Owin Framework Segmentation White Paper

This paper describes the theory behind the way that the Owin Framework Builder builds the Owin Pipeline.

Background

One of the features of the Owin Framework is that middleware components can describe their dependencies on other middleware components and the pipeline builder will figure out how to build the pipeline so that all the dependencies are met.

When middleware declares a dependance it can specify the name of a specific middleware instance it depends on, or the class of middleware it depends on. It can also indicate whether the dependant is required or optional.

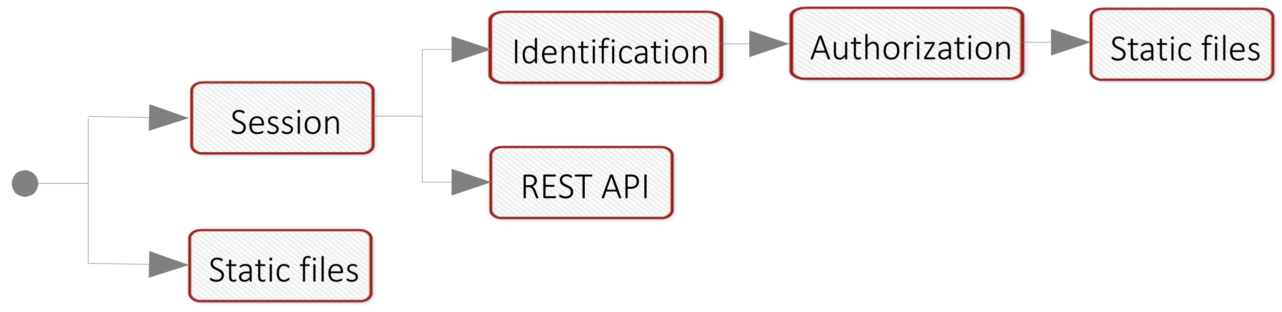
For a required dependant, the middleware it depends on must be present for this middleware to function correctly. An example of this type of dependance is the the authorization middleware which checks if the caller has permission to perform the requested action depends on the identification middleware that identifies the caller. Unless we identify the caller there is no way to determine if they have permission or not, so this is a required dependant.

For an optional dependant, the middleware will make use of the middleware it depends on if it is present, but if it's not configured then this middleware will still work just fine without it. An example of this type of dependency is the static files middleware optionally depends on the output cache. In this case if there is no output cache configured the static files middleware will still serve static files just fine, but when the output cache is configured the static files middleware will let it know that the static files can be cached, and if the file is modified it will tell the output cache not to use the cached version.

In the simplest case where the Owin pipeline is a list of middleware components linked into a single chain, all that we have to do is build a dependency tree then walk the tree bottom up to figure out the order to run the middleware components in. There is a class in the Owin Framework that does this.

This paper is concerned with the much more complicated case where the Owin pipeline splits into multiple segments, and therefore there are several paths (or routes) that requests can take through the pipeline.

This is an example of a segmented pipeline with multiple routes:



In this drawing:

- The rectangles are middleware components. The labels inside the rectangles describe what kind of middleware it is.

- Request processing starts at the left hand side, and follows the arrows from left to right, executing middleware components in the order they are encountered.
- Where there are multiple paths that can be taken (for example after the session middleware) there is a routing decision. This decision is based on examining the OWIN context. For example the decision could be based on the path of the request, the presence of a certain header, the user agent string, something added to the OWIN context by other middleware (such as session variables) etc. This decision logic is written by the application developer.
- This pipeline has 5 segments. The first segment at the left side of the drawing has no middleware in it, an empty segment in this case. The second segment contains only session middleware, the third segment contains only static files, the fourth segment contains identification, authorization and another instance of static files, and the last segment contains the REST API.
- This pipeline has 3 routes, these are the three possible paths that you can take from the left side of the diagram to the right side of the diagram by following the arrows.

This drawing describes an application in which:

- There are some static files that are served directly by the static files middleware without any other processing taking place.
- There are some other static files that have restricted access. The access restriction is provided by the authorization middleware which depends on identification middleware which in turn depends on session.
- There is a REST API that has no security restrictions but needs session.

Note that this is just one example, there are limitless possibilities for how middleware can be arranged to provide the functionality the application needs without executing any unnecessary steps.

