Technical Walkthrough - ION

Nichlas Karlsson, Senior Software Engineer, Shared Technology

Agenda 1(3)

- M3 BE ION enablement – overview
  - Message flows
- Applications
  - Deliverables
  - Event rules
  - BOD mappings

Agenda 2(3)

- MEC details
  - Communication
    - ION
    - Event Hub
  - Control properties
  - Check Order
  - Error handling
  - MEC Utilities
- Mapping tool
  - New Eclipse plugin
  - Graphical programming

Agenda 3(3)

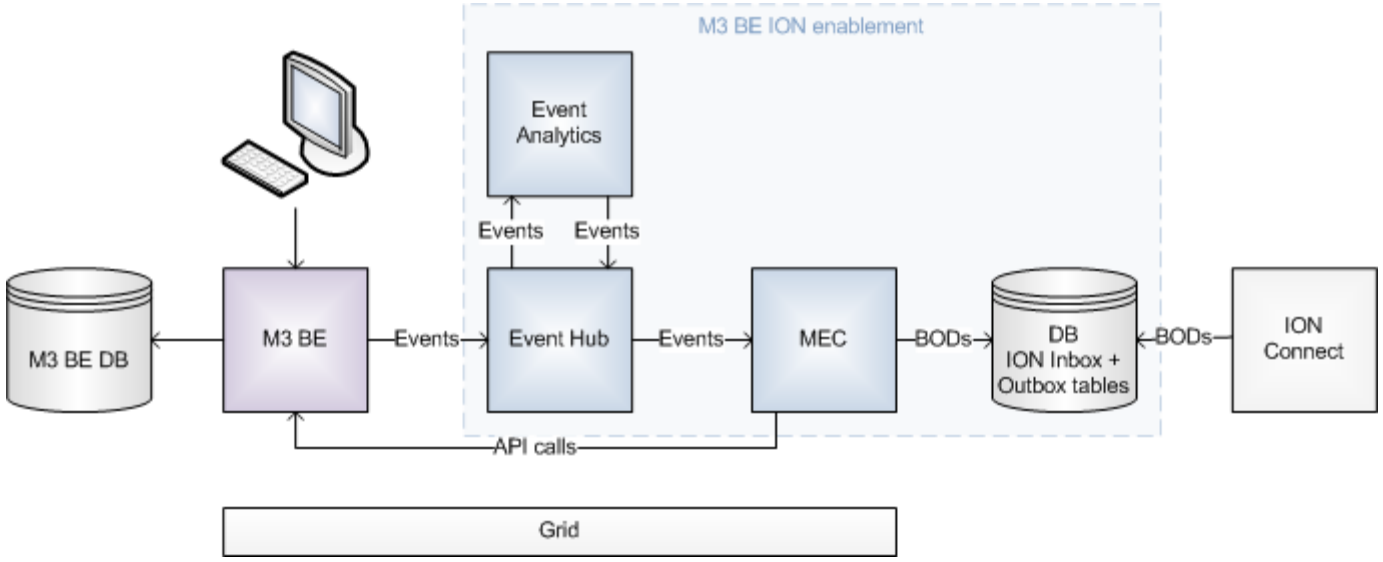
- Mapping tool
  - Eclipse plugin
  - Graphical programming

Technical Walkthrough - ION

M3 BE ION enablement – overview

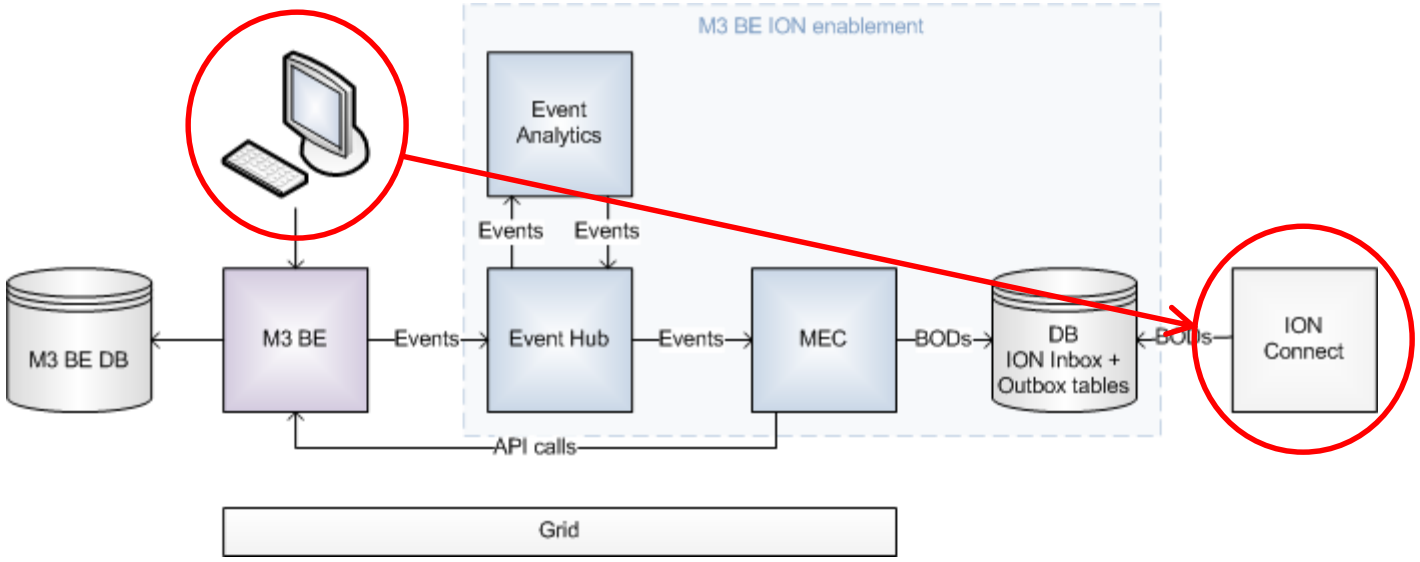
M3 BE ION enablement – overview

- The following products are used for M3 BE ION enablement:
  - M3 Enterprise Collaborator (MEC) v9.2
    - Runs on Grid
  - Event Hub (including Event Analytics) v1.2
    - Requires that M3 BE runs on Grid



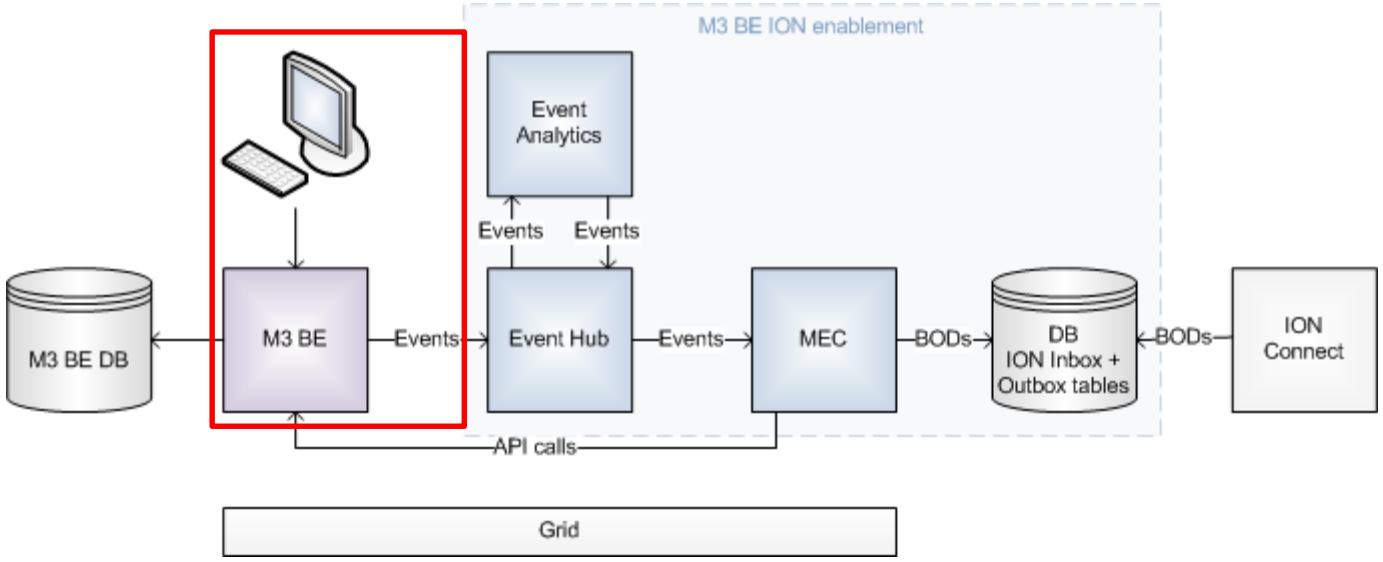
Message flows – outbound

- When an item is changed in M3 BE the BOD SyncItemMaster will be sent to ION since M3 BE is System Of Record (SOR) for ItemMaster



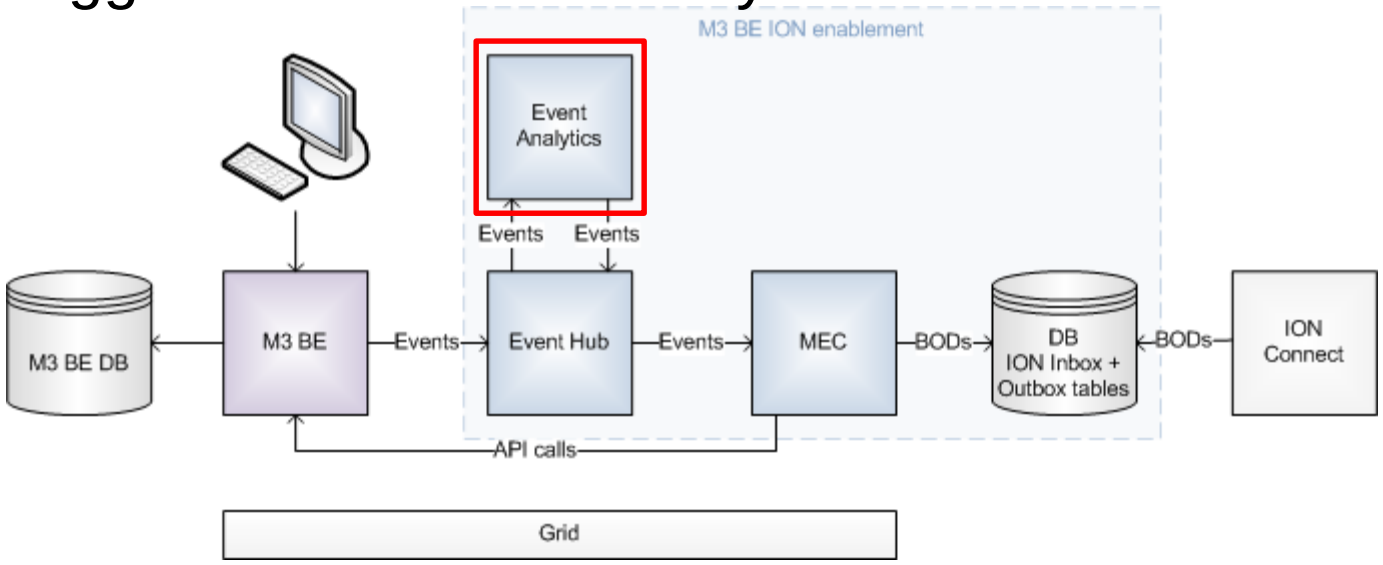
Message flows – outbound

1. A user changes the item in M3 BE
2. An event is created in M3 BE's database layer when the MITMAS (item master) record is updated
3. The event is posted to the Event Hub



