SIS Feed Consumer POC on WebLogic JMS

Draft Revision 0.5 06/09/2007
Draft Revision 0.6 12/03/2015

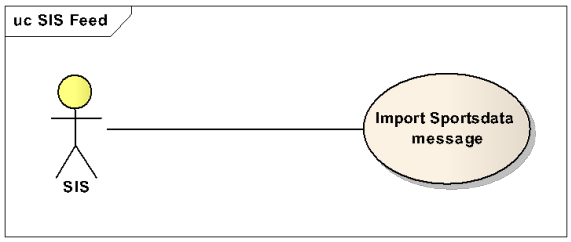
1. Scope and requirements

In this section we will define scope and capture requirements for the OM SIS Feed Components.

1.1. Vision

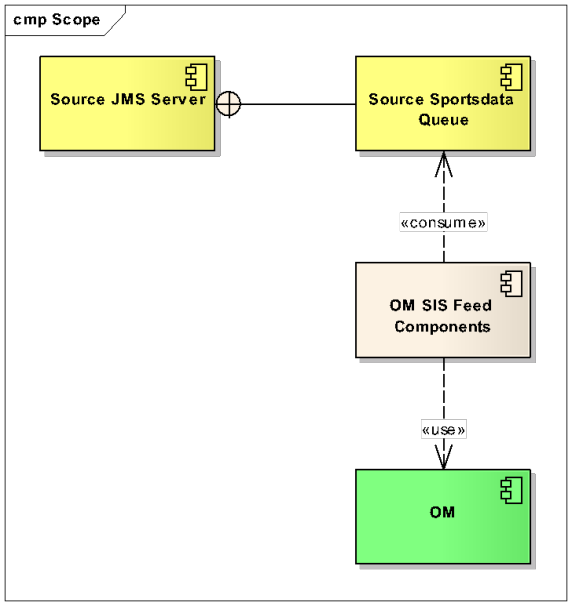
SIS provides a remote queue of Sportsdata messages for consumption by Betting Company. Within Betting Company, Opportunity Management will want to receive updates with the content of these messages in order to create and update opportunities and to send its own notification messages to other applications. The OM SIS Feed components are responsible for consuming raw Sportsdata messages from SIS and using OM API to update Opportunity Management while meeting a number of specific requirements.

1.2. Use case



Sportsdata messages are consumed from a Source Sportsdata Queue provided by SIS.

