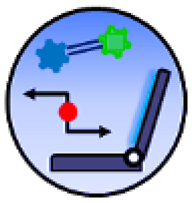


Licas All-in-One Scientific GUI

Version 1.2

[A guide to using the GUI application for viewing or testing networks running on the licas system]

Kieran Greer,
Email: kgreer@distributedcomputingsystems.co.uk.
http://distributedcomputingsystems.co.uk/licas.html

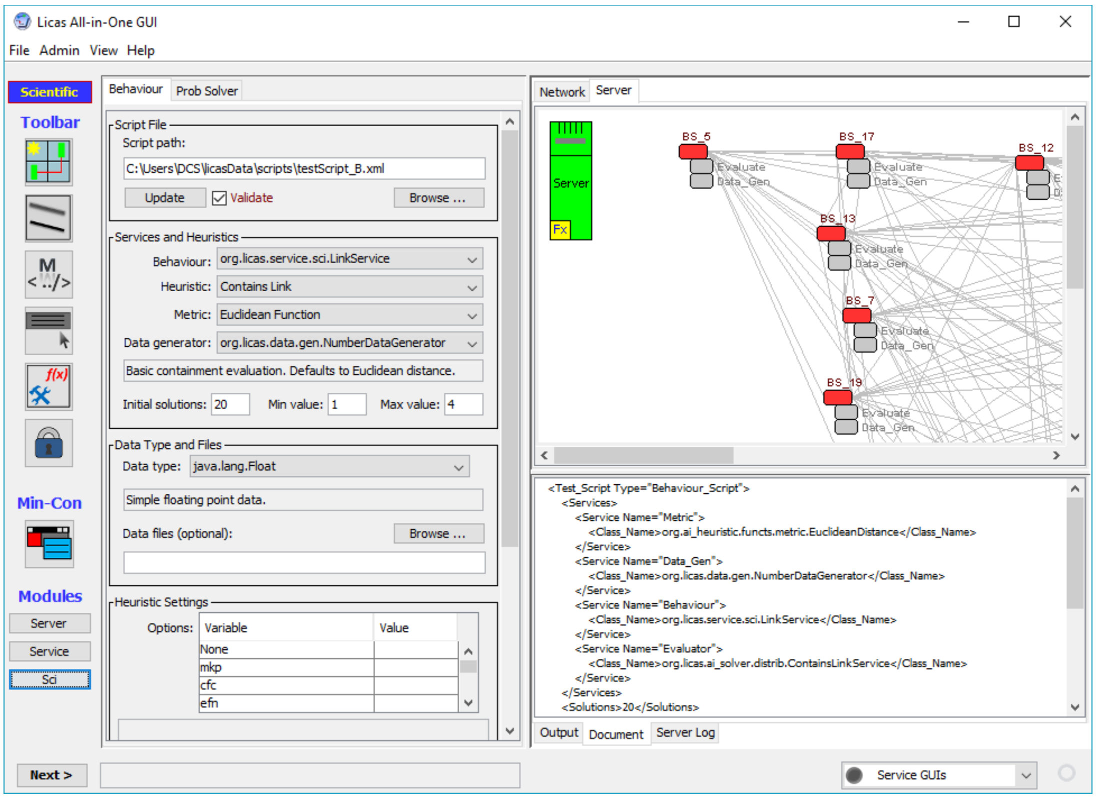
Table of Contents

1 Introduction

This guide describes the free All-in-One GUI application that comes with the licas server system and how to use the scientific functionality. You need to read the 'licasGui' document first for all background information. This document then describes the scientific panels and how to use them.

2 Behaviour Panel

This panel can be used to run or test your own algorithms that might perform some sort of behaviour or self-organisation. Figure 1 shows what the behaviour panel looks like. There is also a `Behaviour Mediator` service that can be run. If it is available, then both the behaviour and problem solving processes will send their information to it, for display purposes. The behaviour mediator is described in more detail in the 'licasService' document.



Script File

Script path: C:\Users\DCS\licasData\scripts\testScript_B.xml

☑ Validate

Services and Heuristics

Behaviour: org.licas.service.sci.LinkService

Heuristic: Contains Link

Metric: Euclidean Function

Data generator: org.licas.data.gen.NumberDataGenerator

Basic containment evaluation. Defaults to Euclidean distance.

Initial solutions: 20 Min value: 1 Max value: 4

Data Type and Files

Data type: java.lang.Float

Simple floating point data.

