Websphere MQ Everyplace Designer

Phill van Leersum
WMQE Development
IBM Hursley Laboratories

Change History

Version	Change
First Draft	

Notes

- **In this document 'Websphere MQ Everyplace' is often abbreviated to 'WMQe'**
- **WMQE Designer is currently undergoing development. Some of the facilities mentioned in this document have changed. The documentation will be corrected when the designer code stabilises, but is still included as 'better than nothing'.**

1.0 WMQE Designer

This is a tool designed to allow the design and comprehension of WMQE networks. The tool provides the following features:

- Model WMQE Network graphically
- Save and Load Models (xml format)
- Manipulate the view of the model to expose various levels of complexity
- Calculate queue route resolutions
- Independant of WMQE - requires only java 118 or later

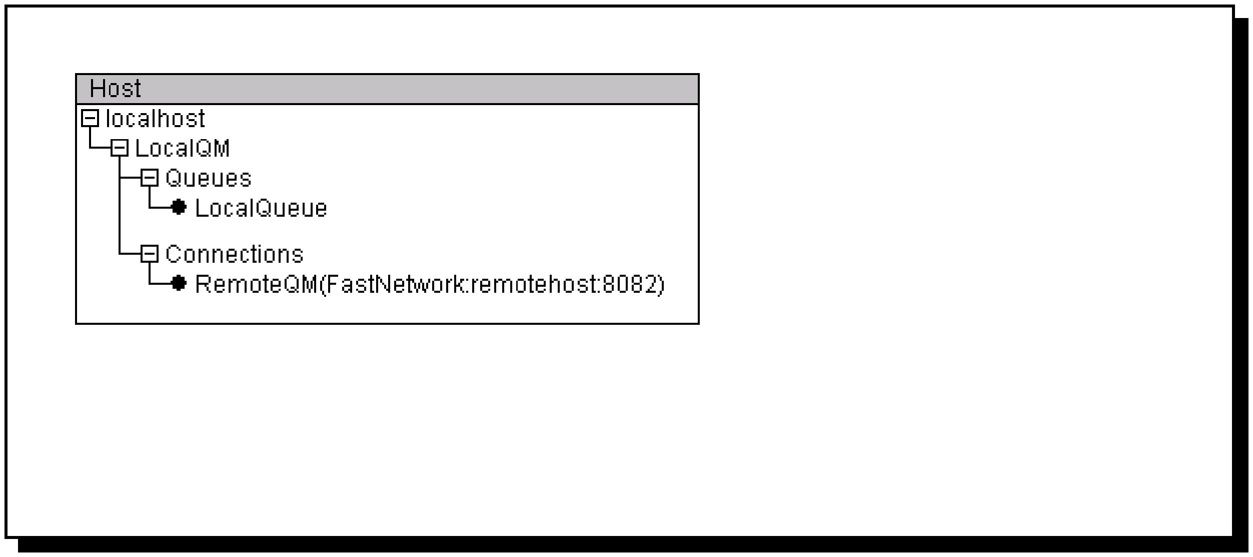
Optional extensions allow the 'Investigator' functionality:

- Reading an existing WMQE network into the model
- Administering an WMQE network
- Creating an WMQE network from a model defininition file.

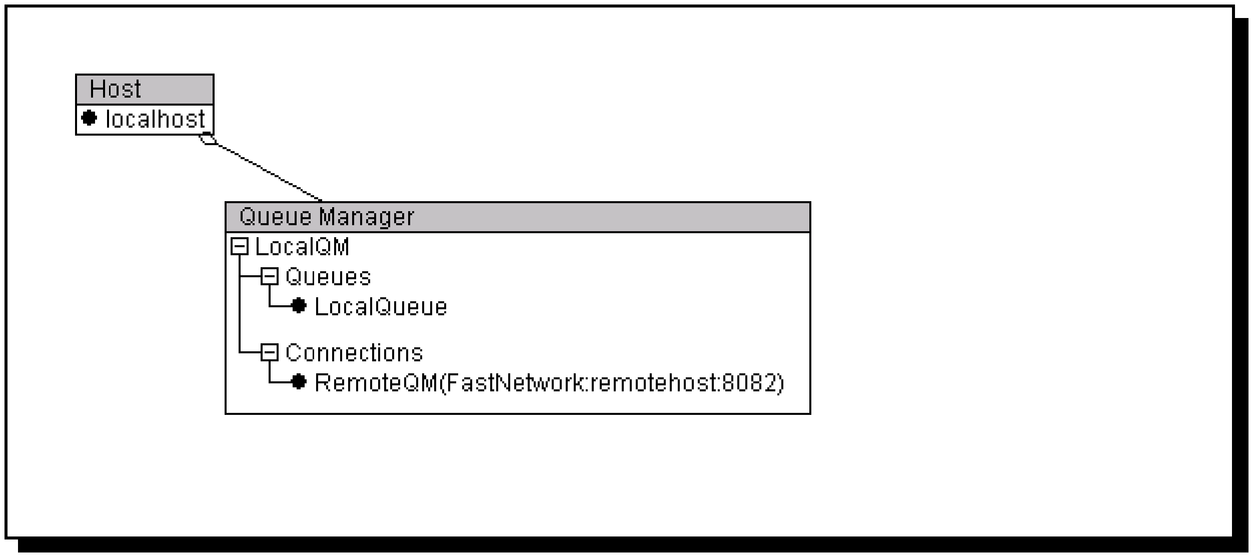
2.0 Basics

2.1 Notation

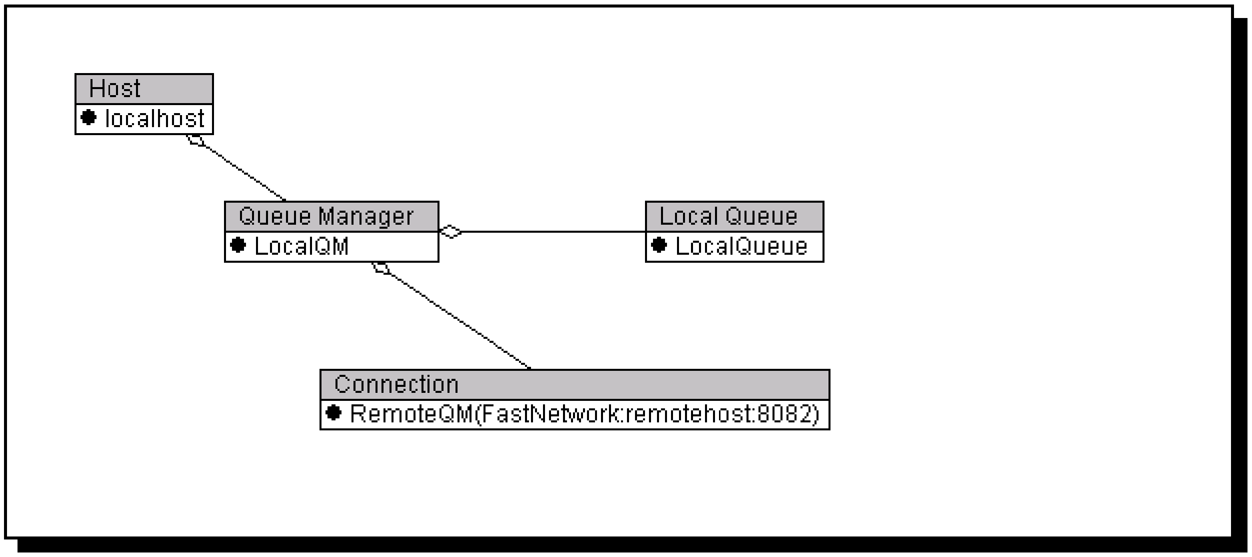
The WMQE Designer interface is a graphical representation of the WMQE Resources (queue managers, queues etc) in a WMQE network. It combines tree views into a graphics view. The trees are 'decomposable' into sub trees, simply by dragging an WMQE Resource from its parent. This is easier to show diagramatically. First here is a simple diagram showing a host and the WMQE Resources on it:



and the same system where the queue manager 'LocalQM' has been detached from its parent host:



The line with a diamond shape shows that the queue manager is the child of the host. This preserves the parent/child relationship from the tree, that would otherwise be lost by detaching. Any resource shown a shadow is detachable from its parent. The queue manager has two 'categories' within it - these are not detachable. If the categories are empty, then they will not be shown:



Here the local queue has been detached from the queue manager, so the empty queue category has been hidden. It will be replaced as soon as it is required. Categories cannot be detached from their parents. The ability to separate out the trees into components makes diagramming of particular aspects of a WMQE network simple. Re-attaching the child to its parent is simple. Drag the child so that the mouse pointer is over the parent and release. The target area for reattachment includes any visible, attached childrten of the parent. If the parent is still attached to *its* parent, then any part of the gand-parent will accept the reattachment.

2.2 Editing

The diagram can be edited by using the popup menus and the edit menu on the menu bar. All WMQE resource types can be manipulated. There are two modes for editing - 'normal' and 'expert'. MQeDesigner always starts in 'normal' editing mode. In 'normal' editing mode some model editing actions are dependant upon the resources available. This constrains the editor to making only valid WMQE configurations. The constraints are as follows:

- To create a 'normal' connection definition, there must be a target listener. The GUI only allows you to create a 'normal' connection definition on a queue manager by selecting an existing listener on another queue manager.
- To create a 'via' connection definition there must be a normal connection definition, and a suitable queue manager as target. Suitable queue managers are those for which there is currently no connection definition. Via connections can be used as targets for via connections.
- To create a remote queue there must be both an ultimate local queue (on another queue manager) and a connection definition ('normal' or 'via') to the queue manager owning the ultimate local queue. The GUI will then allow you to select the local queue, or any of its aliases.

- To create a home server queue there must be a suitable store and forward queue. A suitable store and forward queue is one that is reachable (ie there is a connection or set of connections to reach the S&fQueue), that has an appropriate queue manager entry (same name as queue manager on which home server queue is being created), and that does not already have a home server queue on the current queue manager.