Owin Framework Pipeline Configuration

Pipeline configuration has the following aspects:

- Application developers build their applications by installing third-party middleware and/or writing middleware as part of their application.
- Within the middleware itself, there can be a declared dependency on another type of middleware (for example authentication depends on identification).
- Dependencies can be required (the dependant must exist before this middleware on any routes that include this middleware) or optional (the dependant does not have to exist on the route, but if it does then it must come before this middleware).
- The application developer can define additional middleware dependencies that are required by the application. These can be a dependency on a certain type of middleware or a specific instance. These application defined dependencies can be either required or optional.
- The application developer defines the routes that they want in their application. If the application developer does not define any routes, then the Owin Framework Builder constructs a pipeline with only one segment.
- The application developer must assign at least one middleware component to each route.

Middleware components that are not explicitly assigned to routes by the application developer will be placed by the Owin Framework Builder. The builder will place middleware so that all dependencies are satisfied and additional functionality is not introduced where is was not wanted by the application developer.

- The application developer can assign a middleware component to more than one route.
- The application developer can create multiple instances of the same middleware type and configure each instance independantly.

The subject of this white paper is: how can the builder correctly construct segments and place middleware into those segments given the configuration information that is described above.

Pipeline Building Rules

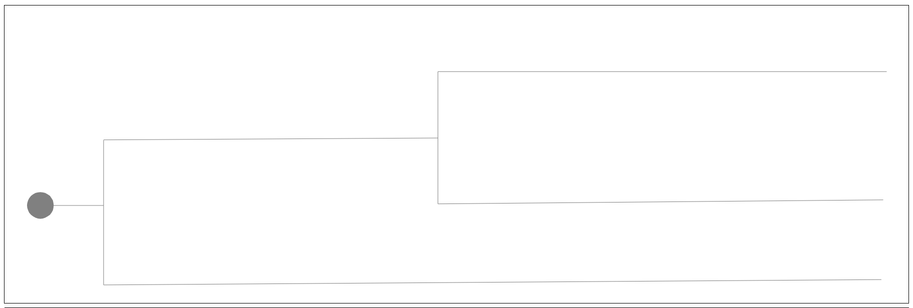
The Owin Framework Builder must obey the following rules in order to build a correct pipeline:

1. Middleware assigned to a specific route by the application developer must be present on that route. Note that application developers can assign a single instance of a middleware to multiple routes, in which case the same instance can be wired into both routes, or it can be placed on a segment that is shared between all the routes it is on.
2. Where middleware instances have required dependants, these dependants must be present on all routes that contain the middleware itself, and the dependants must be encountered before the middleware that depends on them when the pipeline is traversed from left to right.
3. Where middleware instances have optional dependants, these dependants must not be encountered after the middleware itself when the pipeline is traversed from left to right.
4. Routes must only contain the middleware that was explicitly assigned by the application developer and required dependants.

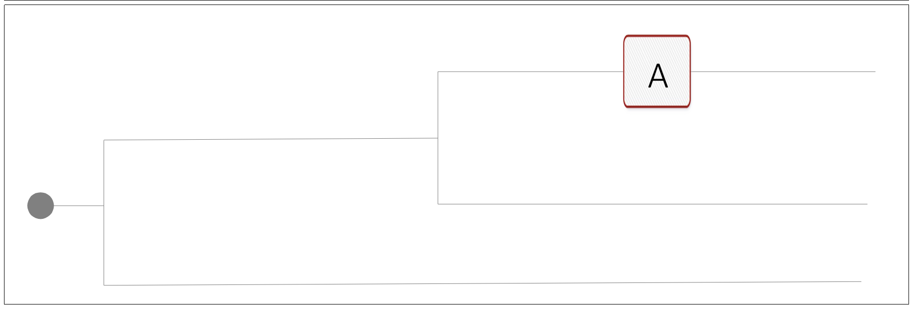
Use Cases

This section contains example configurations and how the pipeline should be built for each configuration. These use cases are good for evaluating the proposed algorithm, and can also be the basis for a set of unit tests.