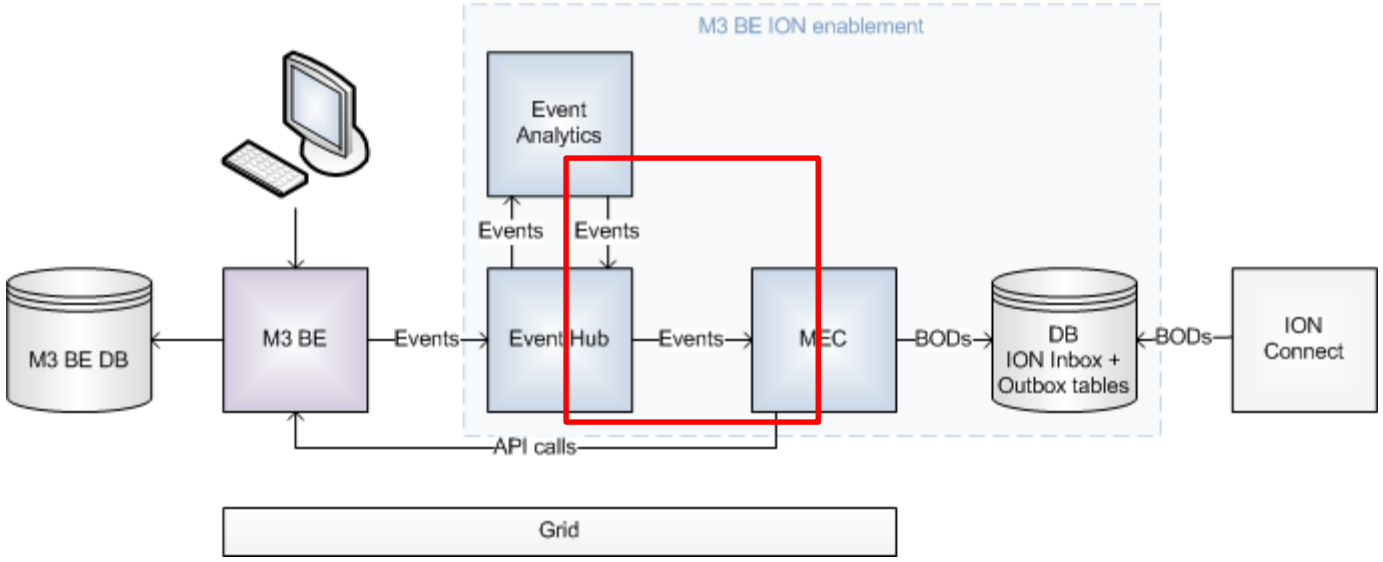
Message flows – outbound

4. Event Analytics subscribes to the MITMAS event
5. One or several rules are fired in Event Analytics when the MITMAS event is received
6. One of the rules creates a new event that has the purpose to trigger MEC to create a SyncItemMaster BOD



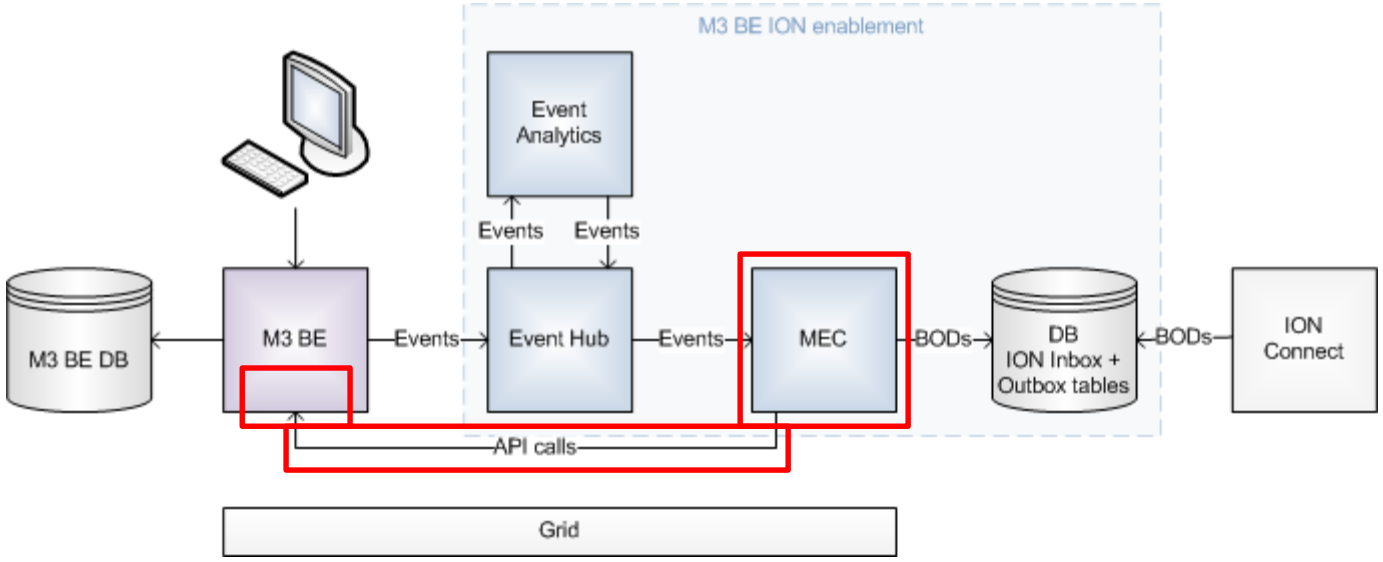
Message flows – outbound

7. The new SyncItemMaster event is posted to the Event Hub
8. MEC subscribes to the SyncItemMaster event
9. MEC detects a partner agreement, which defines processes to execute



Message flows – outbound

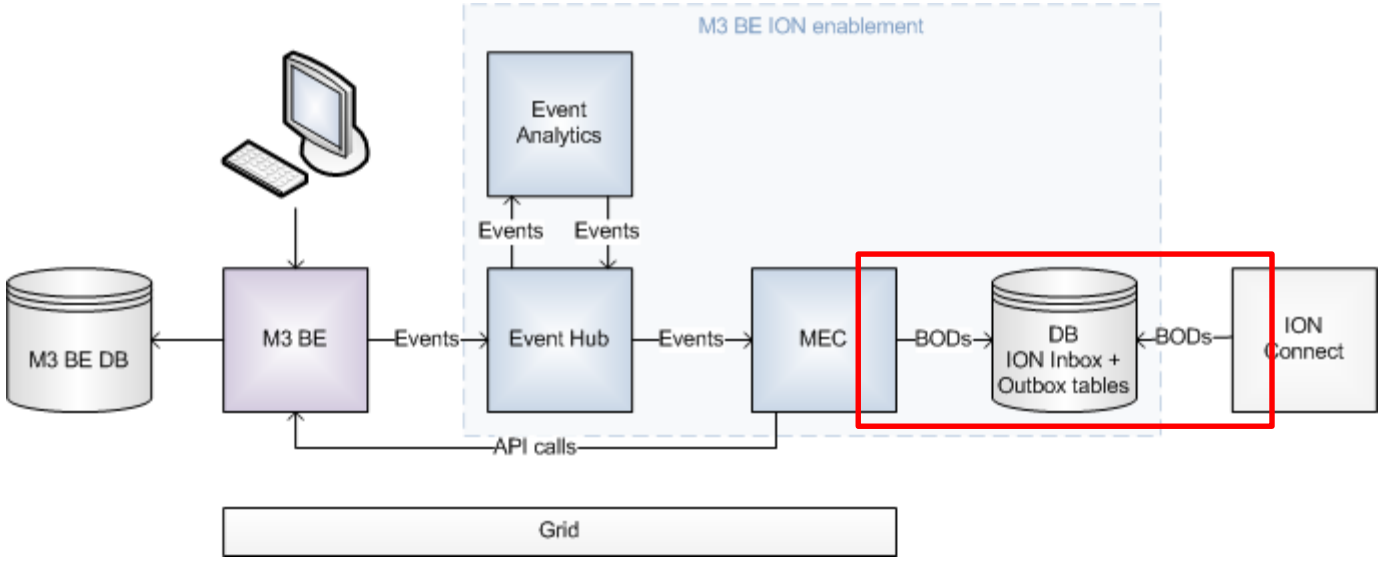
10. An XML Transform process semantically maps the event data to a SyncItemMaster BOD
    - The event data is complemented by M3 BE data by calling M3 BE APIs
    - All semantic transformation and data formatting, translation and restructuring take place in this process



Message flows – outbound

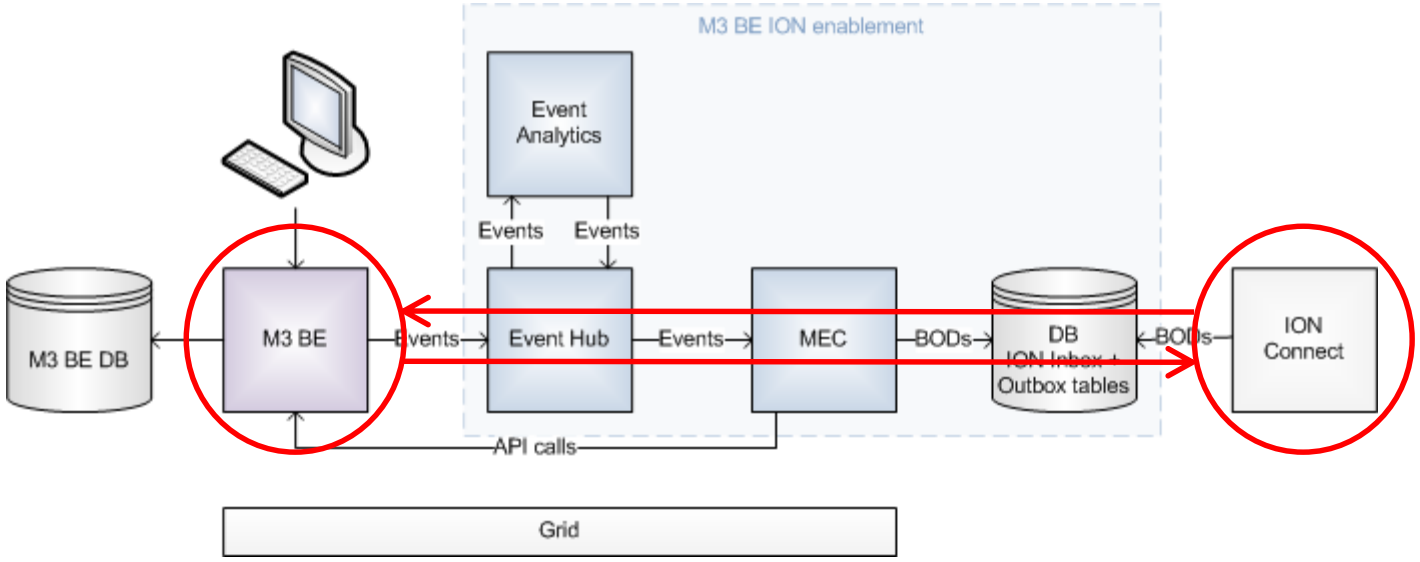
11. A send process sends the SyncItemMaster BOD to ION via the ION outbox database tables

