ISLA®

Common Domain Model

A FRAMEWORK FOR EVENT NEGOTIATION

CHRIS.RAYNER@ISLAEMEA.ORG

ISLA

1 Introduction

The purpose of this document is to provide a technical overview of how to use the objects and functions in the CDM to negotiate a new business event. The example we use is the negotiation of a new trade execution. The document covers four common scenarios found in real world trade negotiations, providing example JSON (valid for CDM version 4.0.0-dev.4) for each step of the workflow.

Note that JSON is the native data serialisation format for the CDM which is why we use it in this document. Other data formats are available and can be used if preferred.

The document is aimed at engineers and developers and is of a technical nature. Having a basic understanding of the CDM concepts would also be beneficial but is not assumed.

2 Overview

The CDM is built upon the concept of workflows. Each workflow can be broken down into a series of steps. Each step holds the data required to transition a trade from one state to another.

In this document we explain how one party can propose a new step in a workflow, and another party (or parties) can accept or reject the proposal. If the proposal is accepted, then the outcome is a new business event. If the proposal is rejected, then no event is generated, and the workflow stops at this point.

The example we use is the negotiation of a new securities lending trade; we assume that the borrower has already identified the need for the trade and determined the lender they wish to use to source the securities.

The borrower will propose the new trade execution to the lender, sending them the details of the trade as they see it. The lender can then either accept or reject the proposed trade.

If the lender accepts the proposal, then the trade execution was successful, and the trade can be created by both the borrower and the lender.

If the lender rejects the proposal, then the trade will not be executed, and no further action can be performed on this proposal.

The lender is also able to send a counter proposal back to the borrower. This would happen when the lender is happy to continue with the transaction but wants to change some of the terms within it. An example of this would be where the lender would like a different rate on the trade.

When the borrower receives a counter then they can accept it or reject it. If they want to change some of the terms of the new counter then they can generate another counter proposal themselves and send that back to the lender.

There is no restriction on the number of counter proposals that can be performed on a workflow, the workflow only stopping either when all parties accept the proposal or at least one party has rejected it.

It is also possible for either the lender or the borrower to initiate the trade negotiation. We use the example of the borrower sending the initial proposal as this is the most common scenario.

Note that there are many other lifecycle events that this workflow can be used for, the process and CDM objects are not restricted to negotiating trade executions. Some potential use cases could be when agreeing prices for mark to markets, updating margins, or negotiating a change to the terms of an existing trade.

In the following sections we describe how to model four of the most common trade execution negotiation sequences:

- **Propose – Accept**
  This is where one party proposes a new trade and the other party agrees to all the details and terms on the trade and accepts the proposal.

- **Propose – Reject**
  One party proposes a new trade but the other party does not want to continue with the transaction at all and rejects the proposal.

- **Propose – Counter – Accept**
  One party proposes the trade, but the other party wants to change some the terms of the trade and sends back a counter proposal. The initiating party is happy with the new details and accepts the counter proposal.

- **Propose – Counter – Reject**
  In this scenario one party proposes a trade but the other party offers a counter proposal. The initiating party is not happy with the counter, however, and no longer wants to continue with the transaction, sending back a rejection.

Before describing the technical details of each of these scenarios we'll briefly run through how to model each potential step in a workflow i.e. how to model a new proposal, and how to accept, reject or counter that proposal.

We'll also show how to track the lineage of the workflow, keeping track of each step along the way.

Detailed descriptions of the CDM objects and functions that are used for each of the scenarios are provided, with JSON examples too.

3 How to model a proposal

The WorkflowStep object allows a lot of detail to be entered. This can be roughly split into information that describes the workflow step itself (i.e. metadata) and the actual content of the workflow step (i.e. the data).

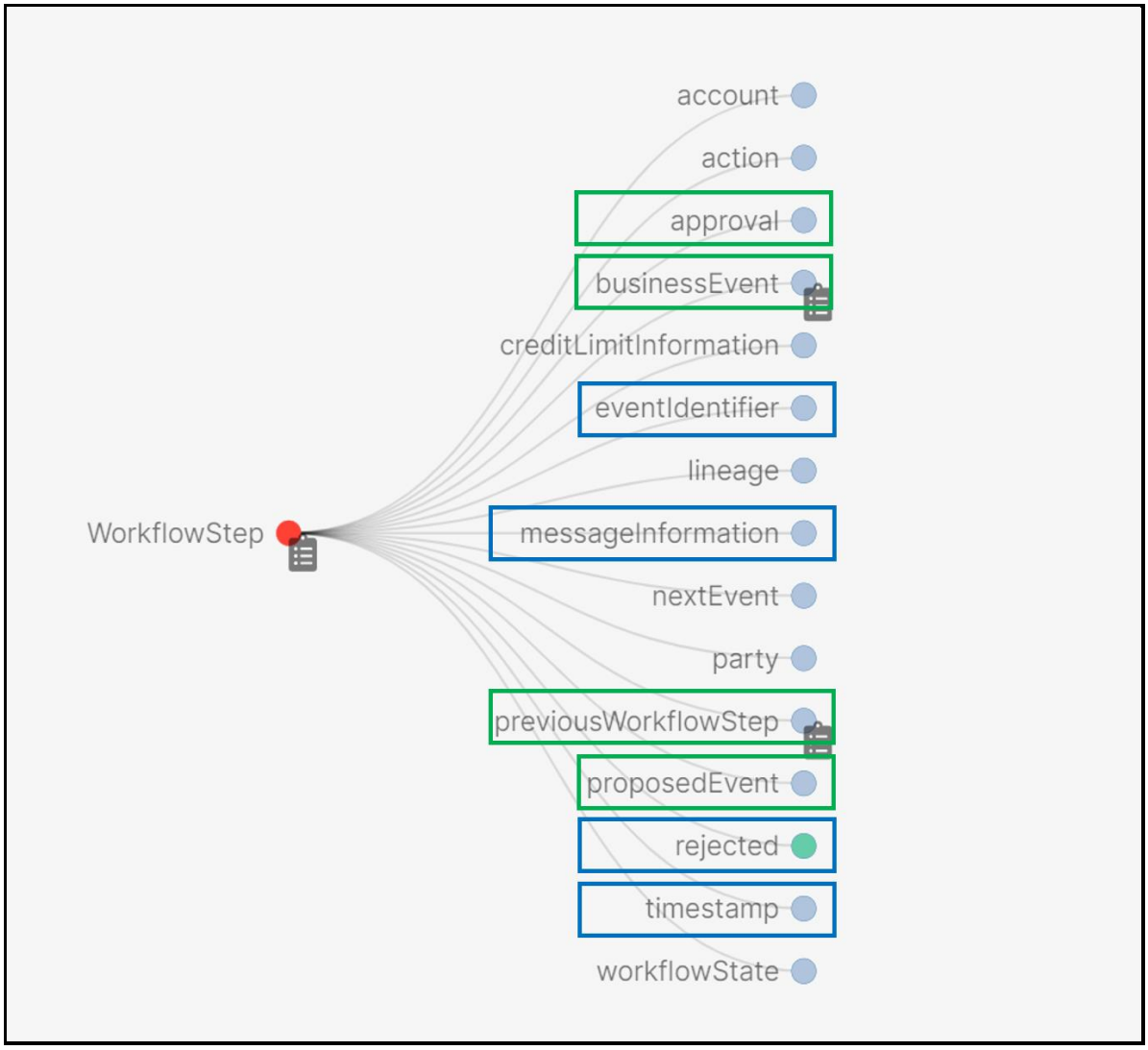

Diagram 1: Overview of the top level WorkflowStep object. Only the objects and attributes we will be referencing are highlighted. Metadata items are shown in blue, whereas objects holding data are shown in green.

In our trade negotiation example, the core data is a new proposed event, which is a trade execution being proposed by the borrower. The details of the execution are held in "proposedEvent" which is an instance of the "EventInstruction" object. This should describe the parties on the trade, the instrument being loaned and the economic terms of the trade.

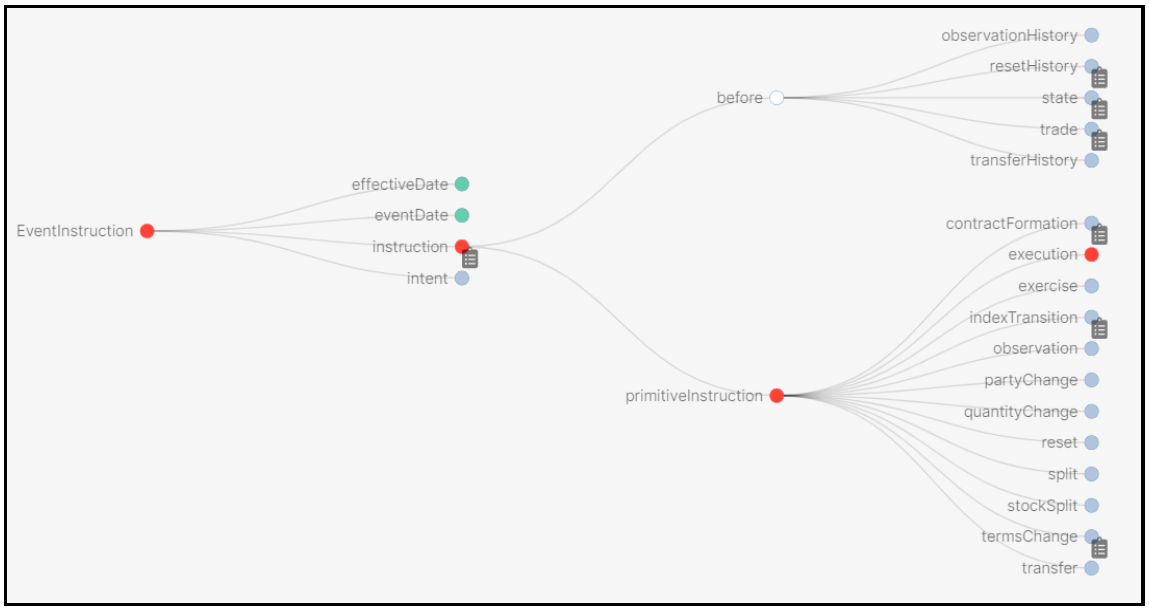

Diagram 2: The "proposedEvent" object is of type EventInstruction. The EventInstruction holds the "instruction" itself, which can include a "before" image of the trade prior to the event, and a "primitiveInstruction" that holds the data to use for the event. For our use case we will be using the "execution" primitive instruction, which is of type ExecutionInstruction.

