Examples how to use TiViPE

Technical Report: Version 1.0.1

Tino Lourens
TiViPE
Kanaaldijk ZW 11
5706 LD Helmond
The Netherlands
tino@tivipe.com

February 4, 2013

Contents

Abstract

The aim of TiViPE is to integrate different technologies in a seamless way using graphical icons [2]. Due to these icons the user does not need to have in depth knowledge of the underlying hardware architecture.

This report is elaborates on the use of TiViPE, the aim of this document is to learn the user how to use TiViPE in the best possible way. This document provides insight upon the basis of questions from users. It will extend over time.

1 Introduction

The aim of TiViPE is to integrate different technologies in a seamless way using graphical icons [2]. Due to these icons the user does not need to have in depth knowledge of the underlying hardware architecture.

This document is meant to provide users insight on how to use TiViPE and is based upon question from different users. Every section in this report provides a different topic that is meant to be independent from other sections.

This report is outlined as follows: Section 2 describes step by step how a library and 2 new modules are created. Section 3 provides debugging opportunities in TiViPE. Section 4 describes problems encountered to the use of a Linux operating system. Section 5 shows how exclusive and non-exclusive type of toggles are used. Section 6 elaborates how to merge several modules to a new module. Special attention is given to the modification of the GUI.

2 Creating a library and two new modules

TiViPE has been created originally keeping in mind that users had created a library and wanted to use the available routines of the library within TiViPE without additional programming.

Many TiViPE users however start using TiViPE and the graphical icons, and find it an easy way to program until, they meet the moment that the graphical icon/module they have in mind to use does not exist. So far they never considered even to create a new module, but as soon as a new module needs to be created it is very likely that the required routine call needs to be programmed as well.

Two simple module examples called `TVPhelloworld` and `TVPcopy` have been constructed and implemented in a library called `TVPexample`. All files for library and two modules can be found in `TVPexample.zip` on the TiViPE website. This zipped file contains 7 files:

1. `Readme.txt`: to explain how a library is constructed and modules embedded.
2. `TVPhelloworld.ci`: TiViPE code information file with content of 10 tabloids.
3. `TVPcopy.ci`: TiViPE code information file with content of 10 tabloids.
4. `src.pro`: project file. Used to compile files to a 'TVPexample' library.
5. `TVPexampledefs.h`: header file to take care of dll import and export.
6. `TVPexample.h`: header file with c++ class `TVPexample`. The class contains 2 routines: `TVPhelloWorld` and `TVPcopy`.
7. `TVPexample.cpp`: a c++ source code file with the implementation of the above 2 routines.

We assume that the user is familiar with or has some basic knowledge of C or C++ programming. Files `TVPexample.cpp` and `TVPexample.h` contain the implementation of the routines `void TVPhelloWorld ()` and `Data5D TVPcopy (Data5D d)`.

2.1 Library construction

The files `src.pro`, `TVPexample.cpp`, `TVPexample.h`, and `TVPexampledefs.h` make up the source files for the library. Create directory `$MYTVPHOME/Libraries/TVPexamples/src` for Unix or `%MYTVPHOME%`
`Libraries`

`TVPexamples`
`src` for windows directory, and in copy the above 4 files to this directory.

In Windows start a command window (cmd.exe) and goto your TiViPE home directory (C: TiViPE). Here type

```
vars.bat
cd %MYTVPHOME%\Libraries\TVPexample\src
qmake
nmake
```

or unix

```
cd $MYTVPHOME/Libraries/TVPexample/src
qmake
make
```

to compile the library. The resulting library can be found in `%MYTVPHOME%` `lib` for windows or `$MYTVPHOME/lib` for unix directory.

2.2 Exporting routines for Microsoft Windows

Windows platforms make use of dll export and import. In the `TVPexample` library we made use of `TVPEXAMPLE_LIB`. My recommendation is to create a name in caps that is the same as the name of your library. Note to set this properly in your `...defs.h` file.

As for the header files in this case only `TVPexample.h` be sure to have included:

```
#include <TVPexampledefs.h>
```

and start every class with:

```
class TVPEXAMPLE\_EXPORT ...
```

The export definition is given in your project file called src.pro it is as follows:

```
win32-msvc*|win32-icc {
DEFINES      += TVPEXAMPLE_LIB
}
```

2.3 Creation of TiViPE modules

A new TiViPE module is created by pressing starting TiViPE and typing `Ctrl-N` or by selecting create `File->Module->New`. A window pops up where an (M) field needs to be filled out and a "New module name" which is a unique name needs to be given. After pressing the OK button an new window pops up with 10 tabloids. Six of these tabloids are used for communication and four are used to describe the routine call used. When OK is pressed in this tabloid, all data is saved to a so-called "code information" file. Pressing the help button provides extensive help on creation and modification of a module.

Two of these files, `TVPhelloworld.ci` and `TVPcopy.ci` are described below.