1.3. Scope

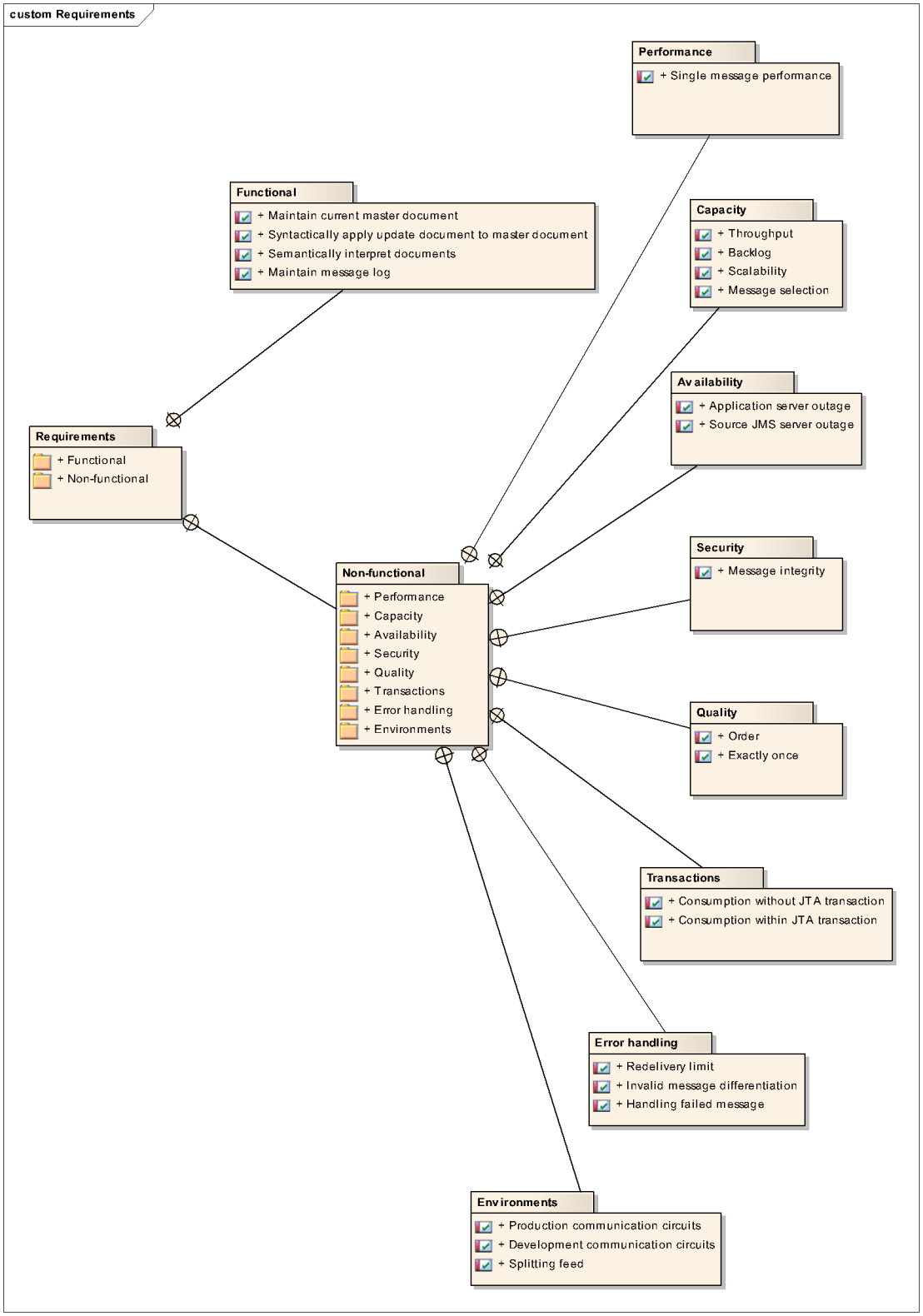


The scope of this effort is to design and build the OM SIS Feed Components. The surrounding components and environment are as follows:

- Source JMS Server is a WebSphere MQ Server hosted at SIS.
- Source Sportsdata Queue is a queue of raw byte messages hosted by Source JMS Server.
- In between SIS and Betting Company is a private network circuit of sufficient capacity.

- OM is an existing Java/Spring component. It needs to be packaged and deployed in an appropriate runtime environment with access to its database, message destination, and perhaps other resources.
- The Sportsdata messages contain XML payload of a format specified by schemas in RELAX NG Compact Syntax format. Message payload is divided into the two categories master and update. A master payload is a document describing a current state of some sports event or similar. An update payload consists of instructions to syntactically update an existing master document. The message received from SIS is a javax.jms.BytesMessage with no specified properties.
- The projected initial load of 12000 messages in an 8 hour period was derived from “representative” test data and discussions with SIS, but is in no way conclusive.
- The application can be deployed in and leverage the features of WebLogic Server 9.2.

1.4. Requirements



These requirements are prioritized according to the "MoSCoW" criteria: Must have (mandatory), Should have (important but may be omitted), Could have (truly optional), Want to have (can wait for later releases).

"The system" below refers to the entire setup on the Betting Company side.

1.4.1. Functional requirements

1.4.1.1. Maintain current master document (Must)

The system shall maintain current master documents.

1.4.1.2. Syntactically apply update document to master document (Must)

The system shall syntactically apply updates to master documents.

1.4.1.3. Semantically interpret documents (Must)

The system shall semantically interpret new master documents and update documents and update OM.

1.4.1.4. Maintain message log (Should)

The system shall maintain logs of received and processed messages in order to resolve any disputes between SIS and Betting Company regarding message delivery and content.

1.4.2. Non-functional requirements

1.4.2.1. Performance

Single message performance (Should)

The system shall process a single new message delivered from the Source Sportsdata Queue completely in less than 1 second.