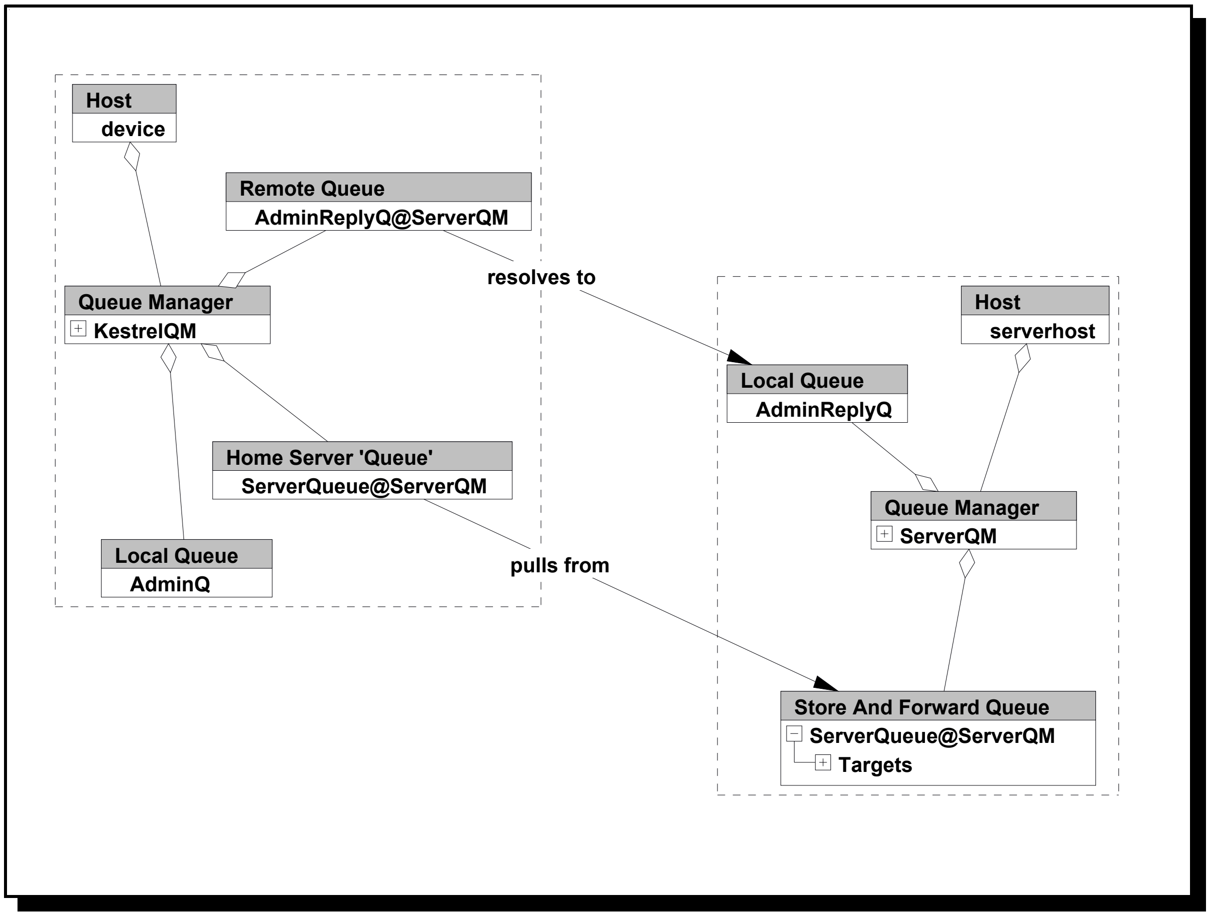
I strongly recommend using 'normal' edit mode whenever possible, as an aid to creating valid WMQE topologies. Some valid topologies cannot be created using the expert menus, and then the 'expert' mode can be used. In expert mode there are no constraints upon the data entered, and so errors are likely - from simple spelling mistakes to subtly broken topologies. The expert mode can be enabled/disabled using the 'Edit' menu.

2.3 Showing and Hiding Glyphs

Individual glyphs can be hidden by using the popup menu. To show a glyph, use the popup menu of its parent to show all children. Host glyphs are children of the windo, so use the main menu to show all hidden glyphs. Hiding a glyp hides all its attached children, but not its detached children.

2.4 Showing ‘Boundaries

Sometimes adding a boundary to a diagam helps to make it clear where one collection of resources starts and another ends. An item in the popup menu (glyph/show boundary), adds a dotted rectangle around all the children of a component. The boundary will not be shown if there are no detached children. It helps to have the owning component (the one whose boundary is being shown) in a prominent position. It sounds more complicated than it really is. Play with it:



2.5 Saving to File

Models can be saved as an xml representation (File-->Save Model). The xml model files can, of course, be loaded (File-->Load Model). Note that loading a file adds to the current model. To load a file into a clean environment use the 'File-->New Model' menu item.

Model Diagrams can only be exported as bitmaps; use the 'File-->Save Diagram' menu item.

2.6 Help

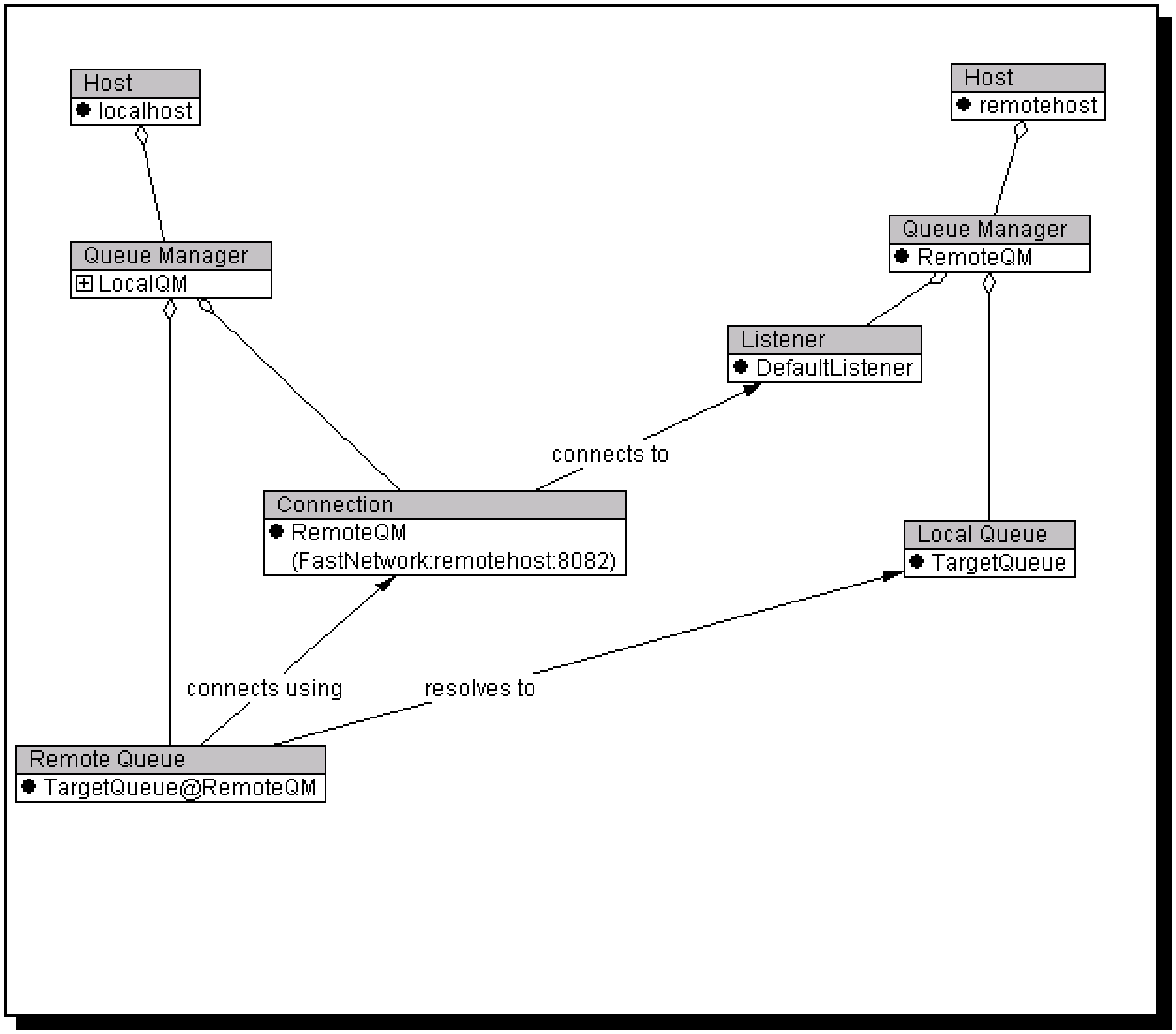
Context sensive help is available for each WMQE resource. Choose help on the popup menu (if the help window is not already visible), or click on the resource if the help windo is visible.