Data files (optional):

Heuristic Settings

Options:

Variable	Value
None	
mkp	
cfc	
efn	

```
<Test_Script Type="Behaviour_Script">
  <Services>
    <Service Name="Metric">
      <Class_Name>org.ai_heuristic.functs.metric.EuclideanDistance</Class_Name>
    </Service>
    <Service Name="Data_Gen">
      <Class_Name>org.licas.data.gen.NumberDataGenerator</Class_Name>
    </Service>
    <Service Name="Behaviour">
      <Class_Name>org.licas.service.sci.LinkService</Class_Name>
    </Service>
    <Service Name="Evaluator">
      <Class_Name>org.licas.ai_solver.distrib.ContainsLinkService</Class_Name>
    </Service>
  </Services>
  <Solutions>20</Solutions>
```

Figure 1: Behaviour Panel

The behaviour panel is somewhat configurable and should allow you to load in different algorithm types or evaluation functions etc. You can also configure each service with a data file, or add a linking service, etc. The following sections will describe more clearly what can be configured. The structure of this panel is similar to the problem solver panel (section 3), but where the problem-solver can also be centralised, behaviours are always distributed.

2.1 Script File Group Box

If you have saved a script previously, you can load it in again from the `Script File` set of options. The script will be displayed in the `Document` tab, from where it can also be edited manually. Use this to browse to an existing script file and load it into the `Documents` text area. A behaviour test does not require a script as much as the problem solver does, but it can store the configuration more easily, allowing you to quickly setup the test environment. Any script that is loaded can then be updated when you enter different data values.

2.2 Create Services Section

The `Create Services` section allows you to select a set of service classes, to use to create the behaviour service. The lists of services can be updated through the `Service Factory`, as described in other documents. Each default class that is added should now have a text description. A class name is not very descriptive, so a text description helps to define what the class is about. Whenever a class type is selected, its description should also be displayed in the related text box. To create a behaviour network using the default classes, you need to define three types of class. These are:

- **Behaviour Class:** this is the class to load as the behaviour service itself. There is a default one in the licas package that contains its own control loop, to periodically evaluate some value and try to link based on the evaluation result. This is represented by the classname itself.
- **Heuristic Class:** the behaviour service can be loaded with an evaluator service. This is passed the value to evaluate and returns the result. While the behaviour controls the overall process, it still needs a service to measure or evaluate the information that it receives. This could be a collective evaluation over all available data, as in the default linking service heuristics. So changing the heuristic will change the behaviour result.
- **Metric Class:** the heuristic service can be loaded with a specific evaluator or metric service. This can evaluate or compare two values differently and so changing this will change the result of the heuristic. The heuristics and metrics are concepts that are generally known and so they are represented by their names instead.
- **Data Generator Class:** the behaviour service can also be loaded with a data generator, to generate random or other data for testing the network with. There are some default data generators that can return string or number-based values, or even parse an XML file to generate some java objects, based on Bag-of-Words structures.

If you are not reading data files, then a data generator is required as well. To create a behaviour network, you then need to select the number of services to create by typing a number into the `Initial solutions` text box. You then also need to include a minimum and maximum value for the data generator. These values should be typed into the `Min Value` and `Max Value` text boxes. If you include a folder with files, then that overrides these settings. The number of files determines the number of services that are created and a data generator is not required because the data is read from each file. The linking heuristic classes are described more in the 'licasProblemSolver' document.

2.3 Dataset Files

This section is an extension of the Service and heuristics section and it is required to define the data type at least. It can be used to define the data type that should be generated and any optional files that should be read. Only the file directory needs to be declared, where all of the files in that directory would then be parsed and loaded. This results in two entries in the script – one for the data type and one for the directory file path. When configuring the script, care should be taken to make sure that the selected data type is compatible with the selected data files. You can also use the `Data Type` combo box, without any folder path, to indicate a data type for a data generator. The default algorithm uses the String data type.

If a data generator is used, you can specify the number of solutions, and the minimum and maximum values, as described in section 2.2. With Strings, for example, a minimum value of 2 and a maximum value of 5 will give the data value of the services a range from 2 to 5 characters. The algorithm then tries to cluster based on similar values.