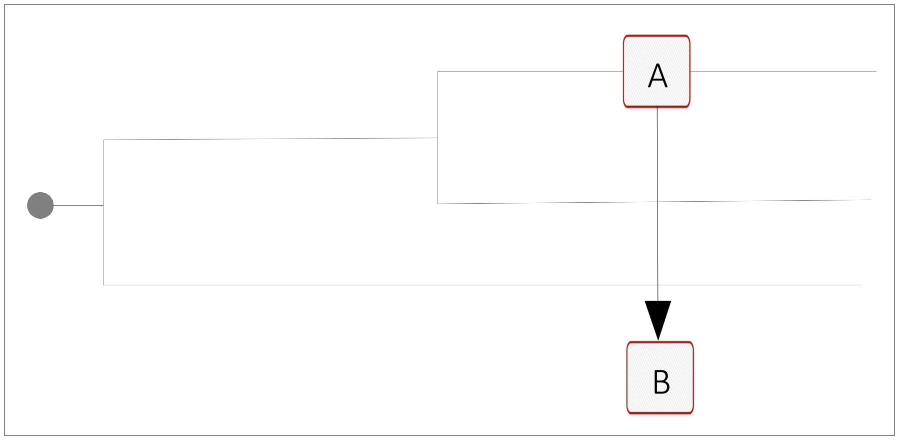
In these use cases the following notation will be used:



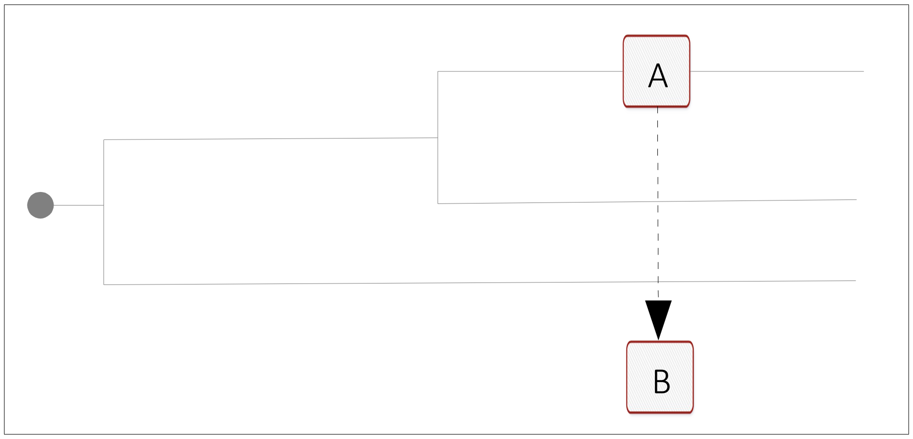
This denotes an Owin pipeline with segments and routes. Each horizontal line is a segment. Each path that can be traced from left to right is a route.



This denotes a middleware instance (A) that has been assigned to a specific route by the application developer.

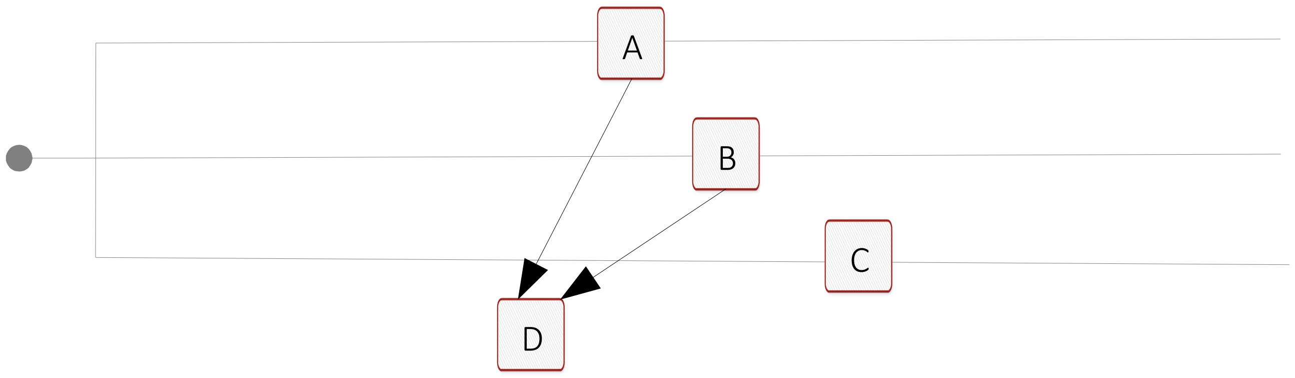


This denotes a required dependency. Middleware A depends on middleware B. Middleware B was not assigned to a route by the application developer and must be placed by the pipeline builder so that it comes before Middleware A in all routes that include middleware A.