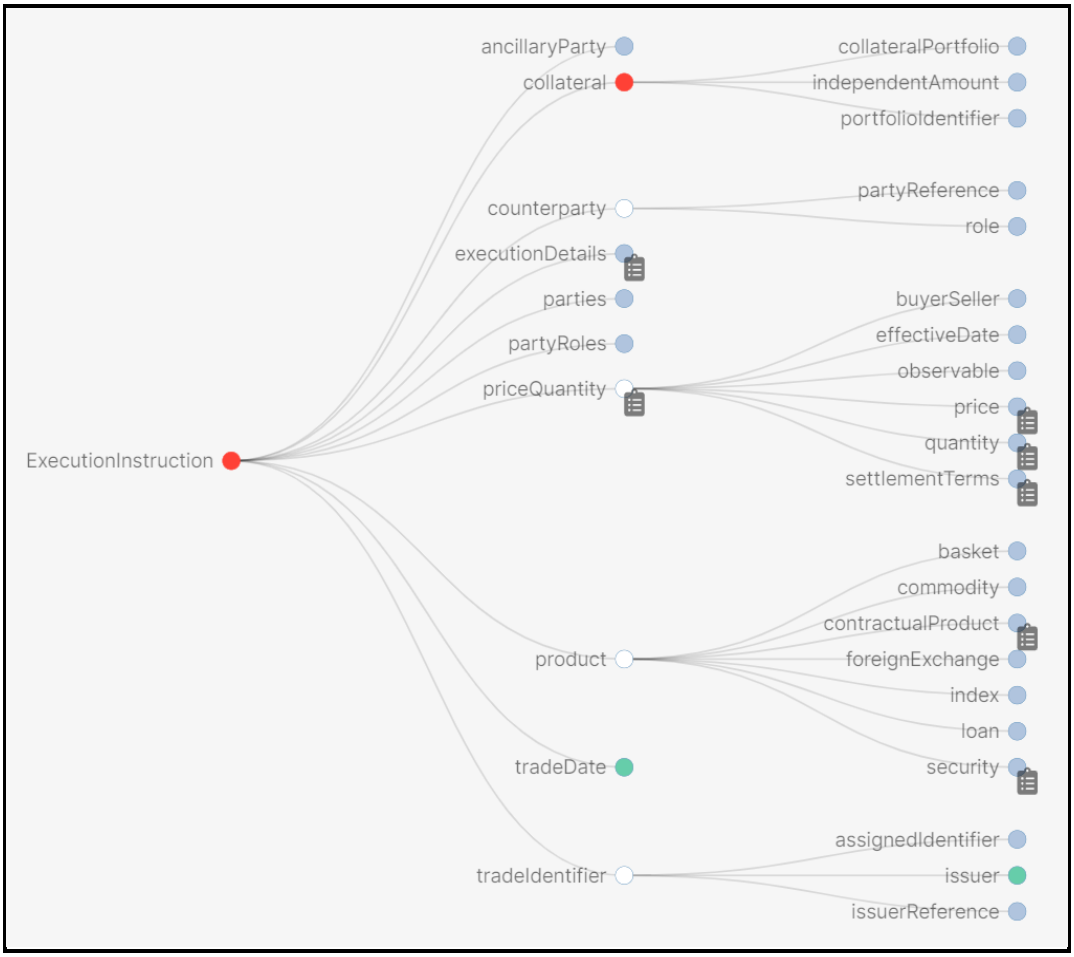

Diagram 3: The "execution" object is of type ExecutionInstruction, shown here. It is under here that the core details of the trade execution can be set.

As this is a new proposed event, there is no business event as yet, so the “businessEvent” object will not be included in this workflow step. Similarly, this is the first step in the workflow for this event, so there will be no previous workflow steps either, so the “previousWorkflowStep” object will also not be included at this point.

There are two main pieces of metadata required. The first is an identifier for the trade, which is held under the “eventIdentifier” object, in the “assignedIdentifier -> identifier” attribute.

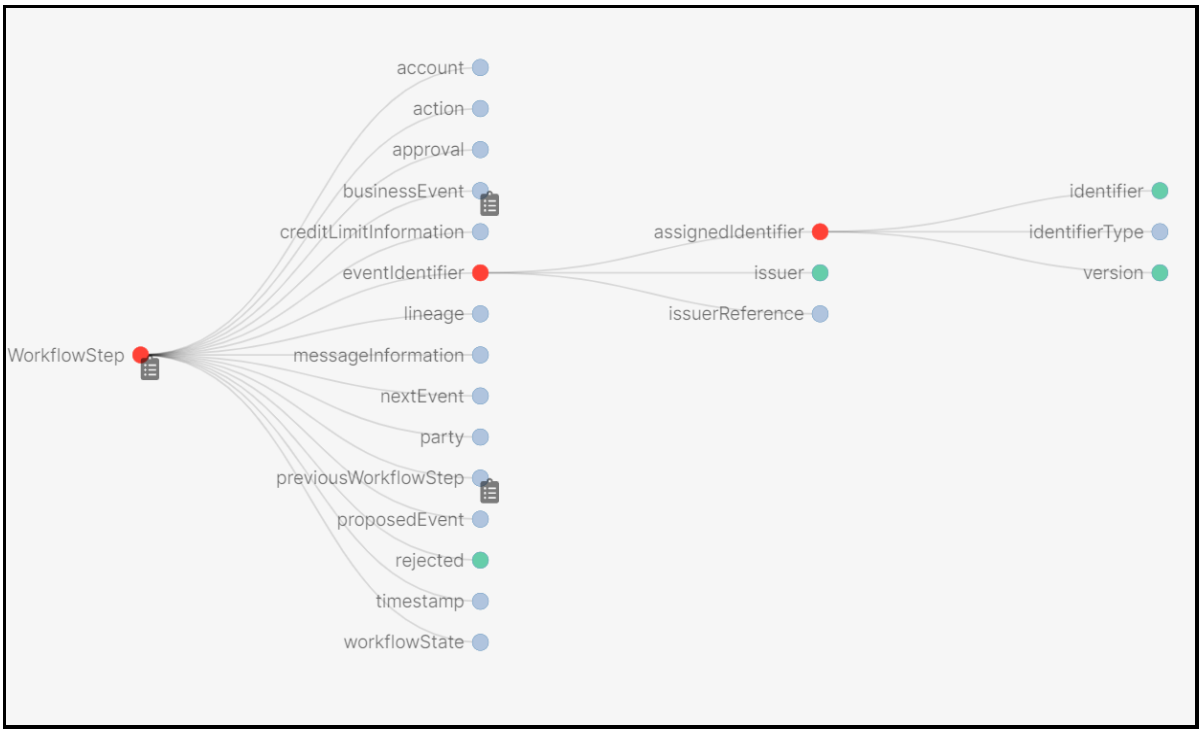

Diagram 4: The “eventIdentifier” object, which holds “assignedIdentifier” where the trade “identifier” is held.

At this stage the identifier will most likely be an internal reference, as the UTI would not necessarily be known yet. This id will not change for the duration of the negotiation process, ensuring all parties can cross reference between all workflow steps associated to this proposed trade.

For a new proposal like this the “version” attribute under “assignedIdentifier” should be set to 1. The “version” is used to count the number of proposals/counter proposals for this proposed event. Setting the version to 1 lets all parties know that this is a new proposed trade execution.

(Note: when a party wants to generate a counter proposal they will increment the “version” number – please see the following descriptions about how to model a counter proposal for more detail on the usage of the “version” attribute).

For any trade negotiation to reach a successful conclusion all parties involved must approve the details of the execution. To allow party approval statuses to be held against a workflow step the “approval” object can be used.

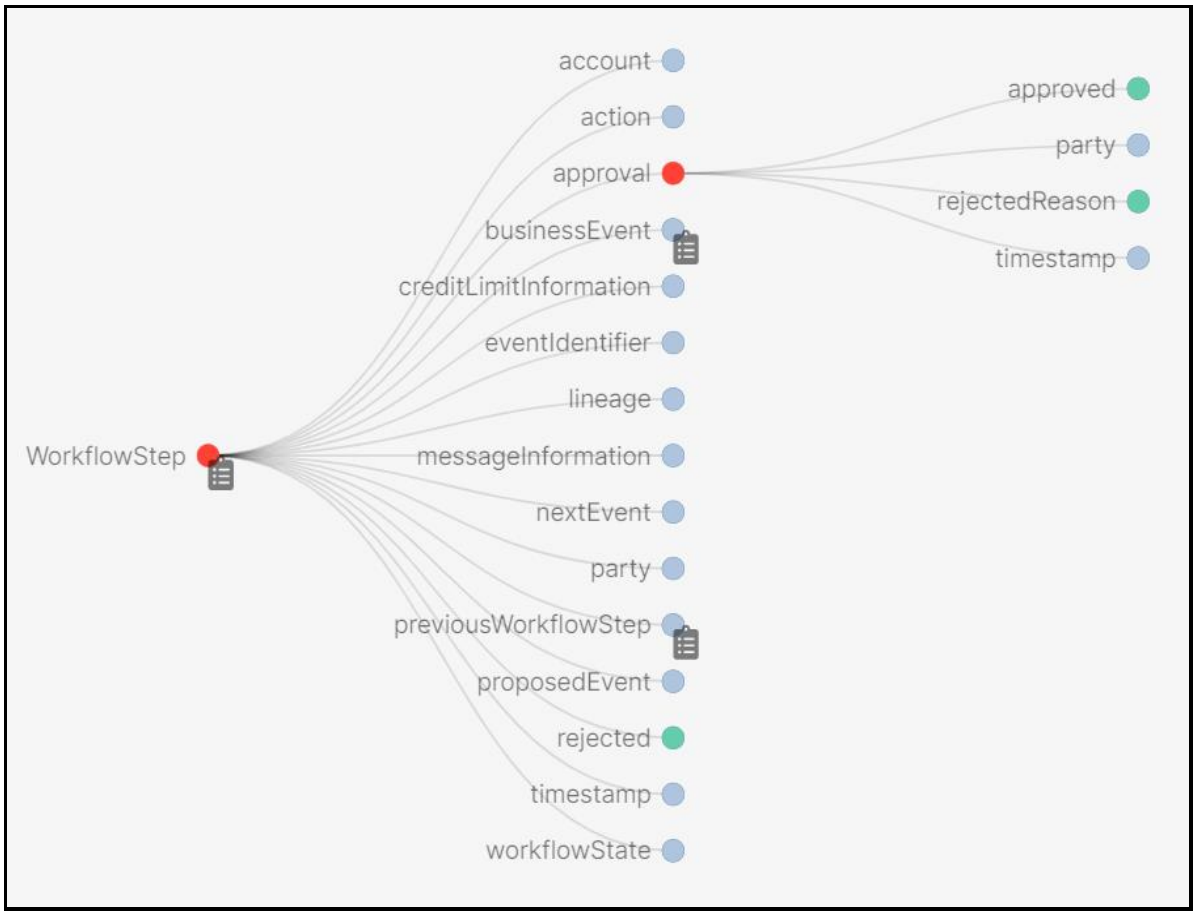

Diagram 5: The "approval" object where party approval statuses are held.

This object allows a list of parties to be included in the workflow step. The parties listed in this object are expected to approve or reject the proposed event. The "approved" attribute must be set for each of the parties in the list, and can be set to either True (i.e. approved) or False (i.e. rejected).

As this is a new proposal being instigated by the borrower, the approval status of the borrower party can be set to True, as they are proposing the execution details and so have already pre-approved them. The other party on the negotiation, the lender, will not have seen this proposal as yet, so the lender party must be included in the list with their approval status set to False.

Party	Approval
Borrower	True
Lender	False

Table 1: For a new proposal the proposing party (the borrower in this use case) should set their "approval" status to True; the other party (the lender), who has not seen the proposal yet, should have their status set to False.

