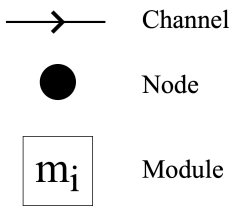


**Figure 3.6:** Constructs used

1. **Sequence** : The simplest of all connectors. A task in the workflow is activated upon the completion of another task along the same branch in the workflow. This connector emulates the functionality of Reo's Sequencer.

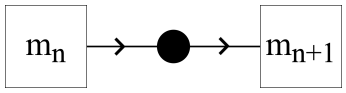


**Figure 3.7:** Sequence connector

As shown in Figure 3.7, by allowing a module to consider the output channel of the previous module, as its input channel, the framework facilitated the sequence workflow pattern.

2. **Parallel Split** : Used to diverge a branch into two or more parallel branches which execute concurrently.

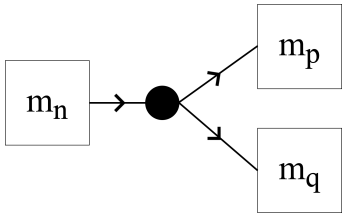


**Figure 3.8:** Parallel Split connector

As demonstrated in Figure 3.8, the framework facilitates this behaviour by allowing a number of separate branches to accept the output channel of a particular module a signal to execute the next modules along each workflow. Go-routines allow concurrent execution of each workflow branch.

3. **Synchronization** : Handles the intersection of two or more branches into a single subsequent branch and control is passed to this subsequent branch.

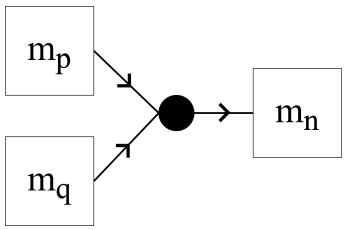


**Figure 3.9:** Synchronization connector

This connector acts as an AND-join. If both of the messages are not received from $m_p$ and $m_q$ output channels, $m_n$ would not execute. Instead it would wait until both messages are received. Thereby, facilitating an AND-join. In the context of the framework, the connector would take the inputs of the two branches, combine them and direct it to an output branch, as depicted in Figure 3.9. Although currently defined to integrate two branches, the user has the ability to compose a similar function following the same code format to accommodate any number of input branches.

4. **Exclusive Choice** : Splits a branch and redirects execution based on some selection mechanism defined in the control thread. The thread of control is passed to one of the outgoing branches based on some condition being met or not. The Reo connector, ”Exclusive Router” functions in a similar manner.

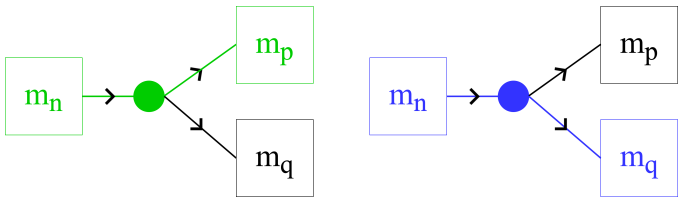


**Figure 3.10:** Exclusive Choice connector

This workflow construct is facilitated through the dynamism allowed within the framework as discussed in Chapter 3.3.2 - Dynamism. Figure 3.10 illustrates an exclusive choice situation; the path of the workflow is amended based on some selection criteria defined within the control thread.

5. **Simple merge** : Converges two or more branches into a single subsequent branch.

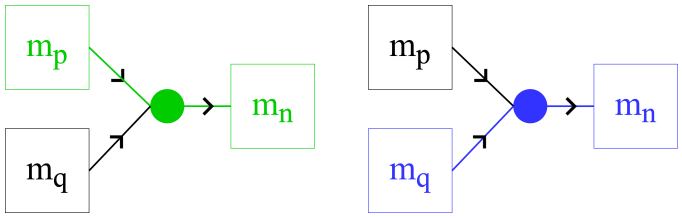


**Figure 3.11:** Simple Merge connector

It is designed to facilitate an XOR-join where if either one of the two incoming modules pass a message to $m_n$, the thread of control would be passed on (See Figure 3.11). It does not wait for both incoming modules to complete as demonstrated with the Synchronize connector. Instead, the $m_n$ module has the capability to accept one incoming message in a "first-come first-serve" fashion.

6. **Multiple Choice** : Used in instances where a branch diverges into two or more branches. When the input branch has completed, the thread of control is immediately passed to one or more of the outgoing branches based on some selection mechanism.

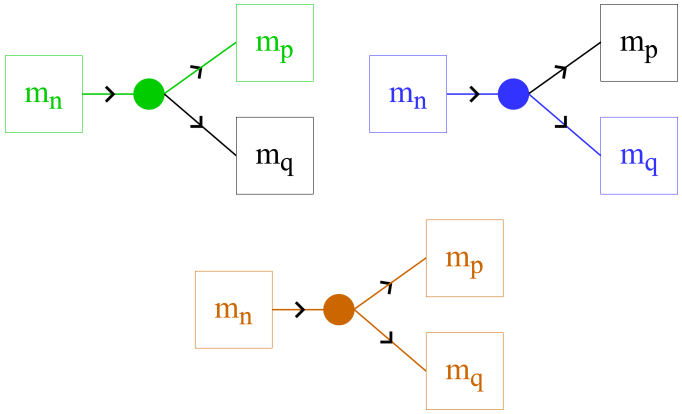


**Figure 3.12:** Multiple Choice connector

The framework provides dynamicity by employing these connectors. The decision based on the selection to pass the thread of execution to a subsequent branch is made at runtime.

The operation of the Multi-Choice pattern is illustrated in Figure 3.12. After module $m_n$ has been triggered, the thread of control can be passed to one or both of mp and $m_q$ depending on the evaluation of the conditions associated with each of them.

7. **Loop** : A special connector supported by the developed framework. Loop can be utilised to execute one or more modules repeatedly for a predefined number of times. The corresponding Reo connector to this is the ”Feedback Loop”.

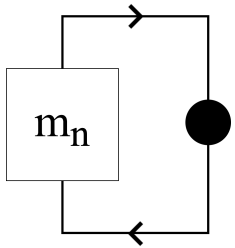


**Figure 3.13:** The Loop connector

As depicted in Figure 3.13, module mp executes n times. The output message from each iteration would act as the input to the next iteration.

The relevant code segments of all aforementioned connectors are included in Appendix A - Code Implementation of Connectors.