THE USE OF WIZARDS IN CREATING CONTROL APPLICATIONS

Philip Duval, DESY MST, Hamburg, Germany
Vladimir Yarygin, IHEP Protvino, Russia

Abstract

In modern times control systems are becoming more complex at the same time that control groups are tightening their belts and staff are consisting of fewer and fewer people. Hardware specialists and machine physicists (and not computer scientists!) are frequently being called upon to develop and maintain control system applications. There are at least three approaches a control system engineer can take to make life easier for such developers: 1) 'No programming' - servers are database driven, clients are wide-interface or widget driven. 2) 'Do it yourself' - developers must be able to use an application programmer's interface (API) and know how to program in the designated language 3). 'Wizard based' – developers enter application criteria into a 'wizard' which produces code (i.e. a project), which can run as is and/or be used as a starting point for further development. We report here on the third approach in the context of the TINE control system at DESY. The TINE server wizard will be presented, which generates server-side projects in C or Visual Basic. The TINE client wizard will also be presented, which generates client-side projects in Visual Basic, Java, and DDD (DOOCS Data Devices).

1 INTRODUCTION

Consider the following two cases.

One: An existing front-end data acquisition system needs to be integrated into the control system. If the control system is of the "no-programming," database-driven variety, then the data-acquisition system either needs to be a "perfect fit" or the control system engineers might have to invest considerable time matching device drivers, IO addresses, control algorithms, command structures, alarm information, archiving information, etc. to the database. If the control-system is of the "do-it-yourself" variety, then the control system engineers will have to understand how to integrate what already exists into the control system (with possibly the same investment in time) or they can present the engineer responsible for the data acquisition system with the control system API and ask him to integrate it.

Two: A machine physicist wants to write a diagnostic application. He knows what control system data he wants to use and he wants to be able to combine and manipulate the data in various ways. Widget-driven tools are more than likely useless, unless they do exactly what the machine physicist had in mind. The machine physicist must then present his wishes to the control system staff and hope that something will happen, or he must himself be able use the control system API on some platform he can understand.

In the cases above, the hardware engineer and the machine physicist might be willing to use the control system API themselves as long as the "learning curve" is shallow or non-existent, tantamount to producing results quickly. Indeed, the control system staff itself will welcome any tools which increase productivity.

Ideally, the non-specialists would not have to learn an API at all. Rather, the desired functionality could be achieved by answering friendly questions in a setup wizard, which would create the interface (generate the code) needed for the platform in question. The specialists likewise tend to welcome such setup tools, as they provide a head start in application development (the alternative typically being to copy code from working examples and editing it into something relevant).

Finally, consider the following case.

Three: A comprehensive console application exists, covers all the needs of the operators, machine physicists, and engineers. However, as it was written in Visual Basic it cannot run on non-windows platforms and is therefore not available to Controls Group B, who use Linux machines as the standard console platform.

If the above application is wished say as a Java application, then either the control system staff must rewrite it or regenerate it.

Here too there is an ideal situation. Namely, if the application in question exists as a "meta-application" (such as an XML-file containing the information and instructions needed to render the application) then converting it to Java (or C++, or HTML, etc.) becomes trivial if there is a "renderer" capable of transferring the XLM instructions into code. Indeed, the (meta-) application becomes separated from its many different "views," be they console programs, interactive web pages, voice-applications using Voice XML, etc.

Below we shall describe the current status of the TINE server wizard and TINE client wizard in use at DESY. For more detailed discussion on the TINE control system, please see reference [1] as features of the control system will be only briefly mentioned in this article.

2 TINE SERVER WIZARD

TINE is object-based to the extent that device servers offer front-end information in the form of properties and devices. TINE properties can be read-only, write-only, or read-write and should be thought of as corresponding to methods (perhaps get/set methods) as in some cases (e.g. property "initialize"), data need not be exchanged at all. All properties are available via a variety of data access methods.

The current TINE server wizard addresses only the basic server functionality and not hardware IO. The goal is to present the server developer with a setup tool where he can input the functionality the server is supposed to have. The generated project will not have information as to the hardware IO and therefore contains numerous "TODO" statements at strategic locations in the code. Until the developer modifies the code to interface to the real hardware, the data generated for the properties will be simulated. As an example consider the input parameters shown in figure 1 below.