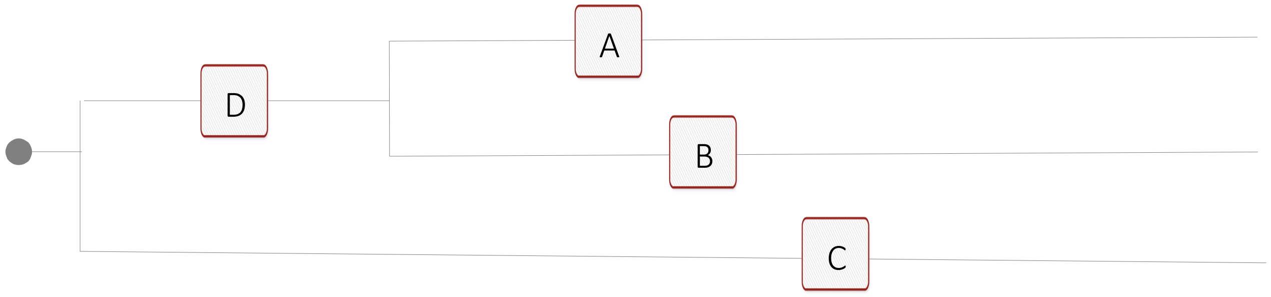


This denotes an optional dependency. Middleware A optionally depends on middleware B. If middleware B is on the same route as middleware A then B must come before A in the pipeline.

Additional segments use case



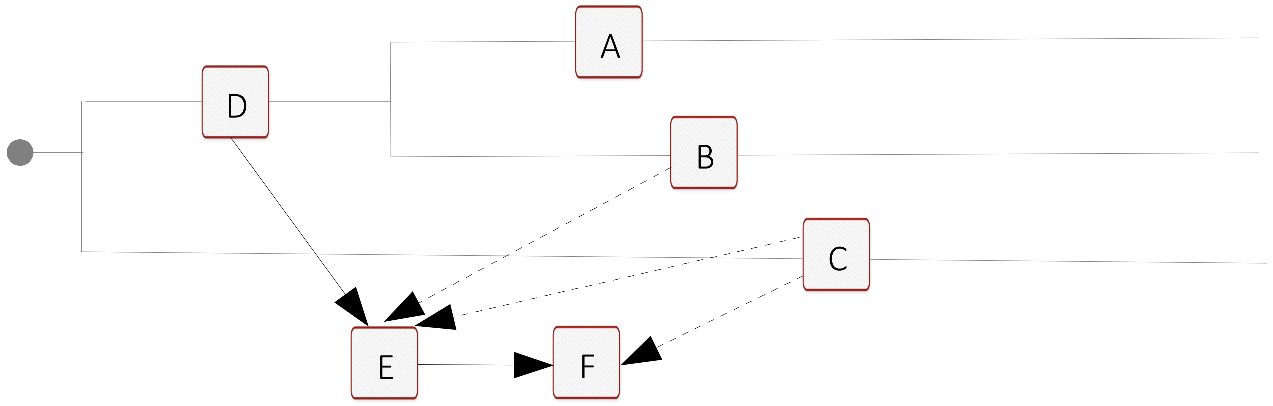
In this use case there are three middleware instances on separate routes but two of these instances share a required dependance. The pipeline should be built like this:



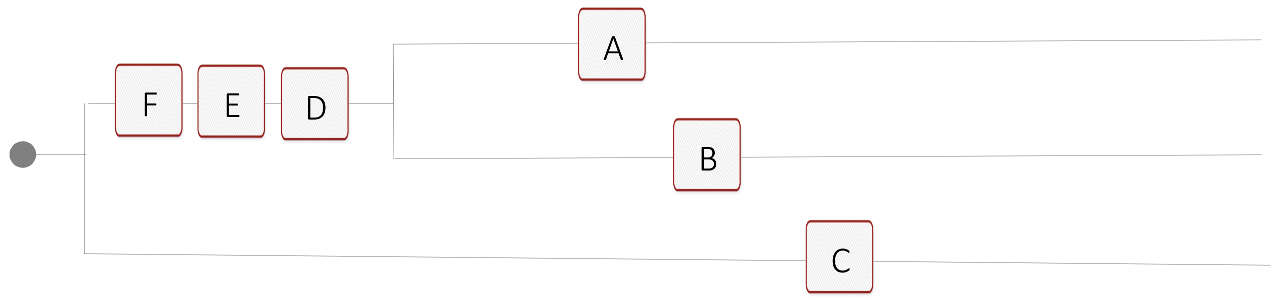
In this case the pipeline builder needs to introduce an additional segment for middleware D so that it is common to A an B but does not run on the same route as C.