1.4.2.2. Capacity

Throughput (Must)

The system shall keep up with an average message flow of 12000 messages in an 8 hour period, equivalent to 0.42 messages per second.

Backlog (Should)

The system shall be able to process a day's backlog of messages (12000) in less than one hour, equivalent to 3.4 messages per second.

Scalability (Should)

The system shall be scalable to accommodate an order of magnitude (10 times) future increase in the number of messages without code changes.

Message selection (Could)

The system shall be configurable in such a way that messages can be selected, ignored or prioritized according to properties embedded in its payload.

1.4.2.3. Availability

Source JMS server outage (Must)

The system shall recover automatically from transient unavailability of the Source JMS Server due to either network outages between SIS and Betting Company or service outage at SIS.

Application server outage (Must)

The system shall recover automatically from the failure of any single application server instance on which it is deployed.

1.4.2.4. Security

Message integrity (Should)

The system must not be vulnerable to manipulation (deletion or insertion of messages) at any point.

1.4.2.5. Quality

Order (Must)

The system shall process related messages or items in messages in a specified order.

In the most basic case, source messages must be processed strictly sequentially in the order they are delivered from the Source Sportsdata Queue. For messages that contain a list of identifiers, each identifier is processed in the order it occurs in the message.

It is also possible define a partial order based on attributes found in the body of each message. This would permit parallel processing of several individually ordered message flows.

The id field present in all Sportsdata messages defines such a partial order. A single message may contain a list if ids and would thus have to be split into one for each occurring id.

Exactly once (Must)

The system shall process each message exactly once. Messages must not be lost and must not be processed more than once even if in order as they cannot be assumed to be idempotent (“pp=p”).

1.4.2.6. Transactions

Source consumption without JTA transaction (Must)

The system shall be able to consume the Source Sportsdata Queue without a WebLogic JTA transaction. According to sources at IBM, WebSphere MQ Server participating in a JTA transaction with WebLogic Server 9.2 as the transaction manager is not supported.

Source consumption within JTA transaction (Should)

The system shall be able to consume the Source Sportsdata Queue within a WebLogic JTA transaction. If this is possible, we should be able to reap all the benefits.

1.4.2.7. Error Handling

Redelivery limit (Must)

The system shall enforce a limit to the number of times a particular message can be redelivered for any reason. In a messaging system, an unhandled exception thrown during the processing of a message can cause a message rollback and subsequent redelivery. This is acceptable or even desirable if the cause of the exception is transient. However, if the message will always cause the exception, it will keep bouncing indefinitely and block the progress of other messages. This is also known as a poison message scenario. Even if other measures are in place, this catch-all measure must remove a poison message after a maximum number of redeliveries.

Invalid message differentiation (Should)

The system shall correctly differentiate between an invalid message and a transient error.

An invalid message is a message such that its content alone causes an exception every time it is processed. It should be handled but should not cause a rollback.

A transient error that is not caused by the content of the message alone could simply cause a rollback in order to be retried.

Handling failed message (Could)

The system should discover and handle failed messages appropriately. A failed message is a message that has either been identified as invalid or has reached its redelivery limit. It should be logged or forwarded appropriately.

1.4.2.8. Environments

Production communication circuits (Must)

There should be two circuits from the SIS Production Source Sportsdata Queue to Betting Company; one to the Production environment and one to the Disaster Recovery environment, but they must never be active at the same time.

Development communication circuits (Should)

There should be one circuit from the SIS Development Source Sportsdata Queue to the Development and Test environments.

Splitting feed (Could)