2.3.1 Hello World

The ”hello world” example uses the routine call:

```
void TVPexample::TVPhelloWorld ()
```

form the compiled TVPexample library. In the code information file the library and location need to be given, a header file where the routine is given, a description of the parameters used, and the class plus routine call need to be provided. These are all the elements required to embed a routine call into TiViPE:

tabloid libraries

```
-L$(MYTVPHOME)/lib -lTVPexample
```

tabloid includes

```
$(MYTVPHOME)/Libraries/TVPexample/src:TVPexample.h
```

tabloid GUI is empty since there is no argument given nor any return value required.

and for the tabloid routine this is

```
TVPexample::TVPhelloWorld
```

2.3.2 Copy

The copy routine

```
Data5D TVPexample::TVPcopy (Data5D d)
```

requires the description of an input and an output parameter. Hence the main difference is in the GUI and of course the routine call itself:

tabloid libraries

```
-L$(MYTVPHOME)/lib -lTVPexample
```

tabloid includes

```
$(MYTVPHOME)/Libraries/TVPexample/src:TVPexample.h
```

tabloid GUI

```
Input("Input", "-in", DATA5D, DATA5D_ANYTYPE) [1];
Output("Output", "-out", DATA5D, DATA5D_ANYTYPE) [F];
```

and for the tabloid routine this is

```
TVPexample::TVPcopy
```

The GUI contains one input and one output. To set the arguments properly, the indexes are provided [1] for first argument and [F] for function return parameter. The syntax of the GUI is quickly learned by using the available buttons in the GUI tabloid. Clicking these buttons gives a pop up window, where the syntax is described in detail. The syntax of the GUI parameters is described extensively in the documentation that is found by pressing the help button in the tabloid window.

3 Debugging

Within TiViPE itself there is no way to debug code as you are used to do. The reason is that there is no textual code within TiViPE that is coded by human. TiViPE makes use of internal code generators, so no opportunity to touch code at this level. The aim of TiViPE was to integrate existing (and tested) library calls into TiViPE. Graphical debugging is easy you see a module crash. Once you use graphical modules and merge them to new modules you rarely encounter an error, there is the true advantage of graphical programming in combination with code generators.

Nevertheless a module that has been created might crash. Most of the time this is in the routine you just developed and the crash occurs somewhere in the textual library. How to debug?

As example the TVPexample library is taken where two simple module examples called `TVPhelloworld` and `TVPcopy` have been constructed and implemented in a library called `TVPexample`. All files for library and two modules can be found in `TVPexample.zip` on the TiViPE website.

Uncomment the line 23 in the file `TVPexample.cpp`:

```
//#define TVPEXAMPLE_DEBUG
```

by removing the two slashes. Save the file and recompile the library by typing `nmake` on a windows OS or `make` on unix or mac. After every routine call a print statement is given.

When running Microsoft Windows you might see not anything until the module is compiled with a console option enabled. For instance, the created TVPcopy module is stored in

```
%MYTVPHOME%\Modules\<Entryname>\TVPcopy\src
```

Next open file `src.pro` and for windows remove the sharp (#) to enable the console

```
# CONFIG += console
```

save the file and open a command window (cmd.exe) and compile the module

```
c:\TiViPE\vars.bat
cd %MYTVPHOME%\Modules\<Entryname>\TVPcopy\src
qmake
nmake
```

Rerun the module/icon TVPcopy within TiViPE. The module will print "`TVPexample::TVPhelloWorld`" every time the module is being executed. This output is found, by clicking the lower right rectangle in the graphical icon.

3.1 Debugging without TiViPE

A module can be debugged using the standard debug tooling used in for C/C++. This is done by

1. compiling your library in debug mode by editing the src.pro file in your module directory by removing sharps from `CONFIG += warn_on` and `CONFIG += debug` and comment `CONFIG += release` by placing a sharp (#) at the beginning of the line.
2. compiling the module in debug mode also by editing the `src.pro` module in the same way.
3. run `TVPMTVPcopy`[1] with a bunch of parameters from your preferred debugging tool.

[1] Every TiViPE module is prefixed with `TVPM`.

4 Linux

Question: why certain TiViPE modules are not working in my Linux distribution?

Linux comes in many flavors and different precompiled packages. TiViPE will provide support for Ubuntu and CentOS. TiViPE starting from version 2.1.0 will provide support for specific distributions only. For instance, if CUDA support is required, a production release is available for Ubuntu 10.04 and 11.10, this implies that the TiViPE modules using CUDA might not work if another distribution is used. On the other hand your hardware might not provide the right support, or if a library is missing, you might need to install this software package.

4.1 Problem starting TiViPE

Question: I get an error when starting TiViPE, it seems that there is a problem with Qt. How can this problem be fixed?

Answer: In your

```
$HOME/.bashrc
```

file, please include

```
if [ -f $HOME/TiViPE/vars ]; then
. $HOME/TiViPE/vars
fi
```

Every time you start up a terminal this file is read and thus also the `$HOME/TiViPE/vars` script is started.

In this vars file you will find a setting for Qt. Most likely you need to change this to the path where your qt is installed.

