Project No. 611391

DREAM
Development of Robot-enhanced Therapy for Children with Autism Spectrum Disorders

TECHNICAL REPORT
Guideline For the YarpGenerator

Date: **18/05/2015**

Technical report lead partner: **Plymouth University**

Primary Author: **E. Senft**

Revision: **1.4**

Project co-funded by the European Commission within the Seventh Framework Programme		
Dissemination Level		
PU	Public	
PP	Restricted to other programme participants (including the Commission Service)	**PP**
RE	Restricted to a group specified by the consortium (including the Commission Service)	
CO	Confidential, only for members of the consortium (including the Commission Service)	

Contents

Summary **2**

Principal Contributors **2**

Revision History **2**

1 Yarp Component Generator **3**
1.1 Concept 3
1.2 Utilisation 4
1.3 Integration in a Yarp project 5
1.4 Example component generation 6

Summary

This technical report presents the documentation for the second version of the YarpGenerator.

Principal Contributors

The main authors of this document are as follows (in alphabetical order).

Paul Baxter, Plymouth University
Emmanuel Senft, Plymouth University

Revision History

Version 1.0 (E.S. 03-02-2015)
Initial description

Version 1.1 (E.S. 12-05-2015)
Updated description to reflect changes in code

Version 1.2 (P.B. 13-05-2015)
Small clarifications etc.
Version 1.3 (E.S. 13-05-2015)
Small clarifications etc.

Version 1.4 (P.B. 18-05-2015)
Added example component generation.

Version 1.5 (E.S. 23-05-2016)
Update with new version of YarpGenerator and remove the Software Installation part.

1 Yarp Component Generator

1.1 Concept

In order to change a single port in a Yarp component, as specified in the software engineering standards, multiple parts of the code need to be modified, in different source files. But this process is a fixed one and can be automated. The purpose of the YarpGenerator is to provide an easy way to generate the structure of a component that conforms to the standards, including the different files and folders required, the code needed to use Yarp ports and the comments according to the standards.

This software enables the generation of compilable code, which can be added to a project (such as the DREAM integrated system) with minor additional modifications. Furthermore, it reduces the complexity of using Yarp ports by centralising the send/receive functions, and by providing if desired a simpler interface to access to the Yarp functionalities. For input ports, callbacks are used to enable event-based processing, without having to manually check for new information.

This software has been developed using Qt, is directly usable on windows and can be modified using QtCreator. This project contains five folders: components, descriptions, models, release and finally the sources.

The folder *components* is the folder where the generated components are located. A new folder with the component name is created and can be then just moved in the desired location.

Each time a component is generated, a description file with the all the parameters needed for the generation is created in the folder *descriptions*. These files can be loaded in the software using the user interface by filing the lower text box with the name of the component followed by 'Description', as demonstrated when the software is started.

The folder *models* contains all the templates used to generate the other files. In the previous version, it appeared multiple times in the project, but this has been fixed, and now only one folder contains all the files used.

The folder *sources* contains all the sources used for the project and can be modified using Qt Creator. Some adjustment of the project have to be made to be able to run smoothly, such as changing the location of the build, the run command and so on.

Finally *release* contains the executable and the libraries needed by the tool.

As accepted in Brussels, the naming convention used for the port generation is: `/componentname/portname:i-o`.

This software will generate up to four folders and ten files. The files config.ini, name.h, name.cpp, nameConfiguration and nameMain.cpp are the ones defined in the Software Development Guide, and are required to allow the use of Yarp. In addition to these files described in the Software development guide, we propose to add four files to allow a simpler use of Yarp. The idea is to have the main code in nameController.cpp (with nameController.h as header) which does not require any Yarp functionalities, and a nameYarpInterface class providing a really simple interface between the main code and the files used for the Yarp communication. All the source files are located inside the src folder in the component.

In nameYarpInterface.cpp, each input port is linked to a function which will be called each time a message is received, and the message content is transmitted as a parameter. And then in this class, some simple conversions can be done to native c++ variables and then call the related function in nameController.cpp. The same idea is applied for output port: functions in Yarp interface can be called from the controller, do some conversion to Yarp type variables and send the message by calling functions in nameComputation.cpp. The other files (computation, configuration, main and the header) do not require to be modified for using Yarp.

1.2 Utilisation

Figure 1 shows the GUI of the YarpGenerator utility.