A time stamp for the workflow step is required, which should be set to the current date and time.

Additional details about the transfer of the workflow step between parties can be specified in the "messageInformation" object. Our example does not rely on these attributes, but they can be useful when determining the direction and flow of messages between parties.

The "rejected" attribute should not be included in the workflow step at this point. It should only be included when a workflow step is actually being rejected, which is described in more detail in a later section.

4 How to accept a proposal

Once the lender receives a proposal they can then decide whether they accept the terms of the trade execution or not.

If the lender decides that the terms are acceptable then they will need to send a message back to the borrower confirming this. The message will be another WorkflowStep object, but this time with a "businessEvent" object in it as opposed to a "proposedEvent" object.

The lender is essentially agreeing the terms of the execution as held in the "proposedEvent" from the workflow step that they received from the borrower. As the terms are acceptable, the lender will need to send back a WorkflowStep with the same terms as the "proposedEvent" object they received. This will be in a "businessEvent" object, as both parties will have now approved the event so it can be executed.

The borrower application can then compare the terms they sent in the "proposedEvent" against those that they received back in the "businessEvent"; if they match then the borrower can confirm that the lender has agreed to the terms.

Note that the comparison of the "proposedEvent" and the "businessEvent" is entirely at the discretion of the application. Where there is a trusted relationship between a borrower and lender then it would be feasible for the borrower application to skip this validation, potentially leading to improved processing times. This could be beneficial at peak times where there is a high volume of trading.

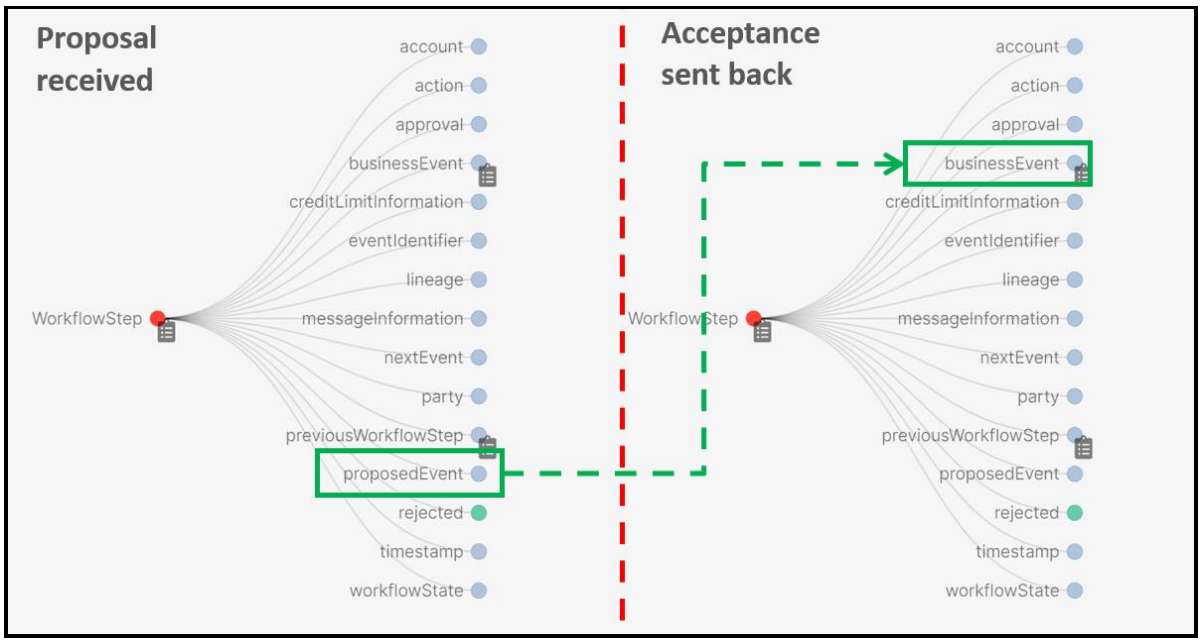

Diagram 6: If the proposal in the "proposedEvent" object is accepted then this will be used to create the "businessEvent" object that is sent back.

The new WorkflowStep must also now include the “previousWorkflowStep” object. The WorkflowStep object that the lender received holding the proposed trade execution must be copied into the “previousWorkFlowStep” object. This preserves the lineage of the negotiation and will allow both the lender and borrower applications to inspect the entire negotiation process should they need to.

Note that the details from the original “proposedEvent” must not be updated when they are put into the “previousWorkflowStep”. This is because the JSON that the CDM generates can contain metadata references. These references allow sections of the JSON to be cross referenced by other objects, reducing data duplication.

The approval status of the lender party in the new workflow step should now be updated to True. The approval status of the borrower party should already be set to True; thus the “approval” object in the new workflow step should now have both the borrower and lender approval statuses set to True.

Party	Approval
Borrower	True
Lender	True

Table 2: The lender has now approved the execution details; both parties should now have their “approval” status set to True.

There are functions available in the CDM that can be called to facilitate the creation of this “accept” workflow step. These are explained in more detail in the following sections.

5 How to counter a proposal

If a lender receives a proposal but does not find the terms acceptable, then they can offer a counter proposal. A counter would be used when the lender still wants to fulfil the request but wants to change some of the terms of the trade, for example offering a different rate.

Note that if the lender does not want to continue with the trade negotiation then they should reply to the borrower with a rejection and not a counter.

The borrower will have sent the lender a WorkflowStep with a “proposedEvent” in it holding the terms that the borrower would like. The lender will need to take that “proposedEvent” object, update the terms that they want to change in it, and then put it into a new WorkflowStep that they will return to the borrower. The borrower application can then compare the terms they sent against those that they received back, which will highlight the terms that the lender has changed.

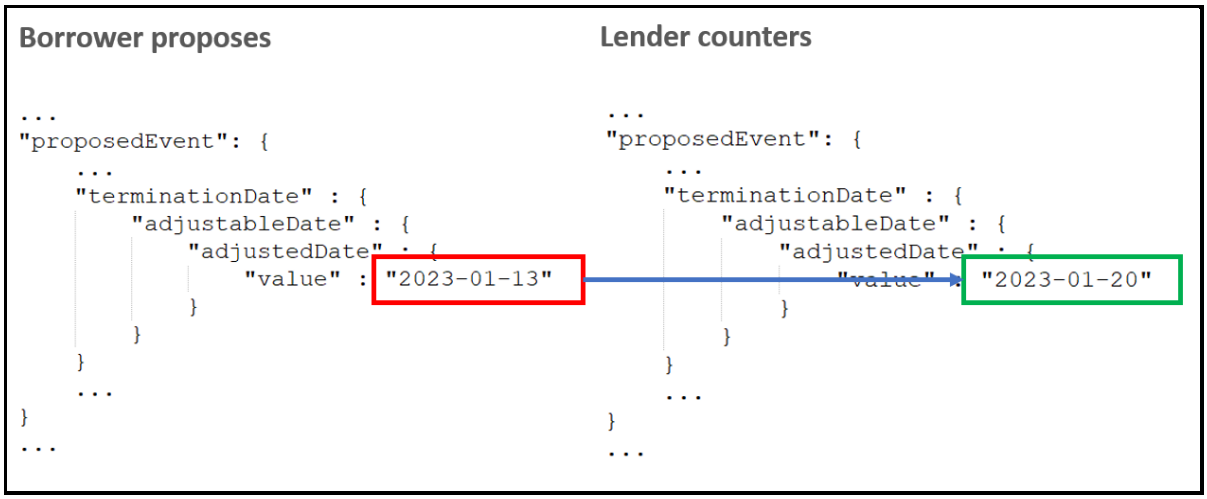

Diagram 7: A simplified example showing the borrower proposing a termination date of 13th January but the lender countering the proposal with a date of 20th January.

In order to make it clear that this is a counter the lender must also increment the "version" number of this WorkflowStep. This is held under the "eventIdentifier" object, in the "assignedIdentifier -> identifier" attribute.

For example, the borrower sends out a new proposal but the lender wants to change the rate. The lender will send back a new WorkflowStep with the same identifier but with the "version" incremented by 1:

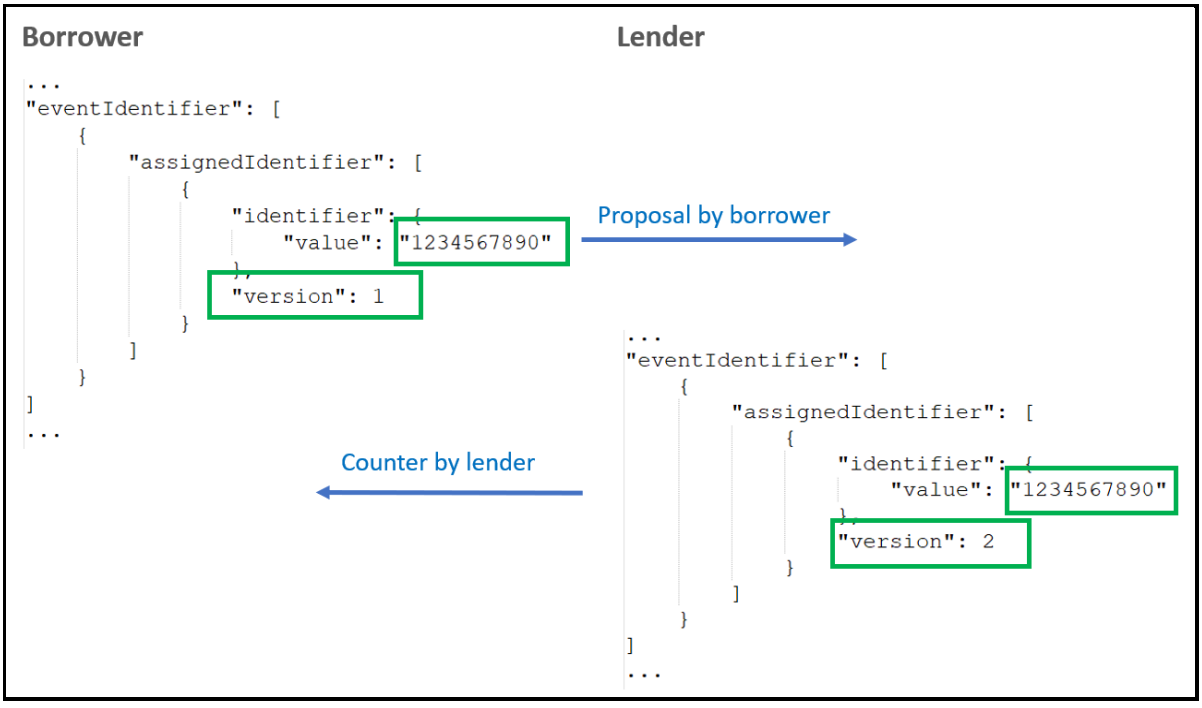

Diagram 8: For a counter proposal the trade identifier does not change but the version number is incremented.

If the borrower is not happy with the new terms in the counter proposal then they too can offer a counter proposal back to the lender. Once again, the borrower would update the details in "proposedEvent" and increment the "version" before sending the counter proposal back to the lender.