2.7 Default Relationships

Some relationships between resources are very useful, and so these are displayed automatically when relevant. These are called the 'Default Relationships' and are:

- The connection or connection alias used by a remote queue
- The connection used by a 'via' connection
- The listener to which a connection refers
- The store and forward queue that a home server queue pulls from

Some of these can be seen on the following diagram:



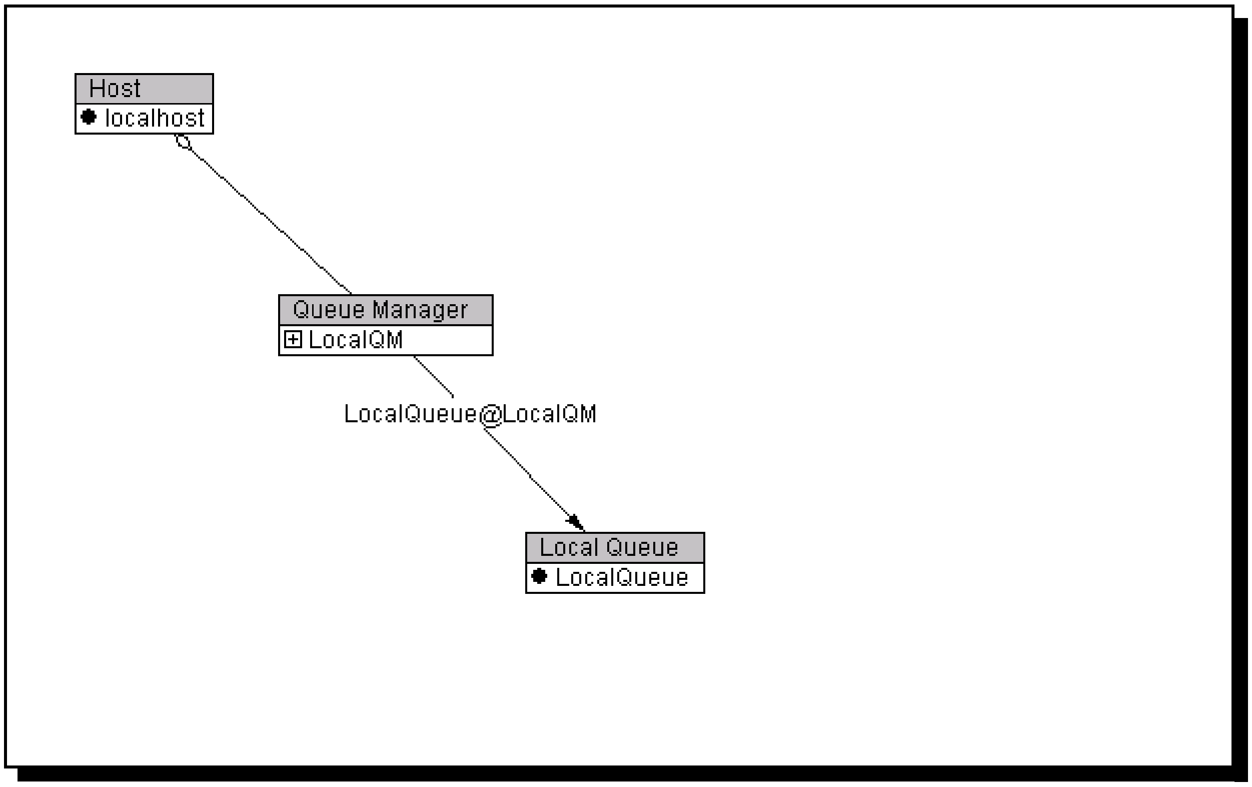
Default relationships are enabled/disabled globally by a menu option (View --> Show Default Relationships).

3.0 Message Resolution

The designer can show how WMQE will resolve message routing. There are three reource type that will allow message resolution:

- QueueManager: you will be asked to provide the names of the target queue and target queue manager. This is the most 'natural' of the message resolution as it mimics the standard putMessage() function available on the WMQE API.
- Queue: The names of the target queue and target queue manager are entirely defined by the queue.
- QueueManagerEntry (on store and forward queue): The name of the target queue manager is the name of the queue manager entry. You will be asked to specify the target queue name.

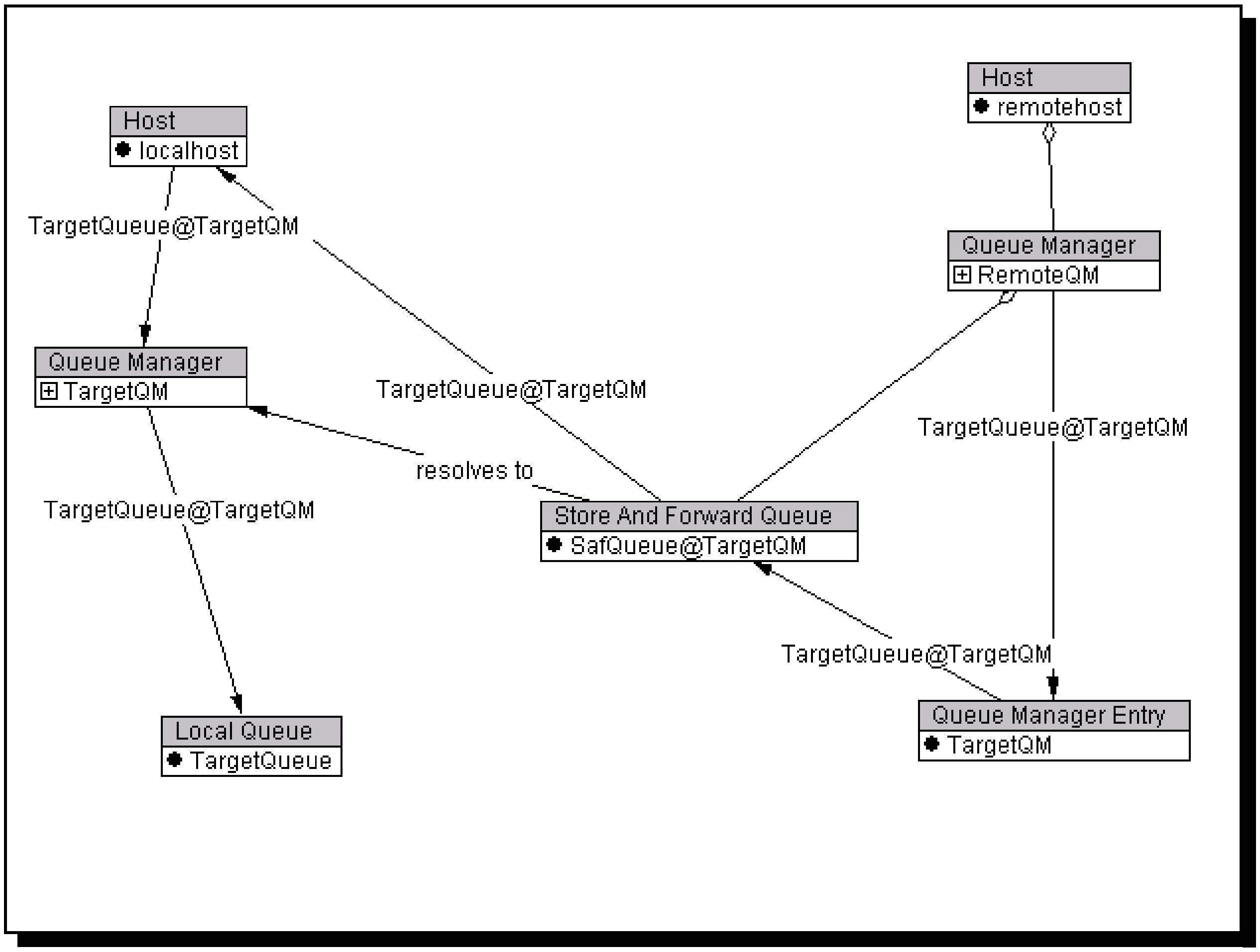
The message route resolution is invoked by using the popup menu on the resource. Note that the resource must be detached from its parent in order be the origin of a message resolution. The following shows a simple local message put:



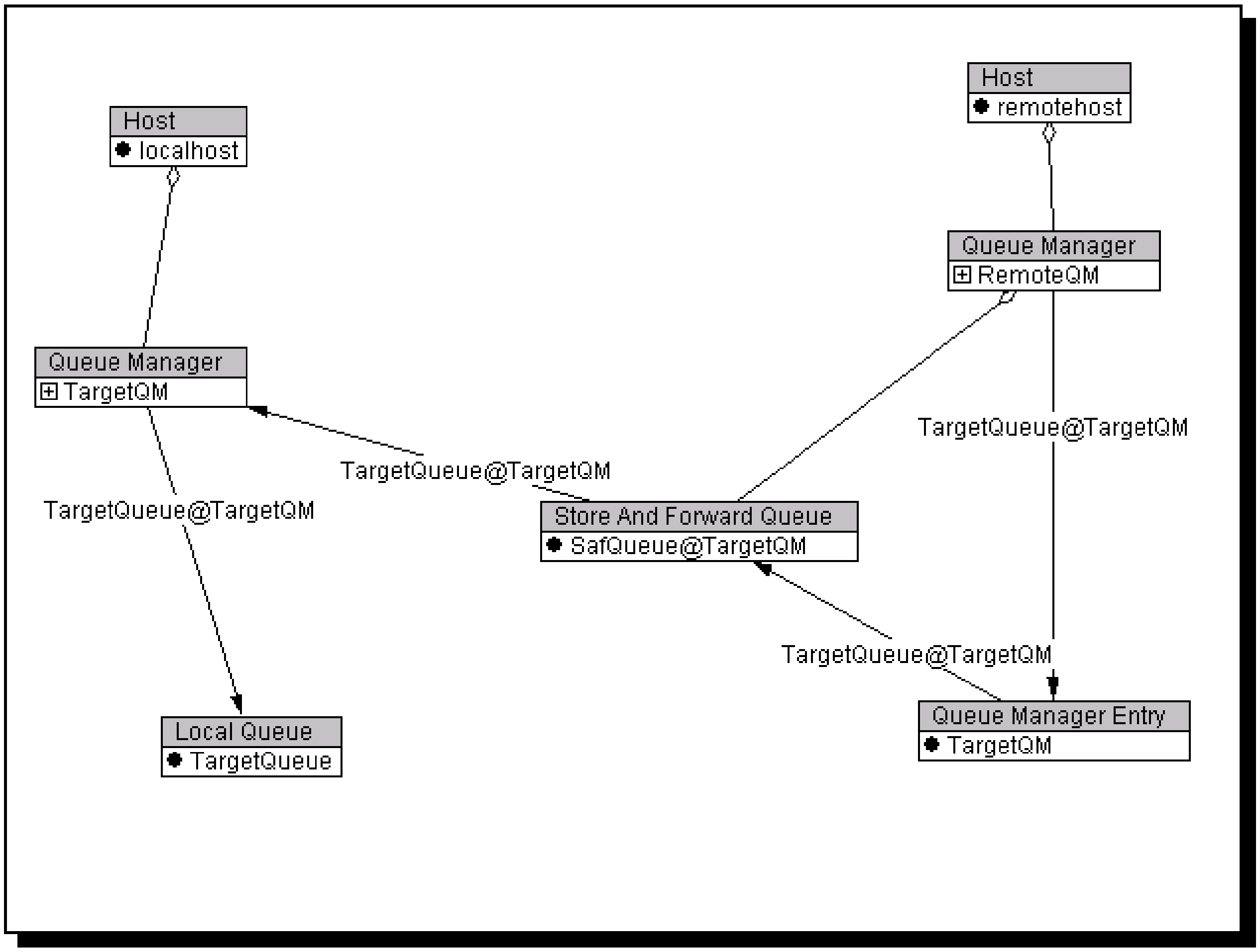
The message route is shown with an arrow labelled with the message route name. The arrow indicates the direction in which the message flows, and the colour of the label indicates that the message is being 'pushed'. The message route can be hidden using the 'hide message route' option on the popup menu.

