MicroBlock and EpochBlock Component

Overview

An EpochBlock is constructed at the end of every epoch (every 12 hours) to summarize the epoch. It includes voting results in the form of elected delegates, a summary of all transactions that were post-committed in the form of the MicroBlock tip, and all the transaction fee collected in the epoch. (In later releases, more information will be added to the EpochBlock). Intermediate epoch MicroBlocks are used to simplify the process of constructing a large epoch block. These MicroBlocks contains summaries of the transactions that occurred in a portion of the epoch. With MicroBlocks, nodes would be able to periodically ensure they are fully synched at a much higher frequency than if they solely relied on the full epoch blocks.

Both the MicroBlocks and EpochBlocks are created by delegates periodically. They must be approved by the delegate consensus before propagated to the network. The MicroBlock and EpochBlock component is responsible for handling all MicroBlocks and EpochBlocks related operations, this includes:

- Build MicroBlocks conforming to the IDD and initiate consensus sessions.
- Verify proposed MicroBlocks as delegates, and verify approved MicroBlocks as non-delegate network nodes. The two verification logic should be the same.
- Maintain the Microblock tip and the number of Microblocks in an epoch.
- Keep the sum of transaction fees in an epoch.
- Maintain a list of delegate candidates and their stake and voting power.
- Build EpochBlocks conforming to the IDD and initiate consensus sessions.
- Verify proposed EpochBlocks as delegates, and verify approved EpochBlocks as non-delegate network nodes. The two verification logic should be the same.
- Store valid MicroBlocks and EpochBlocks in the Database.
- Maintain timers for proposing MicroBlocks and EpochBlocks

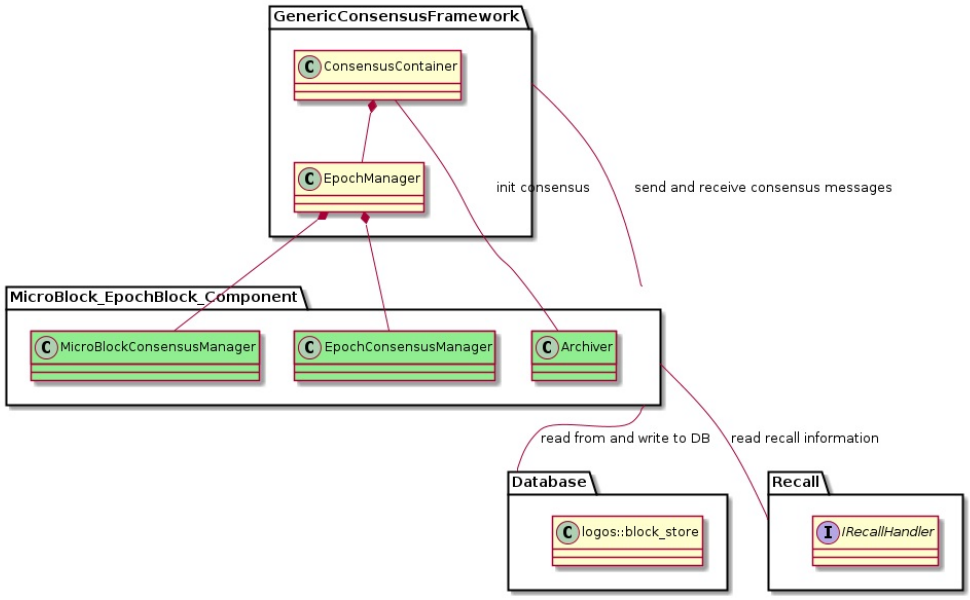
Execution Concept

The MicroBlock and EpochBlock component uses timers to propose MicroBlocks periodically. When a timer expires, one of the boost thread pool thread will call the proper callback function which will build a MicroBlock with the information read from the local database, and initiate a consensus session. Similar to the BatchStateBlock consensus, when backup delegates receive a pre-prepare message for a MicroBlock, they validate the block and reply with a prepare (and commit later) if the block is valid. When the last MicroBlock is approved by the delegates, the EpochBlock is proposed and a consensus session is initiated.

To reduce the unnecessary redundant MicroBlock and EpochBlock consensus sessions, the delegates agree on a default primary delegate of every MicroBlock or EpochBlock (for MicroBlocks, it is computed based on the hash value of the previous block; for EpochBlocks, it is the most voted delegate.). The default primary will propose first. Other delegates put the MicroBlock or the EpochBlock to their waiting lists with a timer. If the timer expires and the expected MicroBlock or the EpochBlock has not been post-committed, then the MicroBlock or the EpochBlock is proposed by the local node.

