***Dispatcher***

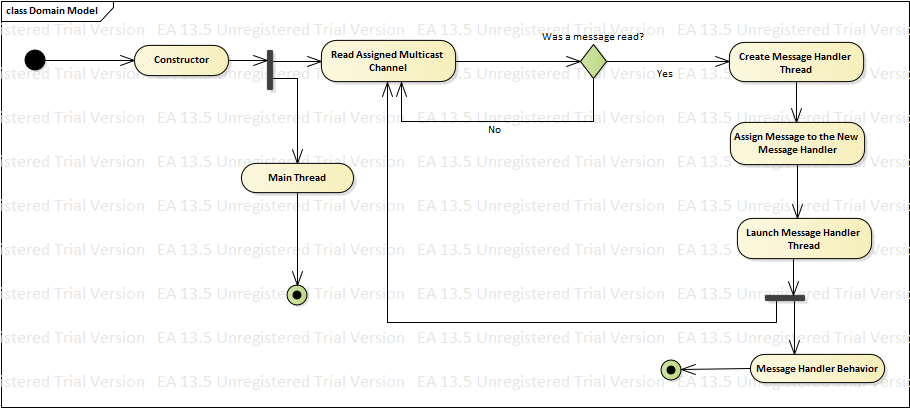
This implementation uses a hybrid of event driven and threaded architectures.

The dispatcher is divided into 2 key components the dispatcher reader and the dispatcher message handler.

The dispatcher reader the main source of concurrency in the event system, it is a threaded method that lives in an infinite loop trying to read its assigned multicast channel. When a message is read from the channel the dispatcher reader creates an instance of the message handler and gives it the read message to process, this message handler is then launched in an independent thread and continues reading. While creating a thread has inherent overhead and there might be a slightly better way to handle this we considered that this implementation fitter out design goals well enough, those goals being to minimize as much as possible the time the dispatcher reader is not reading the socket while maintaining architecture clean and easy to understand and iterate upon.

The dispatcher message handler’s behavior is not fundamental to the dispatcher’s concurrent architecture and as such we will not describe here how it behaves as it is a very simple message parsing and method invocation based on the parsed message. However, the message handler does play a key role in the concurrent behavior of the GETCHUNK and PUTCHUNK sub protocols, the role is notifying an active threads waiting for response messages, as the way this is done is specific to each of the sub protocols we will no go into the architecture in this section, we will reserve this for later explanation when describing the concurrent architectures of the 3 main protocols of the system.

Bellow we can see an activity UML diagram of the dispatcher.



UML Activity Diagram of the Dispatcher