Energy Mashup Lab Rabbit MQ Implementation
Fall 2021

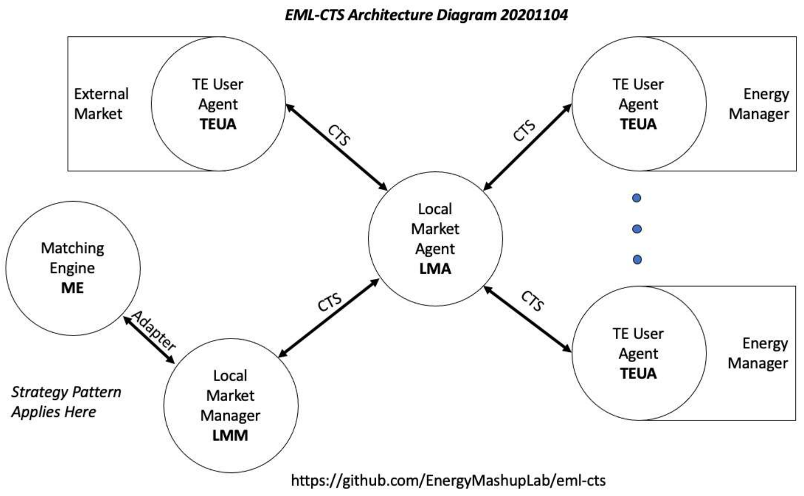
Professor:
Osama Eljabiri

Sponsor:
William Cox

Author:
Shane Calderwood

Table of Contents

1. Introduction
    a. Project Architecture Overview



The above diagram depicts the EML-CTS implementation architecture. The TE User Agent (TEUA) is the actor through which a user interacts with the system. The job of the TEUA is to transfer messages between EML-CTS and an energy manager or external market. The Local Market Agent (LMA) essentially serves as a router. It does not actually edit any of the messages that are sent to it. Instead, it forwards them on to the corresponding party. The main purpose of the LMA is to ensure that responses are returned to the TEUA that sent the original request. The Local Market Manager (LMM) forwards messages from the LMA onto the Parity Matching Engine (ME). The matching engine is not implemented by the EML-CTS project. Instead, the project uses an extension of the Parity Matching Engine which is external to the project.[1] The main purpose of the LMM is to serve as a bridge between the Parity Matching Engine and the EML-CTS project. It reformats messages from the LMA (tenders) so that they conform to the standards of the Parity Matching Client. Similarly, it reformats messages from Parity so that they conform to the EML-CTS standards.

The Project is currently driven using Postman[2]. The process begins when Postman sends a ClientCreateTender request to the TEUA. This request is a simplification of the standards-based Common Transactive Services EiCreateTenderPayload. The TEUA uses the ClientCreateTender request to build an EiCreateTenderPayload request which is then sent to the LMA. The LMA receives the EiCreateTender payload and forwards it on to the LMM. The LMM then converts the EiCreateTender to a MarketCreateTender and sends it to Parity. It also replies to the LMA with an EiCreatedTenderPayload, notifying it that the

[1] https://github.com/EnergyMashupLab/parity-cts
[2] https://www.postman.com/

tender was created successfully. The LMA then takes the response and forwards it to the corresponding TEUA.

When the Parity Matching Engine matches two tenders (a buy and a sell) it creates a MarketCreateTransactionPayload and sends it to the LMM. The LMM converts it to an EiCreateTransactionPayload, and sends it to the LMA. The LMA then forwards the transaction onto the participating TEUAs where it is converted to a ClientCreateTransaction and sent to the Energy Manager (Postman in this case).

b. Rabbit MQ in EML CTS

Rabbit MQ is a message broker. The current iteration of the EML-CTS project sends messages using Restful operations. The issue with the Restful implementation is that it is not scalable. Restful operations are blocking which means that clients must wait for a response from the server before continuing execution, or in the alternative, create a complex asynchronous input/output system.. Since EML-CTS is designed to be a distributed system the Restful operations will need to be replaced.
RabbitMQ has been chosen as the susbstitue for the Restful operations as it is standards-based and open source. RabbitMQ will handle the plumbing for EML-CTS in the future. It is important to note that the overall structure and logic of the project will remain the same. Only the messaging system is changing.

2. Design
    a. Detailed EML-CTS Code Analysis

This section will dig into the actual EML-CTS code to show how the messaging system is currently implemented.
All code analysis and description is for the EML-CTS *poc-rabbitmq* branch for this Proof of Concept.
The controller files for the TEUA, LMA and LMM are stored in the complete/src/main/java/org/theenergymashuplab/cts/controller directory. We will start by looking at the code in the TeuaRestController.java file. Line 52 maps the URI of the TeuaRestController class to "/teua". Each method inside the class is also mapped to a specific URI. For example, the postClientCreateTender method is mapped to the URI: "/teua/{teuaId}/clientCreateTender". Posting to a method's URI executes the method. Inspection of one of the Postman scripts will show that a URL is listed for every message. For example, the first message in the clientCreateTender collection is posted to the URL: http://localhost:8080/teua/1/clientCreateTender. Posting to this address executes the TEUA 1's postClientCreateTender() method. The parameter for the method is is a ClientCreateTenderPayload. The postClientCreateTender method then converts the ClientCreateTenderPayload to an EiCreateTenderPayload and posts

it to the URL that maps to the LMA's postEiCreateTender() method. The post operation is done using the statement on line 334.