12. ION fetches the SyncItemMaster BOD from the outbox database tables



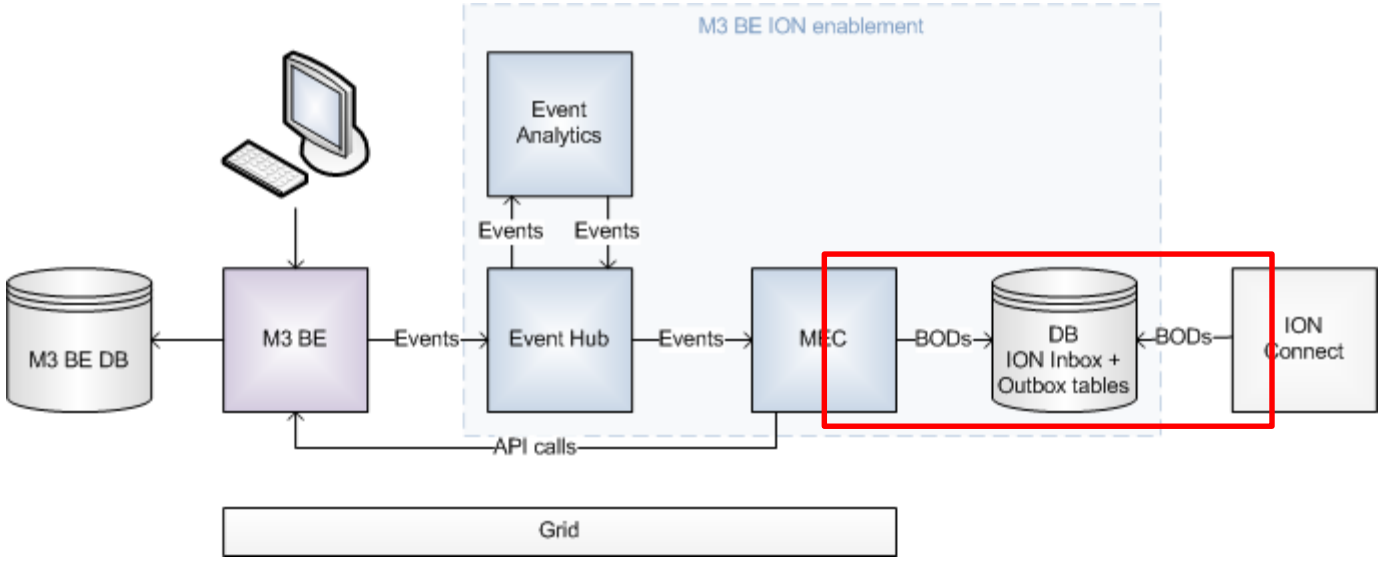
Message flows – inbound

- A non-SOR application sends a ProcessItemMaster BOD to M3 BE
- M3 BE will send back the reply BOD AcknowledgeItemMaster



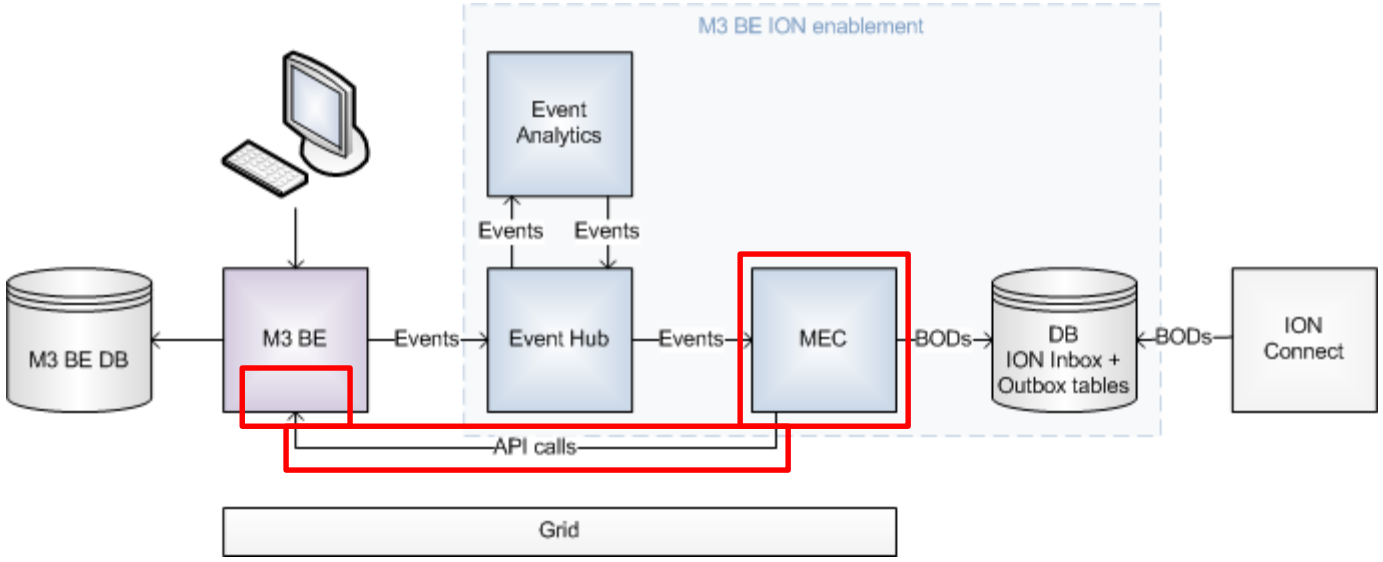
Message flows – inbound

1. ION puts the ProcessItemMaster BOD into the inbox database tables
2. MEC fetches the BOD from the ION inbox database tables
3. MEC detects a partner agreement, which defines processes to execute



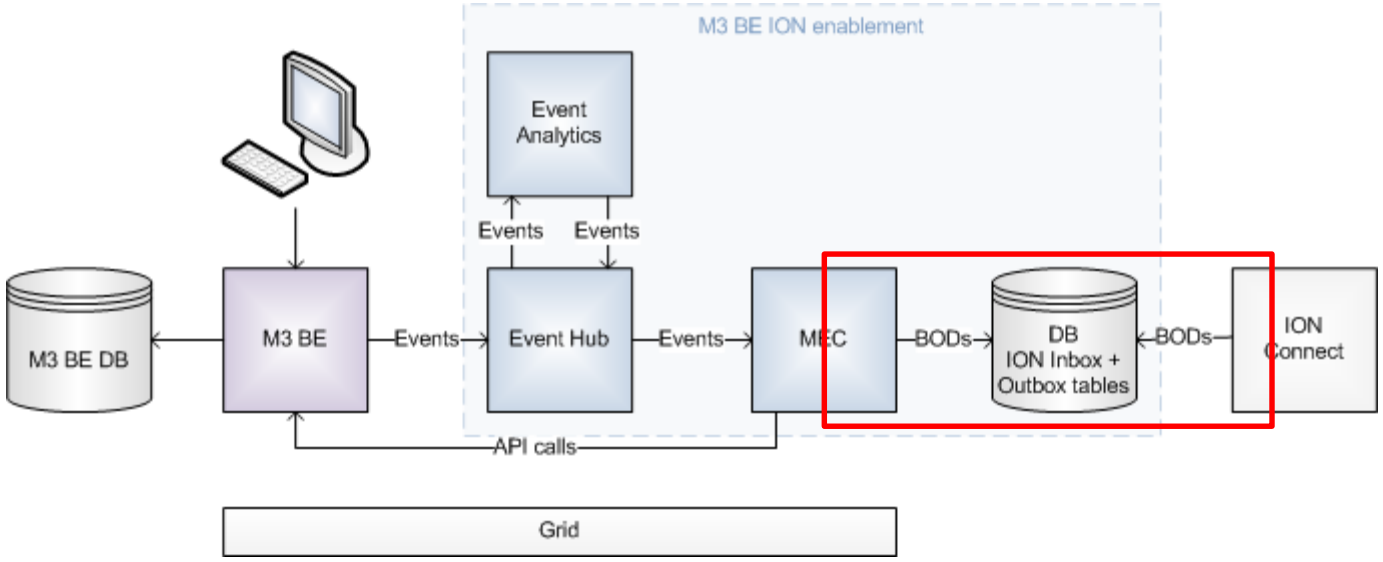
Message flows – inbound

4. An XML Transform process semantically maps the ProcessItemMaster BOD data to M3 BE format
   - M3 BE is updated by calling M3 BE APIs
   - All semantic transformation and data formatting, translation and restructuring take place in this process



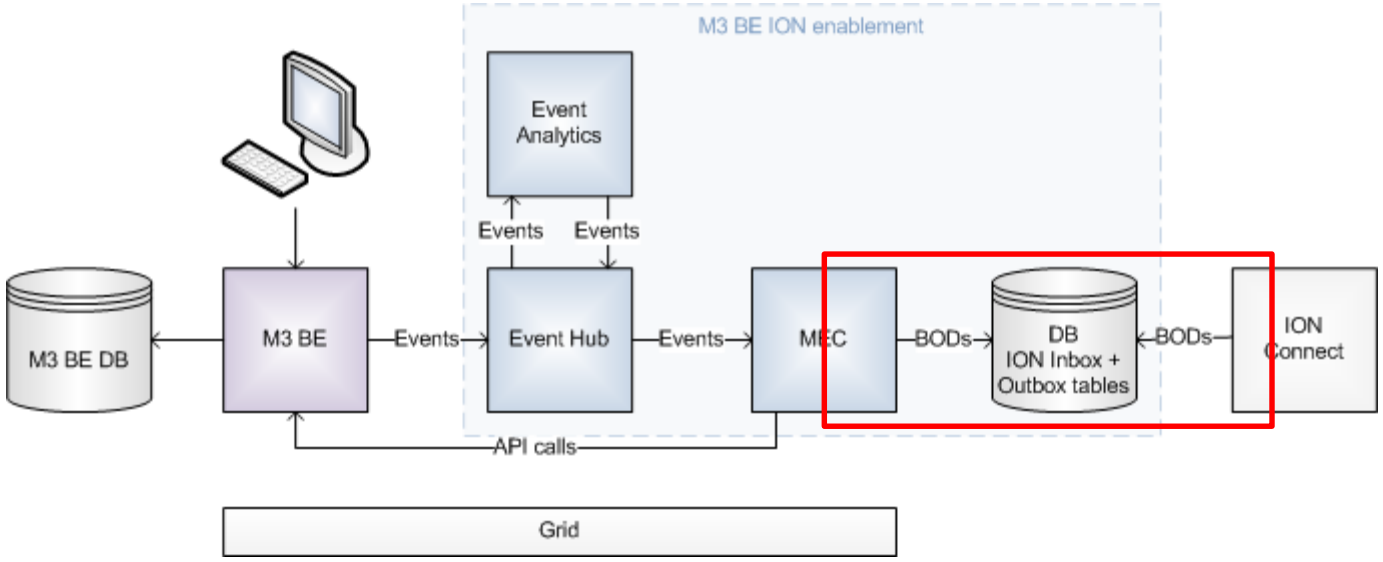
Message flows – inbound

5. The reply BOD AcknowledgeItemMaster can either be created synchronously by the same XML transform process and then be sent back to ION, or asynchronously when an M3 BE event triggers a separate outbound message flow