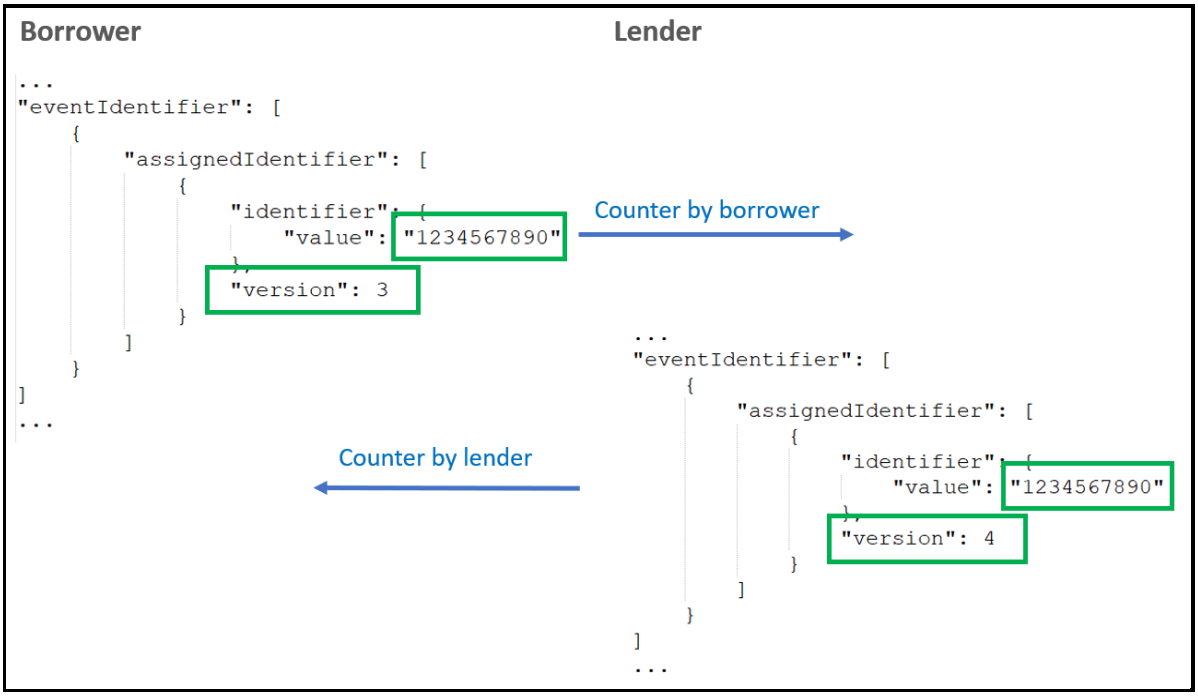

Diagram 9: Both the lender and the borrower can make counter proposals

Using this process the borrower and the lender can elicit any number of counter proposals, allowing the negotiation to continue until the terms of the trade are agreeable to both parties.

Once again the WorkflowStep will need to be copied into the "previousWorkflowStep" object in order to preserve the lineage of the negotiation.

If a counter proposal is being made then the party that is sending the counter should update their approval status to True in the "approval" object. The approval status of the other party in the negotiation must now be set to False, as the other party will not have seen these new terms.

Party	Approval
Borrower	True
Lender	False

→

Party	Approval
Borrower	False
Lender	True

Table 3: The proposal by the borrower has been countered by the lender. The "approval" status of the borrower must now be set to False, as they have not approved the counter proposal yet, and the status of the lender should be set to True.

6 How to reject a proposal

If the lender receives the execution but no longer wishes to continue with the negotiation then they can send back a rejection. This can be achieved by setting the "rejected" attribute to True.

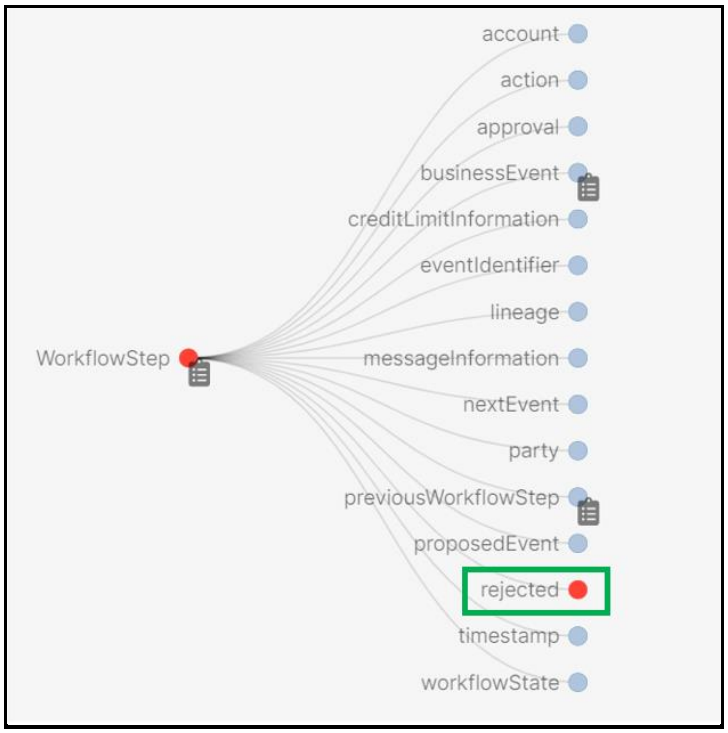

Diagram 10: The "rejected" attribute under "WorkflowStep"

The "proposedEvent" that was in the original workflow step should be left as it is. It should also be put into the "previousWorkflowStep" to preserve lineage.

The approval status for the lender party in the "approval" object should also remain as False. The approval status for the borrower party, which should currently be True, can remain as it is, as the lender is now closing the negotiation so approval is not required i.e. this is not a counter proposal that needs to be approved, this is a termination of the negotiation.

Note that the borrower can also reject a counter proposal from a lender using the same process. If the terms that the lender proposes in their counter proposal are not acceptable to the borrower, or the borrower has a different reason for not continuing the negotiation, then they too can reject the proposal.

There are functions available in the CDM that can be called to facilitate the creation of this "rejected" workflow step. These are explained in more detail in the following sections.

7 How to track the workflow steps – lineage

During the process of the negotiation each workflow step should be retained for reference and lineage purposes. This allows each party on the event to have a complete overview of the negotiation and for applications to make decisions based upon it.

The "previousWorkflowStep" object in the model is used to hold all the workflow steps for a negotiation. When a workflow step is received by any party then they will need to put that workflow step into the "previousWorkflowStep" object. This is true whether the party is accepting, countering or rejecting a workflow step they have received.

Note that on a negotiation that has several steps involved then “previousWorkflowStep” will end up with a tree of several workflow steps defined within it.

Applications should also retain the WorkflowStep objects that are being passed between the parties. Preserving the objects themselves allows the applications to check that the previousWorkflowStep objects that they receive in any new workflow step objects do indeed match the previous objects that they have received previously.

In reality the objects received from other parties on the negotiation should be correct, and thus should not need to be stored by the applications. However, there is an element of trust involved in the sending of data between parties, and having the previous objects stored within the applications will support additional data validation and audit trails if required.

8 Who creates the actual Business Event?

In our negotiation scenarios it should always be the lender who creates the business event. This is because the lender needs to have the “last look” on the trade as they are providing the shares as requested by the borrower.

A successful trade execution negotiation process will have been achieved once the lender party has received an object that accepts the terms of the execution. At this point the lender application can generate a “businessEvent” object rather than a “proposedEvent” and send this out to the other parties on the negotiation.

The “businessEvent” marks the end of the negotiation and no further changes can be made to the trade execution. The trade will be executed by the lender and the borrower and further lifecycle processing on the trade can continue e.g. settlement.

9 Scenario 1 – Propose – Accept

In general trading this should be the most common scenario, as we are expecting that the borrower and lender will have already completed their preliminary trade checks before entering into the negotiation process. The overall process can be split into 3 steps, as expressed in the following diagram:

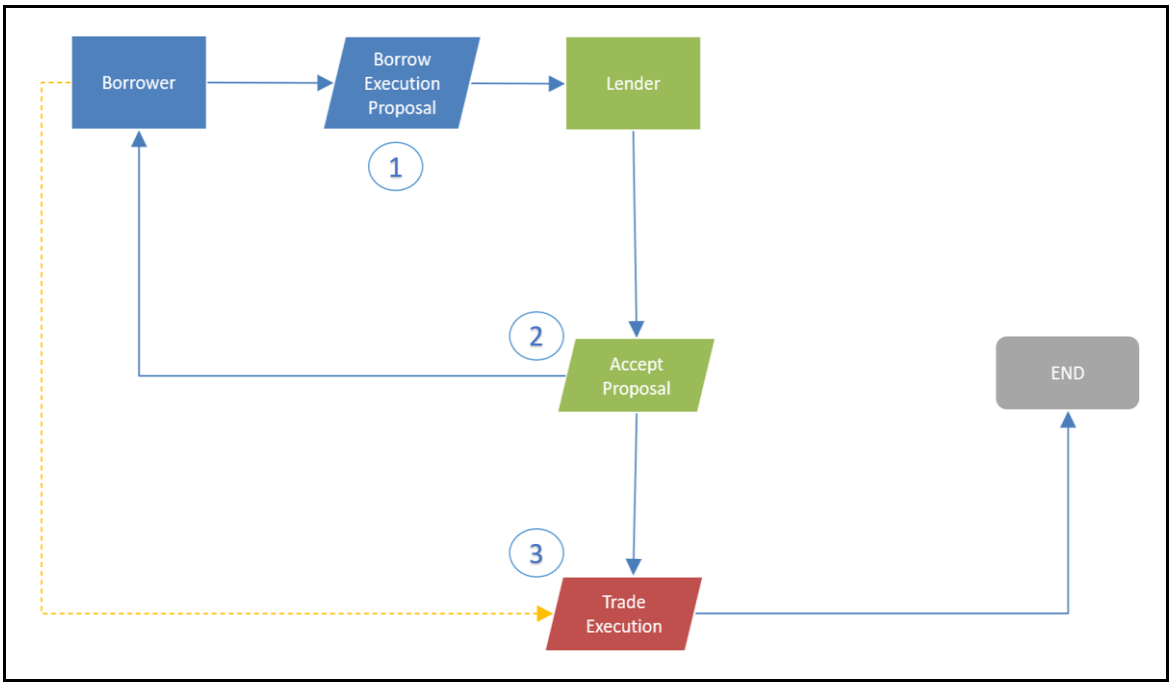

Diagram 11: Overview of the Propose-Accept workflow

9.1 Step 1 – Borrower creates the initial proposal

To start the negotiation process the borrower will need to create a new "WorkflowStep" object with the details of the trade execution in the "proposedEvent" object within it. This can be done using the "Create_ProposedWorkflowStep" function in the CDM.

```
func Create_ProposedWorkflowStep: <"Represents the proposal to create a business event that results in a workflow step
    containing an instruction, message details, identifiers, event timestamps, parties and accounts. The optional
    previous workflow step input provides workflow lineage to where there has been a correction or cancellation to the
    proposed step. The action is constrained so that when a previous workflow step is specified, the valid actions are
    as follows; New -> Correct and Correct -> Cancel. When a previous workflow is not specified, the action must be
    New.">
    [creation WorkflowStep]
    inputs:
        messageInformation MessageInformation (0..1) <"Contains all information pertaining the messaging header">
        timestamp EventTimestamp (1..*) <"The dateTime and qualifier associated with this event.">
        eventIdentifier Identifier (1..*) <"The identifiers that uniquely identify this lifecycle event.">
        party Party (0..*) <"The specification of the parties involved in the WorkflowStep.">
        account Account (0..*) <"Optional account information that could be associated to the event.">
        previousWorkflowStep WorkflowStep (0..1) <"Optional previous WorkflowStep that provides lineage to
            WorkflowStep that precedes it.">
        action ActionEnum (1..1) <"Specifies whether the event is new or a correction. The action cannot be a
            cancellation or new if the previous step is also new.">
        proposedEvent EventInstruction (1..1) <"The proposed instruction for the step to initiate a workflow e.g.
            Clearing Instruction or Allocation Instruction">
        approval WorkflowStepApproval (0..1) <"The approval status of all parties on the event.">

    output:
        proposedWorkflowStep WorkflowStep (1..1) <"Proposed WorkflowStep populated with the proposed instruction">
```