It is important to note that the return value form this post method is an EiCreatedTenderPayload. The Spring Boot framework allows a programmer to send a response through a return statement. Therefore, instead of posting the EiCreatedTender back to the TEUA using the restTemplatePostForObject() method, the LMA simply returns it. The TEUA then converts the LMA's response from a EiCreatedTenderPayload to a ClientCreatedTenderPayload and uses a return statement to send it back to Postman.

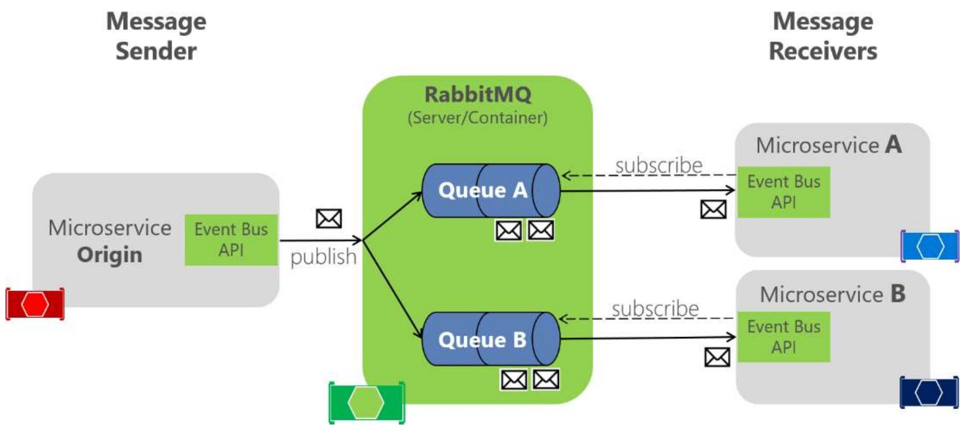
Now let's look at the LMA's postEiCreateTender() method, which can be found in the LmaRestController.java file. The method begins execution when a message is posted to the URL: http://localhost:8080/lma/createTender. The parameter for the method is a EiCreateTenderPayload. The LMA simply takes the parameter and forwards it onto the LMM using the restTemplate.postForObject() method on line 105.

The LMM's postEiCreateTender method is in the LmeRestController.java file. It is mapped to the URL: http://localhost:8080/lme/createTender. The method takes in the EiCreateTenderPayload posted by the LMA and puts it into a queue named queueFromLme. The queueFromLme will be touched on in the next section. The rest controller also maps the tender IDs to their EiCreateTenderPayload. Finally, the postEiCreateTender() method returns a EiCreatedTenderPayload to the LMA, which is then forwarded back to the corresponding TEUA.

EiCreateTenderPayloads are pushed to the queueFromLme by the LmeRestController, and they are picked up by the LmeSocketClient which resides in the complete/src/main/java/org/theenergymashuplab/cts directory. The LME socket client converts these EiCreateTender payloads to MarketCreateTenderPayloads and sends them off to the Parity Matching system.

When the Parity Matching system matches a buy and sell request, it sends a MarketCreateTransaction to back to the LmeSocketServer, which is then converted to a EiCreateTransactionPayload and pushed back to the LmeRestController using the eiCreateTransactionQueue. The EiCreateTransactionPayload is then sent back through the plumbing in the opposite direction as the EiCreateTenderPayload.