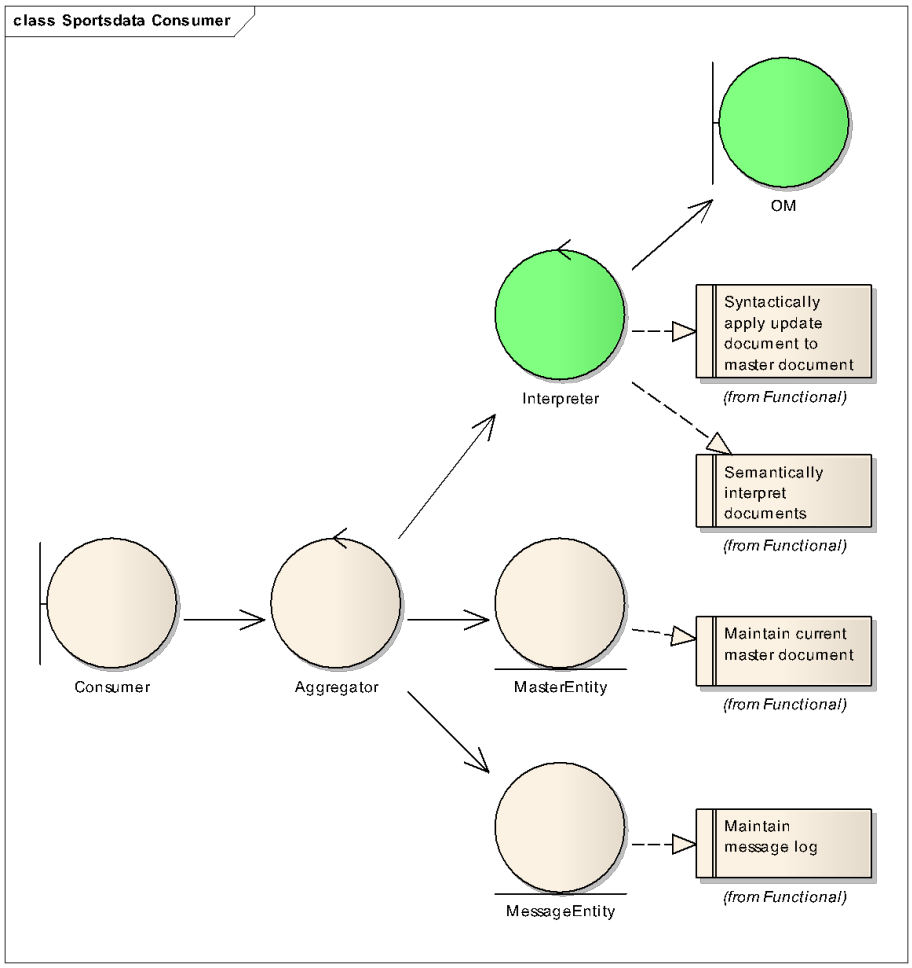
The system should be able to provide a raw Sportsdata feed to several clients while consuming from a single Source Sportsdata Queue.

2. Analysis

In this section we will look at a first cut of analysis classes to realize the functional requirements and relevant attributes of the messages to be imported. The rest is deferred until design.

2.1. Functional requirements realization

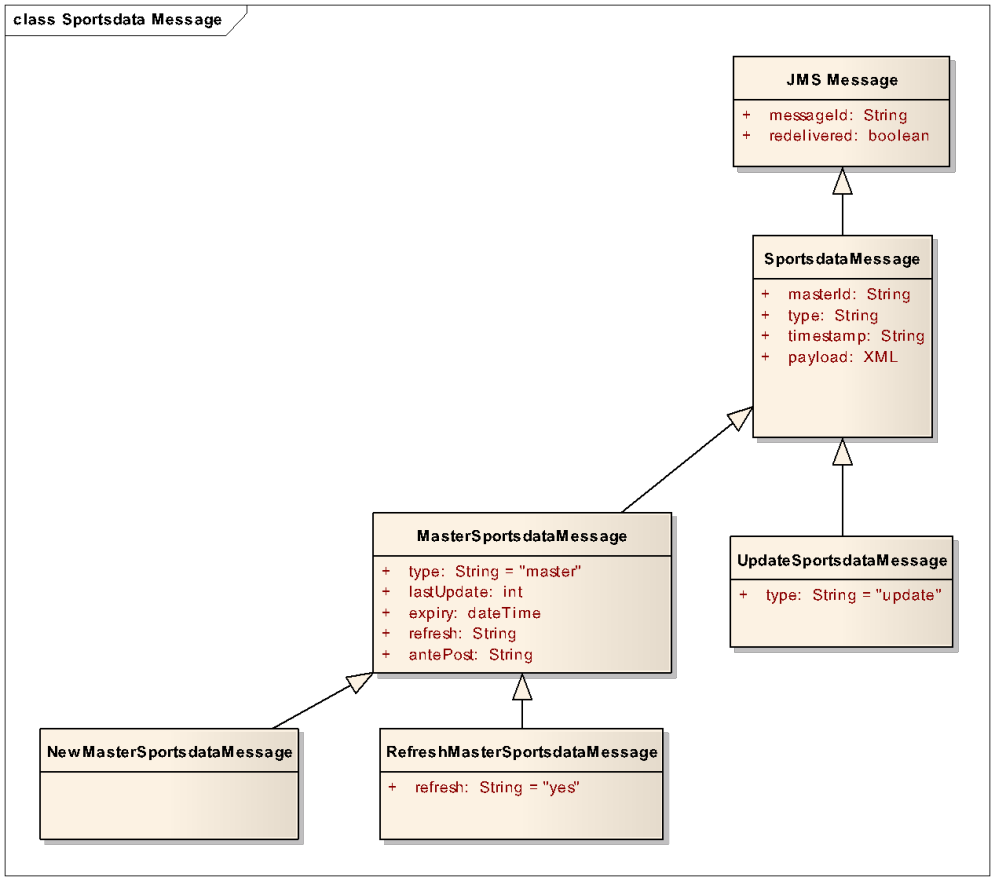
The functional requirements are realized by the following analysis classes.