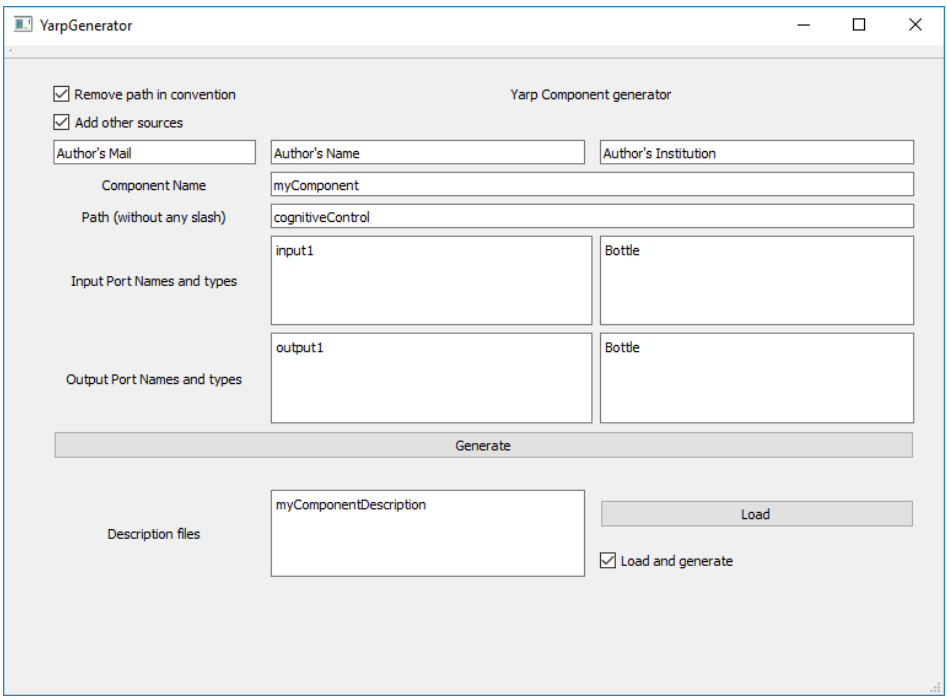

Figure 1: Interface of the YarpGenerator.

The first line of the interface contains information about the author, which will be directly included in the doxygen comments. The next two lines contain the name of the component and the path used for the port naming convention (not the generation). The first checkbox in the top right corner allows a switch to the DREAM convention. A second checkbox enables/disables the generation and use of the two additional classes proposed: yarpInterface and the Controller.

The next four boxes correspond to the different port names and their type (only yarp-compatible types should be used). In theory the number of port is not limited (i.e. a limit to the number of specified ports has not been encountered in testing), BufferedPort will be added in any case, so only Bottle, VectorOf<int>, VectorOf<float> and VectorOf<double> should be used.

The generate button generates the folder and files as described in figure 4. When a component is generated, a description file is created in the description folder and can be reused later to create a new component with the same ports and name. This can be useful if ports subsequently have to be renamed/added/removed after the first creation of the component: after regenerating the source files with the new port names, the yarp-standard interface code can be replaced without affecting the main computation source code. Only the functionalities required to get other information from the config files have to be added again.

The lower part of the interface can be used to generate components from the description file. Several files can be added in the text box and used at once. These description files should be put in the `/description` folder and follow this structure (instructions in italic and parameters in bold):

Remove path: **Value**

Add files: **Value**

Author's name: **Firstname Surname**

Author's institution: **Institution's name**
Author's mail: **e-mail address**
Component name: **Name**
Path: **Path**

Input Port Name and Type
input1 inputtype1
input2 inputtype2

Output Port Name and Type
output1 outputtype1
output2 outputtype2

The parameter **value** can take 0 or 1 and describes for the first line if the path should be removed when generating port names according to the convention selected (normally yes), and for the second line if the two secondary class should be created. A line should be inserted between the path line and the first input line, as well as between the last input line and the first output line. Similarly to the interface, only Yarp compliant types should be used. This is an example of description file content:

Remove path: 1
Add files: 1
Author's name: Emmanuel Senft
Author's institution: PLYM
Author's mail: emmanuel.senft@plymouth.ac.uk
Component name: myComponent
Path: cognitiveControl

Input Port Name and Type
in1 Bottle
in2 VectorOf<int>
in3 Bottle

Output Port Name and Type
out1 Bottle
out2 VectorOf<float>

1.3 Integration in a Yarp project

Once all the files are generated, only a few steps are required to be able to use the component in a Yarp project.

First, the main folder generated needs to be added in the project, and the CMakeList in `/DREAM/release/components` needs to be modified to include this new component and then rerun CMake to update the Visual Studio project 1.4. Then the main xml file describing the network (`/DREAM/release/app/app.xml`) needs to be changed: the new module needs to be added as well as each connection between its ports and the others specified.

1.4 Example component generation

The generation of a sample component is described in this section to demonstrate the main steps involved, and the main output files generated (with automatically generated contents). Figure 2 shows the definition of three ports for this example component (two input ports and one output port), and a text file name where the description is stored for later reload if required. The content of the description file is presented in figure 3.