4.2 Ubuntu 10.04

Question: I found that Ubuntu 10.04 does not support Qt 4.7.3, but Ubuntu 11.04 does. Is this correct?

Answer: I think that Qt 4.7.3 is supported on Ubuntu 10.04, but you need to compile it yourself. For Ubuntu it might be easier to upgrade to a newer distribution.

5 Toggles

TiViPE supports two types of toggles, an exclusive type and a non-exclusive type. In the two toy examples below the differences between these tow toggles are described.

The GUI of module *ColorGenerator1* is as follows:

```
Output("Output image", "-o", DATA5D, DATA5D_UCHAR) [F];
BeginGroup(2);
  Integer("Width", "-w", 640, 1, INT_MAX) [1];
  Integer("Height", "-h", 480, 1, INT_MAX) [2];
  Integer("Red", "-r", 128, 1, 255) [3];
  Integer("Green", "-g", 128, 1, 255) [4];
  Integer("Blue", "-b", 128, 1, 255) [5];
  Toggle("Enabled colors", NONEXCLUSIVE);
    ToggleElement("Blue enabled", "-be", CHECKED) [8];
```

```
    ToggleElement("Green enabled", "-ge") [7];
    ToggleElement("Red enabled", "-re", CHECKED) [6];
  EndToggle();
EndGroup();
```

and the routine call used in C++ is as follows:

```
Data5D C_Color::colorTest (int w, int h, int r, int g, int b,
                           bool re, bool ge, bool be)
```

The GUI of module *ColorGenerator2* is as follows:

```
Output("Output image", "-o", DATA5D, DATA5D_UCHAR) [F];
BeginGroup(2);
  Integer("Width", "-w", 640, 1, INT_MAX) [1];
  Integer("Height", "-h", 480, 1, INT_MAX) [2];
  Integer("Red", "-r", 128, 1, 255) [3];
  Integer("Green", "-g", 128, 1, 255) [4];
  Integer("Blue", "-b", 128, 1, 255) [5];
  Toggle("Enabled colors", EXCLUSIVE) [6];
    ToggleElement("Red enabled", "-re");
    ToggleElement("Green enabled", "-ge");
    ToggleElement("Blue enabled", "-be", CHECKED);
  EndToggle();
EndGroup();
```

and the routine call used in C++ is as follows:

```
Data5D C_Color::colorTest (int w, int h, int r, int g, int b,
                           int  ec)
```

In the 2 examples above the syntax of `Toggle` and `ToggleElement` is similar, but not exactly the same. There is the obvious difference in choice being either `EXCLUSIVE` or `NONEXCLUSIVE`. In an exclusive toggle exactly one element can be checked, while in a non-exclusive toggle, zero, one, or multiple elements can be checked. An element is checked when the extra argument `CHECKED` is added.

In case a non-exclusive toggle is used, all toggle elements are passed on as arguments to the routine call, while the toggle itself is not used as an argument. In the GUI of module *ColorGenerator1* this is done by providing an argument number between square brackets before the semi column. In a non-exclusive toggle multiple elements can be checked.

In case of an exclusive toggle, only the toggle itself is passed on as argument. In the GUI of *ColorGenerator2* this is illustrated again by an argument number that is given between square brackets before the semi column. The elements are numbered sequentially starting at 0 with an increment of 1, hence in the example GUI above, `ec` will get the value 2. Since only one element can be checked at a time `ec` will pass on a unique number for every toggle element. Modification of this parameter (and parameters in general) in done in the parameter window, that is opened by left-clicking on the right upper square of the appropriate module icon in the network, an example of such a network is given in Figure 1a. The parameter window for *ColorGenerator2* is illustrated in Figure 1c.

Figure 1: *(a) TiViPE network* `togglemerge.net` *to demonstrate the use of toggles and how to merge the green modules to* ColorImageGeneratorM. *(b) Parameters of* ColorImageGenerator1. *(c) Parameters of* ColorImageGenerator2. *(d) Paramaters of* DoubleOperand. *(e) Paramaters of* ColorImageGeneratorM.

6 Merging modules

The power of TiViPE is that multiple modules can be merged to a single module without any additional programming, but what does that imply?

The TiViPE module merging engine (MME) takes up that part of the network that has been marked (in green) for merging. It firstly computes the order in which these modules are executed. Once the order is determined, the code-information files with extension `.ci` are parsed and processed to a new code-info file. After this process has completed the user can edit the tabloids, as usual, with the exception of the *Include*, *Library*, and *Routine* tabloids. These three will be fully taken care of by the MME. The tabloids *Brief*, *Full*, *Input*, *Output*, *Parameters*, and *Examples* contain the documentation of all individual modules belonging to that particular tabloid. This documentation is expected to be altered by the user for every individual tabloid.

The *GUI* tabloid contains syntax specific elements describing the parameters of all the merged modules. The number of parameters can grow quickly, hence in this section an example is provided, how these parameters can be fixed or overlayed.