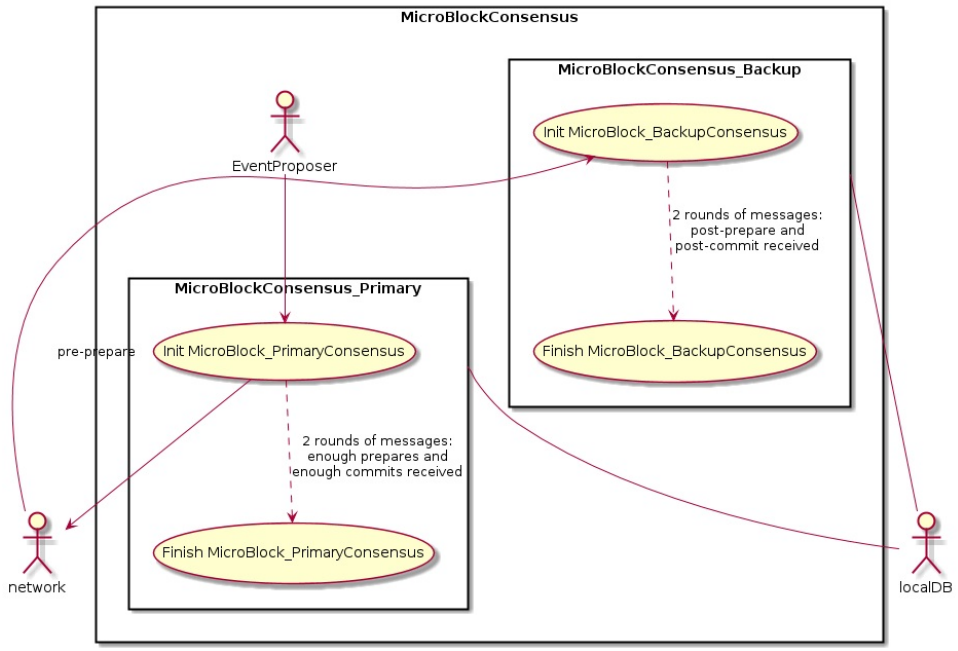
Interfaces

MicroBlocks and EpochBlocks are created by delegates and must be approved by the delegate consensus. Hence, as the interface figure shows, the MicroBlock and EpochBlock Component has an Archiver class which creates MicroBlocks and EpochBlocks. After that, the Archiver uses the ConsensusContainer of the generic consensus framework to initiate consensus sessions. The specialized functionalities, such as the verification of a MircroBlock proposed by another delegate, are implemented by the specialized ConsensusManagers (such as the MicroBlockConsensusManager). In addition, the MicroBlock and EpochBlock Component uses the database, and reads recall information from an IRecallHandler. (A more detailed figure is in the Classes section.)