Diagram 12: The inputs and output of the "Create_ProposedWorkflowStep" function as expressed in the Rosetta DSL

The function expects several items as input, which will be used to generate the output, which will be a new "WorkflowStep" object. For this initial proposal the parameters to the function should be populated as follows:

messageInformation
Although this optional object is not specifically used in our scenarios it can hold useful metadata describing the message itself. This data can be useful for applications that receive these WorkflowStep objects, as they can be used to confirm who sent the object and who the intended recipient was.

timestamp
This must be included in the object and should be set to the date/time that the workflow step was generated. Example JSON for this object would be as follows:

```
"timestamp": [
      {
            "dateTime": "2023-01-11T14:18:49.1016752+01:00",
            "qualification": "EVENT_SENT_DATE_TIME"
      }
]
```

Diagram 13: Example of a "timestamp" object

Note that this object is a list so that many dates and times can be included. This allows multiple different date structures or timezones to be included. For more details of the options available here please refer to the "EventTimestamp" type in the CDM.

eventIdentifier
This mandatory object is used to store a unique reference identifier for the trade execution that must not change throughout the negotiation process. It also holds the "version" number that is used when determining whether this is a new proposal or a counter proposal.

The unique identifier for this trade negotiation is put into the "eventIdentifier -> assignedIdentifier -> identifier" attribute. The "identifierType" attribute associated to the "identifier" allows the id to be defined as a UTI; however, at this stage of the negotiation this will not be a UTI, as the trade has not been executed yet, so the "identifierType" attribute should not be included.

For the initial trade proposal the "eventIdentifier -> assignedIdentifier -> version" number should be set to "1". The version number should only be incremented by 1 when the workflow step holds a counter proposal. For more details on updating the version number please refer to the "Propose-Counter-Accept" scenario later in this document.

An example of a valid "eventIdentifier" object for a new proposal would be as follows:

```
"eventIdentifier": [
      {
            "assignedIdentifier": [
                  {
                        "identifier": {
                              "value": "1234567890"
                        },
                        "version": 1
                  }
            ]
      }
]
```

Diagram 14: Example of an "eventIdentifier" object

The id given to this negotiation workflow is thus "1234567890". This will not change throughout the negotiation process of this specific trade execution.

party
The "party" object is optional at this level but it is recommended that the parties on the negotiation are defined here. These can then be referenced through metadata links further down in the object tree.

The "party" object can contain multiple parties. In our scenarios we have two parties, the borrower and the lender, so there must be two parties described in the "party" list. An example of a "party" object containing two parties is as follows:

```
"parties": [
      {
            "partyId": [
                  {
                        "identifier": {
                              "value": "549300TVHZO5P05IDJ86"
                        },
                        "identifierType": "LEI"
                  }
            ]
      },
      {
            "partyId": [
                  {
                        "identifier": {
                              "value": "549300FS4OSSFUCNC234"
                        },
                        "identifierType": "LEI"
                  }
            ]
      }
]
```

Diagram 15: Example of a "parties" object with two parties described in it

account
The account is another optional object that could be specified here or elsewhere in the object tree. In our scenarios the account details are specific to an individual party, not the workflow step itself, so it is recommended that this object is left empty here.

previousWorkflowStep
This object must be empty at this point as this is the first step in the negotiation process i.e. there have been no previous workflow steps in this negotiation.

action
This attribute is used to describe the status of the event within the workflow step. We are always describing a new event in each workflow step so this attribute should always be set to "NEW".

proposedEvent
This is the core of the initial proposal as it holds the details of the trade execution. Within the "proposedEvent" object there are currently four items that can be defined, the most important of which is the "proposedEvent -> instruction".

The borrower is proposing a new trade execution so the "proposedEvent -> instruction -> primitiveInstruction -> execution" object should be populated. This holds details of the collateral, parties, economics and payout terms of the trade. These will differ on a trade-by-trade basis so the object will not be expanded upon further here.

For more information on how to model a trade execution please refer to the CDM documentation and the notations in the model itself. Example JSON can also be found in the Visualisation tool in the Rosetta platform provided by REGnosys.

The "proposedEvent -> intent" should not be specified; we have used the "execution" instruction which will flag this as a trade execution so an intent is not required.

The "proposedEvent -> eventDate" and "proposedEvent -> effectiveDate" are optional but it is recommended that they are both populated. For our purposes the event date and the effective date will be the same and should be set to the current date.

A simplified example of a "proposedEvent" object could be as follows:

```
"proposedEvent": {
      "effectiveDate": "2023-01-11",
      "eventDate": "2023-01-11",
      "instruction": [
            {
                  "primitiveInstruction": {
                        "execution": {
                              "counterparty": [ ... ],
                              "executionDetails": { ... },
                              "parties": [ ... ],
                              "priceQuantity": [ ... ],
                              "product": {
                                    "contractualProduct": {
                                          "economicTerms": { ... },
                              },
                              "tradeDate": { ... },
                              "tradeIdentifier": [ ... ]
                        }
                  }
            }
      ]
}
```

Diagram 16: A simplified example of a "proposedEvent" object showing the top level attributes that might be specified within it.

Note that the contents of the "execution" object have been simplified for brevity.

approval
The "approval" object is optional as not all workflows will require a step to be approved. Approval is an important part of our negotiation though so this object must be present.

The "approval" object holds a list of the parties whose approval of a workflow step is required. These parties must be defined in the "party" object for the workflow step as they must be taking part in the negotiation. Associated to each party in the list is an approval status.

When creating the first step in a new negotiation, the "approval" object must be populated with entries for each of the parties in the negotiation. In our scenarios we always have two parties, a borrower and lender, so the borrower will need to create the initial "approval" object with two items in it, one for themselves and one for the lender.

As the borrower is initiating this proposal they can pre-approve the trade details and set the approval status for their party to True. However, they must set the approval status to False for the lender, as the lender has not seen the terms of the execution yet.

The JSON below shows an example of the "approval" object for a new trade execution between a borrower and a lender:

```
"approval": [
      {
            "approved": true,
            "party": {
                  "partyId": [
                        {
                              "identifier": {
                                    "value": "549300TVHZO5P05IDJ86"
                              },
                              "identifierType": "LEI"
                        }
                  ]
            },
            "timestamp": [
                  {
                        "dateTime": "2023-01-11T14:18:49.1016752+01:00",
                        "qualification": "EVENT_SENT_DATE_TIME"
                  }
            ]
      } , {
            "approved": false,
            "party": {
                  "partyId": [
                        {
                              "identifier": {
                                    "value": "549300FS4OSSFUCNC234"
                              },
                              "identifierType": "LEI"
                        }
                  ]
            },
```

```
            "timestamp": [
                  {
                        "dateTime": "2023-01-11T14:18:49.1016752+01:00",
                        "qualification": "EVENT_SENT_DATE_TIME"
                  }
            ]
      }
]
```


Diagram 17: An example of the "approval" object showing the approval status of the two parties on the negotiation.

The borrower – LEI "549300TVHZO5P05IDJ86" – has their approval status set to "true"; the lender – LEI "549300FS4OSSFUCNC234" – has their approval status set to "false".

Once the required JSON has been prepared then the "Create_ProposedWorkflowStep" function can then be called. The result of the function call should be a new "WorkflowStep" with all the details specified within it. This object is the initial trade execution proposal that the borrower would then send on to the lender.

If you are using the Rosetta platform you can use the "Functions" bottom menu option to select the "Create_ProposedWorkflowStep" function and upload the JSON against it for testing purposes. You can also use the "API Export" bottom menu option and use a tool like curl to perform the same task through a desktop application.

The CDM is also available as a set of Java jar files and these can be used directly to call the functions.

Further details on how to setup the CDM for test or implementation purposes is beyond the scope of this document.

9.2 Step 2 – Lender accepts the proposal

The lender will receive the "WorkflowStep" object created by the borrower. This will hold the details of the trade execution in the "proposedEvent" object within it.

The lender application will need to check the terms described in the "proposedEvent", and, if happy with them, they can commit the trade on their system. They can commit the trade as the borrower has already pre-approved the execution details by setting their approval status in the "approval" object to "true".

The lender application will need to notify the borrower that they have accepted the proposal. They can do this by generating a new workflow step that holds a "businessEvent" object. This can be done by using the "Create_AcceptedWorkflowStepFromInstruction" function in the CDM.

```
func Create_AcceptedWorkflowStepFromInstruction: <"Represents the acceptance of a proposed instruction that results
    in a workflow step containing a business event, message details, identifiers, event timestamps, parties and
    accounts. The previous workflow step input must exist to provide workflow lineage. The instruction from the
    previous workflow step should be used with a [creation BusinessEvent] function to create the input business
    event passed into this function e.g. PartyChangeInstruction from the previous step is used with
    Create_PartyChange to produce the business event which should used as an input to this step.">
    [creation WorkflowStep]
    inputs:
        proposedWorkflowStep WorkflowStep (1..1) <"WorkflowStep as instruction.">
    output:
        acceptedWorkflowStep WorkflowStep (1..1) <"Accepted WorkflowStep populated with the business event and
            associated details about the message, identifiers, event timestamps, parties and accounts involved in
            the step.">
```

Diagram 18: The inputs and output of the "Create_AcceptedWorkflowStepFromInstruction" function as expressed in the Rosetta DSL

Note that there is another function in the CDM called "Create_AcceptedWorkflowStep" which has different input requirements to "Create_AcceptedWorkflowStepFromInstruction". The advantage of using "Create_AcceptedWorkflowStepFromInstruction" is that it requires only one input parameter, this being the "WorkflowStep" object that the lender wants to accept. This makes the acceptance of a workflow step a lot simpler from the implementation perspective.

The "Create_AcceptedWorkflowStepFromInstruction" performs several tasks for us.

Firstly, it will create a new "WorkflowStep" object with the same "action", "messageInformation", "timestamp" and "eventIdentifier" objects as the original workflow step that the lender is accepting.

It will also put the original "WorkflowStep" object into the new workflow step's "previousWorkflowStep" object, preserving the lineage of the negotiation.