As an example the three green modules, given in Figure 1a, are merged to a module called *ColorImageGeneratorM*. Besides the two modules given in Section 5, there is another module *DoubleOperand* involved, the GUI of this module is as follows:

```
Input ("Input", "-i", DATA5D, DATA5D_ANYTYPE) [1];
Input ("Input 2", "-j", DATA5D, DATA5D_ANYTYPE, OPTIONAL) [2];
Output ("Output", "-o", DATA5D, DATA5D_ANYTYPE) [F];
BeginGroup (2);
  Toggle ("Operation", EXCLUSIVE, 2) [5];
    ToggleElement ("Absolute diff", "-absdiff", CHECKED);
    ToggleElement ("Add", "-add");
    ToggleElement ("Blend", "-blend");
  BeginGroup (2);
    Float ("Blending value", "-blendval", 0.5, 0, 1, 10) [6];
    ToggleElement ("Divide", "-divide");
    ToggleElement ("Divide into", "-divideinto");
    ToggleElement ("Maximum", "-maximum");
    ToggleElement ("Minimum", "-minimum");
    ToggleElement ("Modulus", "-modulus");
    ToggleElement ("Multiply", "-multiply");
    ToggleElement ("Power", "-power");
    ToggleElement ("Subtract", "-subtract");
    ToggleElement ("Subtract from", "-subtractfrom");
    ToggleElement ("Norm", "-norm");
  EndToggle ();
    EmptySpace (2);
    Float ("Real value", "-realval", 0, -DBL_MAX, DBL_MAX, 10) [3];
    Float ("Imaginary value", "-imagval", 0, -DBL_MAX, DBL_MAX, 10) [4];
  EndGroup ();
EndGroup ();
```

and the routine call used in C++ is as follows:

```
Data5D Arithmetic::DoubleOperand (Data5D i, Data5D j, double re, double im,
                                  int op, double blend)
```

The code generator has merged the GUI of these three modules to

```
Output("Output", "-oM1", DATA5D, DATA5D_ANYTYPE) [2:F];

BeginGroup(2);
  Integer("Width", "-w", 640, 1, INT_MAX) [0:1];
  Integer("Height", "-h", 480, 1, INT_MAX) [0:2];
  Integer("Red", "-r", 255, 1, 255) [0:3];
  Integer("Green", "-g", 255, 1, 255) [0:4];
  Integer("Blue", "-b", 255, 1, 255) [0:5];
  Toggle("Enabled colors", EXCLUSIVE) [0:6];
    ToggleElement("Red enabled", "-re");
    ToggleElement("Green enabled", "-ge", CHECKED);
    ToggleElement("Blue enabled", "-be");
  EndToggle();
EndGroup();
BeginGroup(2);
  Integer("Width", "-wM0", 640, 1, INT_MAX) [1:1];
  Integer("Height", "-hM0", 480, 1, INT_MAX) [1:2];
  Integer("Red", "-rM0", 255, 1, 255) [1:3];
  Integer("Green", "-gM0", 255, 1, 255) [1:4];
  Integer("Blue", "-bM0", 255, 1, 255) [1:5];
  Toggle("Enabled colors", NONEXCLUSIVE);
    ToggleElement("Blue enabled", "-beM0", CHECKED) [1:8];
    ToggleElement("Green enabled", "-geM0") [1:7];
    ToggleElement("Red enabled", "-reM0", CHECKED) [1:6];
  EndToggle();
EndGroup();
BeginGroup(2);
  Toggle("Operation", EXCLUSIVE, 2) [2:5];
    ToggleElement("Absolute diff", "-absdiff");
    ToggleElement("Add", "-add");
    ToggleElement("Blend", "-blend");
    BeginGroup(2);
      Float("Blending value", "-blendval", 0.5, 0, 1, 10) [2:6];
      ToggleElement("Divide", "-divide");
      ToggleElement("Divide into", "-divideinto");
      ToggleElement("Maximum", "-maximum", CHECKED);
      ToggleElement("Minimum", "-minimum");
      ToggleElement("Modulus", "-modulus");
      ToggleElement("Multiply", "-multiply");
      ToggleElement("Power", "-power");
      ToggleElement("Subtract", "-subtract");
      ToggleElement("Subtract from", "-subtractfrom");
      ToggleElement("Norm", "-norm");
    EndToggle();
    EmptySpace(2);
    Float("Real value", "-realval", 0, -DBL_MAX, DBL_MAX, 10) [2:3];
```

```
    Float("Imaginary value", "-imagval", 0, -DBL_MAX, DBL_MAX, 10) [2:4];
  EndGroup();
EndGroup();

Output("Output image", "-o", DATA5D, DATA5D_UCHAR, FIXED) [0:F, 2:2];
Output("Output image", "-oM0", DATA5D, DATA5D_UCHAR, FIXED) [1:F, 2:1];
```

Since we do not have another connection (that has not been merged to) the output of *ColorImageGenerator1* or *ColorImageGenerator2* the MME has added the `FIXED` parameter to the outputs of both modules, given by the last two lines in the GUI. This causes these two outputs not to be visible as an output connection. Removing the `FIXED` argument will make the connection visible, and thus connectible by other modules.