Optional dependency

In this use case C has an optional dependency on F. This should not add F to the C route. The configuration looks like this:

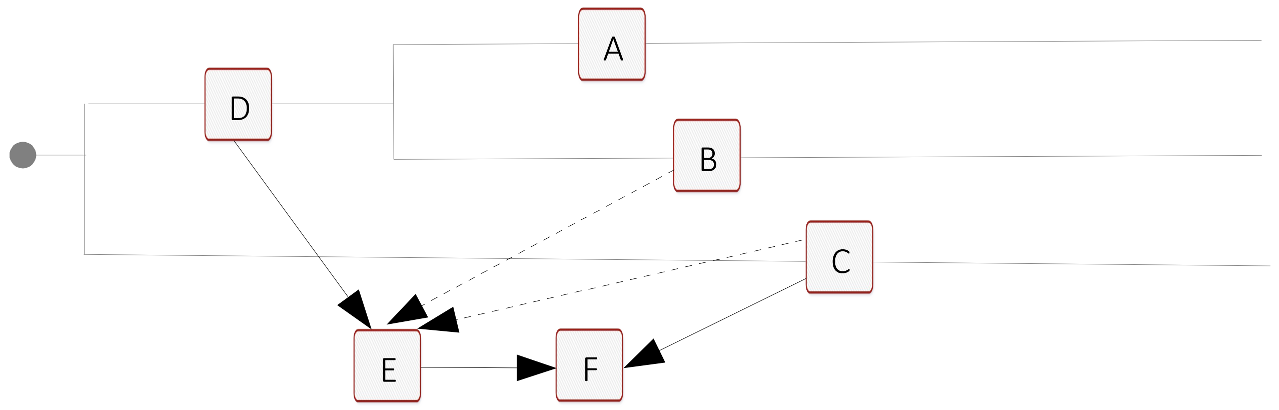


And the builder should build this pipeline:

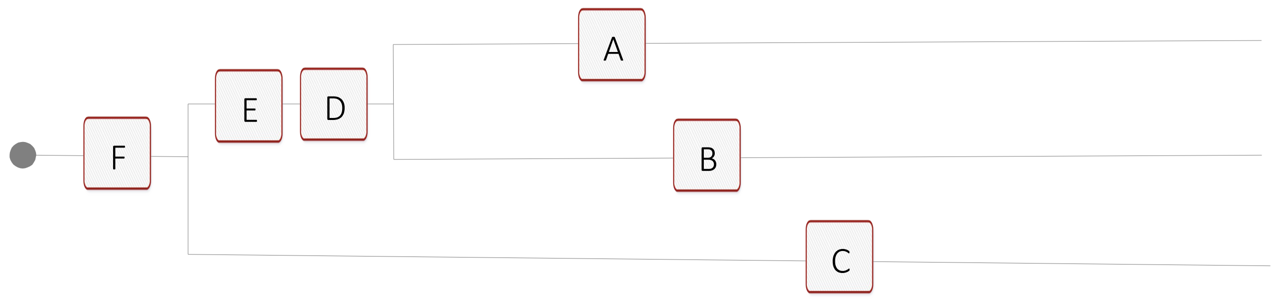


Required dependency

This use case is exactly like the previous one except that the dependency between C and F is required. The configuration looks like this:

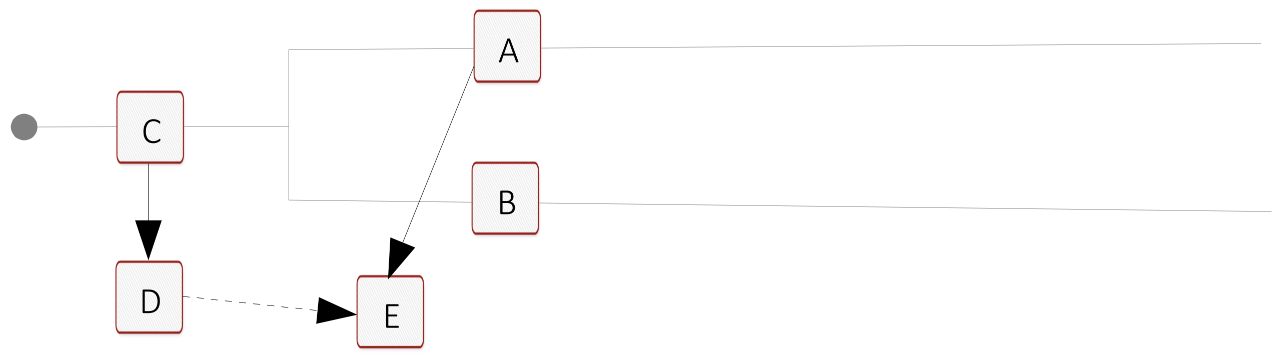


And the builder should build this pipeline:

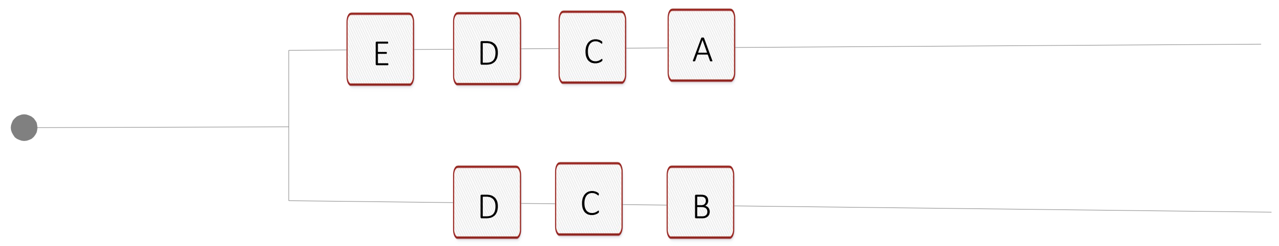


Duplicate to satisfy optional dependency

This use case demonstrates an example of where the builder must duplicate a middleware instance across two segments in order to satisfy the optional dependency. The configuration looks like this:

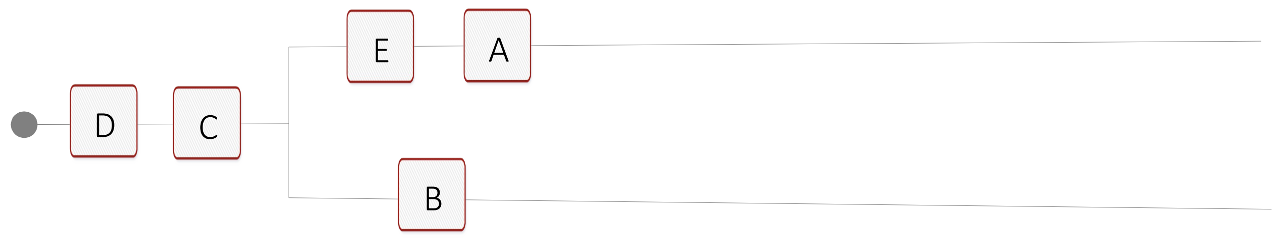


And the pipeline should look like this:



Note that there is an implied optional dependency from A->C and B->C because the application developer put C on a segment that precedes A and B. This must be taken into account when ordering the components within the segment.

Without the optional dependency between D and E, the pipeline would look like this instead:



Proposed Algorithm

This is an algorithm that appears to satisfy all of the requirements.

1. Construct a pipeline containing all the segments defined by the application developer and assign middleware instances to segments based on the application configuration.
2. Check required dependencies between nodes that are already assigned and where required dependencies are not met, duplicate middleware onto segments containing middleware that depends on them. Repeat until there are no more broken required dependencies for the application configured middleware.
3. Find the middleware instances that are not already assigned to a segment but have assigned

instances with a required dependency on them. Assign these middleware instances to each of the segments containing middleware that depends on them. Repeat this until no more assignments are made.

4. Find middleware instances that are not already assigned to a segment and have required dependencies. Place these on the segment that is closest to the left side of the pipeline and where all dependant middleware is left of this one or in the same segment. Repeat this until no more assignments are made.
5. Place all nodes not assigned to a segment into the root segment (at the far left of the pipeline).
6. Find all middleware instances that are assigned to multiple segments. If these segments have a common prior segment then move the middleware instance into the prior segment. Only move middleware that does not have any required dependants in the same segment. If there are other branches off the prior segment and these sibling segments do not contain the duplicated middleware then create a new segment in between this segments and the prior one. Repeat until no more movements are made.
7. Find all middleware instances where their optional dependencies are violated, i.e. There exists an optional dependant in a segment that is to the right of the middleware with the dependency. Move these middleware instances and duplicate into all segments to their right. Also move any other middleware in the same segment that has a required dependency on the one you are moving/duplicating. Repeat until their are no more violated optional dependencies.