2.4 Heuristic Settings

With the introduction of the new metric classes, some might require additional configuration parameters. These would be specified here and you only need to enter a value for any that you need to use. You also need to highlight the ones that you need to use. When you update the script, these should get added. When you select a heuristic, a small description should be displayed that may contain some shorthand at the end.

- SIB means that it will evaluate a smaller value as better.
- LIB means that it will evaluate a larger value as better.
- After that, you might get an initialisation parameter name in shorthand as well. MKP, for example, is displayed for the Minkowski heuristic. If you look at this section, you can see that there is an `mkp` entry. So in that case, to use the heuristic, you also need this parameter value, which you must set and then `Update` the script first, before running. To set, click on the `Value` column and enter a value in the related row.

2.5 Constructors

Each behaviour service can be initialised with an admin script, plus passwords. You can browse to add the admin script from a file. The passwords are then set automatically and may be ‘anon’. You need to create the script file first, which can be done through the Service Factory form, for example.

2.6 Other Configurations

The `Other Config` section allows for easy configuration of other values. The `Server` button will ask if you want to copy the details of the currently running server to your script. The Dynamic links button will open a form to allow you to enter the threshold and other values related to initialising the linking services. To make sure that a link is created, set the increment value to be larger than the threshold values – to 10 in the original form, for example. If you click the `Remove` check box, you can remove currently saved values instead.

2.7 Script Validation

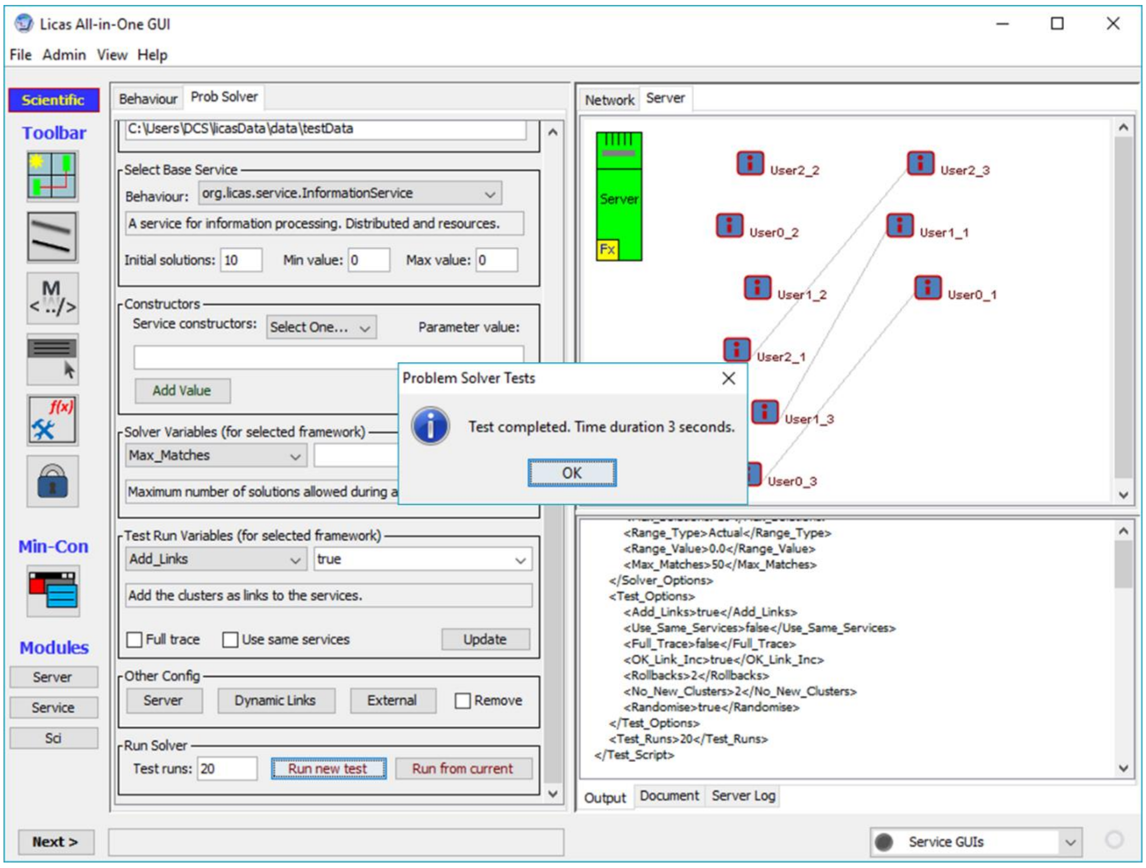
The `Update` button can be used to display and validate the script before running it. If the `Validate` check box is selected, the script is evaluated for missing or incorrect values. This is also the case with the problem solver panel and script. If the validation report looks incorrect, you can ignore it by unchecking this box. Note that a full script validation is not possible because there are different ways to execute a test, but missing values should be flagged. For example, all 3 service classes are required for a behaviour network, but the problem solver might not require all 3 classes, where they can default to ‘none’. The script checker only makes sure that the base behaviour service is defined. When configuring the problem solver, the behaviour panel values are added first, before the problem-solver specific values are added, so the behaviour panel is required for both types of problem.