The parameter descriptions of these output connections have been modified to `[0:F, 2:2]` and `[1:F, 2:1]`. To understand what this means exactly, it is useful to understand how the routine calls are processed:

0 ColorImageGenerator2 calls routine colorTest

1 ColorImageGenerator1 calls routine colorTest

2 DoubleOperand calls routine DoubleOperand

Due to the grey (control) connection, *ColorImageGenerator2* will be executed before *ColorImageGenerator1*. The routines are executed with the following parameters

```
c2 = C_Color::colorTest (w2, h2, r2, g2, b2, re2, ge2, be2)
c1 = C_Color::colorTest (w1, h1, r1, g1, b1, ec1)
do = Arithmetic::DoubleOperand (c1, c2, re, im, op, blend)
```

The dependencies between parameters are given by the connections between the modules, and in this case are `c1` and `c2`. The routines are numbered starting at 0. Hence in the GUI the parameter description `0:F, 2:2]` denotes the function parameter (F) of routine 0, and the second parameter of the second routine. So in the example this corresponds to variable `c2`. In a similar way this holds for the parameters of the last GUI line `[1:F, 2:1]` represent the function return parameter of the first routine and the first parameter of the second routine, which corresponds to variable `c1`.

The MME has removed two input connections and overlayed them with the output connection. In the example `2:2` and `2:1` are the input connection that have been overlayed with `0:F` and `1:F`, hence the number of arguments is still the same.

The MME has also modified the argument of the last output connection from ``` ``-o'' ``` in *ColorImageGnerator1* to ``` ``-oM0'' ``` in the merged GUI. The reason is that every argument needs to have a unique name. In case multiple ``` ``-o'' ``` arguments exist the MME will append `M0` to the argument, and in case that one exists `M1` will be appended, or `M2`, etcetera, until a unique argument is constructed. The user can modify this argument by hand, but has to ensure that it contains a unique name.

6.1 Overlaying parameters

In the example there is a clear segregation of the group of parameters, not only by order, or by evaluating the first number between the square brackets, but in this case also given by groups:

```
BeginGroup(2);
  ...
EndGroup();
```

In the newly generated GUI, these group sections can be modified to generate one group. This is useful especially when the merged module is merged with another set of modules.

Both *ColorImageGnerator1* and *ColorImageGnerator2* modules contain a width and a height parameter that have to be the same for both modules. It implies that the width and height parameters will be overlayed, to achieve this overlaying, the GUI of the following commands

```
  Integer("Width", "-w", 640, 1, INT_MAX) [0:1];
  Integer("Height", "-h", 480, 1, INT_MAX) [0:2];

  Integer("Width", "-wM0", 640, 1, INT_MAX) [1:1];
  Integer("Height", "-hM0", 480, 1, INT_MAX) [1:2];
```

is modified to

```
  Integer("Width", "-w", 640, 1, INT_MAX) [0:1, 1:1];
  Integer("Height", "-h", 480, 1, INT_MAX) [0:2, 1:2];
```

6.2 Fixating parameters

In mergeModule the maximum color component is taken from *ColorImageGnerator1* and *ColorImageGnerator2* modules. The maxima are obtained using the *DoubleOperand* module, since there is no intention of changing any of these parameters, an additional `FIXED` argument is added to all parameters that contain a parameter index number, in this case `[2:x]`, where `x` is ranging from 3 to 6.

```
BeginGroup(2);
  Toggle("Operation", EXCLUSIVE, 2, FIXED) [2:5];
    ToggleElement("Absolute diff", "-absdiff");
    ToggleElement("Add", "-add");
    ToggleElement("Blend", "-blend");
    BeginGroup(2);
      Float("Blending value", "-blendval", 0.5, 0, 1, 10) [2:6];
      ToggleElement("Divide", "-divide");
      ToggleElement("Divide into", "-divideinto");
      ToggleElement("Maximum", "-maximum", CHECKED);
      ToggleElement("Minimum", "-minimum");
      ToggleElement("Modulus", "-modulus");
      ToggleElement("Multiply", "-multiply");
      ToggleElement("Power", "-power");
      ToggleElement("Subtract", "-subtract");
      ToggleElement("Subtract from", "-subtractfrom");
      ToggleElement("Norm", "-norm");
    EndToggle();
    EmptySpace(2);
    Float("Real value", "-realval", 0, -DBL_MAX, DBL_MAX, 10, FIXED) [2:3];
    Float("Imaginary value", "-imagval", 0, -DBL_MAX, DBL_MAX, 10, FIXED) [2:4];
  EndGroup();
EndGroup();
```

The MME will automatically fixate the *Blending value* parameter since it is nested within the toggle, but one is free to add `FIXED` to this parameter as well. The MME will determine if specific layout parts in the GUI can be removed. In the case all `BeginGroup`, `EndGroup`, and `EmptySpace` commands are removed.