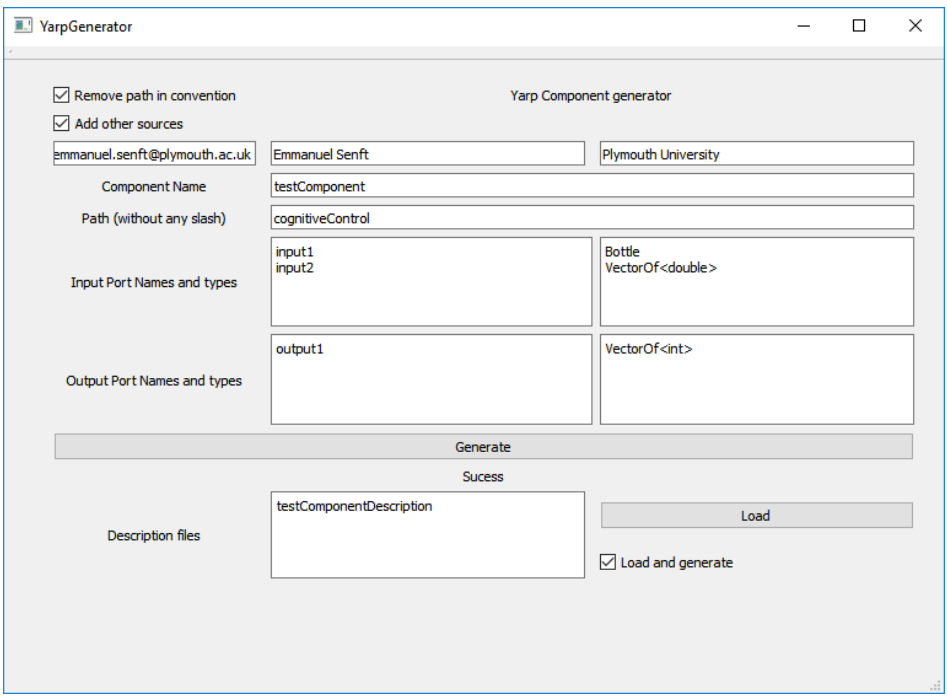

Figure 2: Definition of example component ports and types.

testComponentDescription.txt

```
1  Remove path: 1
2  Add files: 1
3  Author's name: Emmanuel Senft
4  Author's institution: Plymouth University
5  Author's mail': emmanuel.senft@plymouth.ac.uk
6  Component name: testComponent
7  Path: cognitiveControl
8
9  Input Port Name and Type
10 input1 Bottle
11 input2 VectorOf<double>
12
13 Output Port Name and Type
14 output1 VectorOf<int>
```

Figure 3: Description file associated with the example presented in figure 2.

Generating a new component with these settings results in a file structure as shown in figure 4. Unlike the previous version, now a flat structure is adopted for the files, all the sources (the 4 mandatory and the 4 optional) are in the same folder: src. Note also that the xml test application is not automatically generated at the moment, and must be made manually.

In addition to the source code contents, the yarpGenerator utility also automatically generates the

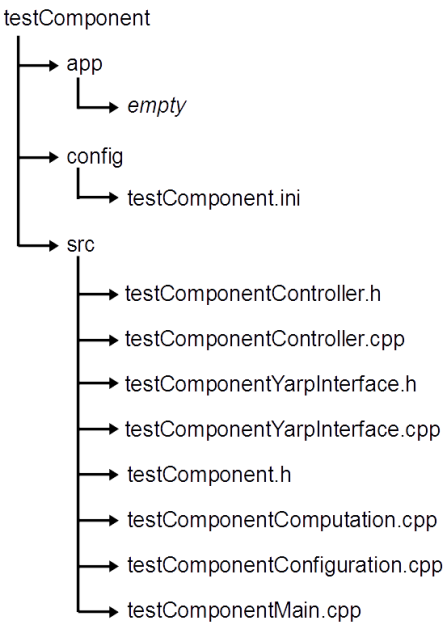

Figure 4: File structure created for the testComponent. An xml application file is not created automatically; in the current version of the yarpGenerator, this has to be created manually. This is the same structure as described in the wiki.

Doxygen-compatible comments in the header of the testComponent.h file based on the instantiated ports. A sample of this is shown in figure 5.

```
* <b>Configuration File Parameters </b>
*
* The following key-value pairs can be specified as parameters in the configuration file
* (they can also be specified as command-line parameters if you so wish).
* The value part can be changed to suit your needs; the default values are shown below.
*
* Key | Value
* :-- | :----
* _input1In | /testComponent/input1:i
* _input2In | /testComponent/input2:i
*
* - description
*
* Key | Value
* :-- | :----
* _output1Out | /testComponent/output1:o
*
* - description
*
* \section portsa_sec Ports Accessed
*
* - None
*
* \section portsc_sec Ports Created
*
* <b>Input ports</b>
*
*  - \c /testComponent
*
*  - \c /testComponent/input1:i
*  - \c /testComponent/input2:i
*
* <b>Output ports</b>
*
*  - \c /testComponent
*
*  - \c /testComponent/output1:o
*
* <b>Port types </b>
*
* The functional specification only names the ports to be used to communicate with the component
* but doesn't say anything about the data transmitted on the ports. This is defined by the following code.
*
* \c BufferedPort<Bottle>     input1In;
* \c BufferedPort<VectorOf<double>>       input2In;
* \c BufferedPort<VectorOf<int>>      output1Out;
```

Figure 5: Sample of the automatically generated comments in the main test component header, showing the port definitions.