service
  Implementations (db=Database, file=FileStorage)
Storage_Type=db
Database_JDBC_Driver=org.gjt.mm.mysql.Driver
Database_ConnectionProvider=jdbc:mysql
Database_ConnectionHost=localhost
Database_ConnectionPort=3306
Database_ConnectionDatabase=NbPersistence

FileStorage_BaseDirectory=C:/NBStorage/filebased/persis
  tent


Database_WSRM_Database=wsrm
#Database_WSRM_username=username
#Database_WSRM_password=password
```

Initializing the client service

- You can initialize the configurations associated with services in your session using the following

```
String config =
   "D:/NaradaSources/config/ServiceConfiguration.txt";
SessionService.setServiceConfigurationLocation(config);
```

- Initialize the ClientService instance using the entity Id

```
ClientService clientService =
   SessionService.getClientService(entityId);
```

- Note that last 2 method calls listed above throw the NaradaBrokering `ServiceException` if it encounters problems.

Initialize Broker communications

- It is very simple to initialize and load communication libraries in NB clients.
- Create a Properties object and load t he appropriate values.
- If the broker is running on localhost and listening to TCP connections on 3045.

```
Properties props = new Properties();
props.put("hostname", "localhost");
props.put("portnum", "3045");
clientService.initializeBrokerCommunications(props,
                                              "niotcp");
```

Initializing the consumer role – (I)

- Every consumer needs to implement the **`NBEventListener`** interface.
  - This contains the **`onEvent(NBEvent nbEvent)`** method that is invoked by the substrate upon receipt of an appropriate event.
- To create a consumer and register with substrate do the following

```
EventConsumer consumer =
  clientService.createEventConsumer(this);
```

Note that **`this`** refers to the class, which implements the **`NBEventListener`** interface.

Initializing the consumer role – (II)

- Next, you need to specify your subscription.
  - Here we deal with the simplest form which is String based.
- This is done by the creation of a Profile

```
int profileType =
  TemplateProfileAndSynopsisTypes.STRING;
Profile profile =
   clientService.createProfile(profileType,
                                      "Movie/Casablanca");
```

- Next proceed to subscribe

```
consumer.subscribeTo(profile);
```

Initializing the consumer role – (III)

- Note that there is no limit on the number of consumers that can be created from a client service.
- There is also no limit on the number of subscriptions that you can subscribe to on a given consumer.
- A given consumer can have subscriptions of different types, such as XPath, Regular expressions etc.

Initializing the producer role

- Creation of the event producer is done by invoking the following method.

```
EventProducer producer =
    clientService.createEventProducer();
```

- You can suppress redistribution of generated events by using the following

```
producer.setSuppressRedistributionToSource(true);
```

- A sample of other utility methods include

```
producer.generateEventIdentifier(true);
producer.setTemplateId(12345);
producer.setDisableTimestamp(false);
```

Generating and Publishing events

- To generate events, one needs to specify the event type, the content synopsis and the payload for the event.

```
int eventType = TemplateProfileAndSynopsisTypes.STRING;
String synopsis = "Movie/Casablanca";
byte[] payload;
NBEvent nbEvent =
   producer.generateEvent(eventType, synopsis,payload);
```

- To publish an event simply use the following method.

```
producer.publishEvent(nbEvent);
```

Dealing with the receipt of events

- Events that an entity receives are delivered using the `onEvent(NBEvent nbEvent)` method.
- Processing logic associated with received events can be put here in this method.
  - Note that an entity can inspect this event to retrieve its headers, synopsis, payloads etc.
- In the simplest case, you can print the event’s payload.

Dealing with other profiles/templates

- NaradaBrokering provides support for other profiles and event types.
- We will take a look at some of these. These include
  - Integers
  - <tag, value> pairs based on equality.
  - XPath queries and XML events
  - Regular expressions' based subscriptions

Availing of Quality of Services

- Quality of Services (QoS) pertaining to compression, fragmentation, reliable delivery, replay etc. in NaradaBrokering.
- Here we discuss building applications which can avail of these services.
- Generally, this involves the creation of `ProducerConstraints` & `ConsumerConstraints`.
  - These constraints are associated with the publishing and consumption of events.

Creation of Consumer Constraints

- ConsumerConstraints are created by the `EventConsumer` by using the `Profile` on which the constraints are to be specified.