- Interpreter is the OM component that interprets and manipulates message documents. It uses another OM component to update and notify OM.
- MasterEntity represents the mutable master document that is updated by subsequent messages. The key is a single master id.
- MessageEntity represents received and processed message content for audit purposes. A unique message id would be part of the key.
- Aggregator takes the message content and decides how to handle it depending on the division of message type. Master message content is either “new” or “refresh”, and are simply fed to Interpreter and stored as a MasterEntity (in no particular order). Update message content requires the previous master document to be retrieved before Interpreter is used to manipulate it using the update message content. The message content is logged, possibly including status about the processing by Interpreter.

- Consumer receives raw messages and uses Aggregator to process their content.

2.2. Sportsdata message content



A raw Sportsdata message is a JMS message containing XML payload that completely represents the message content. The content type is divided into categories by attributes in the message content. All message content is either “master” and “update”, and “master” is further divided into “new” and “refresh”.

The JMS Message itself contains several properties, among the unique JMS Message ID and a flag indicating if the provider has previously attempted to deliver this particular message.

The attributes shown are only those directly relevant to a minimal implementation; other attributes could be extracted and used for message selection and prioritization.

2.3. Major decisions

2.3.1. Transactional consumption from Source Sportsdata Queue

2.3.2. Intermediate destination

2.3.3. Intermediate queue or topic

2.3.4. Intermediate distributed destination

2.3.5. Detailed design

In the production scenario, the Source JMS Server is the WebSphere MQ Server hosted at SIS, and the contained Source Sportsdata Queue is its MQ Queue.

In a test scenario, the Source JMS Server might be a local WebSphere MQ Server or a local WebLogic JMS Server. The Source Test Producer can then be a simple JMS client, or in the WebLogic JMS case even just the WebLogic Console.

There is a choice between the OM Source Sportsdata Consumer and the OM Intermediate Sportsdata Consumer.

The OM Source Sportsdata Consumer consumes directly from the Source Sportsdata Queue and uses the OM SIS Feed Handler.

The OM Intermediate Sportsdata Consumer consumes the Intermediate Sportsdata Destination hosted by the Intermediate JMS Server and uses the OM SIS Feed Handler.