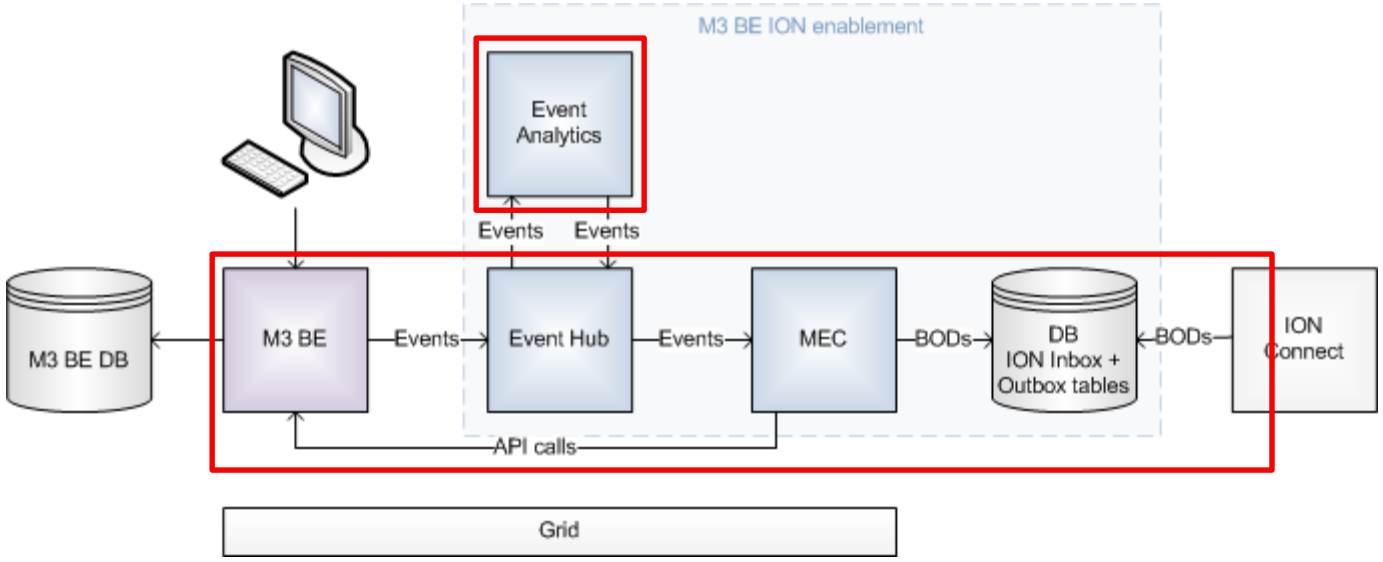
Message flows – inbound

6. If there is an error in the XML transform process a ConfirmBOD is sent to ION as well as an AcknowledgeItemMaster BOD with actionCode “Rejected”
   - The agreement’s error processes takes care of this!

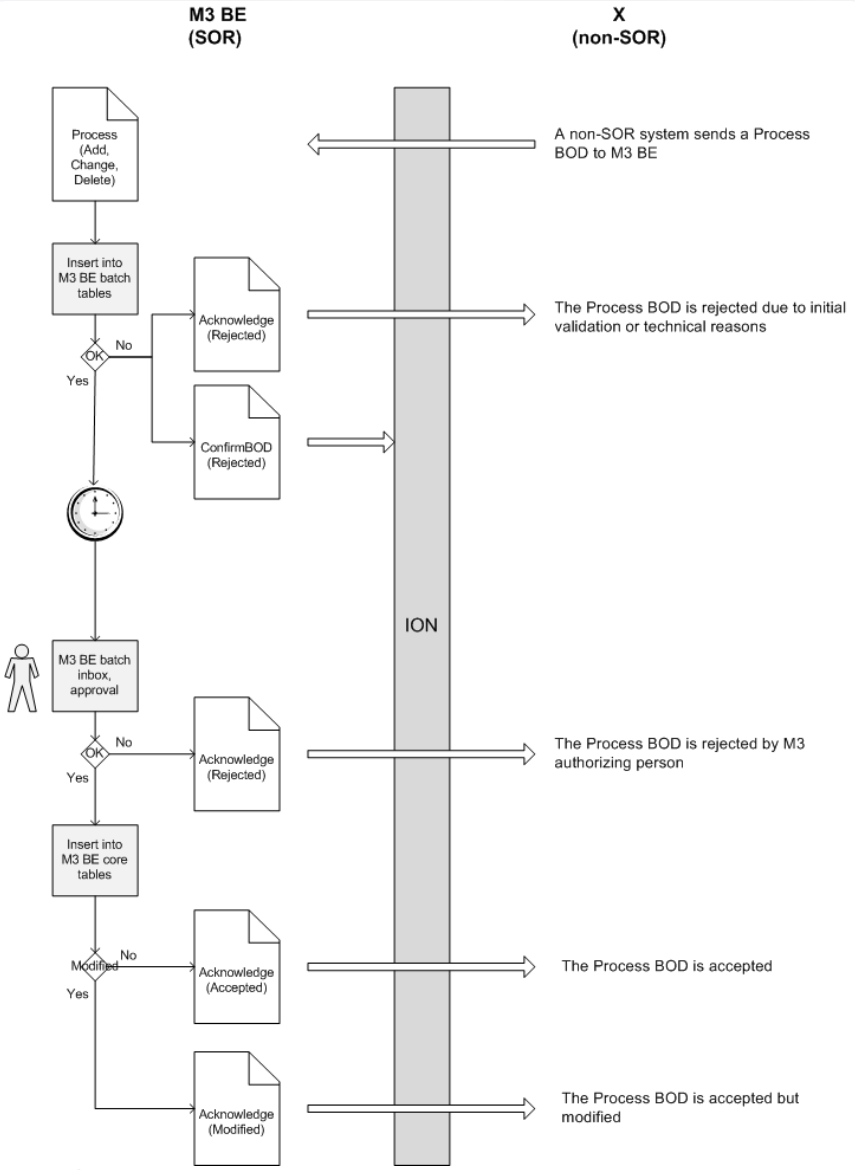


Message flows – inbound

7. Since an update of the MITMAS record in M3 BE is done by the XML transform process an application event triggers a separate outbound message flow for the SyncItemMaster BOD



Message flows – Async. Acknowledge BOD



Technical Walkthrough - ION

Applications

Applications – Deliverables

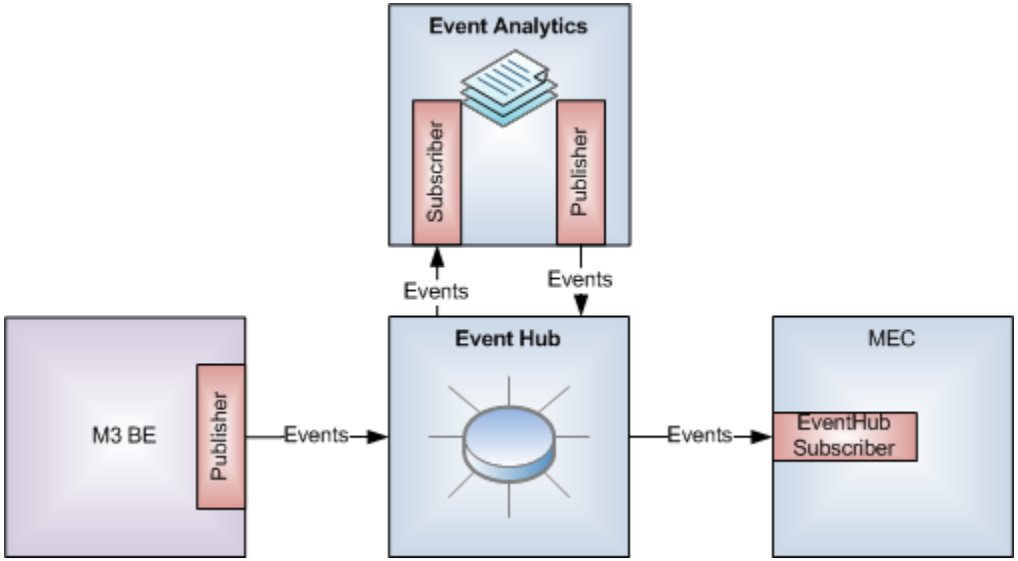
- The delivery package for ”a BOD” consists of
  - MEC mapping
  - Event Analytics rules for outbound BODs
- Common deliverables are
  - BOD Mappings and Descriptions documentation in PDF format
  - Installation and setup documentation
    - Includes references to required M3 BE MCE packages

Applications – Event rules

- Problems:
  - The M3 BE database structure does not match BOD data structures
  - Modifications of a record in one M3 BE table can trigger zero, one or several BODs
  - The triggering of BODs may also depend on the data itself
    - For example you do not want to send a BOD to ION when only data that is *not* included in the BOD has changed in M3 BE
- Solution: Send M3 BE database events to a rules engine!
- Event Analytics subscribes to M3 BE database events and applies rules on these
  - You can write everything from basic to very sophisticated rules using the built-in JBoss Drools Expert rules engine

Applications – Event rules

- The purpose of these rules is to create new events that correspond to the BODs to be created
- These events are posted to the Event Hub and MEC then subscribes to them



Applications – Event rules

- Here are the rules delivered for the noun ItemMaster:
- Changes in the following M3 BE tables trigger a SyncItemMaster BOD to be sent to ION:
  - MITMAS (ItemMaster/ItemHeader)
  - MITLAD (ItemMaster/ItemHeader)
  - MITFAC (ItemMaster/ItemHeader)
  - MITBAL (ItemMaster/ItemLocation)
- MHIMAS triggers an AcknowledgeItemMaster BOD to be sent to ION

Active	Name	Manage
☑	MITBAL_CREATE_DELETE	Edit Upload
☑	MITBAL_UPDATE	Edit Upload
☑	MITFAC_UPDATE	Edit Upload
☑	MITLAD_CREATE_DELETE	Edit Upload
☑	MITLAD_UPDATE	Edit Upload
☑	MITMAS_ACTIVATE	Edit Upload
☑	MITMAS_CREATE	Edit Upload
☑	MITMAS_DELETE	Edit Upload
☑	MITMAS_UPDATE	Edit Upload
☑	MHIMAS_UPDATE	Edit Upload