2.8 Executing the Behaviour Services

If you click the `Run Script` button the behaviours will be loaded onto the network. They are also automatically started, so the main thread of the `Auto` class is automatically started. This is because a behaviour is assumed to be more autonomous and should run its own loop. The services will continue to run until the program code indicates that they should shut down. With the default clustering algorithms, they will run continuously and update dynamic link sets. This provides a changing picture of the permanent links between services and is indeterminate. To manually stop the services from running, you can click the `Stop Threads` button. The example behaviour service (LinkService) is distributed, where it looks up the addresses of other services and asks them directly, rather than accessing through the central server. It also configures itself at the start, through the admin script that is setup to include the evaluator and data generator class types. The behaviour service uses the ‘finaliseInitialisation’ method to create and add these services.

3 Problem Solver Panel

The problem solving package can either be run through the GUI, or on a separate program, but using the GUI server and display, for example. Tests are run in the same way as previously, by creating a test script and then executing that script. Details on all of the script variables can be found in the 'licasProblemSolver' guide. The GUI panel can be used to configure or change the script and then execute it to perform the specified tests. Figure 2 shows what the problem solver panel looks like.



C:\Users\DCS\licasData\data\testData

Select Base Service
Behaviour: org.licas.service.InformationService
A service for information processing. Distributed and resources.
Initial solutions: 10 Min value: 0 Max value: 0

Constructors
Service constructors: Select One... Parameter value:

Solver Variables (for selected framework)
Max_Matches
Maximum number of solutions allowed during a

Test Run Variables (for selected framework)
Add_Links true
Add the clusters as links to the services.
Full trace Use same services

Other Config

Run Solver
Test runs: 20

Problem Solver Tests
Test completed. Time duration 3 seconds.

```
    <Range_Type>Actual</Range_Type>
    <Range_Value>0.0</Range_Value>
    <Max_Matches>50</Max_Matches>
  </Solver_Options>
  <Test_Options>
    <Add_Links>true</Add_Links>
    <Use_Same_Services>false</Use_Same_Services>
    <Full_Trace>false</Full_Trace>
    <OK_Link_Inc>true</OK_Link_Inc>
    <Rollbacks>2</Rollbacks>
    <No_New_Clusters>2</No_New_Clusters>
    <Randomise>true</Randomise>
  </Test_Options>
  <Test_Runs>20</Test_Runs>
</Test_Script>
```

Figure 2. Problem Solver Panel

The process for using this panel is as follows:

- You can either load in an existing script or generate an empty one. If no script exists, an empty script is automatically generated when you press the `Update` button. You can use the example script as a template, for example, as some of the variables might be unfamiliar. Read the 'licasProblemSolver' document for more details.
- After each change, you can then press the Update button again and the change should appear in the script displayed in the `Document` text area.

- Note that the heuristic-specific variables should be displayed in the table, where the second column is editable. If a variable requires a value, then you must enter it there. In Figure 2, for example, `Mutate Percent` or `Crossover Percent` would require a value, but the currently selected do not. Only the selected variables are transferred to the script.

It is also possible to load in your own base classes that can be used to change the heuristic options that are available. This has not been fully tested, but it does work and can allow these classes to be changed to ones in a remote jar file. The following sections describe this further.

3.1 Script File Group Box

This is the same as for the `Behaviour Panel` of section 2.1, but it will read a different script and display a slightly different set of values.

3.2 Solver (Select / Change) Classes Group Box

You can use these options to change the base solver classes that determine what heuristics can be used. You might have overridden the `SolverFactory`, to add your own new heuristics, for example. You can load in a new jar file from the `Service Factory` and select the classes to add. The options allow you to:

- **Solver Factory:** this allows you to load in a different solver factory. This is the base object that controls all of the other configurable objects that are returned for a particular heuristic value. See the ‘licasProblemSolver’ document for details on the solver factory and what classes can be changed.