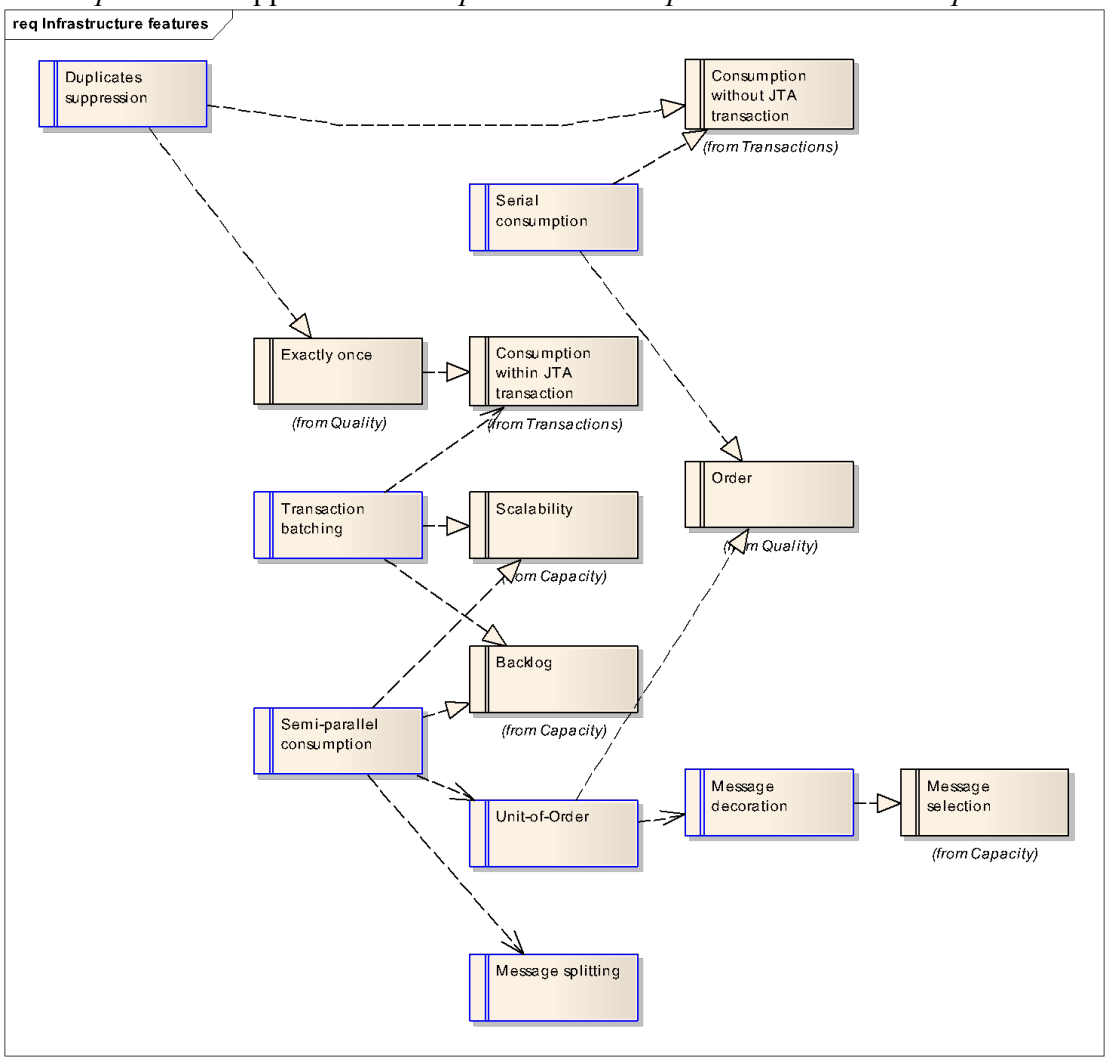
The Sportsdata Importer consumes a message from the Source Sportsdata Queue, splits and decorates it, and produces to the Intermediate Sportsdata Destination.

3. Design

In this section we will identify features, map them to requirements, design the remaining classes and package them into components.

3.1. Features

A number of features that realize the non-functional requirements can be identified. Two of the most important features are *Duplicates suppression* and *Semi-parallel consumption*. The opposite of *Semi-parallel consumption* is *Serial consumption*.



3.1.1. Duplicates suppression

Duplicates suppression is the feature of keeping track of messages seen in order to detect and ignore duplicate messages. Duplicate messages will occur in some outage situations in the absence of a JTA transaction that spans the message consumption and acknowledgement. It is always good practice to provide supplicates suppression, especially when it can be achieved at little cost.
This feature realizes *Consumption without JTA transaction* and *Exactly once*.
This feature may be implemented by keeping track of the unique JMS Message ID assigned by the messaging provider and associated with every message.