3.1 Filtering Message Route Resolutions

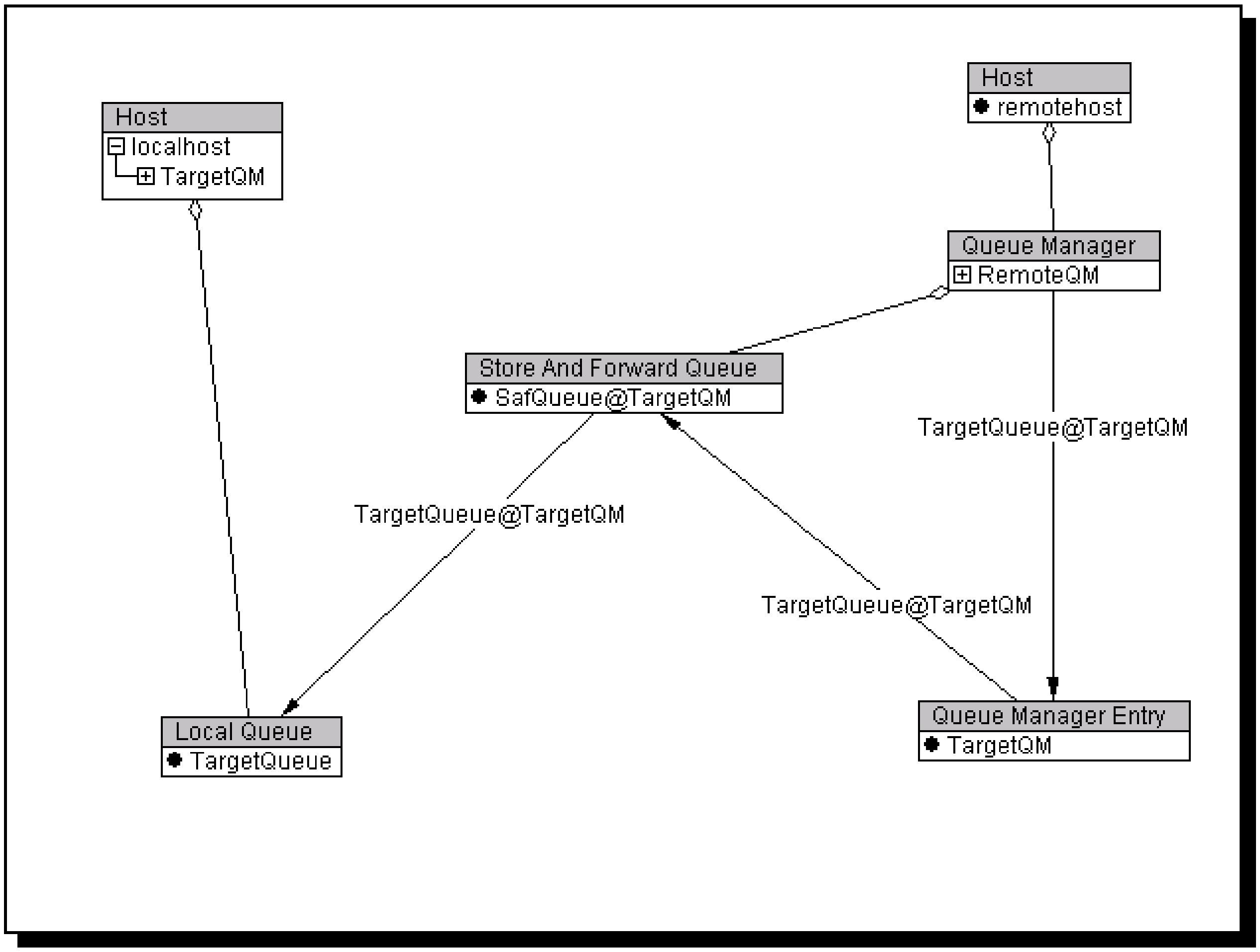
Message route resolutions can get quite complex. Here is a route resolution using a pushing store and forward queue:



It is quite a busy diagram. Forget the logic of the route for a momment, and we can filter out some details. For a start, resolution via localhost is not really of interest. There is a menu item under the view menu on the main menu bar to include/exclude hosts from the message resolution:

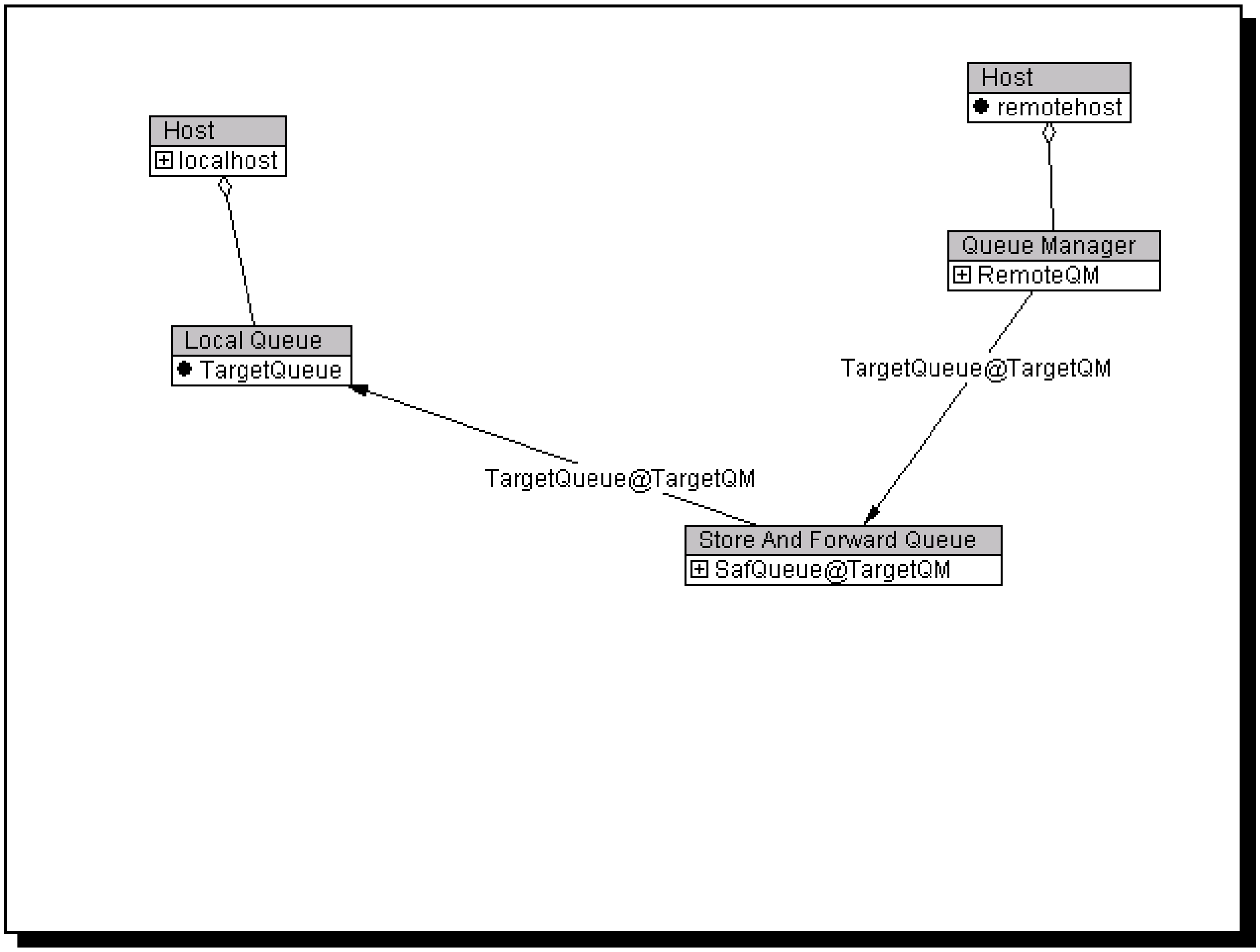


There is a similar menu item to remove queue managers from the route resolution, but there is also a more general way to do this: a route will not be shown via any resource that is still 'attached' to its parent. So if we drag the queue manager 'TargetQM' back into its host we get:



The advantage of dragging the queue manager is its ease and selectivity - route resolution can be 'turned off' for individual queue managers. The disadvantage is that it can be unclear which queue manager on a host a queue or connection belongs to. With only one queue manager on host 127.0.0.2 there is no possibility for confusion here.

Filtering of message resolution can provide a much simplified view:



4.0 Message Routes

There is a higher level construct that we can create from the raw WMQE components, and these are called 'Message Routes'. Each message path shows the ability of messages to move from one queue manager to another. There are three kinds of message path:

- Single message path: These can move messages for a defined queue on a defined queue manager. For example targetQueue@targetQM. These paths are constructed from a remote queue (eg targetQueue@targetQM), a connection definition (eg targetQM) on the same queue manager, and a receiving queue on another queue manager. The receiving queue must be a queue that will accept messages from the sending queue. It may be a local or remote queue (eg targetQueue@targetQM) or a store and forward queue with an aprpriate queue manager entry (eg targetQM). If all of these WMQE resources are present then together they form a single message path. This path will move messages targeted at one queue (eg targetQueue) and one queue manager (eg targetQM).
- Multiple message path: These can move messages for a any queue on one or more defined queue managers. For example *@targetQM. These paths are constructed from a pushing store and forward queue (eg safq@targetQM, with a queueManagerEntry called targetQM) a connection definition (eg targetQM) on the same queue manager, and one or more receiving queues. The receiving queue might be a store and forward queue with entries for the destination queue manager(s) defined by the source queue, in which case the route wil move all messages for the destination queue manager(s). If there is no s&f receiving queue, then the messages will be put to any of the queues accepting messages for the specified queue managers. If all of these WMQE resources are present then together they form a multiple message path.
- Pulling Message Route: A pulling message path allows messages for any queue on a specified queue manager to be pulled to that queue manager. A pulling message path consists of a home server queue (eg hsq@remoteQM), a cnnection definition (eg remoteQM) and a store and forward queue at remoteQM with an apropriate queueManagerEntry.