```
ConsumerConstraints constraints =
consumer.createConsumerConstraints(profile);
```

- The QoS constraint on the subscription is propagated using the following

```
consumer.subscribeTo(profile, constraints);
```

Creation of Producer constraints

- `ProducerConstraints` first require the creation of a `TemplateInfo`.
  - This requires the specification of the templateId, templateType and template.

```
int templateId = 12345;
int templateType =
  TemplateProfileAndSynopsisTypes.STRING;
Object template = "Movie/Casablanca";
TemplateInfo templateInfo =
      clientService.createTemplateInfo(templateId,
                                       templateType, template);
```

- Next this is used to create the appropriate `ProducerConstraints`.

```
ProducerConstraints producerConstraints =
producer.createProducerConstraints(templateInfo);
```

Using the producer constraints

- This producer constraints are specified along with any events that need to be published.
  - Thus the constraints can be specified on a per-event basis.

```
producer.publishEvent(nbEvent, producerConstraints);
```

Compression/Decompression

- This is the simplest QoS available for applications.
- The QoS constraints are associated with producer.
  - The system automatically decompresses the payloads prior to delivery.

```
Properties compressionProperties =  new Properties();
compressionProperties.put("compressionAlgo", "zlib");
producerConstraints.
 setSendAfterPayloadCompression(compressionProperties);
producer.publishEvent(nbEvent, producerConstraints);
```

Reliable Delivery

- Setting up of the Reliable Delivery Node
- You first need to install mySQL 4.0. This is available from http://www.mysql.com/ .
  - If you do not wish to install this you may also use the files-storage based implementation of the NB storage service.

Setting up the MySQL database

- If you have installed mySQL 4.0 you first need to create the database. Use the following command to create the database utilized by NB.
  - `mysql create database NbPersistence;`
- Next go the `$NB_HOME/bin/mysql` directory. Double click on AutoNbDb.bat.
  - You may need to comment the first line in this files using a "**#**" if it is the first you are creating tables.

Setting up the RDS node

Initialize reliable delivery consumer

- Creating the subscription constraints

```
ConsumerConstraints constraints =
      consumer.createConsumerConstraints(profile);
constraints.setReceiveReliably(templateId);
consumer.subscribeTo(profile, constraints);
```

- Also, to retrieve events after a failure or disconnect one needs to
  - Implement the `NBRecoveryListener` interface.
  - Initiate recovery by invoking the following method.

```
long recoveryId= consumer.recover(templateId, this);
```

`this` corresponds to the class which implements the aforementioned `NBRecoveryListener` interface.

Initialize reliable delivery producer

- Initializing the constraints

```
TemplateInfo templateInfo =
      clientService.createTemplateInfo(templateId,
                             templateType, template);
producerConstraints =
   producer.createProducerConstraints(templateInfo);
producerConstraints.setSendReliably();
producer.publishEvent(nbEvent,producerConstraints);
```

- Also, to reinitialize producer after a failure or disconnect one needs to
  - Implement the `NBRecoveryListener` interface.
  - Initiate recovery by invoking the following method.

```
long recoveryId= consumer.recover(templateId, this);
```

`this` corresponds to the class which implements the aforementioned `NBRecoveryListener` interface.

Exactly-once delivery of events

- This uses the NaradaBrokering Reliable Delivery Service.
- This mandates no changes to the NaradaBrokering reliable delivery producer.
- On the consumer side specify both reliable and ordered delivery.