6.3 Result of the GUI

In the resulting some names have been modified slightly, and some of the lines have been reordered. The final GUI has become as follows

```
Output("Output image", "-o", DATA5D, DATA5D_ANYTYPE) [2:F];

BeginGroup(2);
  Integer("Width", "-w", 640, 1, INT_MAX) [0:1, 1:1];
  Integer("Height", "-h", 480, 1, INT_MAX) [0:2, 1:2];
  Integer("Red 1", "-r1", 255, 1, 255) [1:3];
  Integer("Green 1", "-g1", 255, 1, 255) [1:4];
  Integer("Blue 1", "-b1", 255, 1, 255) [1:5];
  Integer("Red 2", "-r2", 255, 1, 255) [0:3];
  Integer("Green 2", "-g2", 255, 1, 255) [0:4];
  Integer("Blue 2", "-b2", 255, 1, 255) [0:5];
  Toggle("Enabled colors of 1", NONEXCLUSIVE);
    ToggleElement("Blue enabled 1", "-be1", CHECKED) [1:8];
    ToggleElement("Green enabled 1", "-ge1") [1:7];
    ToggleElement("Red enabled 1", "-re1", CHECKED) [1:6];
  EndToggle();
  Toggle("Enabled colors of 2", EXCLUSIVE) [0:6];
    ToggleElement("Red enabled 2", "-re2");
    ToggleElement("Green enabled 2", "-ge2", CHECKED);
    ToggleElement("Blue enabled 2", "-be2");
  EndToggle();
EndGroup();

Output("Output image 2", "-o2", DATA5D, DATA5D_UCHAR, FIXED) [0:F, 2:2];
Output("Output image 1", "-o1", DATA5D, DATA5D_UCHAR, FIXED) [1:F, 2:1];

Float("Blending value", "-blendval", 0.5, 0, 1, 10, FIXED) [2:6];
Toggle("Operation", EXCLUSIVE, 2, FIXED) [2:5];
  ToggleElement("Absolute diff", "-absdiff");
  ToggleElement("Add", "-add");
  ToggleElement("Blend", "-blend");
  ToggleElement("Divide", "-divide");
  ToggleElement("Divide into", "-divideinto");
  ToggleElement("Maximum", "-maximum", CHECKED);
  ToggleElement("Minimum", "-minimum");
  ToggleElement("Modulus", "-modulus");
  ToggleElement("Multiply", "-multiply");
  ToggleElement("Power", "-power");
  ToggleElement("Subtract", "-subtract");
```

```
  ToggleElement("Subtract from", "-subtractfrom");
  ToggleElement("Norm", "-norm");
EndToggle();
Float("Real value", "-realval", 0, -DBL_MAX, DBL_MAX, 10, FIXED) [2:3];
Float("Imaginary value", "-imagval", 0, -DBL_MAX, DBL_MAX, 10, FIXED) [2:4];
```

The MME has made a division between, input and output, parameters, fixed output connections, and fixed parameters. Hence even in modules with many parameters it should be relatively easy to change a GUI manually.

7 Robots

TiViPE has set up a textual robot language, and aims to interface different type of robots.

7.1 NAO robot

TiViPE has interfaced the NAO robot using Aldebaran's naoqi library. This implies that the robot commands that are used are send to the robot from the local computer, and thus an IP-address and port number are required. The IP-address is obtained by pressing the chest button on the NAO robot briefly, and the port is fixed to 9559 for all NAO robots. The *NaoRobot2* or the *NaoRobot* modules used in TiViPE thus run on the local machine. In both modules the user can modify IP-address and port in the parameter settings.

7.1.1 Hello world

A basic example is given in Figure 2

The *NaoRobot* or *NaoRobot2* modules also contain a polling parameter, it is used as a refresh rate for a new set of robot commands. Providing a number between larger than 0, for instance 50, 100, or 200 (ms) is appropriate, see Figure 2c. The Key is used to communicate a key in shared memory (other processes can make use of it) for instance *UpdateStates2* or *UpdateStates*, and in that way one can make virtual loops in a graphical environment. Both parameters therefore are used in a so-called state space model. In Figure 2 neither the polling nor the key variables are used actively. Note that the *NaoRobot* module is turned into a red module, by right clicking on the module. A red module means that the module stays active after having received input.

7.1.2 States

The example given in Figure 3 shows a basic state example, where "say(ping)" is state one, and state two is "say(pong)". In this way also one get let the robot alternate between sitting and standing. The modules need to be merged to a single module to ensure that all states have been evaluated in one update cycle. Figure 3a illustrates how 4 modules are merged to a single one my first drawing a rectangle around the modules which marks them as indicated by the green color. Clicking on the merge button will do the rest.