Note that these message paths do not exist as single entities within the WMQE system, but are higher level posibilities iplied by the presence of certain resources. If the resources are present then we can talk about the path existing. If any of the required resources is missing, or incorrectly configured then the path does not exist.

Message paths are only considered for asynchronous message movements.

Message paths are calculated dynamically by the designer tool.

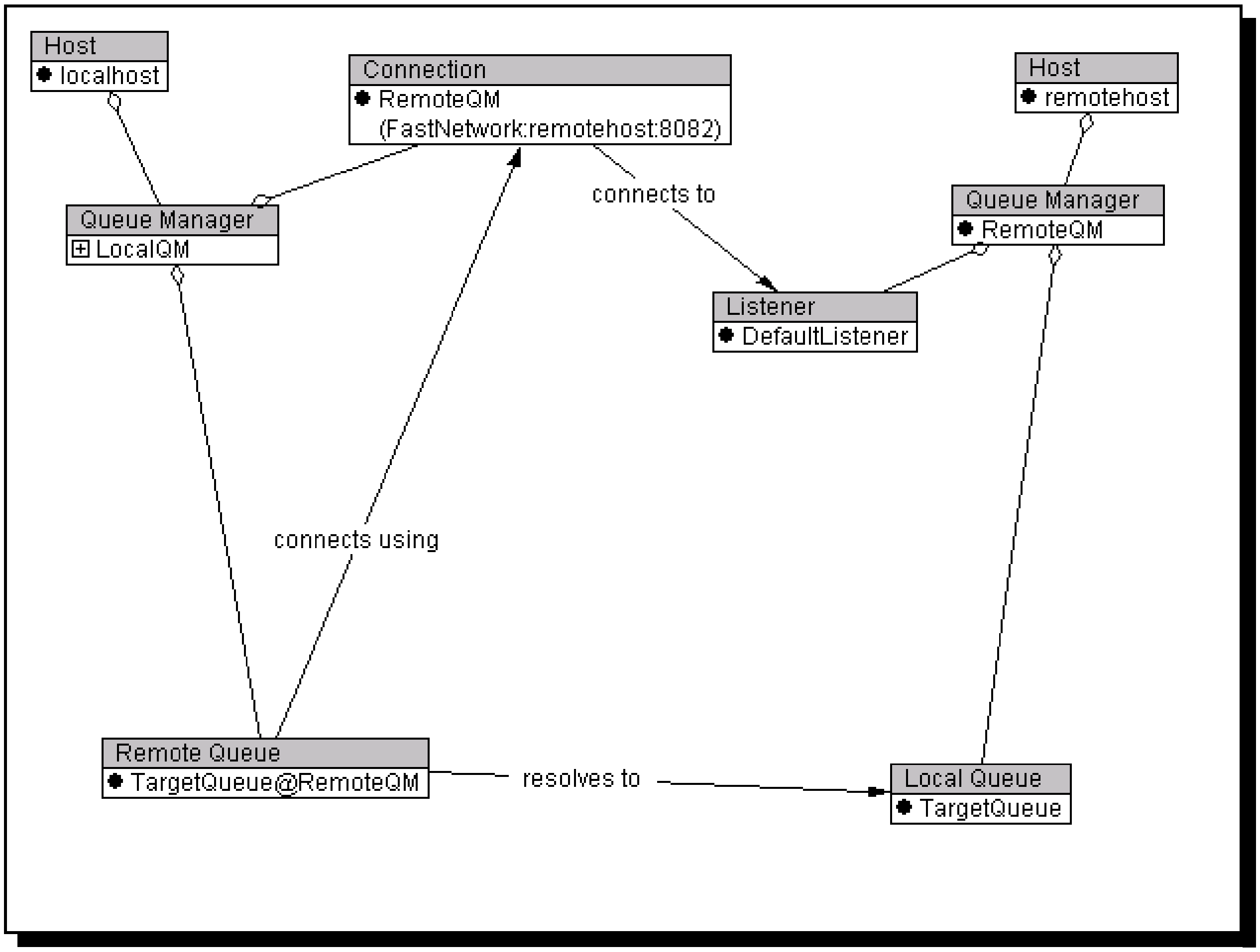
4.1 Business Purpose

Message routes show a high level view of the underlying components. The higher level constructs often relate directly to a business purpose. Message Routes and queues can be labelled with a business purpose in order to reflect this. Since each message route is derived from a source queue, the label for the message route is the same as that of the source queue. (Also useful as message routes are not persistent, but derived from the lower level components so the business purpose can be stored with the source queue). The purpose can be set either from the source queue or from the message route. If a purpose has been set for a source queue, then the user can choose to show either the route/queue name or the business purpose. This decision affects both the source queue and the message route.

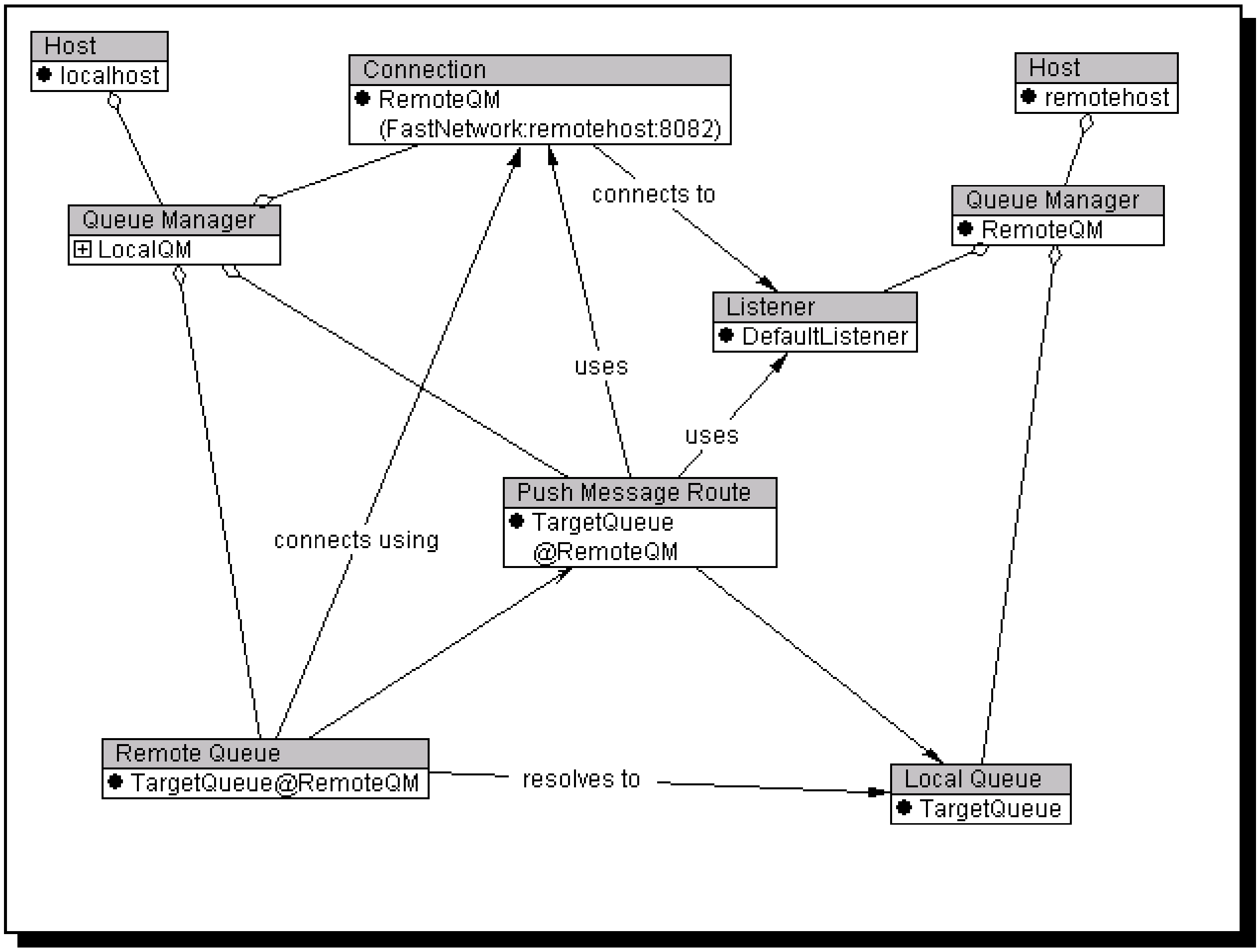
Message route type	source queue
single push	Remote Queue
multiple push	Store and Forward Queue
pull	Home Server Queue