In the problem solver panel, you can then browse to and select your solver class from this section.

3.3 Framework and Heuristics group Box

The base framework that is used to evaluate the script can be changed. You can now select between the grid-based hyper-heuristic or hill-climbing, or a new distributed linking options. The linking option should work in the same way as a behaviour, but instead of running continuously, it only executes the specified number of distributed comparisons. A framework should then list a number of compatible heuristics and variables. This allows the heuristic, metric or evaluation function and the types of evolution, or other options, to be selected. The

SolverFactory can be changed by adding a new module and specifying a new solver factory class in the `Service Config` form's 'Add other Class' section. You then select the same class for the same function in the problem solver panel.

3.3.1 Heuristic Options Table

As in section 2.4, these values are very important and they can cause the test to fail if they are missing. If you select a particular heuristic type, the description might indicate that a configuration value is necessary. The shortened description should match one of the options in this table. You need to both enter a value, make sure it is set and also select or highlight the table row, for the value to be added to the script. If a value is not required, you just need to highlight the row, or all rows that are to be used. The heuristic description also indicates if a smaller or a larger value is better, leading to the following shortened notation:

- SIB means that it will evaluate a smaller value as better.
- LIB means that it will evaluate a larger value as better.
- After that, you might get an initialisation parameter name in shorthand as well. The genetic algorithms require a Crossover and/or Mutate setting, for example. Then `Update` the script first, before running.

3.4 Dataset Files

This section can be used to define the data type that should be generated and any files that should be read, in a similar way to the behaviour component of section 2.3. It is actually required for the default problem solving test however, where it is optional for the default behaviour test. This panel also includes an additional `File/URI` check box. The default is to browse to a folder and read each file in the folder as the data for a service. For a large dataset, for example, there might be only one file to read, as for the SOM, for example. So clicking this check box allows you to select a single file instead of a folder. It is then stored as a URI address.

3.5 Behaviour Services Selection

The `Select Base Service` section allows you to select the test service type. Only one service type is selected for the problem solving tasks. The other boxes to store a number of solutions (initially one per service) and arbitrary minimum or maximum numerical values are also provided. Depending on the test, you might not use data files and so these variables are available again, as for the behaviour script. Note that the default classes use the data files however, so to not use them, you need to add your own classes.

3.6 Constructors

Each problem-solving service can be initialised with an admin script, plus passwords, in the same way as for the behaviour services of section 2.5.

3.7 Solver Variables

Each framework can have some additional variables that are specific to the related problem solver. These are listed here as key-value pairs. After each selection, you should click the `Update` button to save the change. One has been added to the `Test Run Variables` section for convenience. Note that there is now a text description of each variable.

3.8 Test Run Variables

Each framework can have some additional variables that are specific to the test process or the test loop that runs. These are listed here as key-value pairs. After each selection, you should click the `Update` button to save the change. One has been added here for convenience. Note that there is now a text description of each variable.

3.9 Other Configurations

This is similar to section 2.6.