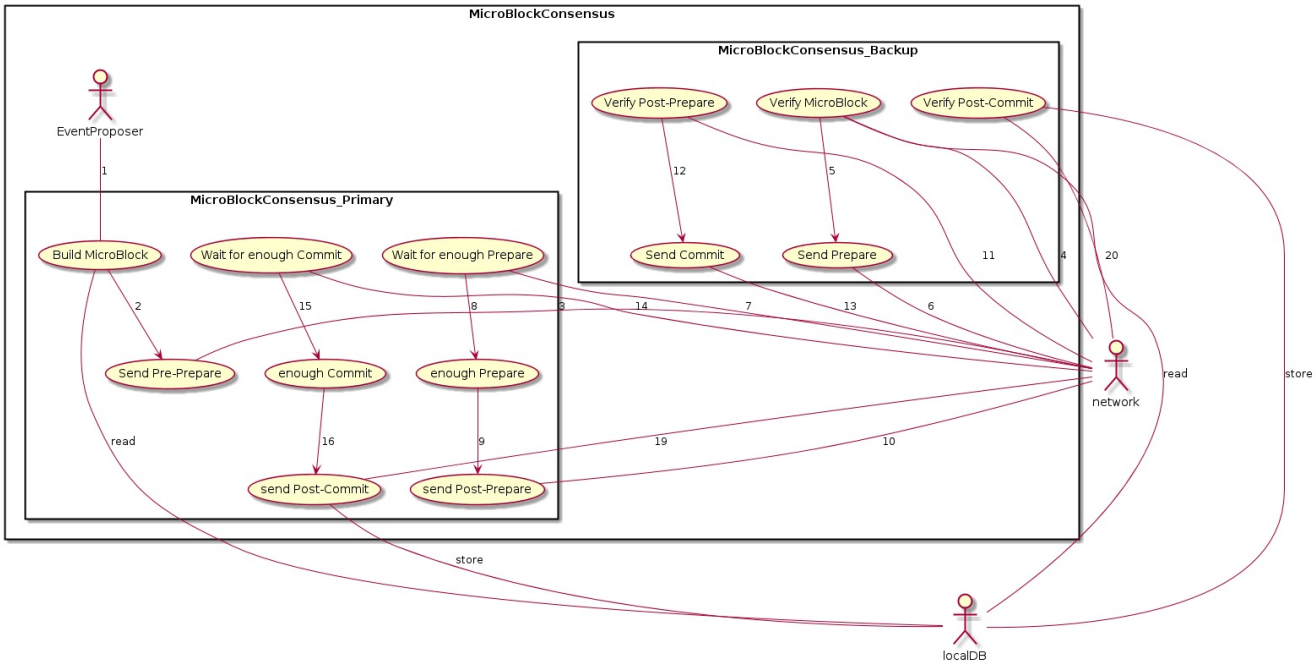


Use Case Diagrams

The use case diagram below show the high level idea of the MicroBlockConsensus. (The EpochBlockConsensus is similar.) The creation of a MicroBlock is triggered by the EventProposer periodically. Then the MicroBlockConsensus is initiated, and the pre-prepare message is sent to all other delegates. Receiving this messages, other delegates will start MicroBlockConsensus sessions as backups. Then the regular consensus protocol will proceed. More details can be found in the Classes section and the Sequence Diagram section.



A more detailed diagram (probably should delete).

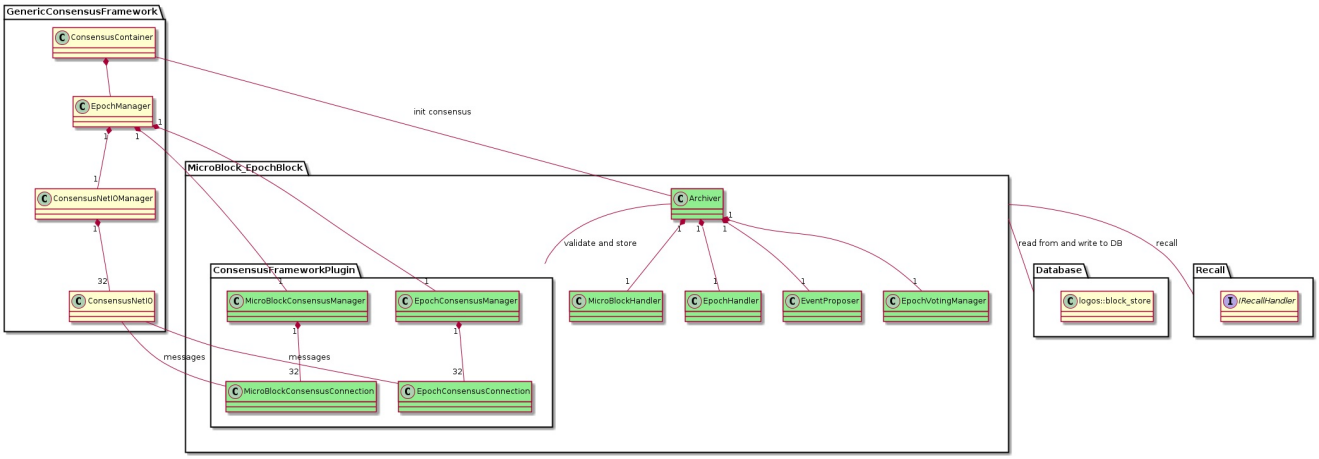


State Diagrams

The MicroBlock consensus and EpochBlock consensus use the generic consensus framework. No new states are introduced.

Classes

The figure below shows the important classes (green ones) of the MicroBlock and EpochBlock component, and their relationships with other classes. We will discuss them in the following subsections.



MicroBlockHandler

Class Overview

The MicroBlockHandler class contains utility functions for build, validate, and store microblocks.