Applications – Event rules – example

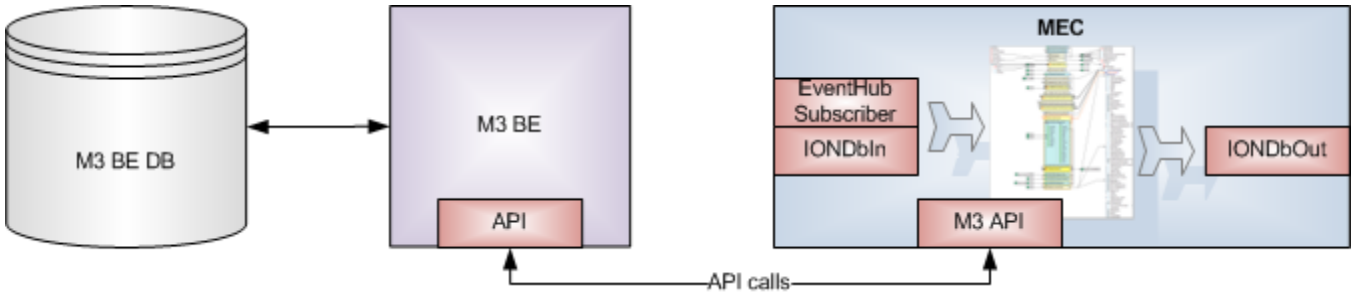
```
. . .

declare HubEvent
   @typesafe(false)
end

rule "annotations (MITMAS to MITMAS_UPDATE)"
   @subscription(M3:MITMAS:U)
   @subscription1(VF_MITMAStoMITMAS_UPDATE:MITMAS:U)
   then
end

rule "Create virtual fields (MITMAS to MITMAS_UPDATE)"
   no-loop
   when
      event : HubEvent(
         publisher == "M3",
         documentName == "MITMAS",
         (
            operation == EventOperation.UPDATE
         ) &&
         (elementOldValues["STAT"] >= 20 && elementValues.STAT >= 20) &&
(elementValues.FUDS != elementOldValues["FUDS"] || elementValues.CPUN !=
elementOldValues["CPUN"] || elementValues.STAT != elementOldValues["STAT"] ||
elementValues.SALE != elementOldValues["SALE"] || elementValues.INDI !=
elementOldValues["INDI"] || elementValues.STCD != elementOldValues["STCD"] ||
elementValues.CHCD != elementOldValues["CHCD"] || elementValues.BUAR !=
elementOldValues["BUAR"] || elementValues.ITCL != elementOldValues["ITCL"] ||
elementValues.ITGR != elementOldValues["ITGR"] || elementValues.PRGP !=
elementOldValues["PRGP"] || elementValues.UNMS !=
elementOldValues["UNMS"])
      )
   then
      Event $MITMAS_UPDATE = new Event("SyncItemMaster",
event.getOperation());
      $MITMAS_UPDATE.setTrackingId(event.getTrackingId());

      // Key fields
      // Fields
      $MITMAS_UPDATE.addElement("CONO", event.getElementValue("CONO"));
      $MITMAS_UPDATE.addElement("ITNO", event.getElementValue("ITNO"));

      // Calculation fields

      // Virtual fields
      // Unique key
      String $keyValue = event.getElementValue("keyValue");
      $MITMAS_UPDATE.addElement("keyValue", $keyValue);

      // Origin key value
      String $originKeyValue = event.getElementValue("originKeyValue");
      if($originKeyValue == null){
         $originKeyValue = event.getElementValue("keyValue");
      }
      $MITMAS_UPDATE.addElement("originKeyValue", $originKeyValue);

      // Origin fields
      String $originPublisher = event.getElementValue("originPublisher");
      if($originPublisher == null){
         $originPublisher = event.getPublisher();
      }
      $MITMAS_UPDATE.addElement("originPublisher", $originPublisher);

      String $originDocument = event.getElementValue("originDocument");
      if($originDocument == null){
         $originDocument = event.getDocumentName();
      }
      $MITMAS_UPDATE.addElement("originDocument", $originDocument);

      $MITMAS_UPDATE.postEvent();
end
```

Applications – BOD mapping

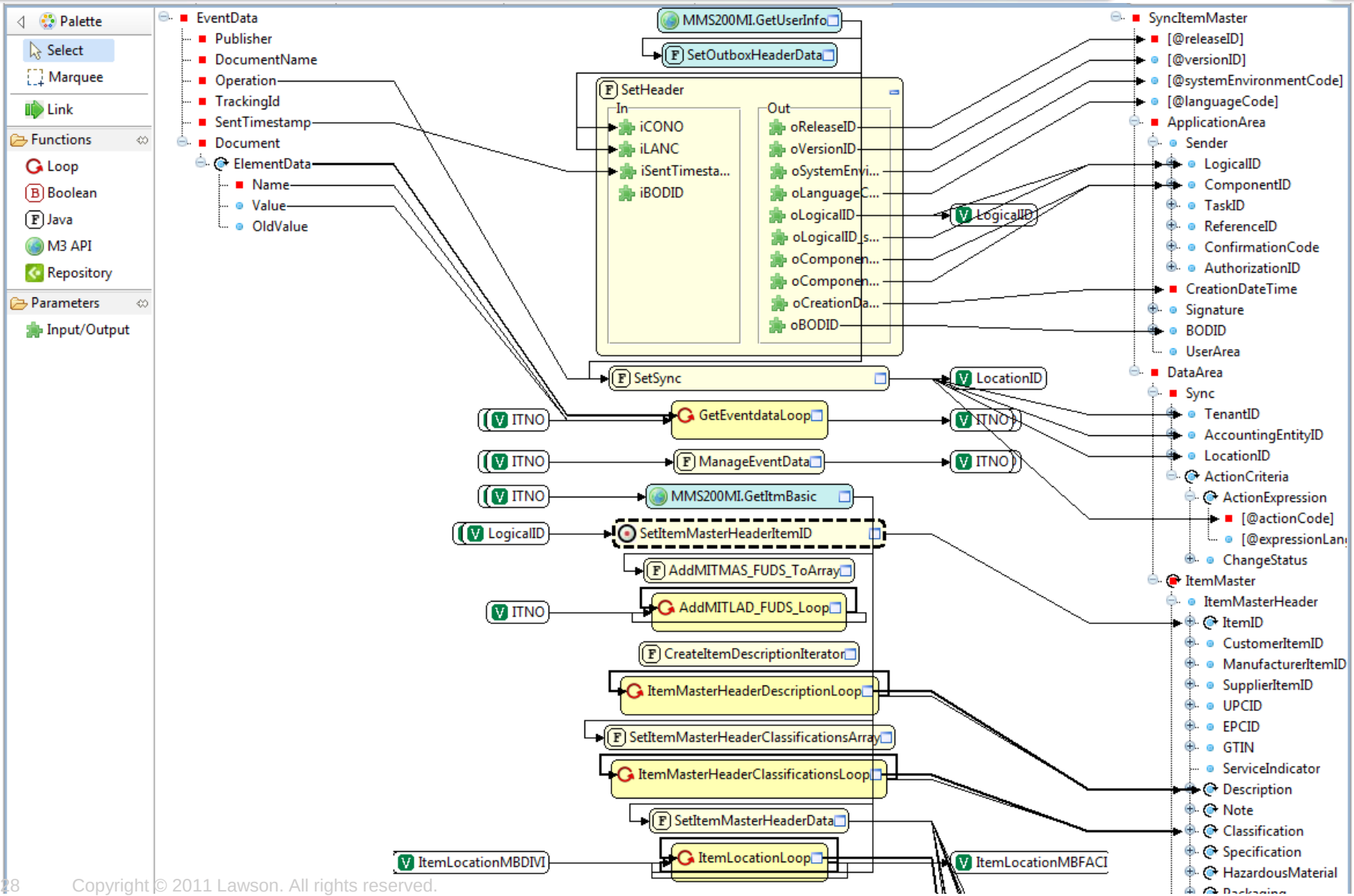
- MEC mappings do the semantic transformation and data formatting, translation and restructuring between M3 BE and BODs
- For outbound messages the input is the event from the Event Hub (in XML format) and the output is the BOD
- For inbound messages the input is the BOD and the output, if exists, is a reply BOD (i.e. Acknowledge or Show)

Applications – BOD mapping

- In the mapping you can fetch data from M3 BE and/or update data in M3 BE via calls to M3 BE APIs
- A MEC mapping is delivered as metadata in a .map file plus the XML schemas (.xsd files) used for the mapping