3.1.2. Serial consumption

Serial consumption is the feature (or constraint) of consuming only one message at a time. A message is only consumed after the previous is completely processed.
This feature realizes *Order* but is incompatible with *Semi-parallel consumption*.
This feature may be implemented by deploying only a single consumer, but care must be taken with failure and failover scenarios. It also adds vulnerability to network latency as there is ever only a single message or acknowledgement in transit at any time.

3.1.3. Transaction batching

Transaction batching is the feature of consuming several messages in the scope of a single transaction. This is beneficial in situations where many messages are waiting to be consumed. Starting and committing a JTA transaction comes at a near constant cost, which can be significant for short transactions. In addition, several persistence frameworks such as WebLogic Entity EJBs and Hibernate can be made to cache data in the scope of a transaction, and as this feature makes the transaction span more work, the potential for caching is greatly increased. This type of caching has no timeout or refresh issues.
This feature realizes *Backlog* and *Scalability* and is only relevant with *Consumption within JTA transaction*.
WebLogic Server 9.2 provides this feature for message-driven beans through WebLogic deployment descriptors.

3.1.4. Semi-parallel consumption

Semi-parallel consumption is the feature of consuming several streams of ordered messages in parallel. This requires defining a partial order for the messages. The master id attribute in every Sportsdata document defines such a partial order.
This feature realizes *Scalability* and *Backlog* and depends on *Unit-of-Order* and *Message splitting*.
This feature may be implemented by deploying several consumers and decorating every message with the master id as unit-of-order.

3.1.5. Unit-of-Order

Unit-of-Order is the feature of decorating messages with an identifier and trust the messaging infrastructure to deliver messages with the same unit-of-order to a single consumer in the order they were sent, even in the presence of multiple distributed consumers.
This feature realizes *Order* (even in the presence of multiple consumers) and requires *Message decoration*.
WebLogic Server 9.2 provides this feature through its WLMessageProducer implementation of the standard MessageProducer. The sending component is coupled to the WebLogic implementation as it has to use WLMessageProducer, but the consumer need not be aware of this but uses a standard MessageListener. It is WebLogic Server JMS that has assumed the responsibility to present the messages in order to the appropriate consumer.

3.1.6. Message decoration

Message decoration is the feature of adding properties to a JMS message before or when sending it in such a way that it can be accessed subsequently without

completely unpacking the message content. A special case is the WebLogic Unit-of-Order property that is attached to the message though using a WLMessageProducer. General message properties can be used in a JMS Message Selector without parsing the XML payload.
This feature is required for *Unit-of-Order* and realizes *Message selection*.
This feature must be implemented by the sender of a message. As the raw Sportsdata messages from Source Sportsdata Queue are not decorated in this fashion, we may not use the features that rely on this when consuming directly from this destination. This drives us to introduce an intermediate destination containing decorated messages.

3.1.7. Message splitting

Message splitting is the feature of splitting a single message referring to several items into several messages referring to only a single item each. The master id attribute in every Sportsdata document may contain a list of master ids. These messages need to be split in order for each master id to be processed in its own ordered stream.
This feature is required by *Semi-parallel consumption*.
This feature may be implemented by an intermediate consumer forwarding split messages to an intermediate destination.

3.2. Design classes

TODO

3.3. Components

3.3.1. Sportsdata Consumer

The core subsystem is the Sportsdata Consumer, which is responsible for consuming a raw or intermediate Sportsdata message and completely processes it using OM components. This component implements a standard MessageListener and is not tied to any container. It realizes *Duplicates suppression* and can be deployed to realize *Serial consumption*.