Class Implementation and Algorithms

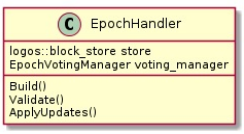
The core algorithm for building and verifying a MicroBlock is to find the list of tips (hashes) of delegatesâ€™ BatchStateBlocks, one tip per delegate. The algorithm include following steps:
- get the cutoff time of the MicroBlock being built.
- get the previous tips included in the previous MicroBlock.
- find the list of latest tips. The latest tips are stored in the database. (note that the latest tips are usually proposed after the cutoff time)
- for each of the latest tips, find the BatchStateBlock, which has the previous BatchStateBlockâ€™s hash. With this hash, one can find an previous BatchStateBlock. So on and so forth, one can walk the BatchStateBlock chain of every delegate. When we find a BatchStateBlock which is (1) proposed before the cutoff time and (2) proposed after the tip of the same delegate in the previous MicroBlock, this BatchStateBlockâ€™s hash is one of the list of tips we are looking for. Note that if no BatchStateBlock is approved during a MicroBlock period, the tip is NULL.

The MicroBlockHandler also access the database to retrieve information to build the MicroBlock and to store approved MicroBlocks. It also accesses an IRecallHandler to read recall information. For example, if a recall happened, the EpochBlock will not be proposed at the 12 hours boundary.

EpochHandler

Class Overview

The EpochHandler class contains utility functions for build, validate, and store EpochBlocks.



Class Implementation and Algorithms

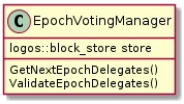
The core algorithm of building and verifying an EpochBlock is to come up with a list of most voted delegates by summarizing the votes processed by the delegate consensus protocol. This algorithm is actually implemented by the EpochVotingManager. The EpochHandler simply calls EpochVotingManager::GetNextEpochDelegates().

The EpochHandler also access the database to retrieve information to build the EpochBlock and to store approved EpochBlocks.

EpochVotingManager

Class Overview

The EpochVotingManager class is mostly a placeholder for managing voting related functionalities. Currently it maintains a fixed list of 64 delegates and rotates them. When GetNextEpochDelegates() is called, 32 delegates are returned, 8 of them are new. The ValidateEpochDelegates() function is used to validate the set of delegates in an EpochBlock.

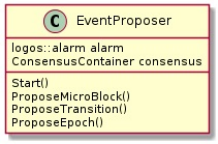


EventProposer

Class Overview

The EventProposer class contains timers for triggering the event for proposing MicroBlocks, and epoch transition. Once the last MicroBlock of an epoch is consensus approved, the EpochBlock of the epoch is proposed.

The EventProposer is started (by calling Start()) when the local node starts.



Class Sequence Diagram

Please see the overall Sequence Diagram in the Scenarios and Sequence Diagrams section.

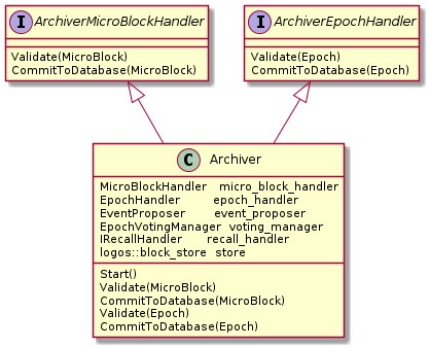
Class Implementation and Algorithms

In the implementation, a single logos::alarm object is used for all the timers. The ConsensusContainer object is implicitly held by callbacks.

Archiver

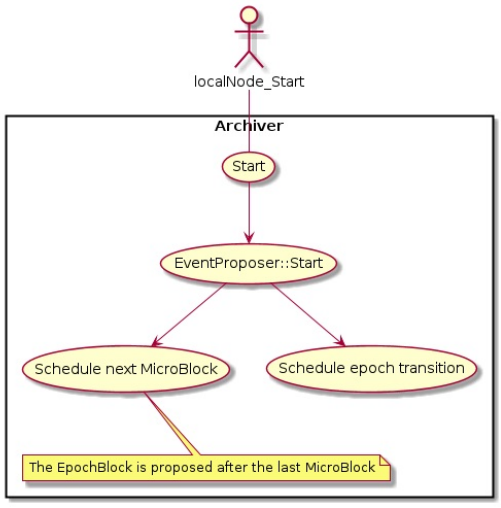
Class Overview

The Archiver class is composed of classes listed above, namely MicroBlockHandler, EpochHandler, EventProposer, and EpochVotingManager. It also implements the functions for validating and storing MicroBlock and EpochBlock, inherited from two interfaces as the diagram shows.