The program performs the following steps:

1. UpdateStates2 starts the program and activates state 1.

2. CommandState2 has state id 1 and lets the robot say "ping". when ready with execution the module activates state 2.

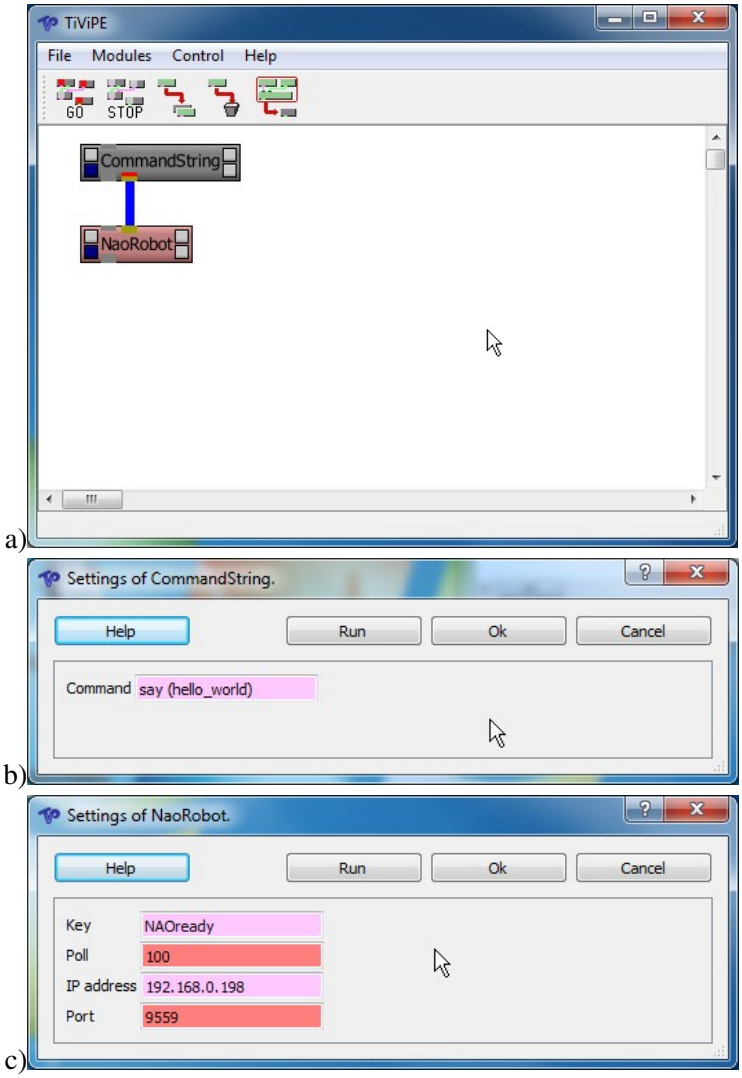

Figure 2: *(a) TiViPE network to let a NAO robot say "Hello World". (b) Parameters of* CommandString. *(c) Parameters of* NaoRobot.