b. Architecture for Applying RabbitMQ to EML-CTS

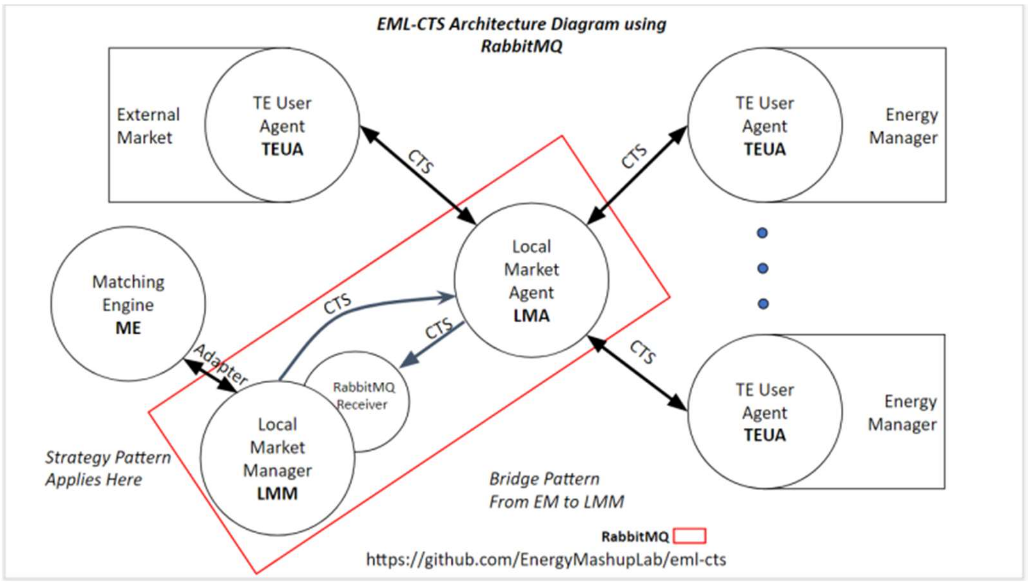


The above diagram depicts the basic RabbitMQ architecture. Clients send requests to topics or exchanges. Topics can contain multiple queues which correspond to different receivers. The RabbitMQ server is responsible for placing an exchange's incoming messages into the correct queue so that they are forwarded to the desired recipient.
Rabbit messages must be in either a string or a byte format. The EML-CTS project requires payload objects to be sent between actors. This can be accomplished by converting the Java objects into JSON strings using the Jackson API. In each case, the message sender will need to serialize the payload using Jackson. The receiver will then deserialize it back into the original object.

The overall architecture for the EML-CTS project is not altered significantly by the transition from Restful Operations to RabbitMQ. The TEUA, LMM, LMA and Client are all handled by a different controller. This will still be the case with RabbitMQ. Each actor will be represented by a RabbitMQ topic. The Rabbit server will then place the message in a queue which corresponds to the method the message should invoke.

The main difference between using RabbitMQ and restful operations is that RabbitMQ does not allow responses to be sent with a simple return statement. Therefore, the responses will need to be posted to the correct exchange. The lack of return statement also creates a correlation issue. The restTemplate.postForObject() method is a blocking operation. RabbitMQ's send method is nonblocking, which increases efficiency. However, responses will no longer be returned directly from the send method making it more difficult to determine which response goes with which request (e.g. which EiCreatedTenderPayload goes with which EiCreateTenderPayload). It is suggested that the tender ID and transaction ID fields be used as a correlation IDs to match responses with their respective requests.
In effect, the synchronous linking of a PostRequest to a PostResponse is replaced by correlation identifiers in the request message and the (asynchronous) reply message.

c. RabbitMQ Demo Description



The above image depicts the EML-CTS architecture of the RabbitMQ demo branch. The link that implements RabbitMQ is the link between the LMA and the LME. Specifically, the tender creation link. The all the other pipelines have been left untouched. Two files were added to the complete/src/main/java/org/theenergymashuplab/cts/controller directory. The MessagingRabbitmqApplication.java file is essentially a RabbitMQ configuration file. It sets up the exchange for the LME and binds a queue to it. It also sets up a listener. The Receiver.java class replaces the LmeRestController's postEiCreateTender() method. Details describing the method can be found in section 2.a.

The LmaRestController.java file has also been edited so that the message is sent to the receiver using the rabbitTemplate.convertAndSend() method. The LmaRestController also reformats the EiCreateTenderPayload to a EiCreateTenderPayloadRabbit. The conversion is necessary because the Jackson json conversion library cannot handle the Instant and Interval datatypes. However, Jackson is able to deserialize the BridgeInstant and BridgeIntervel classes. The EiCreateTenderPayloadRabbit class is simply an EiCreatTenderPayload with the Instant and Interval fields replaced by BridgeInstant and BridgeInterval.

It is also important to note that the receiver does not return anything back to the LMA in this architecture. Instead, the EiCreatedTenderPayload is created by the LMA and set back to the TEUA. This was done to simplify the architecture for a proof of concept and will need to be changed in the future. The EiCreatedTenderPayload should be constructed by the LME not the LMA. The LMA will simply forward the payload onto the TEUA, not create the payload itself. Therefore, the LMA will need an exchange of its own for the LME to send messages to.

3. Closing
    a. Conclusion

The EML-CTS project currently uses restful post operations to send messages between actors. Unfortunately, the current restful implementation is not scalable and will need to be changed. RabbitMQ has been selected to replace the restful operations. The transition to Rabbit will not require a major architectural change in the project. Instead, only the plumbing of the project will need to be updated. A RabbitMQ proof of concept has been created where the tender creation link between the LMA and LME has been updated to use Rabbit. The proof of concept can be found in the git branch named "dev-rabbitmq".

The "poc-rabbitmq" branch implements RabbitMQ on a version of the project that does not use docker. There are two major things that need to be done next. First, all the plumbing in the dev-rabbitmq branch needs to be updated to implement the Rabbit messaging system. The hardest link to implement will be the link between the TEUA and LMA, since it is a many to 1 link. Second, Rabbit will need to be integrated into the version of the project that runs using docker.