4.2 Single Message Route

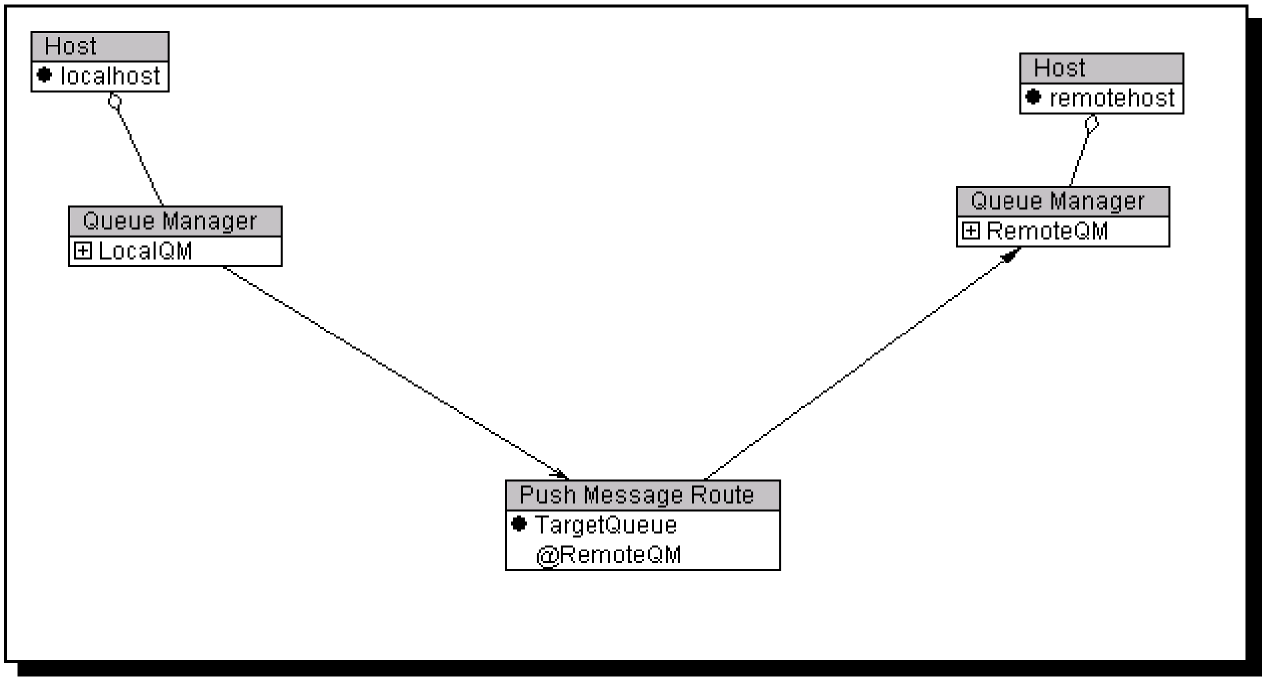
These can move messages for a defined queue on a defined queue manager. For example targetQueue@targetQM. These paths are constructed from a remote queue (eg targetQueue@targetQM) and a connection definition (eg targetQM) on the same queue manager. If both of these WMQE resources are present then together they form a single message path. Here is a diagram of the components of a simple message path:



The components imply a simple message path, which appears as below:



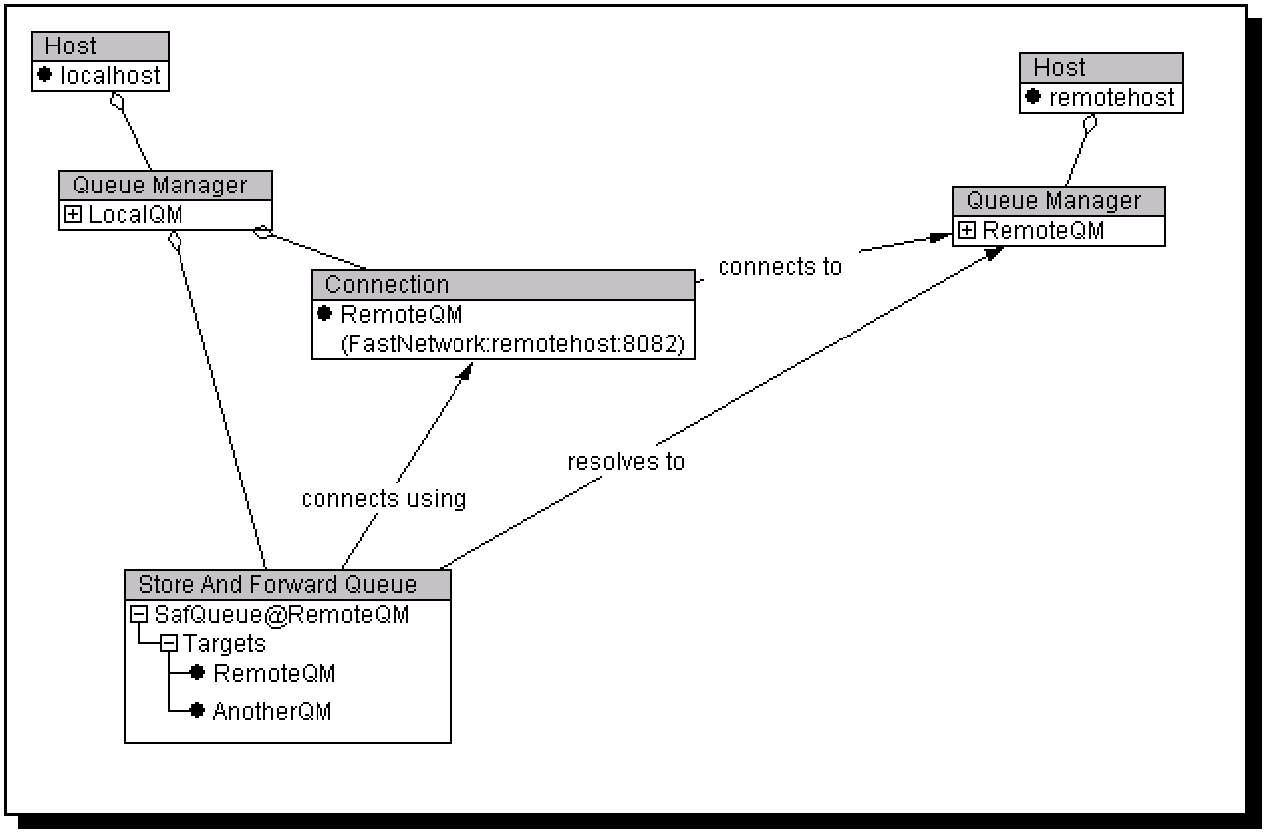
The path is linked to its component parts, but this can complicate the picture. Far better to reattach the component parts to their parents and see the message route:



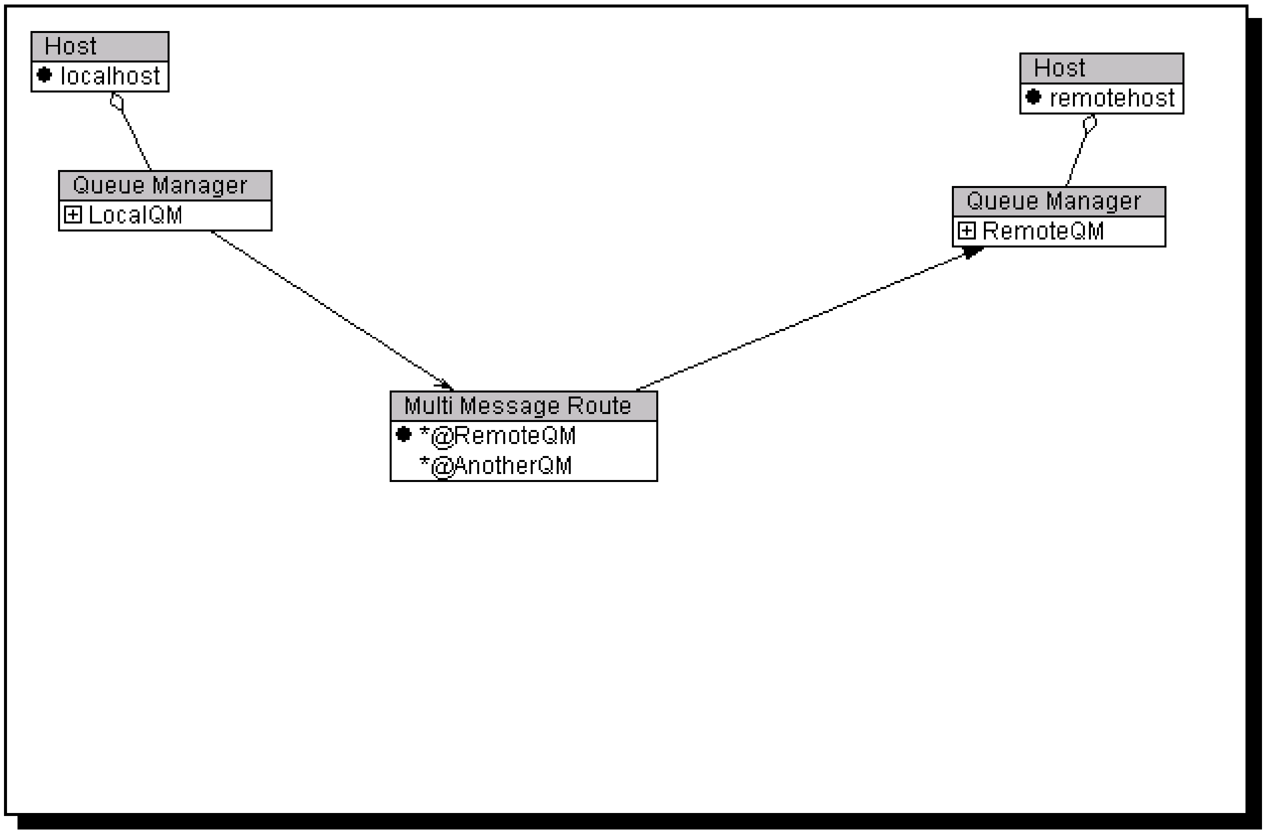
Future examples will not bother with the complex view showing the relationship between the path and its components.