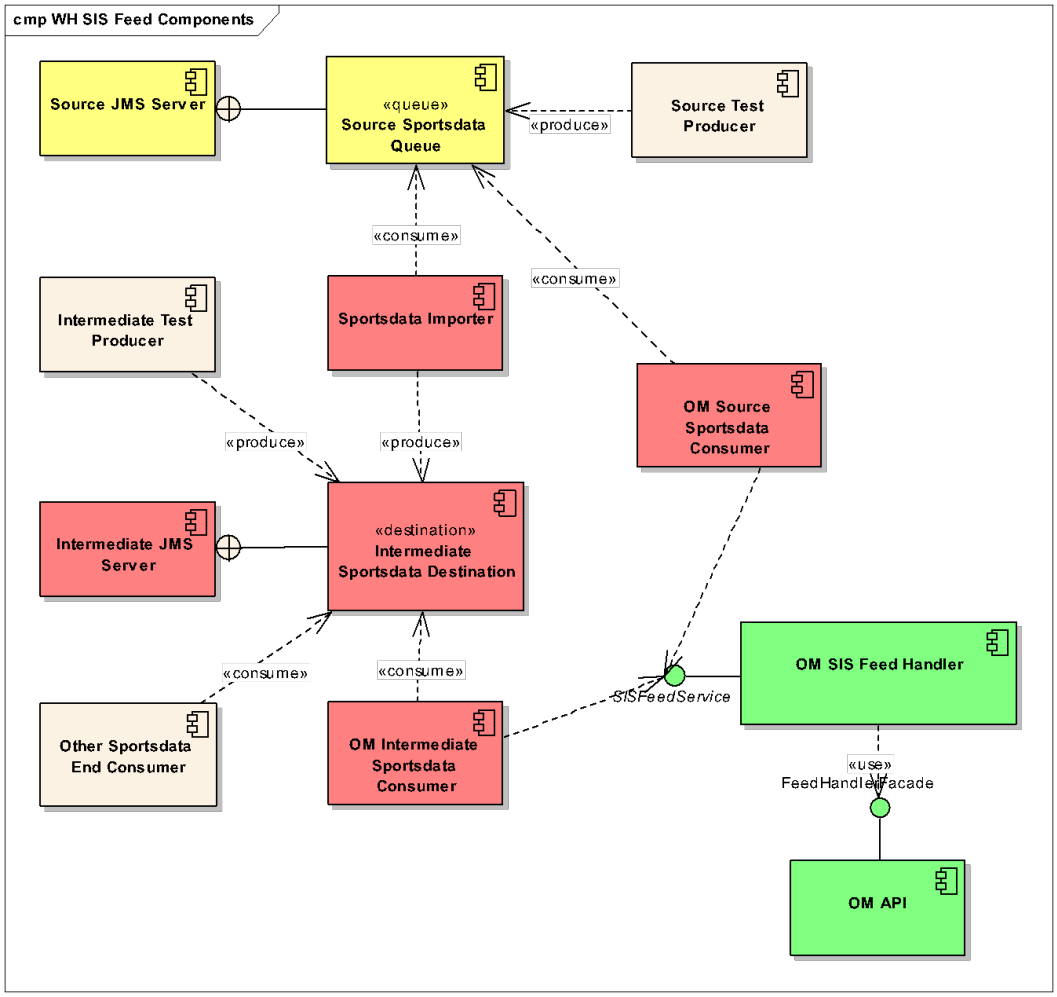
3.3.2. Sportsdata Consumer MDB

The Sportsdata Consumer MDB wraps the Sportsdata Consumer into a message-driven EJB that can be deployed to WebLogic Server. This gives us access to WebLogic Server features such as *Unit-of-Order* and *Transaction batching* without any code.

3.3.3. Sportsdata Importer MDB and Intermediate Sportsdata Queue

Many of the features required for *Semi-parallel consumption* are not available when consuming directly from the Source Sportsdata Queue as raw messages are not decorated appropriately. This drives us to introduce Sportsdata Importer that consumes messages from Source Sportsdata Queue, splits and decorates them, and sends them to Intermediate Sportsdata Queue.

4. Implementation and test



4.1. Automated Build

4.2. Using Spring to create Consumer from Aggregator

4.3. Using Spring to create a ConsumerMDB from Consumer

4.4. Direct implementation of TransportMDB

5. Deployment

5.1.1. Network connectivity SIS – Betting Company

5.1.2. Production with live feed to OM only

5.1.3. Production with shared live feed

5.1.4. Development with shared live feed