Applications – BOD mapping – example

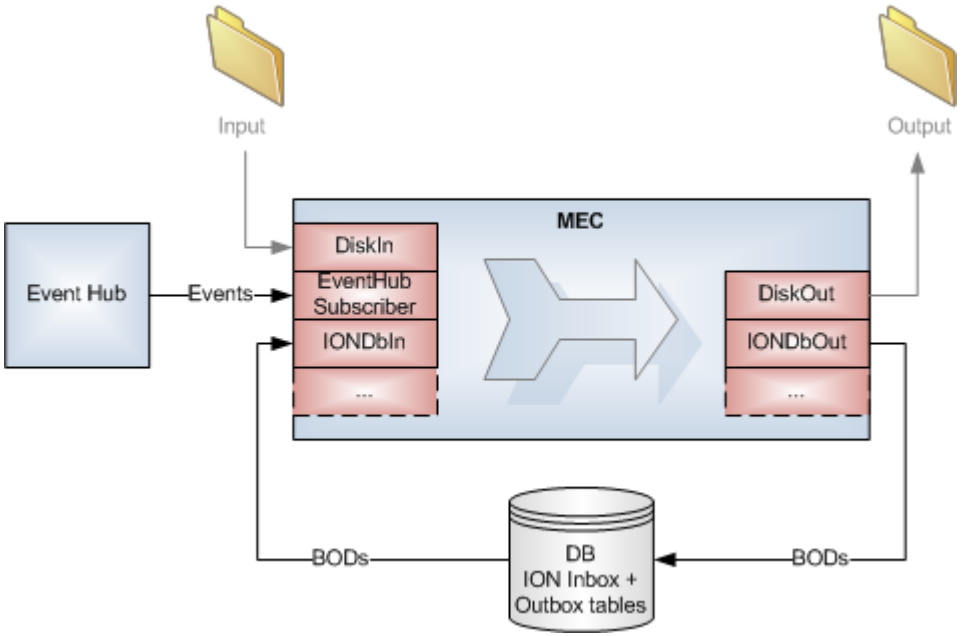


Technical Walkthrough - ION

MEC details

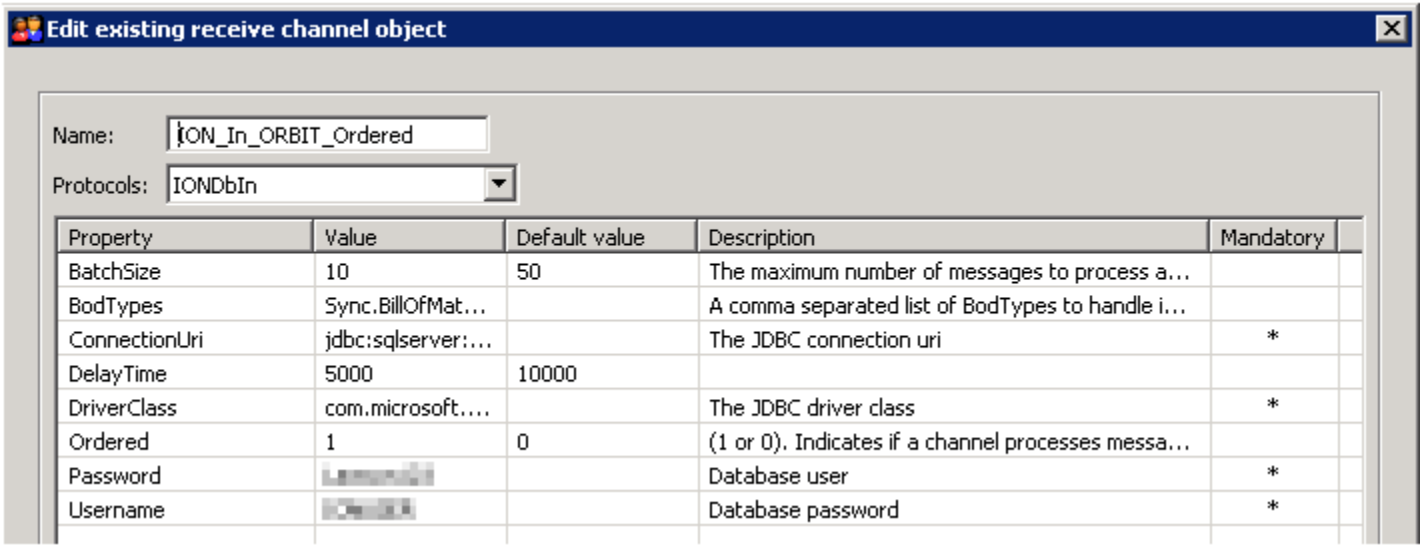
MEC details – Communication

- MEC v9.2 contains two new communication protocols:
  - IONDbIn – polls BODs from ION inbox database tables
  - IONDbOut – puts a BOD into the ION outbox database tables
- MEC v9.2 Event Hub Subscriber receive protocol allows several communication channel instances (for ordering)



MEC details – Communication – IONDbIn

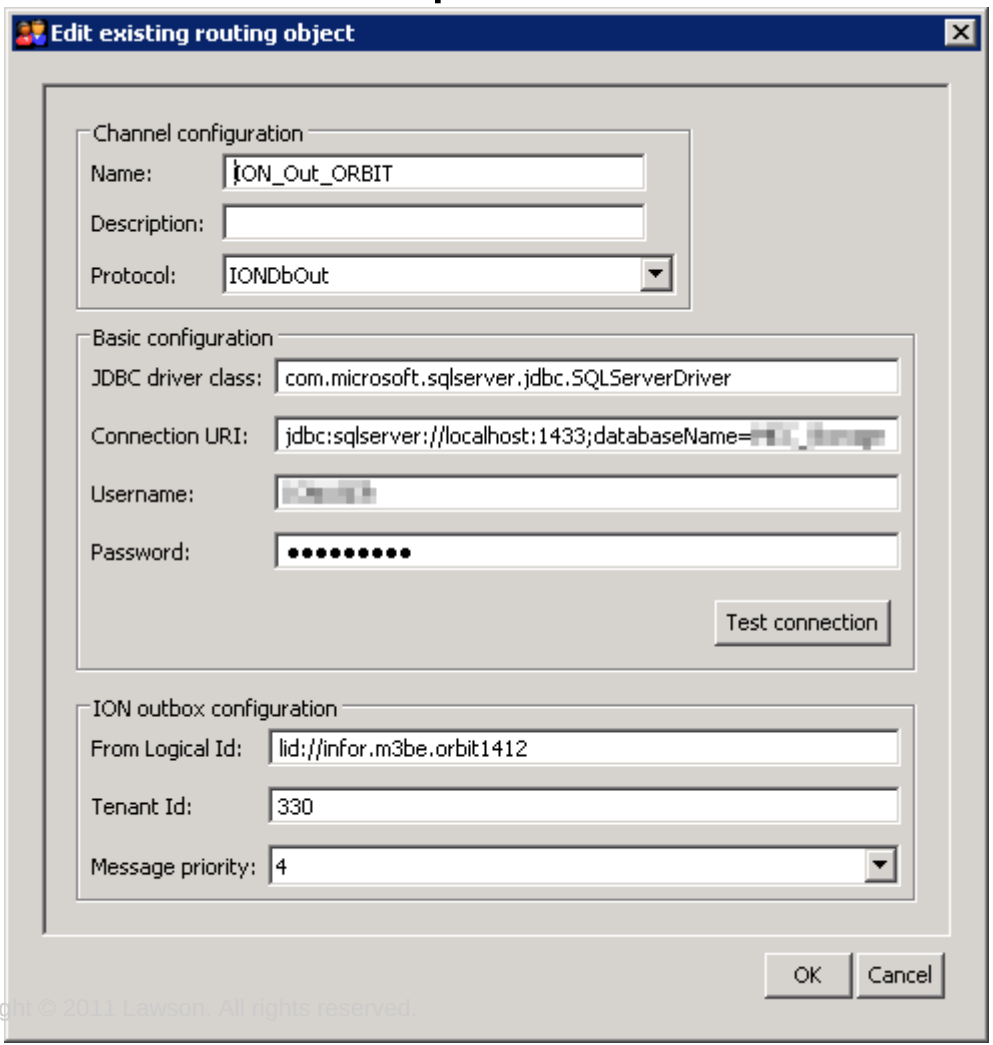
- IONDbIn setup in Partner Admin Tool:



- Runtime communication channels:

MEC details – Communication – IONDbOut

- IONDbOut setup in Partner Admin Tool:



MEC details – Communication – IONDbOut

- The ION outbox tables need to be populated with data in addition to the actual BOD
- This is handled by the Send process when using the IONDbOut protocol
- A lot of information is taken from the manifest and needs to be set correctly
  - Static information for the BOD should be set in the BOD mapping since this is not to be configured at the customer
  - Installation dependable information is set using control properties for the partner agreement, for example from and to logical IDs
- The following slide contains a summary for how this additional data is set by IONDbOut

MEC details – Communication – IONDbOut

- BODType (header)
  1. map:ionBODType * P
  2. agr:ionBODType **
  3. throw exception
- ToLogicalId (header)
  1. if verb is Confirm: “lid://default” P
  2. map:ionToLogicalId *
  3. agr:ionToLogicalId ** P
  4. if verb is Acknowledge or Show: com:ionFromLogicalId *** P (warning if missing)
  5. “lid://default”
- MessageId (header)
  1. map:ionMessageId *
  2. new UUID P
- FromLogicalId (header)
  1. map:ionFromLogicalId *
  2. agr:ionFromLogicalId ** P
  3. send communication object property ‘FromLogicalId‘
  4. throw exception
- MessagePriority
  1. map:ionMessagePriority *
  2. agr:ionMessagePriority **
  3. send communication object property 'MessagePriority’
  4. “4”
- TenantId
  1. map:ionTenantId * P
  2. send communication object property 'TenantId'
  3. “infor”, warning
- Encoding (header)
  1. Current encoding from manifest P
  2. “UTF-8”
- ReferenceId (header)
  1. map:ionReferenceId * P
  2. if verb is Acknowledge or Show: com:ionMessageId *** P (throw exception if missing)
- BODId (header)
  1. map:ionBODId *
  2. agr:ionBODId **
  3. don't set header (not mandatory)
- VariationId (header)
  1. map:ionVariationId *
  2. com:ionVariationId ***
  3. don't set header (not mandatory)

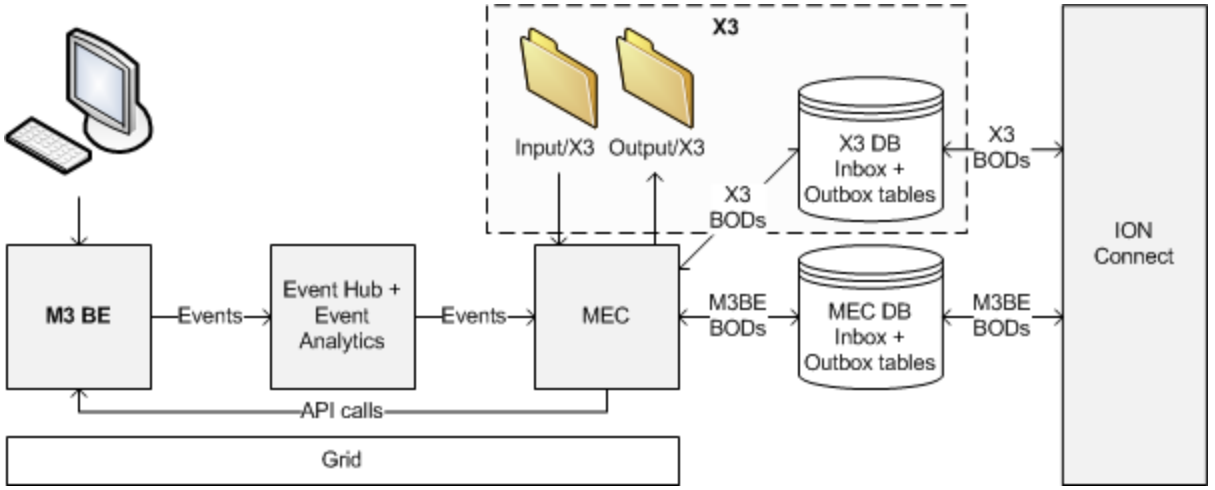
*	Manifest item set in mapping
**	Manifest item set for agreement or group
***	Manifest item set by IONDbIn
P	Preferred

MEC details – Communication – testing

- **Note:** Always test your BOD mappings using the IONDbIn and IONDbOut communication protocols!
- You can start testing using DiskIn and DiskOut, but since the ION communication uses manifest data set by the mapping and by agreement control properties you must also test using ION communication

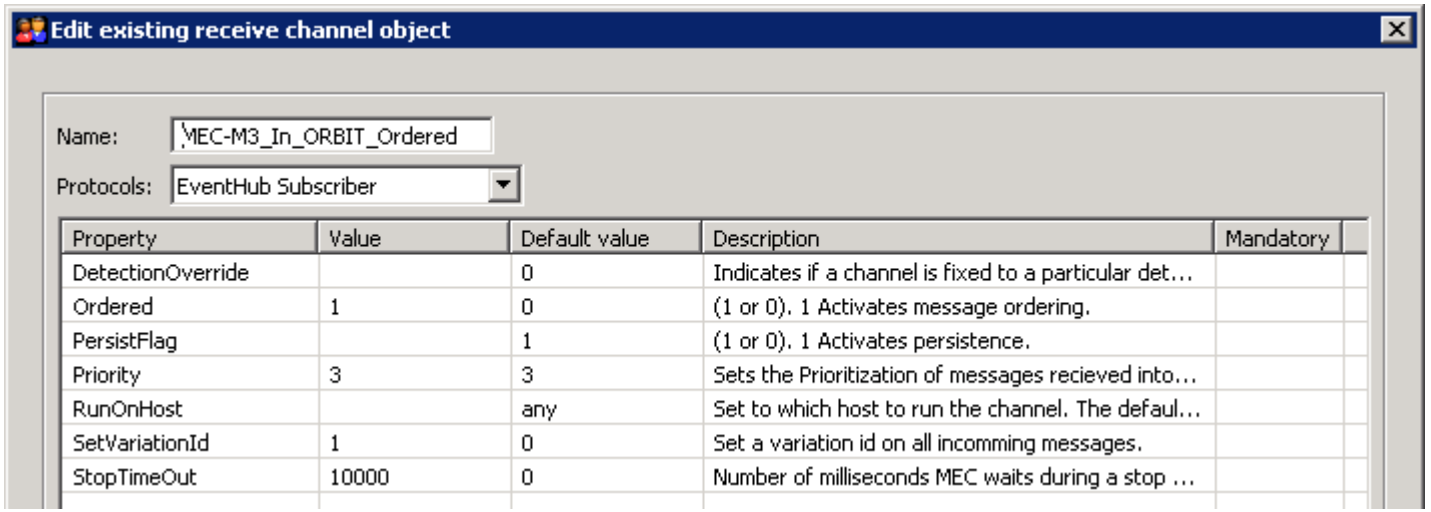
MEC details – Communication – testing

- It is possible to set up a ”virtual” application (X3) in MEC and have ION route documents between M3 BE and X3
  - When dropping a test file into the Input/X3 folder ION will route the BOD to M3 BE
  - When sending a BOD from M3 BE to X3 the XML file will be created in the folder Output/X3
  - The MEC setup is quite schizofrenic, when you understand how to do this then you have graduated!