4.3 Multiple Message Route

These can move messages for a any queue on one or more defined queue managers. For example *@targetQM. These paths are constructed from a pushing store and forward queue (eg safq@targetQM, with a queueanagerEntry called targetQM) and a connection definition (eg targetQM) on the same queue manager. If both of these WMQE resources are present then together they form a multiple message path. Below is a diagram showing the components of a multiple path:

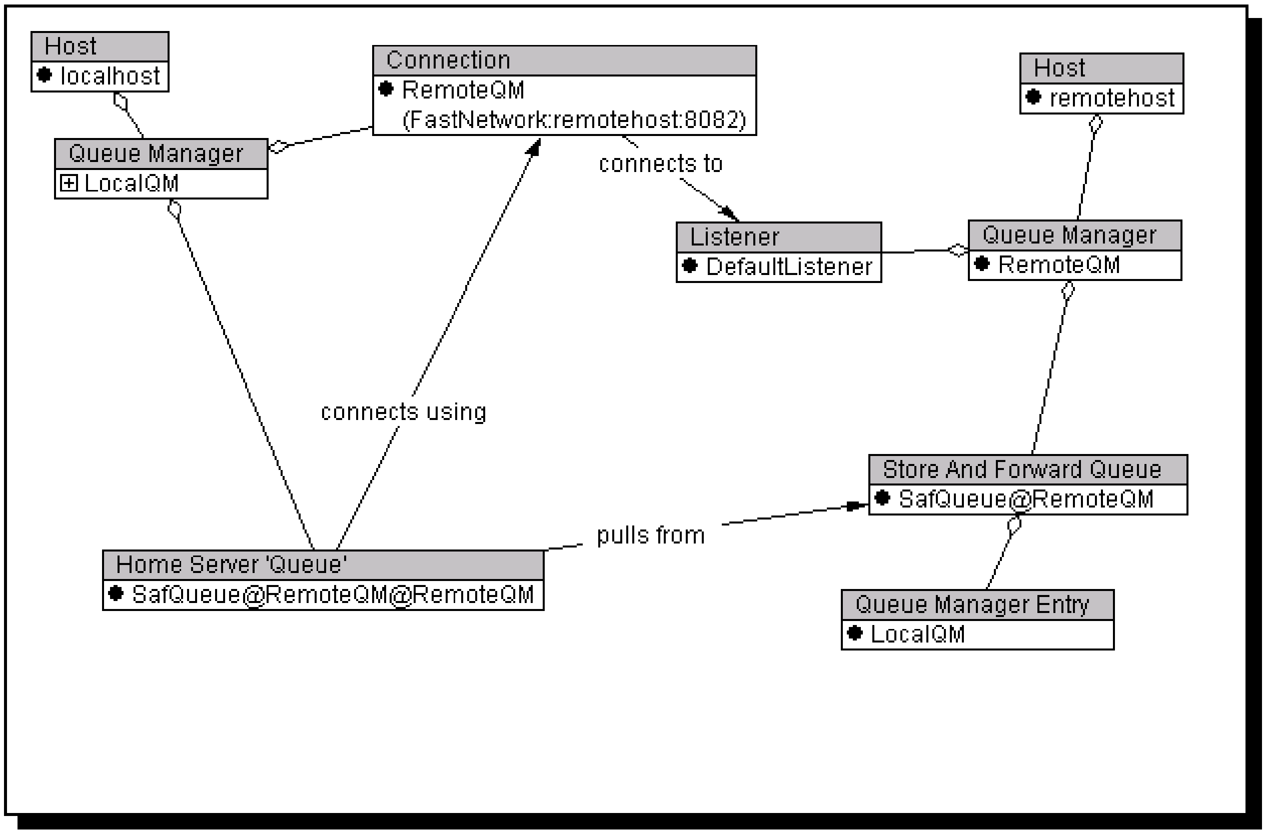


The multiple path itself simplifies the diagram considerably:

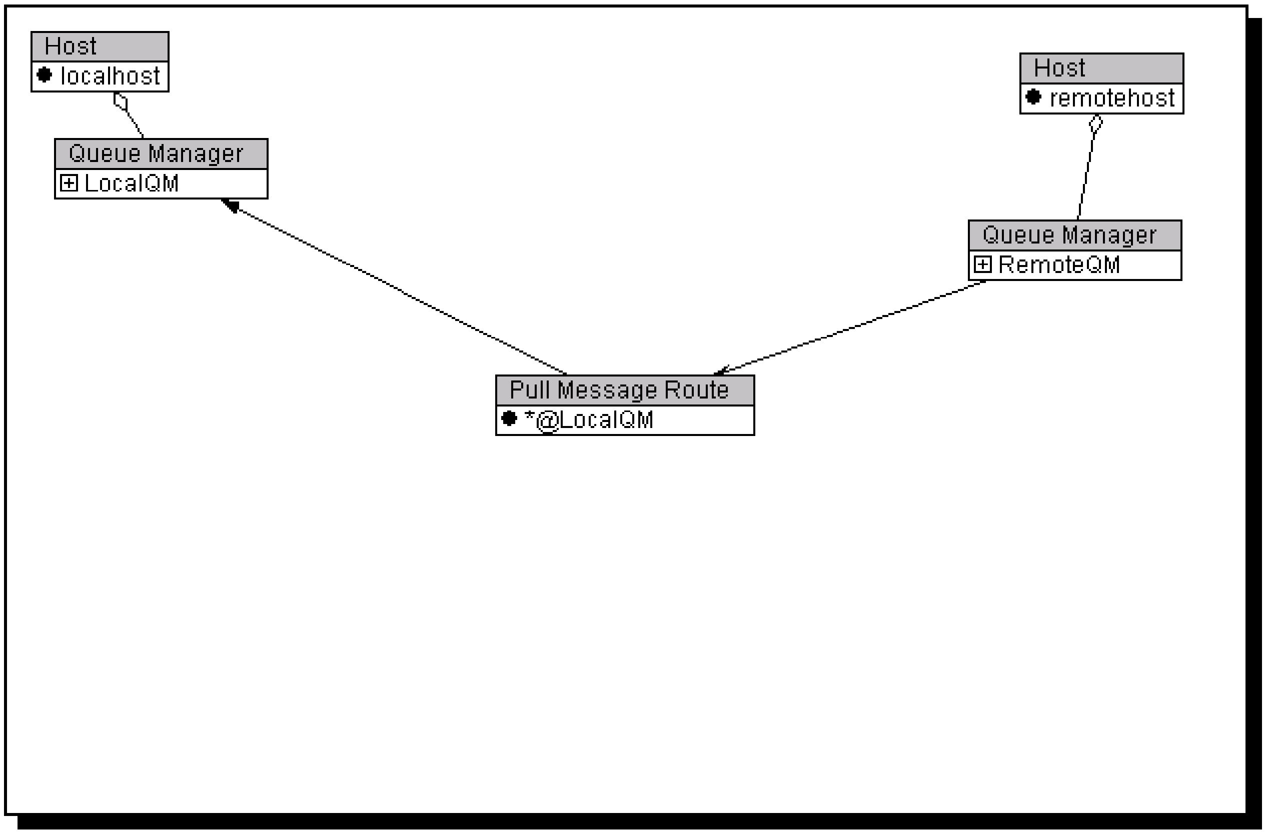


4.4 Pulling Message Route

A pulling message path allows messages for any queue on a specified queue manager to be pulled to that queue manager. A pulling message path consists of a home server queue (eg hsq@remoteQM), a cnnection definition (eg remoteQM) and a store and forward queue at remoteQM with an apropriate queueManagerEntry. The diagram below shows the constituents of a pulling path:

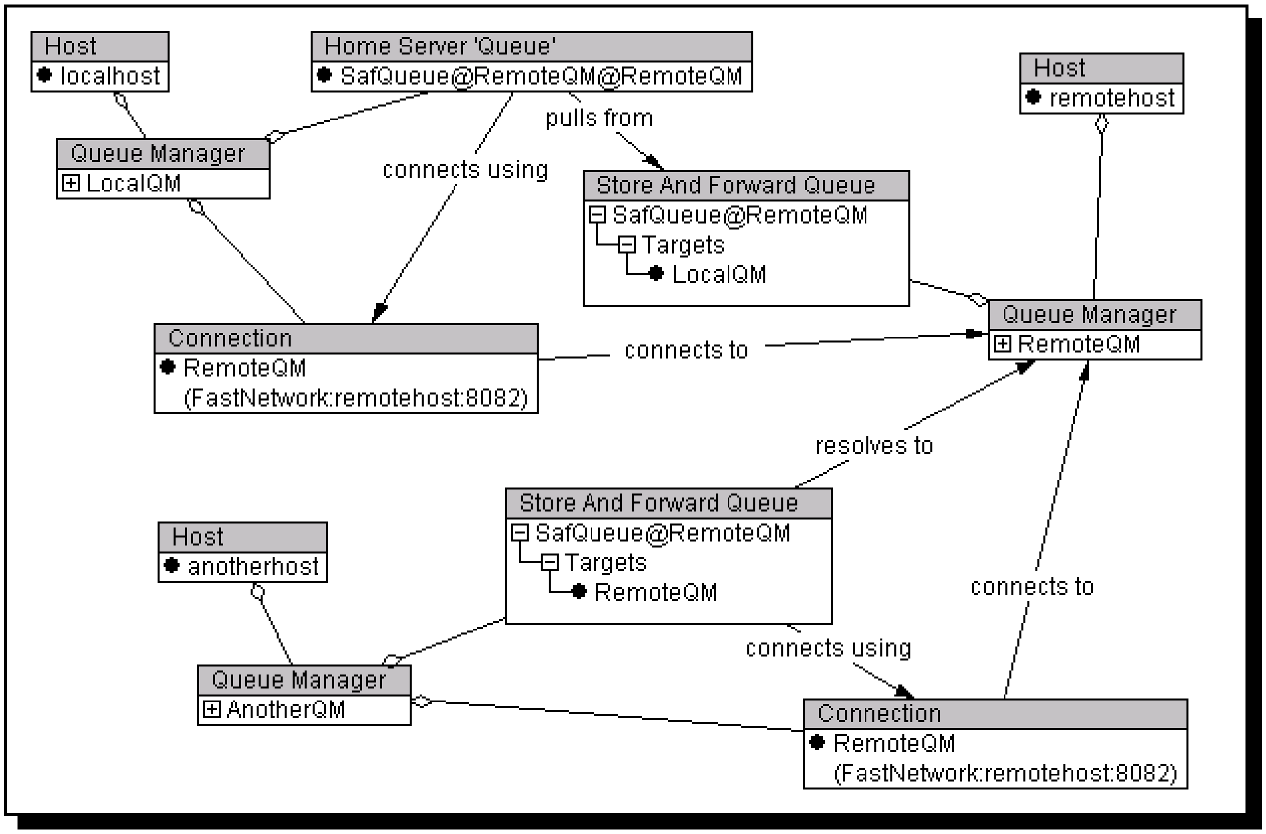


And the maching path representation:

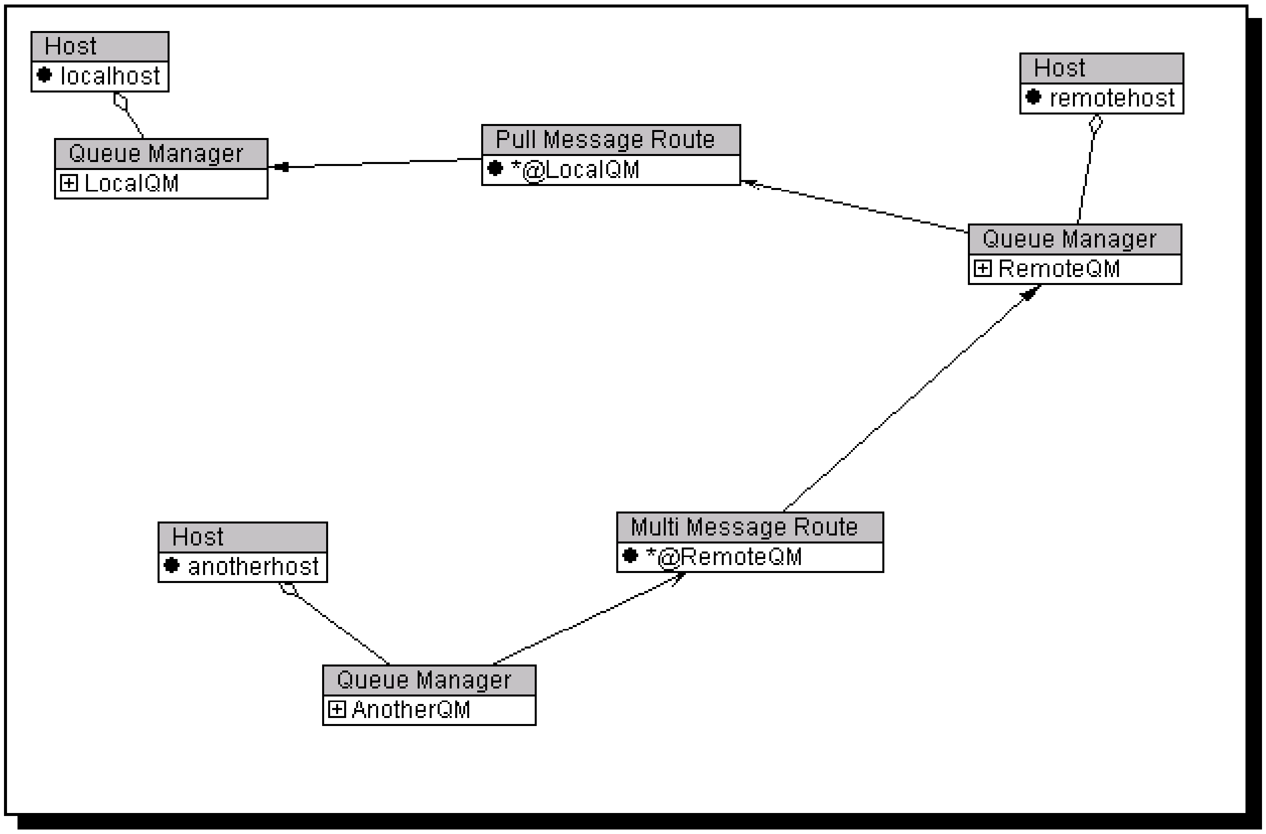


4.5 Example of a Message Route

The folowing diagram shows how message paths can be simplified:



Here we have a system where messages can move from AnotherQM to LocalQM in two hops. The first hop is a mulitple push path (using S&F queue), and the second hop is a pulling path. To see the logic more clearly we can show the paths themselves:



•

5.0 Installation

The tool can be installed in either 'Designer' or 'Investigator' modes. The only difference is in the classes available on the classpath.

5.1 Designer Mode

In 'designer' mode, the tool is independant of WMQE, and requires only that Java is available (118 or higher).

- Choose (or create) a directory for the tool.
- Ensure that the directory contains the following file:
  - designer.jar
- Ensure that java executables are available on the path, and that the java class libraries are available on the classpath. If java has been installed on your machine then these will be set up correctly.
- Ensure that the classpath includes designer.jar eg on windows:
  - classpath=%classpath%;.\designer.jar
- start java using the class designer.MQeDesigner as an argument:
  - java designer.MQeDesigner

You could choose to create a batch file, or to edit the one supplied.

5.2 Investigator Mode

For 'investigator' mode you will need to perform the above actions, but you will also need WMQE classes and an extra jar file in the classpath:

- Ensure that WMQE is on the classpath; for correct operation the MQeMQ bridge class files must be present.
- Ensure that the file investigator.jar is on the classpath
- Start java with the same command as above.

The tool will then allow the 'Administer' action on the popup menu for queue managers **if they have at least one declared listener**. Only 'server' queue managers can be administered (currently). The declared listener *must* match an existing listener on the server queue manager. Selecting the runtime administrator will enable live administration of the queue manager. Resources can be read from the real queue manager. All the normal actions available in designer mode are also avalable in investigator mode. Please note that the administration facilities are based on WMQE v1.x functionality, and so listeners cannot be added/removed/queried remotely.

Please note that WMQE v1.x models Queue Manager Aliases internally as aliases on connection definitions. Creating a queue manager alias in MQeDesigner without first creating the connection definition will succeed in the Designer but fail to be reflected in the 'real' system. So to create an alias for RemoteQM, it is neccesary first to create a connection definition to RemoteQM.

MQ Resources cannot be administrered (maybe one day.....).

6.0 MQeXML

Models are stored in a version of XML called MQeXML. The format can easily be deduced from the files themselves, or the following DTD (data type definition) can be used. Note that the dtd does not (yet) contain attribute definitions

```
<?xml version='1.0' encoding="UTF-8"?>

<!ELEMENT Bridge (BridgeQueueManager)>
<!ATTLIST Bridge name CDATA #IMPLIED >

<!ELEMENT BridgeConnection (BridgeListener)>
<!ATTLIST BridgeConnection name CDATA #IMPLIED>

<!ELEMENT BridgeListener EMPTY>
<!ATTLIST BridgeListener name CDATA #IMPLIED>

<!ELEMENT BridgeQueue (QueueAlias)>
<!ATTLIST BridgeQueue name CDATA #IMPLIED>
<!ATTLIST BridgeQueue queueManagerName CDATA #IMPLIED>

<!ELEMENT BridgeQueueManager (BridgeConnection)>
<!ATTLIST BridgeQueueManager name CDATA #IMPLIED>

<!ELEMENT Connection (ConnectionAlias,ConnectionAdapter)>
<!ATTLIST Connection name CDATA #IMPLIED>

<!ELEMENT ConnectionAdapter EMPTY>
<!ATTLIST ConnectionAdapter name CDATA #IMPLIED>

<!ELEMENT ConnectionAlias EMPTY>
<!ATTLIST ConnectionAlias name CDATA #IMPLIED>

<!ELEMENT HomeServerQueue (QueueAlias)>
<!ATTLIST HomeServerQueue name CDATA #IMPLIED>
<!ATTLIST HomeServerQueue queueManagerName CDATA #IMPLIED>

<!ELEMENT Host (QueueManager,MQQueueManager)>
<!ATTLIST Host name CDATA #IMPLIED>

<!ELEMENT LocalQueue (QueueAlias)>
<!ATTLIST LocalQueue name CDATA #IMPLIED>
<!ATTLIST LocalQueue queueManagerName CDATA #IMPLIED>

<!ELEMENT MQChanel EMPTY>
<!ATTLIST MQChanel name CDATA #IMPLIED>

<!ELEMENT MQLocalQueue EMPTY>
<!ATTLIST MQLocalQueue name CDATA #IMPLIED>
<!ATTLIST MQLocalQueue queueManagerName CDATA #IMPLIED>

<!ELEMENT MQQueueManager
   (MQLocalQueue,MQRemoteQueue,MQXmitQueue,MQChanel)>
<!ATTLIST MQQueueManager name CDATA #IMPLIED>

<!ELEMENT MQRemoteQueue EMPTY>
<!ATTLIST MQRemoteQueue name CDATA #IMPLIED>
<!ATTLIST MQRemoteQueue queueManagerName CDATA #IMPLIED>

<!ELEMENT MQXmitQueue EMPTY>
<!ATTLIST MQXmitQueue name CDATA #IMPLIED>
```

```
<!ATTLIST MQXmitQueue queueManagerName CDATA #IMPLIED>

<!ELEMENT MQeModel (Host)>

<!ELEMENT QueueAlias EMPTY>
<!ATTLIST QueueAlias name CDATA #IMPLIED>

<!ELEMENT QueueManager
   (LocalQueue,RemoteQueue,StoreAndForwardQueue,HomeServerQueue,BridgeQueue,Connection,Bridge)>
<!ATTLIST QueueManager adapter CDATA #IMPLIED>
<!ATTLIST QueueManager channel CDATA #IMPLIED>
<!ATTLIST QueueManager name CDATA #IMPLIED>
<!ATTLIST QueueManager port CDATA #IMPLIED>

<!ELEMENT QueueManagerEntry EMPTY>
<!ATTLIST QueueManagerEntry name CDATA #IMPLIED>

<!ELEMENT RemoteQueue (QueueAlias)>
<!ATTLIST RemoteQueue name CDATA #IMPLIED>
<!ATTLIST RemoteQueue queueManagerName CDATA #IMPLIED>

<!ELEMENT StoreAndForwardQueue (QueueAlias,QueueManagerEntry)>
<!ATTLIST StoreAndForwardQueue name CDATA #IMPLIED>
<!ATTLIST StoreAndForwardQueue queueManagerName CDATA #IMPLIED>
```

It is entirely possible to edit the xml files to create different models. Indeed this could be a powerful technique for creating variations from templates. Direct editing of the XML circumvents the consistency checking applied by the GUI. Under some circumstances this may be desirable.