Finally, and most importantly, the function will create a "businessEvent" object (as opposed to a "proposedEvent" object) using the details passed in the original "WorkflowStep" object. This is the actual business event and marks the end of the negotiation from the lender's perspective.

```
set acceptedWorkflowStep -> businessEvent: <"Assign the business event corresponding to the workflow step.">
    Create_BusinessEvent(
        proposedWorkflowStep -> proposedEvent -> instruction,
        proposedWorkflowStep -> proposedEvent -> intent,
        proposedWorkflowStep -> proposedEvent -> eventDate,
        proposedWorkflowStep -> proposedEvent -> effectiveDate )
```

Diagram 19: Snippet from the "Create_AcceptedWorkflowStepFromInstruction" where the "Create_BusinessEvent" function is called to create a new businessEvent object based upon the details pass in the proposedEvent object.

The "WorkflowStep" that has thus been created can now be sent back to the borrower as confirmation that the trade has been accepted.

9.3 Step 3 – Borrower receives the business event

At this stage the borrower will still be waiting for a response back from the lender, confirming whether they have accepted the terms of the proposal or not. The "WorkflowStep" that they receive from the lender holds a "businessEvent" object, so they know that the terms were acceptable and the trade has been executed by the lender.

The borrower can now commit the trade on their own system using the details from the “businessEvent” object in the workflow step that they received.

10 Scenario 2 – Propose – Reject

Probably the simplest of the negotiation scenarios, this would be where the lender now does not want to continue with the trade at all. This could be where they no longer have the shares for example. The following diagram illustrates the workflow:

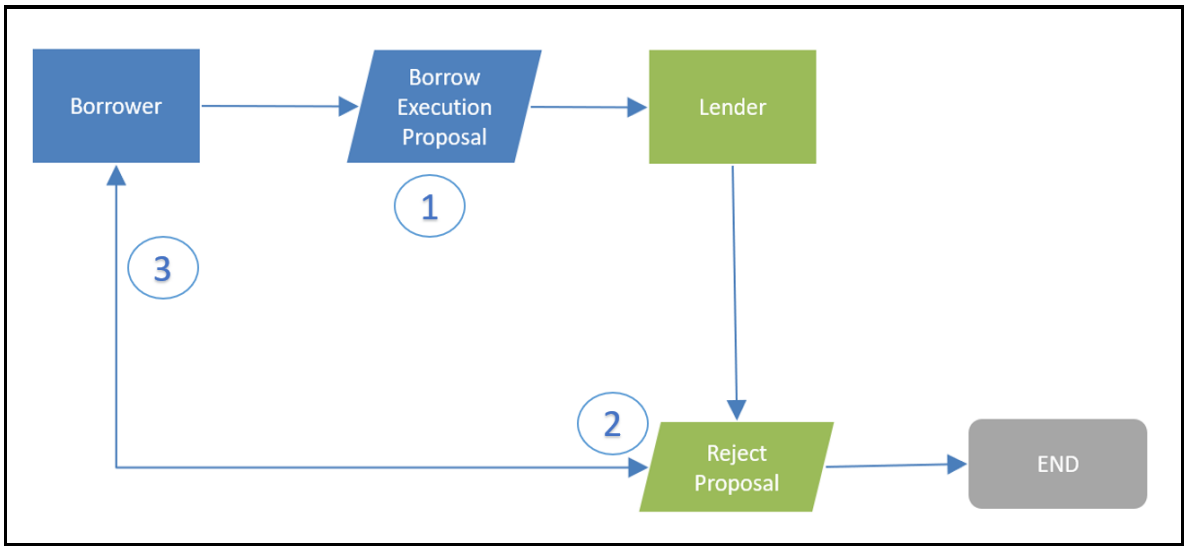

Diagram 20: Overview of the Propose-Reject workflow

10.1 Step 1 – Borrower creates the initial proposal

To start the negotiation process the borrower will need to create a new "WorkflowStep" object with the details of the trade execution in the "proposedEvent" object within it. This can be done using the "Create_ProposedWorkflowStep" function in the CDM.

The details on how to do this are the same as can be found in the section of the same name under the "Propose – Accept" scenario.

10.2 Step 2 – Lender rejects the proposal

The lender will receive the "WorkflowStep" object created by the borrower. This will hold the details of the trade execution in the "proposedEvent" object within it.

The lender application will need to check the terms described in the "proposedEvent". If they are not happy with them and do not want to continue with the trade, or no longer want to continue with the trade for any other reason, then the lender will need to reject the proposal. In this situation the lender will not commit the trade into their application.

To reject the event the lender application will need to generate a rejected workflow step and pass this back to the borrower. This can be done by using the "Create_RejectedWorkflowStep" function in the CDM.

```
func Create_RejectedWorkflowStep: <"Represents the rejection of a proposed instruction that results in a workflow
    step containing the rejection flag, message details, identifiers, event timestamps, parties and accounts
    involved in the step. The previous workflow step input must exist to provide workflow lineage. This function
    will be further developed to provide the reasons for rejection.">
    [creation WorkflowStep]
    inputs:
        messageInformation MessageInformation (0..1) <"Contains all information pertaining the messaging header">
        timestamp EventTimestamp (1..*) <"The dateTime and qualifier associated with this event.">
        eventIdentifier Identifier (1..*) <"The identifiers that uniquely identify this lifecycle event.">
        proposedWorkflowStep WorkflowStep (1..1) <"Required previous WorkflowStep that provides lineage to
            WorkflowStep that precedes it.">

    output:
        rejectedWorkflowStep WorkflowStep (1..1) <"Rejected WorkflowStep with lineage to the proposed step that
            preceded it.">
```

Diagram 21: The inputs and output of the "Create_RejectedWorkflowStep" function as expressed in the Rosetta DSL

The "Create_RejectedWorkflowStep" function expects 4 pieces of information, all of which should be taken directly from the "WorkflowStep" object that the lender is rejecting. These are the "messageInformation" (if populated), "timestamp", "eventIdentifier" and the entire "WorkflowStep" object that is being rejected.

The function will create a new "WorkflowStep" object, setting the "WorkflowStep -> rejected" attribute to "true". This will tell the borrower that the lender no longer wishes to continue the negotiation process for this specific event.

The "WorkflowStep" that has thus been created can now be sent back to the borrower as notice that the trade has been rejected.

10.3 Step 3 – Borrower receives the rejected proposal

When the borrower receives the new workflow step back from the lender they will see that the "rejected" attribute is set to "true". The borrower will know from this setting that the lender does not wish to continue with the negotiation of this event.

11 Scenario 3 – Propose – Counter – Accept

A common situation is where the borrower and the lender will agree the details of the trade but certain terms (e.g. the interest rate) may need to be further negotiated. This would be where a counter proposal by the lender would be made, and it would then be up to the borrower to accept the new rate. The following workflow diagram shows how this could work:

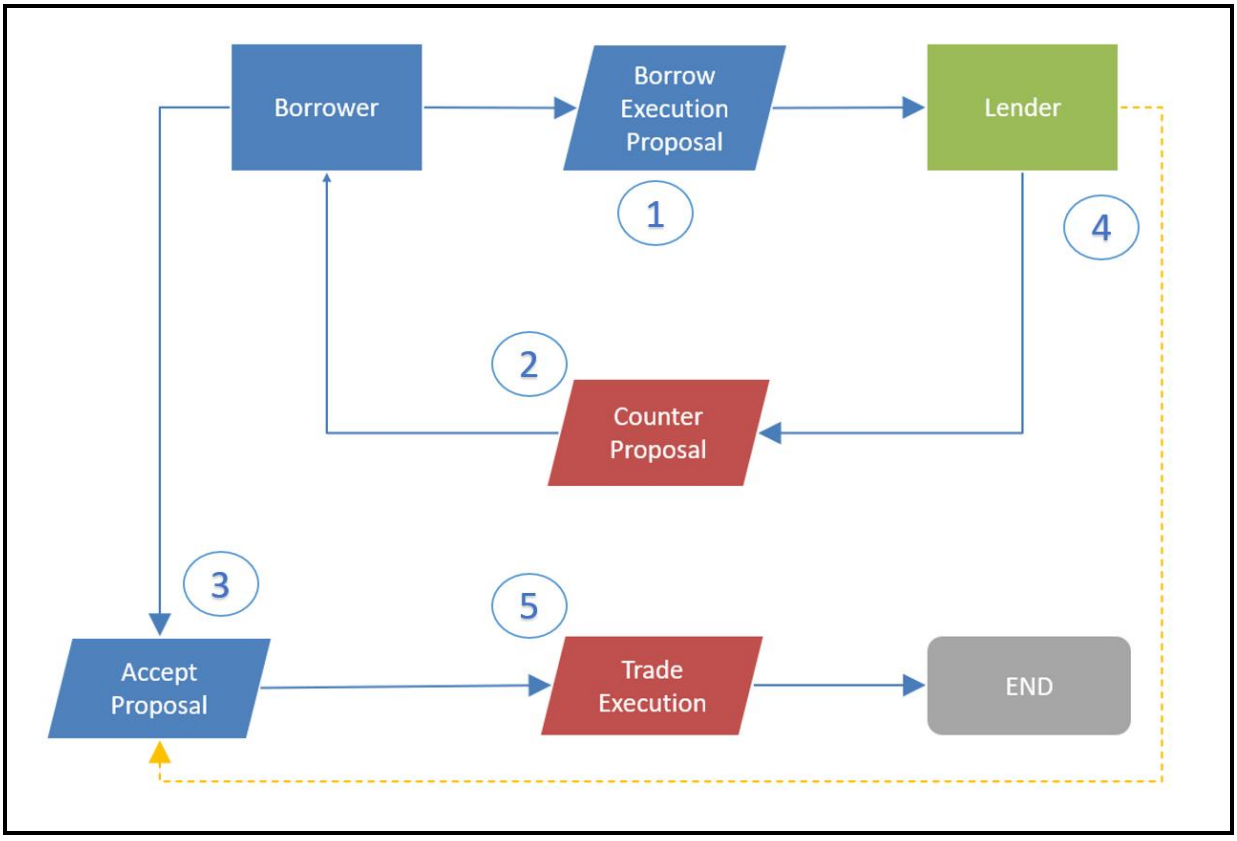

Diagram 22: Overview of the Propose-Counter-Accept workflow

11.1 Step 1 – Borrower creates the initial proposal

To start the negotiation process the borrower will need to create a new "WorkflowStep" object with the details of the trade execution in the "proposedEvent" object within it. This can be done using the "Create_ProposedWorkflowStep" function in the CDM.

The details on how to do this are the same as can be found in the section of the same name under the "Propose – Accept" scenario.

11.2 Step 2 – Lender offers a counter proposal

The lender will receive the "WorkflowStep" object created by the borrower. This will hold the details of the trade execution in the "proposedEvent" object within it.

The lender application will need to check the terms described in the "proposedEvent". In this scenario the lender wants to negotiate some of the terms further e.g. use a more favourable rate. To do this the lender will need to create a counter proposal and pass this back to the borrower for them to check. This will be a new workflow step that holds the terms from the

original proposal that the lender is happy with, along with the updated (or new) terms that they would like to change.