MEC details – Communication – Event Hub

- Event Hub Subscriber setup in Partner Admin Tool:



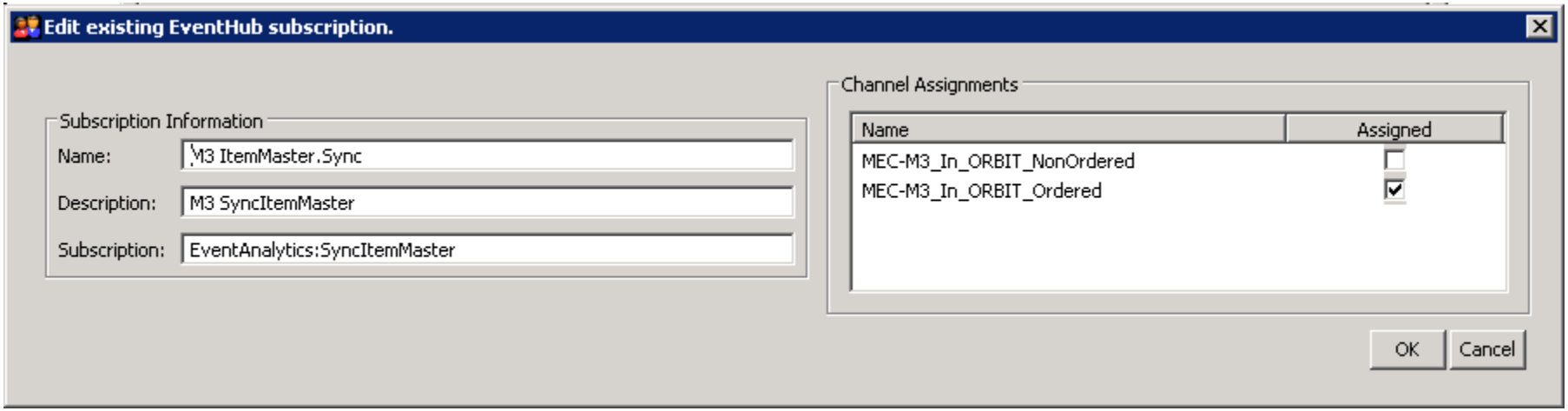
Property	Value	Default value	Description	Mandatory
DetectionOverride		0	Indicates if a channel is fixed to a particular det...	
Ordered	1	0	(1 or 0). 1 Activates message ordering.	
PersistFlag		1	(1 or 0). 1 Activates persistence.	
Priority	3	3	Sets the Prioritization of messages recieved into...	
RunOnHost		any	Set to which host to run the channel. The defaul...	
SetVariationId	1	0	Set a variation id on all incomming messages.	
StopTimeOut	10000	0	Number of milliseconds MEC waits during a stop ...	

- Runtime communication channels:

MEC-M3_In_ORBIT_NonOrdered	EventHub Subscriber	RUNNING	Pause / Resume
MEC-M3_In_ORBIT_Ordered	EventHub Subscriber	RUNNING	Pause / Resume

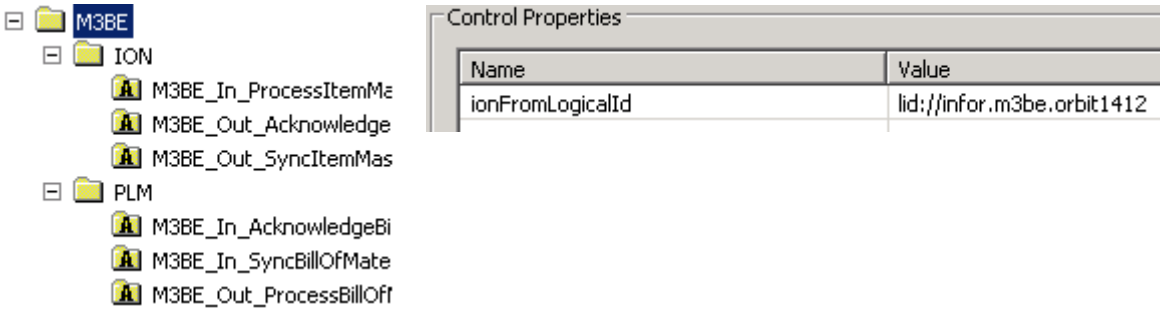
MEC details – Communication – Event Hub

- Event Hub Subscription setup in Partner Admin Tool:
  - You can assign a subscription to a communication channel

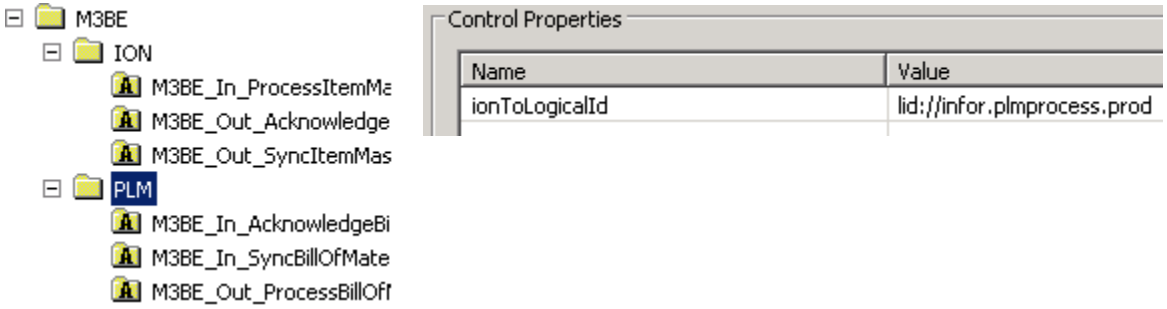


MEC details – Control properties

- Use control properties in Partner Administrator Tool to set up from and to logical IDs for ION
  - Enter from logical ID on the M3BE group:



  - Enter to logical ID on the application group:



MEC details – Check Order

- ION Connect does not guarantee delivery of BODs in the correct order
  - You can for example set up work flows in ION that will delay some BODs
- On the noun ID element there is an attribute ”variationID” that contains a sequence number
- When a Sync BOD is sent out from a system, a variationID is required

```
<ItemMaster>
- <ItemMasterHeader>
  - <ItemID>
      <ID variationID="710053" lid="lid://infor.m3be.orbit1412">NRFIN03</ID>
    </ItemID>
```

MEC details – Check Order

- This variationID can be used at the receiving end to discover messages that are received out of sequence
  - It is in our best interest to try to keep messages in sequence across the bus, but in a multi-threaded environment this cannot be guaranteed
  - The variationID is an integer value that needs to be in the range of -2^63 (-9,223,372,036,854,775,808) and 2^63-1 (9,223,372,036,854,775,807)

MEC details – Check Order – outbound

- We guarantee correct variationID on outbound Sync BODs *per primary key*
- All events for outbound Sync BODs has to be received on an ordered Event Hub Subscriber communication channel
  - Property Ordered = 1
- A unique variationID (manifest item com:ionVariationId) must be generated by the communication channel
  - Property SetVariationId = 1
- The first process in the partner agreement must be ”Check Order”

MEC details – Check Order – outbound

- XPaths to all primary key elements in the event must be given for Check Order
  - For example /EventData/Document/ElementData[1]/Value for CONO and /EventData/Document/ElementData[2]/Value for ITNO
- The Check Order process sequenciates messages with the same primary key – other messages are processed in parallell
  - Thus performance is maximized!

MEC details – Check Order – inbound

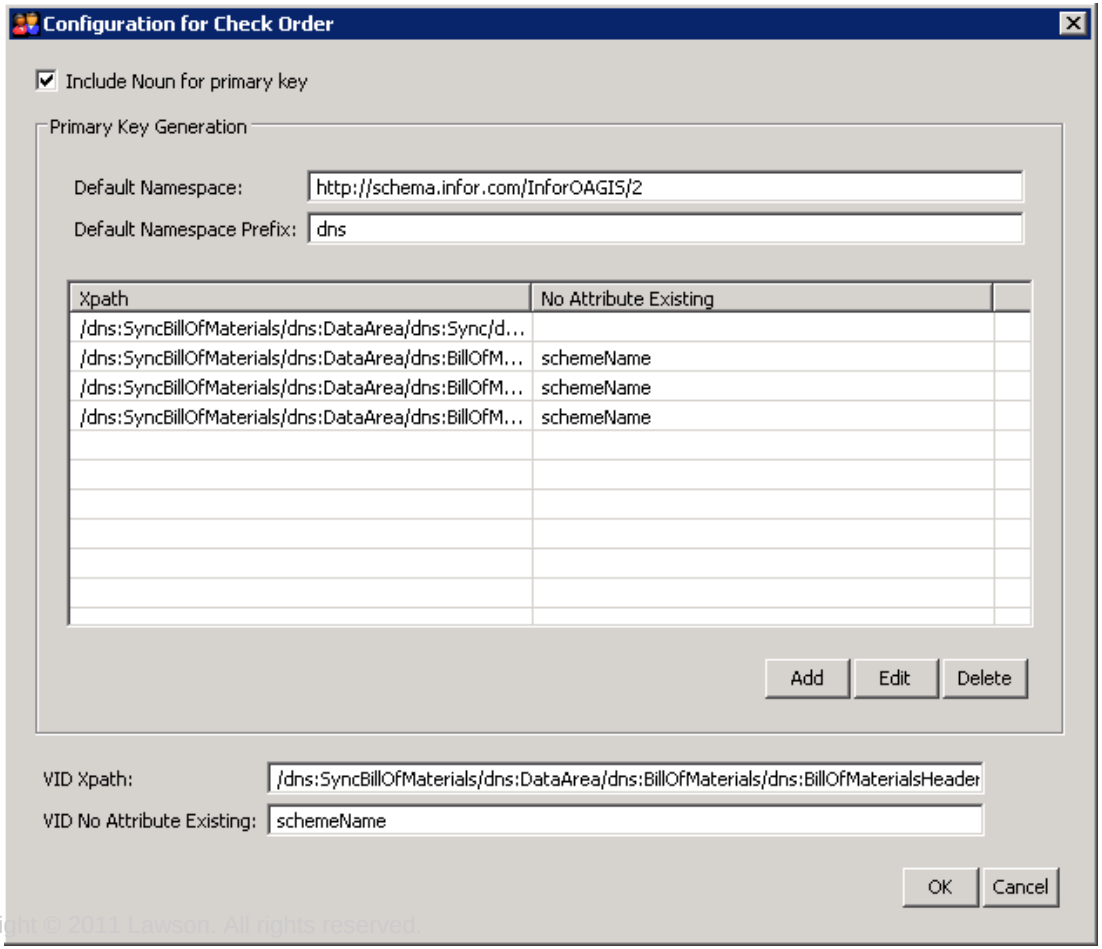
- This is what we do:
  - Keep track of the variationID for the latest successfully processed BOD per primary key
    - Example: Primary key for SyncBillOfMaterials is TenantID (CONO) + ID@accountingEntity (DIVI) + ID@location (FACI) + ID (PRNO)
  - If a received BOD has a variationID *less than* the latest variationID for its primary key it is discarded
    - A Sync BOD always replaces all existing data, hence an old Sync BOD can be discarded
  - If a received BOD has a variationID *greater than* the latest variationID for its primary key, and another BOD with the same primary key is currently being processed, the new BOD will be processed after the other BOD has finished processing
    - Other BODs are processed in parallell to maximize performance
  - If the processing fails the latest variationID for the primary key is reset to its previous variationID