```
ConsumerConstraints constraints =
      consumer.createConsumerConstraints(profile);
constraints.setReceiveReliably(templateId);
constraints.setReceiveInOrder(templateId);

consumer.subscribeTo(profile, constraints);
long recoveryId = consumer.recover(templateId, this);
```

Managing replays – (I)

- Replay Service works with events that have been stored reliably by the NB Reliable Delivery Service.
- Here we first need to use the `ClientService` to create a replay request. There are 3 different ways to do so.
  - Specify `templateId` and the `sequence numbers` to be replayed.

```
long[] sequenceNumbers;
ReplayRequest replayRequest =
  clientService.createReplayRequest(templateId,
                                    sequenceNumbers);
```

  - Specify `templateId`, along with the `start` and `end` values of the sequences to be replayed.

```
ReplayRequest replayRequest =
   clientService.createReplayRequest(templateId,
                                     start, end);
```

  - Specify `templateId`, the range of sequences to be replayed, along with any additional profile constraints for delivery.

Managing Replays (II)

- The replay client needs to implement the `ReplayServiceListener` interface. This has two methods
  - `public void onReplay(ReplayEvent replayEvent)`
  - `public void onReplayResponse(ReplayResponse replayResponse)`
- To initiate replay simply use the following method.

```
consumer.initiateReplay(replayRequest, this);
```

  - The `this` here corresponds to the class implementing the `ReplayServiceListener` interface.

Fragmentation/Coalescing

- Here we break up a large file into smaller fragments and reliably coalesce them at the receiver.
- This scheme is used in the NB-enhanced version of GridFTP.
  - This allows us to initiate file transfers without the recipient being present.
  - Furthermore, this also allows one-to-many transfers.
- The fragmentation/coalescing service requires the NB Reliable Delivery Service.
- See the configuration file to configure the fragmentation/coalescing service parameters.
  - This includes the location of the temporary directories.

Fragmentation Producer

- The fragmentation properties takes two sets of parameters. You can specify one of these sets.
  - **`fileLocation`** and **`fragmentSize`**. This controls the size of the fragments for the specified file.
  - **`fileLocation`** and **`numOfFragments`**. This controls the total number of fragments for a given file.

```
fragmentationProperties.put("numberOfFragments", 300);
fragmentationProperties.put("fileLocation", filename);
```

- Next proceed to send the file across after splitting it into fragments.

```
producerConstraints.
    setSendAfterFragmentation(fragmentationProperties);
producer.publishEvent(nbEvent, producerConstraints);
```

The coalescing consumer

- Here we specify the delivery of the coalesced payload.
  - Note that the large file will be coalesced in the directory specified in the config file.
  - The large file will not be in memory. Instead the user will get a notification saying that the file has be written to the appropriate location.

```
ConsumerConstraints constraints =
      consumer.createConsumerConstraints(profile);
constraints.setReceiveReliably(templateId);
constraints.setReceiveAfterCoalescingFragments();
consumer.subscribeTo(profile, constraints);

long recoveryId = consumer.recover(templateId,this);
```

Writing JMS applications

- We assume here that users are a bit familiar with JMS. There are several excellent books available for that.
- Here we give details regarding the creation of the TopicConnectionFactory.
  - Once this is set up interactions proceed as defined in the JMS specification.

```
Properties props = new Properties();
/** This pertains to setting up a TCP link */
props.put("hostname", hostInfo);
props.put("portnum", portInfo);
NBJmsInitializer ini =
  new NBJmsInitializer(props, “niotcp”, entityId);
 TopicConnectionFactory conFactory =
      (TopicConnectionFactory)ini.lookup();
```

Durable JMS subscriptions

- For every topic that you wish to be durable, set up the RDS node as outlined earlier.
- Further, include the mapping of the templateId to the JMS topic in the properties used for initializing the bridge.
  - This has to be done prior to constructing the **`NBJmsInitializer`**

```
props.put("/Sports/NBA", "34567");
```

- Note that even though you are using NB's reliable delivery service you do not need to import any NB related packages in your JMS application.

Use of NB's JMS mode in the Anabas System