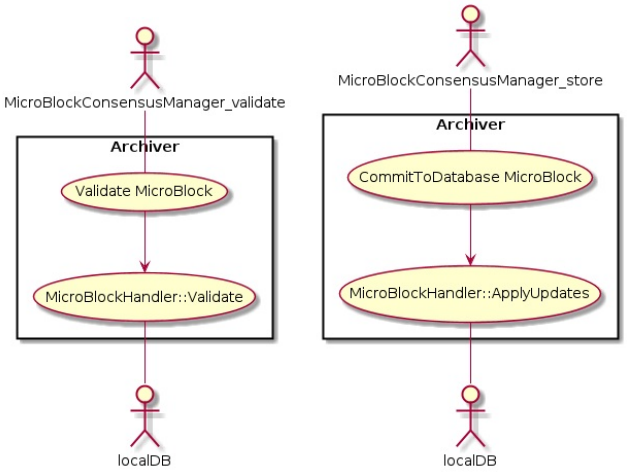


Class Use Case Diagram

When the local node starts, it starts the Archiver which starts the EventProposer. The EventProposer inserts two timers, one for the next MicroBlock, the other for epoch transition.



As the figure below shows, the MicroBlockConsensusManager uses the Archiver to validate or store a MicroBlock. The Archiver will use the MicroBlockHandler for these tasks. (The EpochBlock related use cases are similar.)



Class Sequence Diagram

Please see the overall Sequence Diagram in the Scenarios and Sequence Diagrams section.

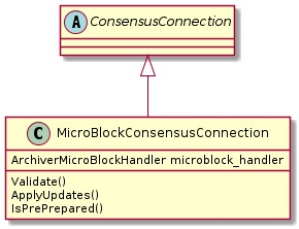
Class Implementation and Algorithms

Most functions are pass-through to the members.

MicroBlockConsensusConnection

Class Overview

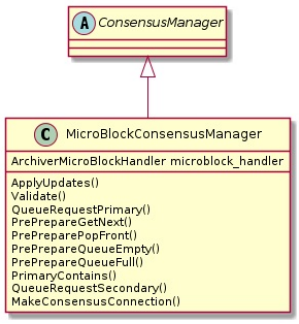
The MicroBlockConsensusConnection class inherits from the generic ConsensusConnection Class, and can be plugged into the generic consensus framework to handle MircoBlock specific functions. It uses the ArchiverMicroBlockHandler, which is the Archiver class explained previously, to validate MicroBlocks and apply updates. IsPrePerpared() returns if a MicroBlock is already pre-prepared, i.e., in an ongoing MicroBlockConsensus session.



MicroBlockConsensusManager

Class Overview

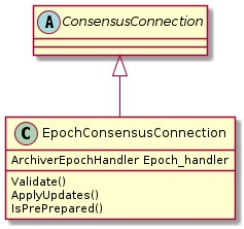
The MicroBlockConsensusManager inherits from the generic ConsensusManager Class, and can be plugged into the generic consensus framework to handle MircoBlock specific functions. It uses the ArchiverMicroBlockHandler, which is the Archiver class explained previously, to validate MicroBlocks and apply updates. The inherited queue related functions are very simple because the queueâ€™s length is at most one.



EpochConsensusConnection

Class Overview

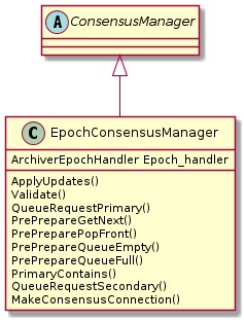
The EpochConsensusConnection class inherits from the generic ConsensusConnection Class, and can be plugged into the generic consensus framework to handle EpochBlock specific functions. It uses the ArchiverEpochHandler, which is the Archiver class explained previously, to validate EpochBlocks and apply updates. IsPrePerpared() returns if a EpochBlock is already pre-prepared, i.e., in an ongoing EpochBlockConsensus session.