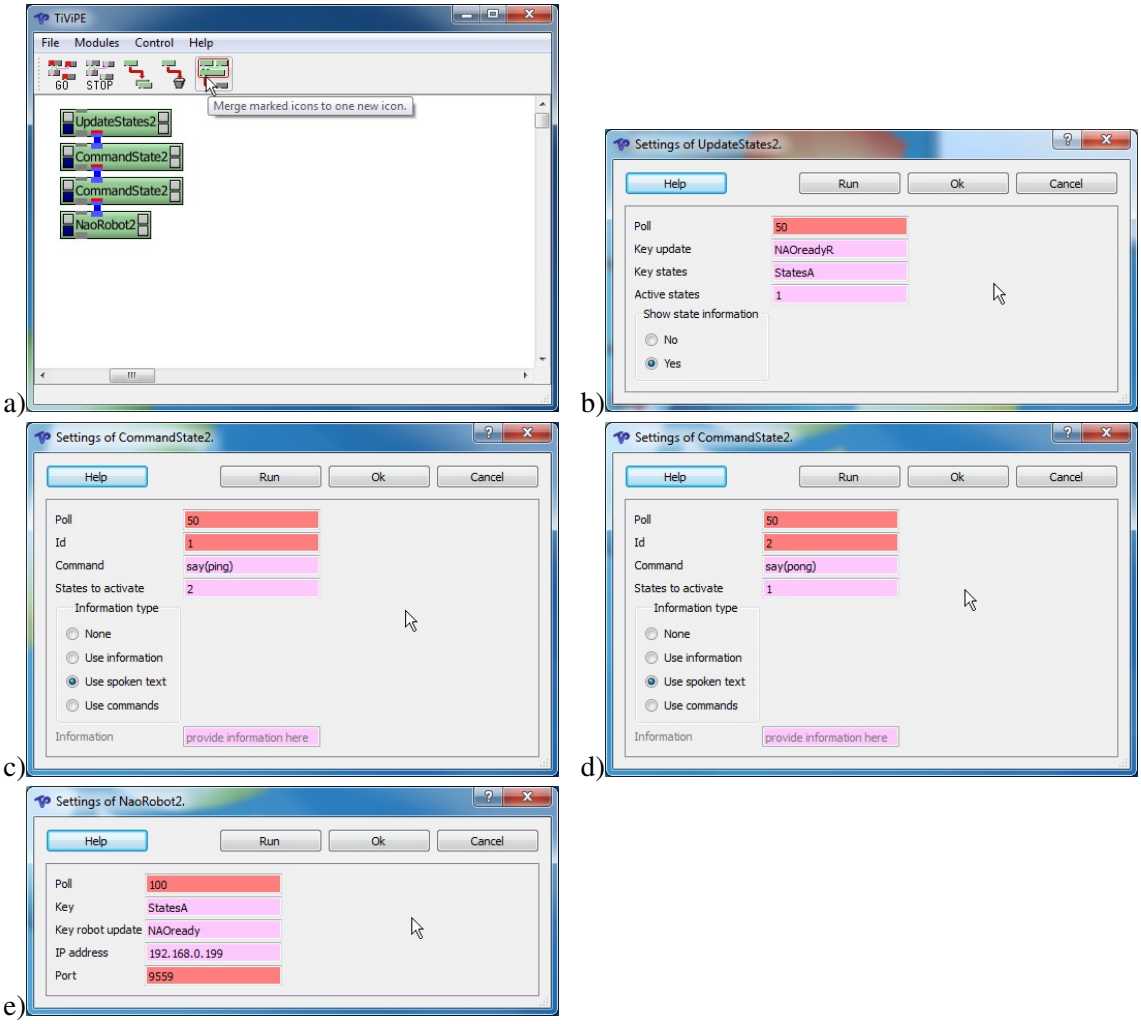

Figure 3: *(a) TiViPE network to let a NAO robot say "ping-pong" until the user stops the program. (b) Parameters of* UpdateStates2. *(c) Parameters of* CommandState2 *for ping. (d) Parameters of* CommandState2 *for pong. (e) Parameters of* NaoRobot2.

3. CommandState2 has state id 2 and lets the robot say “pong”. when ready with execution the module activates state 1.

4. NaoRobot2 receives textual commands as input, in this example either “say(ping)” or “say(pong)”. Once this module is busy, it will not accept new commands, until it is ready. When ready the UpdateStates2 module is triggered and updates the status of active states.

7.1.3 Video from NAO

The FrameGrabber module is the video module that is available on the NAO robot. This is described in the TVProbotics.pdf document [1].

7.2 Robot Operating System (ROS)

ROS is supported on Unix operating systems and is available as a set of open source components. These components all have a different nature, and require a person to have understanding of the different individual components. Scripting type of languages like python or Perl might be used to integrate between these components, and if people find a way to integrate the available packages by means of ROS, an appropriate tool has been found.

ROS but also Player and others provide software tools and libraries for robot and sensor applications. The strength of a tool comes with code re-use, and an architecture of how to include third party libraries, for instance the OpenCV library. The name of ROS is somewhat misleading, since it is neither an operating system nor a real time robot system. Despite the name it can be a useful tool if the concept is understood, from node as a client, to master and graph, and how messages are exchanged between nodes.

It is hard to tell what software is most suitable or best, and over the years and many discussions with people, the conclusion is it all depends how you use it, what your background is, and what you like. TiViPE’s preferences are easy integration of any library, support on multiple operating systems and different types of hardware. The aim of TiViPE is integration and graphical programming. The focus of TiViPE is to let computer users with little programming experience program by connecting graphical blocks. Most of these users are Windows users, having little understanding of clients, servers, messages and data structures, so with TiViPE this complexity is to a large extend hidden in the graphics, a picture says more than a thousand words.

Most commercial users have TiViPE pre-installed and get a basic course of using existing modules in TiViPE. They do not need to develop new modules since we as a company can take care of that, or one could ask a company to develop a software library, and integrate the routine calls within TiViPE. The only concept for them to learn is what robot action needs to be taken and in what order.

TiViPE is not intended for robotics in particular. The CEO of Aldebaran robotics said that he did not see much difference between TiViPE and Choregraphe. For Aldebaran that meant they had no need for TiViPE because with Choregraphe they have a tool to control the NAO robot, to TiViPE that meant that TiViPE can be used to control a NAO robot well enough, but can used for other tasks like data processing, or control other type of robots, since it is a generic integration tool that can embed any C-routine. TiViPE provides infra structures to process data in parallel (GPU by means of CUDA) and Open MP (see education section of TiViPE). As of version 2.1.0 also support is given to OpenCL, in a very similar way as support is given to CUDA. All relevant papers can be found on www.tivipe.com website in both education and research sections.

The concept of TiViPE for robotics has been expressed in for real time and parallel systems in Figure 3 and that means that all actuator commands are provided by a single module called *NaoRobot* or *NaoRobot2*. Further all standard sensory elements are obtained using *NaoSense*, video is obtained

by the *FrameGrabber* module, and audio retrieved from *NAOaudio*. TiViPE uses a simple textual command language to send instructions to the robots. The complexity of a scenario is created by the number of states, where every state might generate a command, but of course only the commands of the active states are passed on to the robot.

References

[1] http://www.tivipe.com/tvpeducation/tvprobotics.pdf.

[2] T. Lourens. Tivipe –tino's visual programming environment. In *The 28th Annual International Computer Software & Applications Conference, IEEE COMPSAC 2004*, pages 10–15, 2004.