To create a counter proposal the lender application can use the same function that the initial proposal was generated with, this being the “Create_ProposedWorkflow” function. A few of the details passed in the input parameters will need to change for a counter proposal, which will be explained here.

In general the “messageInformation” and “timestamp” should be updated to reflect this new workflow step. The “party”, “account” and “action” objects should remain the same. The main objects that the lender would need to change are “eventIdentifier”, “previousWorkflowStep”, “proposedEvent” and “approval”.

The new workflow step that the lender is building is for the same trade execution, just with updated terms. Thus the “eventidentifier -> assignedIdentifier -> identifier” attribute should not be updated. As this is a counter proposal though, the “eventidentifier -> assignedIdentifier -> version” number should be incremented by 1.

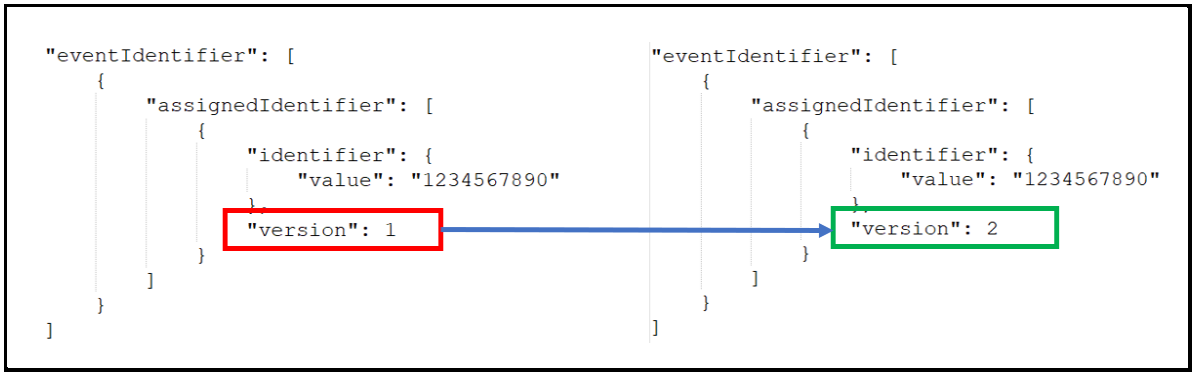

Diagram 23: When countering a proposal the “version” attribute has to be incremented.

In a negotiation where there are multiple counter proposals the “version” could be incremented many times. In all cases the “identifier” must remain the same though so that both parties applications can cross reference each workflow step to a specific negotiation.

Unlike the initial proposal, a counter proposal will always have a previous workflow step. Thus the “WorkflowStep” object that the lender is building a counter proposal for must be put into the “previousWorkflowStep” of the new workflow step.

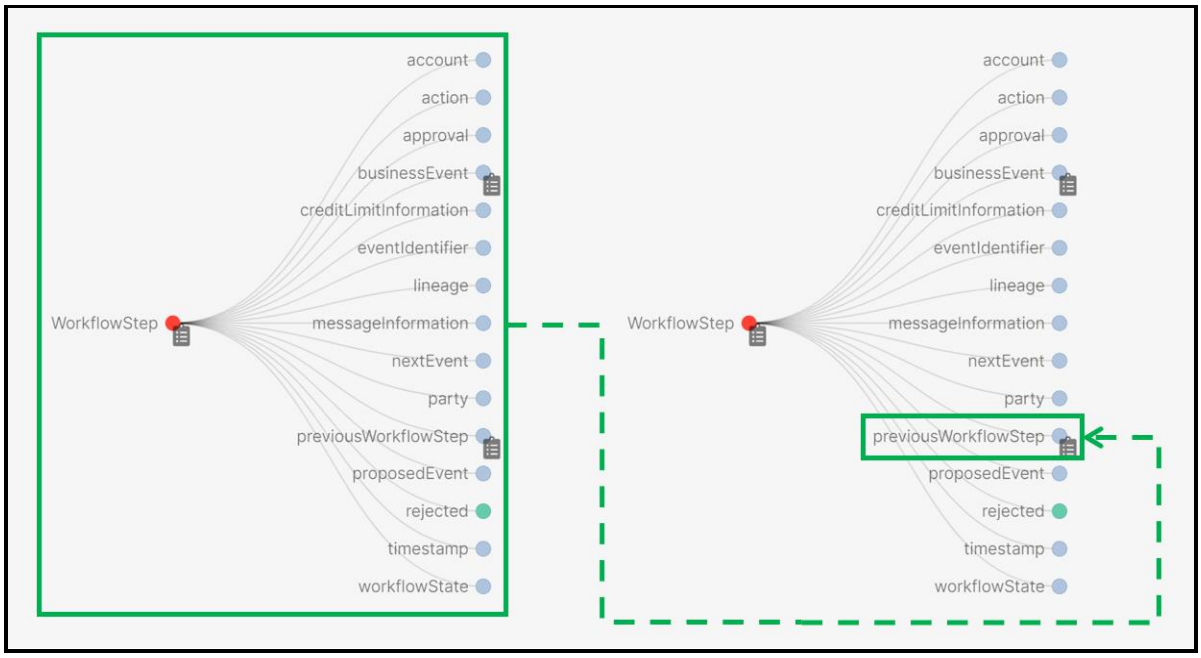

Diagram 24: The original "WorkflowStep" that is being countered by the lender must be put into the "previousWorkflowStep" of the new "WorkflowStep" that the lender will send back to the borrower.

The "proposedEvent" is where the updated terms should be placed. The easiest way to achieve this is for the lender application to use the "proposedEvent" object from the previous workflow step that they are countering and update/add the terms that they wish to change within it.

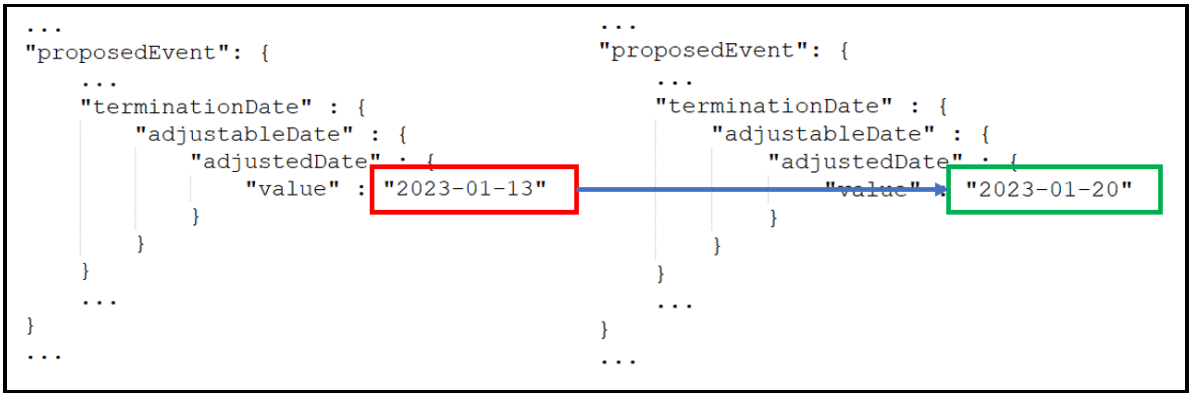

Diagram 25: The "proposedEvent" object in the new "WorkflowStep" should hold the updated terms. In the example above the borrower proposed a termination date of the 13th but the lender is countering with a date of the 20th.

Finally the "approval" object needs to be updated. When the borrower sent the initial proposal through they created the "approval" list with items in it for themselves and the lender, setting their own approval status to "true" but the lender's status to "false".

For the counter proposal the reverse is now true i.e. the lender can now pre-approve the terms of the execution, but the approval status of the borrower must now be reset to "false", as the terms have changed and now need to be approved by them again.

Diagram 26: The borrower (LEI "549300TVHZO5P05IDJ86") sent the initial "approval" object through with their status set to "true" and the lender's status (LEI "549300FS4OSSFUCNC234") set to "false". In the counter proposal the lender must set their approval status to "true" and the borrower's approval status to "false".

Once the required JSON has been prepared then the "Create_ProposedWorkflowStep" function can be called. The result of the function call should be a new "WorkflowStep" with the new terms of the trade specified within it. This can then be sent back to the borrower.

11.3 Step 3 – Borrower accepts the counter proposal

When the borrower receives the new workflow step back from the lender they will see that there is no "businessEvent" in it. From this they can deduce that the terms of the trade were not accepted by the lender.

The borrower application will also see that the "rejected" attribute has not been set, so they will know that the lender still wishes to continue with the trade negotiation.

The borrower application can further tell that this is a counter proposal by confirming that the "identifier" is one that they recognise, and that the "version" number has been incremented.

The next step is for the borrower application to work out the terms that the lender has changed. This can be done by comparing the details in "proposedEvent" and "previousWorkflowStep -> proposedEvent" in the "WorkflowStep" object they received from the lender (the application can also check the "proposedEvent" against the details that it holds internally for the original workflow step that it sent out to the lender).

Diagram 27: The borrower can locate the updated terms in the counter proposal by comparing the "proposedEvent" and "previousWorkflowStep -> proposedEvent" objects in the workflow step that the lender has sent through.

The borrower can then decide whether they are happy with the new terms in the counter proposal. In this scenario the borrower is happy with the new terms suggested by the lender and so will accept them.

To let the lender know that they are happy with the new terms, the borrower application must send a message back to the lender informing them. The way that this should be done is by generating another workflow step with the exact same details in the “proposedEvent” object as the lender sent through. The “approval” status for the borrower should also now be set to “true”.

The borrower application can use the “Create_ProposedWorkflowStep” function to create the new workflow step, passing through the exact same details as on the counter proposal they received from the lender. The one object that would need to be updated in the new workflow step is the “previousWorkflowStep” which should be updated to hold the “WorkflowStep” that held the lender counter proposal.

As a side note, if the borrower did not agree the new terms proposed by the lender, then they can generate a counter proposal themselves and send this back to the lender. To do this they would perform the same as Step 2 in this process again i.e. generate a new “WorkflowStep” with a “proposedEvent” object holding the updated terms in it, increment the “version” number and update the “approval” object accordingly.

11.4 Step 4 – Lender accepts the proposal

The lender application will receive the new “WorkflowStep” object from the borrower. The application will recognise the “identifier” and so will know it is a new step in an existing negotiation.

In addition the “version” number will be the same as the lender set it to when they sent out the counter proposal, allowing them to deduce that this is not a counter proposal, but will either be a rejection or an acceptance of their counter proposal.

As the “rejected” attribute will not be set, the lender can tell that this is an acceptance of the terms that they proposed in their counter proposal. The lender application can further confirm this by checking that the contents of “proposedEvent” and “previousWorkflowStep -> proposedEvent” in the “WorkflowStep” object are identical (the application can also check the “proposedEvent” against the details that it holds internally for the original workflow step that it sent out to the borrower).

Once the lender application is satisfied that the borrower has accepted the new terms they can continue to commit the trade, and notify the borrower. They can do this by generating a new workflow step that holds a “businessEvent” object. This can be done by using the “Create_AcceptedWorkflowStepFromInstruction” function in the CDM.

The process to generate the “businessEvent” is the same as described in the Step 2 from the “Propose – Accept” scenario described previously.