EpochConsensusManager

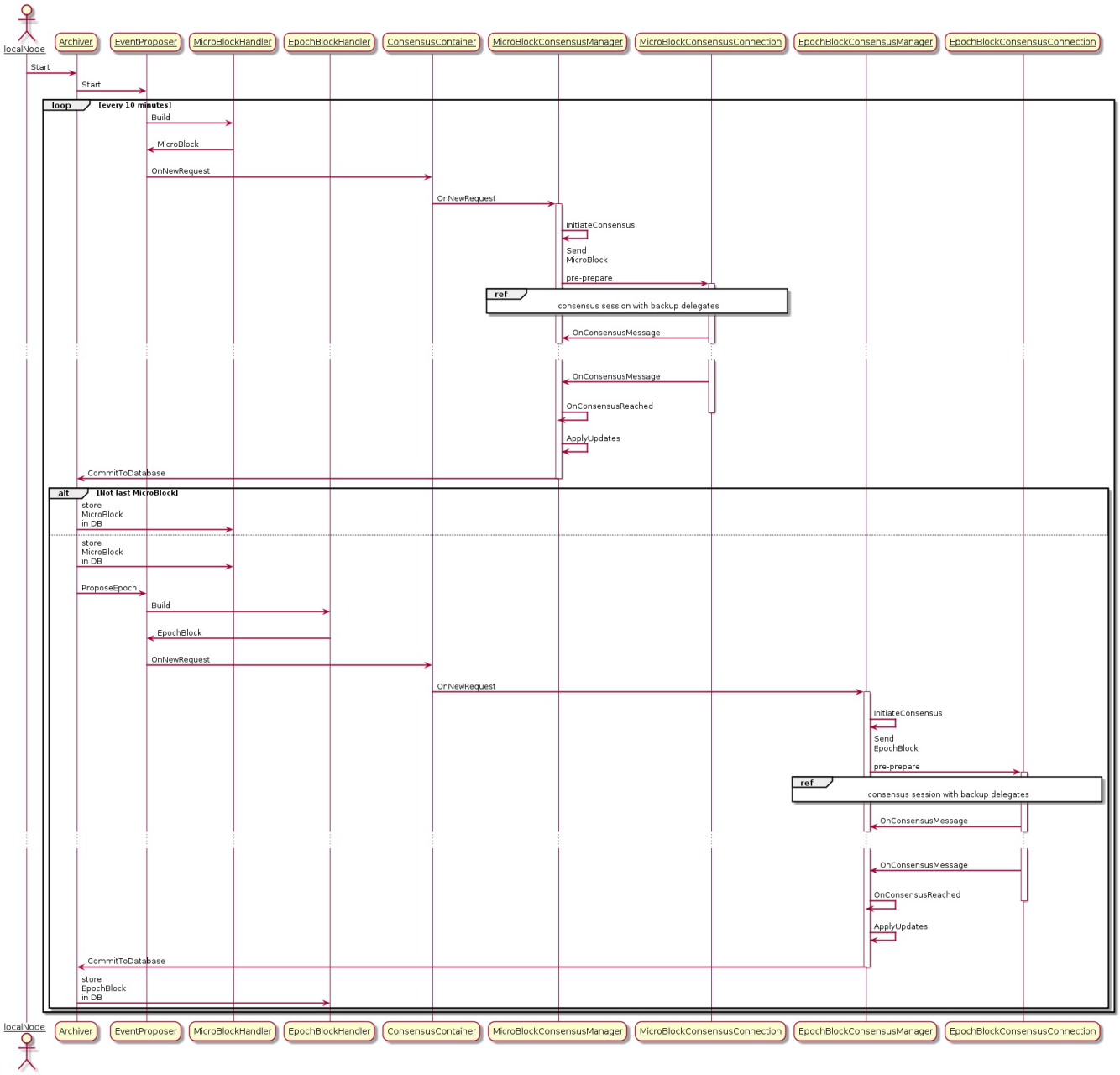
Class Overview

The EpochConsensusManager inherits from the generic ConsensusManager Class, and can be plugged into the generic consensus framework to handle EpochBlock specific functions. It uses the ArchiverEpochHandler, which is the Archiver class explained previously, to validate EpochBlocks and apply updates. The inherited queue related functions are very simple because the queueâ€™s length is at most one.



Scenarios and Sequence Diagrams

As the sequence diagram below shows, the localNode starts the Archiver, which in turn starts the EvenProposer. The EvenProposer schedules a timer, every 10 minutes, a MicroBlock is proposed and runs through the consensus protocol. If a MicroBlock is the last one of an epoch, the EpochBlock is proposed and processed by the delegate consensus. Note that this sequence diagram is from the primaryâ€™s delegate point of view. The backup delegates sequence is not shown.



Note that when the EvenProposer starts, it also schedules another timer for epoch transition. This sequence will be covered by a separate epoch transition design document.

Resources

This component use the network and database, same as the regular consensus.

Implementation and Algorithms

Covered in the previous sections.