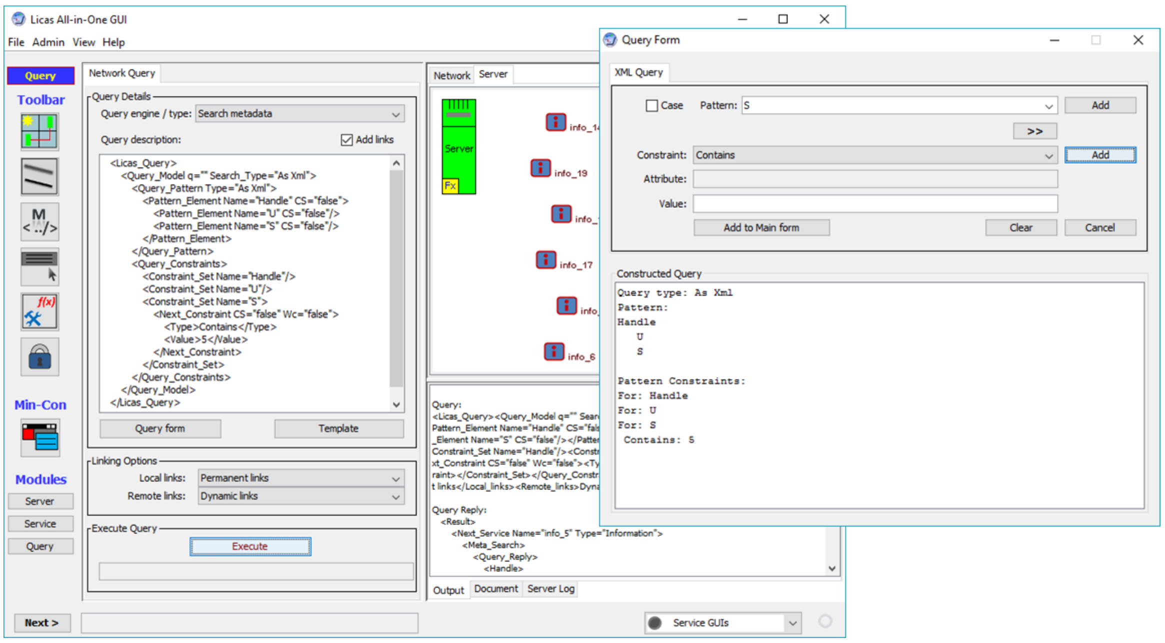
3.10 Run Solver Group Box

The run option has been updated to allow a subsequent test run from the current situation. You can therefore start a test using the `Run new test` option. This will create initial services and solutions and execute the specified number of test runs. By default, the main service loops (Auto class) are not started, so this is completely in contrast with the `Behaviour` panel process. This is because the problem solver is more centralised, where the problem is solved locally and then a mediator sends the result to the services, so that they can update themselves with it. So, it is managed by the central problem-solving algorithm.

If you then click the `Run from current` button, the specified number or test runs will be executed again, but using the current sets of services and solutions. It would also update the current set of dynamic links and does not reset them to zero. Any test uses the test script to define what test to carry out. The check boxes allow you to specify other more specialised features, but default values will work in general.

4 Query Panel

The query panel can be used to execute queries from the system query engines, of which there are now two. The panel has been updated to make use of the different linking options that are available. Figure 3 shows what the query panel looks like. This example uses the default query engine that searches over the service metadata in the server's repository. It has a type of `Search metadata` in the top `Query engine / type` box. The 'licasAdminGuide' document has more details on the structure of a query, but intuitively, you can see that it is asking for service IDs that contain a particular value – '5' in this case. The 'licasUserGuide' document has more details on how the distributed query process over more than one server works.



Query Details
Query engine / type: Search metadata
Query description: [✓ Add links]

```
<Licas_Query>
  <Query_Model q="" Search_Type="As Xml">
    <Query_Pattern Type="As Xml">
      <Pattern_Element Name="Handle" CS="false">
        <Pattern_Element Name="U" CS="false"/>
        <Pattern_Element Name="S" CS="false"/>
      </Pattern_Element>
    </Query_Pattern>
    <Query_Constraints>
      <Constraint_Set Name="Handle"/>
      <Constraint_Set Name="U"/>
      <Constraint_Set Name="S">
        <Next_Constraint CS="false" Wc="false">
          <Type>Contains</Type>
          <Value>5</Value>
        </Next_Constraint>
      </Constraint_Set>
    </Query_Constraints>
  </Query_Model>
</Licas_Query>
```

Linking Options
Local links: Permanent links
Remote links: Dynamic links

Execute Query

```
Query:
<Licas_Query><Query_Model q="" Sear
Pattern_Element Name="Handle" CS="fal
_Element Name="S" CS="false"/></Patte
Constraint_Set Name="Handle"/><Const
xt_Constraint CS="false" Wc="false"><Ty
raint></Constraint_Set></Query_Constr
t links</Local_links><Remote_links>Dyn

Query Reply:
 <Result>
   <Next_Service Name="info_5" Type="Information">
     <Meta_Search>
       <Query_Reply>
         <Handle>
```

Case [ ] Pattern: S
Constraint: Contains
Attribute:
Value:

Constructed Query

```
Query type: As Xml
Pattern:
Handle
   U
   S

Pattern Constraints:
For: Handle
For: U
For: S
 Contains: 5
```

Figure 3: Query Panel

You can query the nodes running on your local server, or if you have registered a number of networks, the query can be run over all of the registered servers. This can be performed automatically if you register the addresses of other servers with the local server. The `scripts` folder in the default 'licasData' location now contains 3 example query scripts. You can use the `Template` button to load one in and then change it as required. In the example, the services have been loaded through the `Service Group` panel (see the 'licasGui' document), creating 20 `Information` services. Added to this panel is a query form to allow you to construct queries manually. This is actually the same query form that the `Textflo` system uses (see the DCS web site). The query displayed produces the XML script shown in the figure panel. Note that you can probably skip sibling elements now and still retrieve the correct XML section. It might also be helpful to download the Textflo system or documentation and test queries over whole documents using that.