11.5 Step 5 – Borrower receives the business event

Having sent back their acceptance of the counter proposal made by the lender, the borrower will be waiting for confirmation back from the lender that the trade has been executed. The “WorkflowStep” that they now receive from the lender will hold a “businessEvent” object, so the borrower will know that the trade has been executed by the lender.

The borrower can now commit the trade on their own system using the details from the “businessEvent” object in the workflow step that they received.

12 Scenario 4 – Propose – Counter – Reject

Even after the borrower and lender have agreed to a trade execution, and have started deciding the specific terms of the trade, it is still possible that one party may not want to continue with the trade. In this scenario we consider the situation where a borrower no longer wants/needs the shares requested and thus wants to stop the negotiation. The following diagram represents the workflow:

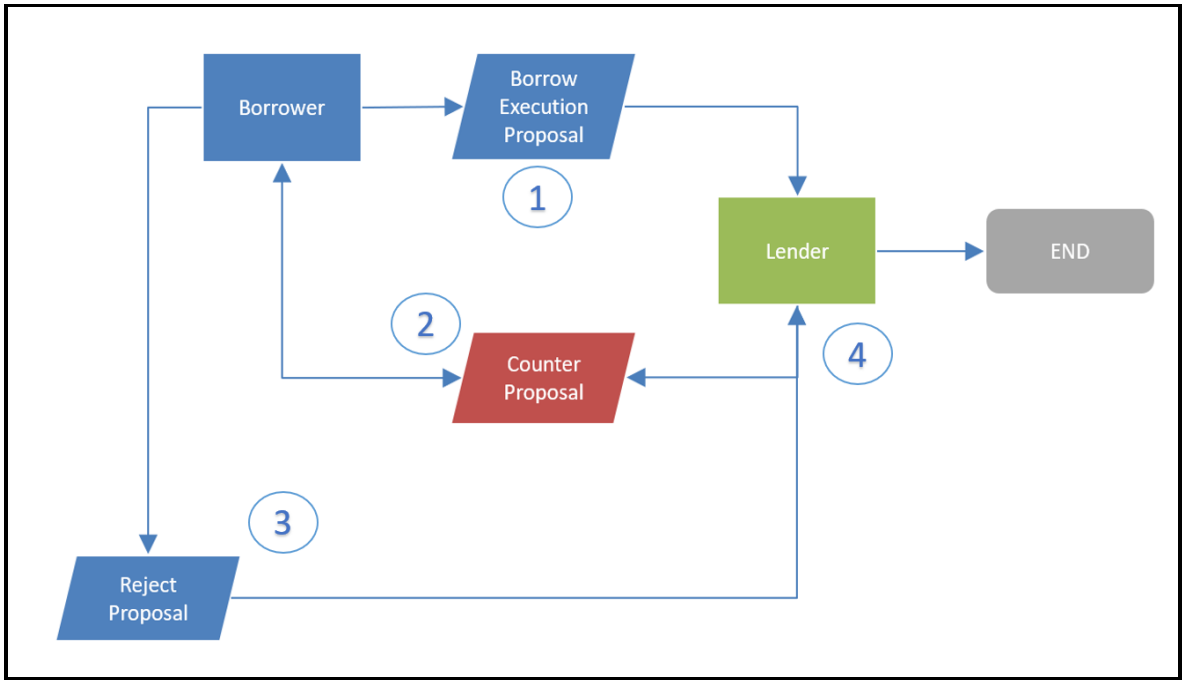

Diagram 28: Overview of the Propose-Counter-Reject workflow

12.1 Step 1 – Borrower creates the initial proposal

To start the negotiation process the borrower will need to create a new "WorkflowStep" object with the details of the trade execution in the "proposedEvent" object within it. This can be done using the "Create_ProposedWorkflowStep" function in the CDM.

The details on how to do this are the same as can be found in the section of the same name under the "Propose – Accept" scenario.

12.2 Step 2 – Lender offers a counter proposal

The lender will receive the "WorkflowStep" object created by the borrower, but decides that they want to negotiate some of the terms within the proposal. They thus build a counter proposal that they send back to the borrower.

The details on how the lender can create a counter proposal can be found in the section of the same name under the "Propose – Counter – Accept" scenario.

12.3 Step 3 – Borrower rejects the counter proposal

The borrower will receive the "WorkflowStep" object created by the borrower. This will hold the counter proposal being made by the lender.

If after checking the terms described in the “proposedEvent” the borrower does not want to continue with the trade, or no longer wants to continue with the trade for any other reason, then the borrower will need to reject the proposal.

To reject the event the borrower application will need to generate a rejected workflow step and pass this back to the borrower. This can be done by using the “Create_RejectedWorkflowStep” function in the CDM.

The details on how the borrower can create a rejected workflow step can be found in the section “Step 2 – Lender rejects the proposal” under the “Propose – Reject” scenario. The fact that this is the borrower rejecting the trade execution is not important, the process to create the rejection is the same.

12.4 Step 4 – Lender receives the rejected proposal

When the lender receives the new workflow step back from the borrower they will see that the “rejected” attribute is set to “true”. The lender will know from this setting that the borrower does not wish to continue with the negotiation of this event. No further action will be taken with this trade execution.

13 Best practices for bilateral trade negotiations

The flexibility and reusability of objects in the model is one of the greatest strengths of the CDM. This makes the definition of best practices and frameworks as described in this document fundamental to the consistent usage of the model throughout the industry.

To support the workflows outlined in this document a few best practices need to be observed by any parties wishing to implement the framework.

1. No further actions are allowed on a rejected proposal

When a party rejects a proposal then they are stating that they do not want to continue with this trade request at all. Applications need to also ensure that any further steps received with the rejected “eventIdentifier -> assignedIdentifier -> identifier” are ignored and preferably generate an error.

2. Both sides need to approve a proposal

If the borrower is happy with a proposal or counter proposal then they can generate a “proposedEvent” and set their “approval” status to true. Once the lender is happy with a proposal or counter they can generate a “businessEvent” and set their “approval” status to true.

Only once both parties have approved a proposal or a counter will it be created as a “businessEvent” by the lender. The lender always has the "last look" (see later point) so they will always be the last party to approve an event.

3. Sending a counter proposal

The original “proposedEvent” will be put into “previousWorkflowStep” for lineage and application validation purposes. A new “proposedEvent” object will be created that will hold the new details, which will be the same as the original “proposedEvent” with any updated terms.

The “eventIdentifier -> assignedIdentifer -> identifier” will remain the same to allow all steps for this proposed trade execution to be cross referenced. However, the “eventIdentifier -> assignedIdentifier -> version” will be incremented by +1, setting this as an update to the original proposal i.e. a counter proposal.

The party sending the counter proposal should set their “approval” status to true. They must set the “approval” status for the other party to false.

4. The Lender always has the “last look”

The borrower will always wait for a "businessEvent" from the lender before creating the trade in their application. This means that the borrower will thus only ever generate "proposedEvent" objects for either proposals or counters.

The lender can generate "proposedEvent" objects for proposals and counters, but will only ever generate "businessEvent" objects when both parties have approved the event.

ISLA

14 More complex negotiations

The scenarios described above are the most common scenarios that would be found during a bilateral trade execution. The framework we have described can also be used for more complex trade negotiations, other trade lifecycle events, or for multi-party negotiations.

It is possible, and in fact highly likely, that a trade negotiation could consist of many counter proposals. In this document we have shown a trade execution with a single counter proposal. The same counter proposal workflow could be used many times on a single negotiation though, allowing several trade terms to be agreed over several iterations of the proposed event details.

We have also assumed that the borrower will be initiating the negotiation. In our examples the borrower has already decided the lender that they want to fulfil the trade. This workflow could be started by the lender as well, where the lender is offering a number of shares at a particular rate to the borrower (or a group of borrowers).

In the same vein, where a borrower is sending out a prospective request to multiple lenders, the borrower could send out the message with the "approval" object not set. This would allow a lender to send back a proposed trade execution with the terms that they would find acceptable. This could be for the full quantity of shares requested by the borrower, or a proportion of them. The borrower could then decide which lender(s) they would like to fulfil the trade and send out proposed events to their selected/preferred lenders. This would facilitate peer to peer auto-borrowing of securities between the borrower and a single or many lenders.

We have focused on bilateral trade executions here, but the workflows described could equally well be used where there are multiple parties on the trade. An example of this would be where the trade execution was for a block trade and the lender had to disclose the funds allocated to the trade to the borrower.

When there are multiple parties on an event then there are a few additional considerations. For example, when a counter proposal is made then the "version" number is incremented in the message that is sent back to the requesting party. If multiple parties come back with counter proposals then the initiating party could receive multiple messages each with the same (or different) version numbers, and each with the same (or different) terms.

To overcome this potential issue with sequencing, specific rules/best practices can be put in place. One way to manage this would be for only the initiating party to handle the version number and approval status of the negotiation. For example, if the initiating party is party1 and they get a message back from party2 that has "version" set to 2, and a message back from party3 that has "version" set to 3, then they should respond with a message "version" set to 4 i.e. the next highest number in the version sequence. If party2 and party3 are happy with the contents of the new event then they will respond with a new message also setting "version" to 4; thus allowing party1 to recognise that the terms of the event have been accepted.

There are also potential issues here with retaining the lineage of the negotiation. Once again, one answer could be to ensure that only the initiating party updates the "previousWorkflowStep" object with the lineage as they see it.

As can hopefully be seen from the suggestions above there are a lot of different use cases for this framework. All that would be required to support some of these would be agreement between the parties on the best practices they want to employ for their negotiations – these can then be promoted throughout the industry for use by all participants.

15 Summary

The workflow processing built into the CDM gives us a framework that can be used for parties to propose, accept, counter and reject business events during a trade's lifecycle. The objects in the model can hold all the terms required for an event, and there are functions available that support the transition of an event through the workflow steps.

The model is flexible by design, allowing event data to be supported at various levels of granularity, the level required being defined by the applications that use the CDM and not the model itself. This flexibility means that best practices and guidelines like the suggestions in this document are critical when determining how a specific event should be negotiated. As more participants start to use frameworks like these, then the closer the industry will move to deciding upon standards for specific event negotiations.

As the CDM is a model, and not an application, there is still some work that needs to be undertaken by any application that wants to use these workflows. Performing basic validation checks and preserving lineage are two aspects of the negotiation process that the CDM would rely upon an application to perform.

Hopefully this document has served to generate some interest in how the CDM can be used to model event negotiation. We've shown some of the objects and functions that are already available in the model, and how to use them for a possible use case. Suggestions for new use cases and the objects and/or functions that would be required to support them are always welcomed by the CDM community.

The CDM is an Open Source model hosted by FINOS. If you would like to become part of the community then please go to the FINOS website and help us shape the standards of the future!

16 Thanks

Modelling diagrams and code snippets all taken from the Rosetta application kindly provided by REGnosys.

The framework itself was derived from the ISLA Trading Working Group, at the time consisting of representatives from the following member firms: Broadridge, FIS, GLMX, Pirum, REGnosys, TradingApps, ShareGain, Wematch.live

Special thanks to Mike Lambert from Broadridge for the workflow diagrams, and Rob Miles from GLMX for the JSON examples.