# THE MESSAGE BUS COMMUNICATION SYSTEM

The server now uses Message Bus system to transfer requests to the services. When an HTTP request is received, it is allocated a unique ID and put on a queue. The server then runs a loop that takes the next message off the queue each (very short) time interval. Each message can be for a specific service or for a service type and gets matched with any service that will accept to process it. If a service is currently busy, then the message can be skipped and the next one processed. This part of the system also links in with the fault system, where if a message is repeatedly refused, it is removed and some details are added to the server fault tree.

## Point-to-Point Request

A service can make a specific request for another service to do work, by including the service's ID in the message path. This is in fact the default setup and most messages are passed this way. For example, if there is a service of type 'sType', with a uuid of 's1', then for a point-to-point request, the client would use a service handle similar to:

```
<Handle><U>http://123.4.5.6:8888</U><S>s1</S></Handle>
```

This would also allow the service password to be used, because the service ID is known and so it in fact would have to be included in the method info. If the service has then registered for point-to-point communications, or has not specified any `Work_Info` in its admin document (see the licasAdminGuide document), then point-to-point is assumed.

## Subscribe Request

Alternatively, a client can make a request to a service type. If a client is asking for a service type to carry out some work, then the service handle might look like:

```
<Handle><U>http://123.4.5.6:8888</U><S>sType</S></Handle>
```

If a type is encountered in the service path, 'sType' in this case, then all services of that type are retrieved and if they have registered, they can be asked to do the work. To register requires that the service is initialised with an admin document that includes a `Work_Info` section and that the `Protocol` is set to either '1' or '2', or 'Subscribe' or 'Both'. The work info can also be hard-coded into the service's initialisation, for example:

```
public MyService extends Service
{
   /** Ctreate a new instance of MyService */
   public MyService()
   {
      super();
```

```
      WorkInfo workInfo = new WorkInfo();
      workInfo.workProtocol = WorkInfo.SUBSCRIBE;
      serviceAdmin.setWorkInfo(workInfo);
   }
}
```

Or the protocol can be set to 'WorkInfo.BOTH' and the server will understand that this service has subscribed to receive requests for its service type.

## Publics - Subscribe

Another implementation is a different publish-subscribe framework. It would allow testing of data flow and basic functionality before maybe considering adding a more complete solution. Client services can subscribe for work as defined by a topic and/or list of keywords. The request is stored under the client service's URI address. Other services can publish the ability to do work, again using a `WorkInfo` object. This can include contract details, for example. Every time a client subscribes, a lookup is done immediately for that client only, and work details are returned if they match. Every time a service publishes its ability to do work, the server looks for matching subscribed requests and sends out the work details to any that match. Matching means to match over the topic and/or any keyword. Each request should therefore have a completely different set of keywords. A service can publish the same work offer any number of times however, when only the first submission is stored, but the work details are sent to any requesting client each time a publication is made.