4.1 Server Metrics or Stats

The second option can be used to search over metrics that get stored for each service method request. Each method that is passed t a service is now counted, as well as the message size. You can use this query engine to retrieve information from a server about what services have received the most invocations, and so on. It is still a work in process and does not return specific values, but comparisons at the moment. This is illustrated in Figure 4.

4.2 Query Details Group Box

Although there is a built-in query engine with the licas system, you can also load in a different query engine as a new service, select its type and use that instead. This should be loaded onto the `HttpServer` server as a base service. If you then retrieve all of the service types from the `Admin Panel` options, you will also retrieve the new query engine service type. This will be included in the `Query engine` combo box, from where it can be selected. Note that the query engine that you provide should implement the `QueryDef` interface, so that the method that is automatically called to execute the query is available and also have a different type to the default ones.

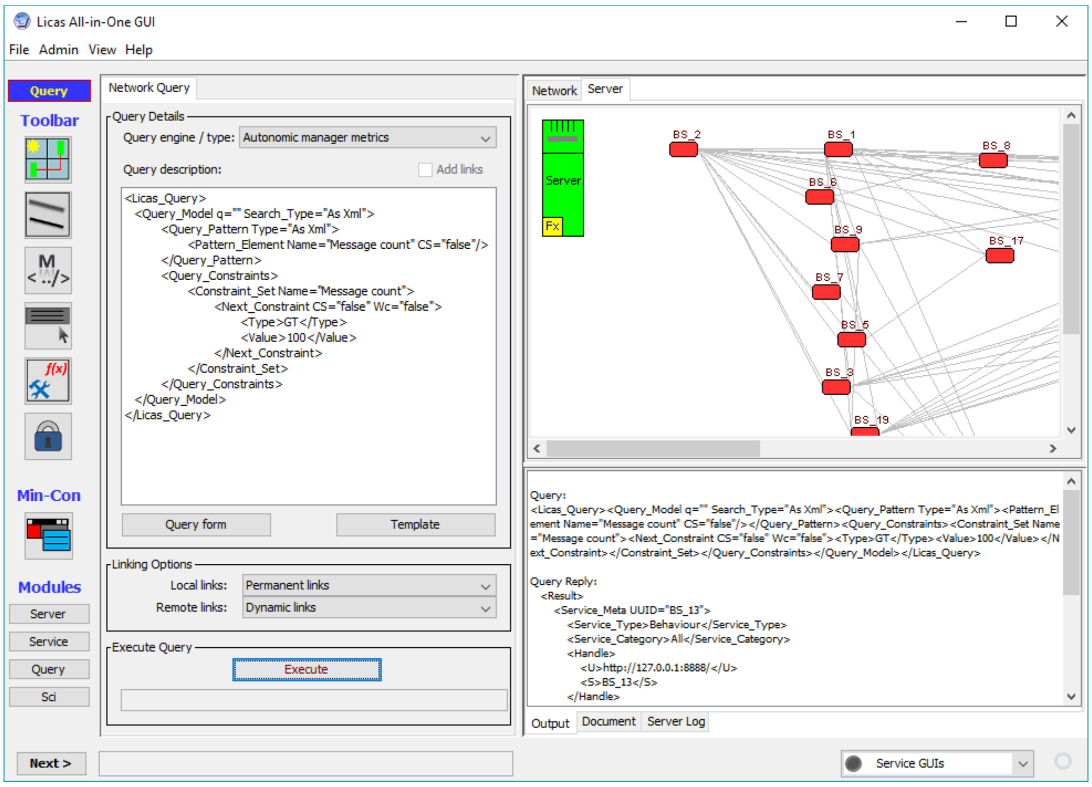
4.3 Linking Options Group Box

This allows you to link up nodes as part of the query process. You firstly need to add a linking service to each node. This can be done using the server popup menu. Right click the server graphic and select the 'Utility service / Linker' option. The `Linker` form opens (see 'licasGui' document). If you just click `OK`, the default linking values are added to each Information service. You can switch on a view of the nested services using the `Admin` panel. The default linker settings indicate a link threshold of 0.4 and an increment value of 0.1. These can then be used as described next.