MEC details – Check Order – inbound

- All inbound Sync BODs has to be received on an ordered IONDbIn communication channel
  - Property Ordered = 1
- The first process in the partner agreement must be ”Check Order”
- XPaths to all primary key elements in the event must be given for Check Order
- XPath for the variationID attribute must also be given
- The Check Order process discards messages with an old variationID and sequenciates messages with the same primary key – other messages are processed in parallell

MEC details – Check Order

- Example on Check Order set up for inbound SyncBillOfMaterials:



MEC details – Error handling

- For all inbound messages (received from ION) that fails in MEC it is required to send a ConfirmBOD BOD back to ION
- There is a new error process available in MEC v9.2, ”Crt ConfirmBOD”, that creates a ConfirmBOD automatically
  - The information given in the MEC error email will be given as error description
  - Crt ConfirmBOD automaticaly includes the received BOD (base64 encoded) in the ConfirmBOD as original BOD according to ION specifications
- The error processes for all inbound agreements must start with the processes Crt ConfirmBOD followed by the process Send with the protocol IONDbOut

MEC details – MEC Utilities

- **IONToolbox** and **IONApplicationArea** are new MEC utility classes included in MEC v9.2
- **IONToolbox** contains methods to be used for
  - ApplicationArea handling – getSystemEnvironmentCode(), getSenderLogicalID(), getMappingVersion(), getBEVersion()
  - LocationID handling – createLocationID(), getFACIFromLocationID(), getWHLOFromLocationID()
  - String handling – isEmpty(), rightTrim()
  - Date handling – normalizeXMLDateTime(), createNormalizedXMLDateTime()
  - ActionCode handling – createSyncActionCode(), createProcessActionCode()
    - Constants for all valid action codes – ACTION_CODE_ADD, ACTION_CODE_CHANGE, ACTION_CODE_REPLACE, ACTION_CODE_DELETE, ACTION_CODE_ACCEPTED, ACTION_CODE_REJECTED
  - Error handling (for Acknowledge BODs) – getReasonFromStackTrace()

MEC details – MEC Utilities

- **IONApplicationArea** is used when you need to send a reply Acknowledge BOD asynchronously
  - You need to store the received message's from logical ID, message ID (to be used as reference ID for the Acknowledge BOD) and ApplicationArea values in the MEC database
    - This data will be used by the Acknowledge BOD mapping
    - A new MEC API DBMap is used for storing data in the MEC database
  - **Note:** You need to store a correlation ID in M3 BE!
    - Default is the MEC message UUID

Technical Walkthrough - ION

Mapping tool

Mapping tool – Eclipse plugin

- A newly developed generic mapping tool, implemented as an Eclipse plug-in
- The new mapping tool has a very generic framework built on plug-ins to separate the basic mapping functionality from the business document format and the backend API format
  - Currently there are plug-in implementations for XML schemas and M3 API to replace the old MEC mapping tool, but other plug-ins can easily be developed
- Use (for example) Subversion for code versioning
- Save the mapping metadata to the MEC database for runtime deployment

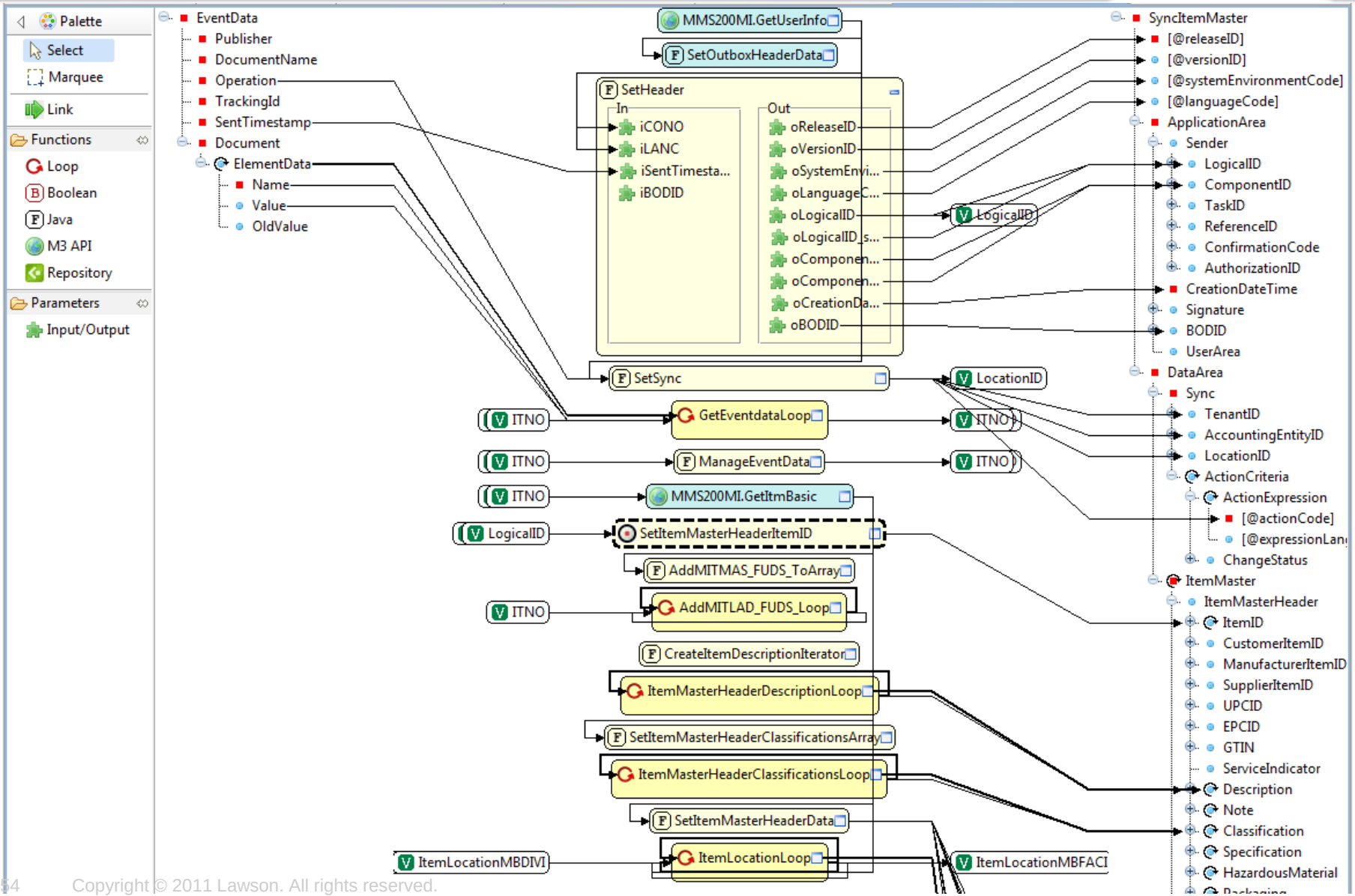
Mapping tool – Graphical programming

- Graphical representation of input and output XML schemas
- Supports ION BOD XML schemas as is
  - Not possible with the old MEC mapping tool
- Use loops, M3 APIs, Java functions, boolean functions, global variables and constants
- Graphical mapping of data and loop control
- Since the BOD XML schemas are used to parse and generate the XML documents the correct XML structure is always applied!
  - You can also automatically perform XML validation in runtime by using the Validate process in the partner agreement

Mapping tool – Mapping standards

- The following mapping functions should be inserted into your repository to be used as standard functions for BOD mappings:
  - **SetHeader** – sets root element attributes plus ApplicationArea values for outbound BODs
  - **GetProcessHeader** – gets ApplicationArea values to be used with *asynchronous* Acknowledge BOD mappings
  - **SetAcknowledge** – sets Acknowledge verb values for outbound *asynchronous* Acknowledge BODs
  - **SetProcess** – sets Process verb values for outbound Process BODs
  - **SetSync** – sets Sync verb values for outbound Sync BODs
  - **SetOutboxHeaderData** – sets manifest items used by IONDbOut send protocol (for outbox header table)
    - Must always be used for outbound BOD mappings!

Mapping tool – BOD mapping – example



Mapping tool – Java code example

```
/**
 * Set root element attributes plus ApplicationArea(created from repository)
 */
private void setHeader() {
    // Set OAGIS release
    oReleaseID = "9.2";

    // Set version of the given BOD definition
    oVersionID = "2.6.2";

    // Set "Test" or "Production" (default) mode
    oSystemEnvironmentCode = IONToolbox.getSystemEnvironmentCode(myMap);

    // Instantiate generic DataTranslator
    String blankDIVI = "";
    dtGen = new DataTranslator(myMap, iCONO, blankDIVI, "ION", "1", "Generic");

    // Set language for the contents of the BOD
    if (iLANC != null) {
        oLanguageCode = dtGen.toMessage("Generic", "languageCode", iLANC, "", "", "CSYTAB","LNCD");
    }

    // Set from logical ID
    oLogicalID = IONToolbox.getSenderLogicalID(myMap);

    // Set MEC mapping version
    oLogicalID_schemeVersionID = IONToolbox.getMappingVersion(myMap);

    // Set business application that issued the BOD
    oComponentID = "M3BE";

    // Set M3 BE version
    oComponentID_schemeVersionID = IONToolbox.getBEVersion(myMap);

    // Set date time stamp that the given instance of the BOD was created (UTC)
    oCreationDateTime = IONToolbox.normalizeXMLDateTime(iSentTimestamp);

    // Set GUID that will make each BOD instance uniquely identifiable, default MEC message UUID
    if (iBODID != null) {
        oBODID = iBODID;
    } else {
        oBODID = strUUID; // Example: "bf941e3f-c28a-4411-9a31-067d4d26c686"
    }
}
```


INFOR™
LAWSON