4.4 Execute Query Group Box

This group box provides the options to allow you to execute a query or retrieve some statistical information. If you click the execute button, the query description will be sent to the server to be executed. The server will look for a child service of the service type specified in the `Query Engine` combo box and then try to execute the `executeQuery` method specified in the `QueryDef` interface on it. This method will be expected to return an XML description of the reply. Figure 3 shows one example of a query that has been executed. The query engine is of type `query engine` and the first service of that type that is found on the network will be used. The network itself is a hierarchical network and an XML-based query has been executed on it. The query reply is also displayed.

The query reply is then written out to the output text area on the GUI. How the reply is generated, or how the query is evaluated depends entirely on what service you provide as a query engine. It only needs to implement the interface to be useable. Some information is output when you execute a query or try to retrieve stats, to let you know that the process is on-going. This is displayed at the bottom of this group box. Note that the default licas services do not store any data themselves, so to query data you would also need to extend the `Service` class or use the Information service possibly. You would then create the network using the new class.



Query engine / type: Autonomic manager metrics

Query description:

```
<Licas_Query>
  <Query_Model q="" Search_Type="As Xml">
        <Query_Pattern Type="As Xml">
              <Pattern_Element Name="Message count" CS="false"/>
        </Query_Pattern>
        <Query_Constraints>
              <Constraint_Set Name="Message count">
                    <Next_Constraint CS="false" Wc="false">
                          <Type>GT</Type>
                          <Value>100</Value>
                    </Next_Constraint>
              </Constraint_Set>
        </Query_Constraints>
  </Query_Model>
</Licas_Query>
```

Local links: Permanent links

Remote links: Dynamic links

```
Query:
<Licas_Query><Query_Model q="" Search_Type="As Xml"><Query_Pattern Type="As Xml"><Pattern_El
ement Name="Message count" CS="false"/></Query_Pattern><Query_Constraints><Constraint_Set Name
="Message count"><Next_Constraint CS="false" Wc="false"><Type>GT</Type><Value>100</Value></N
ext_Constraint></Constraint_Set></Query_Constraints></Query_Model></Licas_Query>

Query Reply:
  <Result>
     <Service_Meta UUID="BS_13">
        <Service_Type>Behaviour</Service_Type>
        <Service_Category>All</Service_Category>
        <Handle>
           <U>http://127.0.0.1:8888/</U>
           <S>BS_13</S>
        </Handle>
```

Figure 4: Autonomic manager metrics, or server stats query.

4.5 Creating Structure through Linking

You select these options through the two `Linking Options` combo boxes. You also need to select the `Update links` check box for linking to be included. The query process will then try to create links based on the query reply. Local nodes, or nodes running on the current server, can be linked up either by creating permanent links immediately, or by creating permanent links from the dynamic ones, when the related weight value passes the top link threshold value, for the Linker service. In the figure, each time the query is executed; nodes related through the query reply have their dynamic link values updated. Remote nodes – running on a different server, can only be represented by the dynamic links or service associations. A structure with permanent links to remote server nodes is not possible.

The query reply can return services or nodes associated through the query execution. In the figure, the query is looking for services with an ID value that contains the number 2. Therefore, services 'i_2', 'i_12' and 'i_20' will be returned for this query. As permanent links were selected, after the first query reply; if you refresh the view, you should see a link between these services. In the figure, if 'dynamic to permanent' was selected, then this requires a reinforcement of the link until the weight value exceeds the link threshold. If you execute the specified query 5 times in a row, the weight value is 0.5 and the threshold is 0.4. Therefore, if you refresh the view after each execution, you will see no change until the 5th execution, when the permanent links will be drawn. That is really all that the dynamic linking process is. You can also retrieve the metadata values during the process to check them.

There is also a possibility to retrieve statistics of the query process by clicking the `Get Stats` button. The implementation of this however needs to be included in a query engine that you provide as the default query engine does not have one. The GUI will then call the `getStats` method on this engine and expect to get an XML reply for the result. This reply will then be written to the output text area as well.