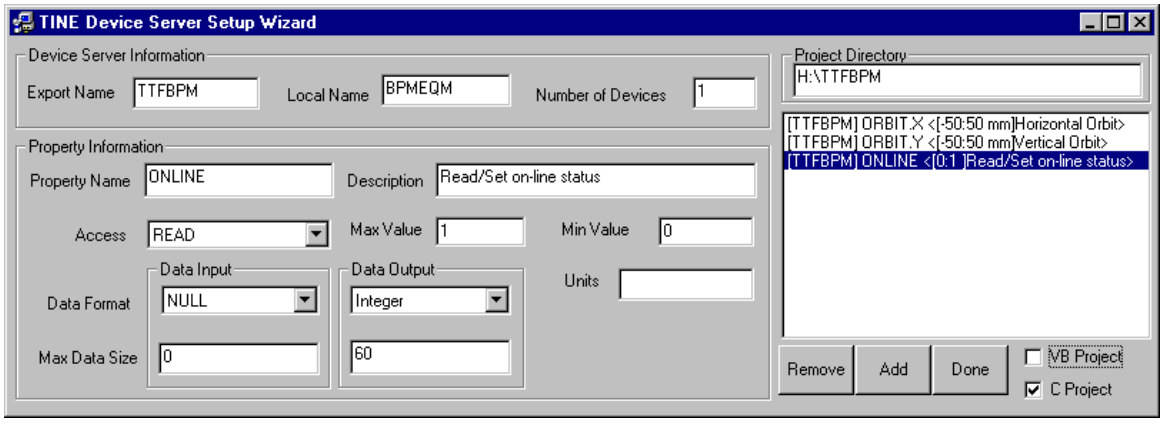

Figure 1: TINE Device Server Setup Wizard with example input.

The wizard selections above will generate either a C project and/or a Visual Basic project (LabView and HPVEE projects will be offered in the next release). A fragment of the generated C project is shown below in figure 2. In this case a generated make file will immediately build a server executable, which will happily deliver simulated data. Using this code as a starting point, the developer can quickly see what he needs to do to interface his hardware data.

The TINE server wizard is currently a "one-pass" wizard. This means that there are no "markers" within the generated code which separate the "hands-off" regions from the code sections the developer is allowed to change.

The code generation process consists of supplying the information as shown in figure 1 through a dialog process. This dialog exists as either a VB program or a TCL script. The dialog then generates an intermediate repository. The server wizard uses the TINE "exports.csv" [1] file as repository, since it is itself useful following the code generation. The repository is then rendered into the desired language.

```
int bpmeqm(char *devName,char *devProperty,DTYPE *dout, DTYPE *din,short access)
{
  int devnr,prpid,i,cc;

  /* TODO: If READ properties take input data, include code to examine the contents of din. */
  /*       If different actions need to be taken at the start or end of a link, examine the */
  /*       'access' parameter against CA_FIRST or CA_LAST.                                   */
  /*       If allow format overloading (you return different data according to the request  */
  /*       format), then replace calls to putDataFromShort() etc with the desired code.     */

  prpid = GetPropertyId(BPMEQM_TAG,devProperty);

  switch (prpid)
  {
    case PRP_ORBIT_Y:
        if (access&CA_WRITE) return illegal_read_write;
        if (dout->dArrayLength > 0)
        {
          if (dout->dArrayLength > PRP_ORBIT_Y_SIZE) return dimension_error;
          if ((cc=putValuesFromFloat(dout,g_orbit_yBuffer,PRP_ORBIT_Y_SIZE)) != 0) return cc;
        }
        return 0;
    case PRP_ORBIT_X:
```

Figure 2: Sample of generated C code give the settings shown in Figure 1.

3 TINE CLIENT WIZARD

Generating server code essentially boils down to providing functionality without worrying about visual components or user-interface dialogs. Not only are console programs visual and have a user interface, but apart from the most trivial cases they tend be specialized. Nonetheless, one has the same goal of supplying design criteria to a client setup wizard, which will generate a client-side information repository to be used to generate (i.e. render) client projects (or even running programs) for the specified platform. In this case, a .csv File is not at all suitable as a repository.

There is already much enthusiasm for wizard-driven user-application development. Two such specifications have been examined for use in the TINE client wizard, namely GLADE [2] and UIML [3]. Both of these make use of XML as information repository. The TINE client wizard is still under development. We can say here that the UIML specification seems most suitable for rendering client-side control system applications, as it has a methodology for handling events and actions. The specification for dealing with the "usual" visual toolkit component objects (i.e. buttons, labels, list boxes, etc.) is already well thought out. It remains to supplement it with data access, and "charting" components (such as ACOP [4] in the VB world). Note that UIML is XML.

Currently we are using a data access specification exemplified by the following UIML snippet:

```
<peers>
 <logic>
  <d-component id="TineTest"
       maps-to="UIMLRenderer">
    <d-method id="GetData"
      return-type="string"
     maps-to="GetData">
     <d-param id="DevServer" type="string"/>
     <d-param id="DevName" type="string"/>
     <d-param id="DevProperty" type="string"/>
     <d-param id="DataFormat" type="string"/>
     <d-param id="DataSize" type="string"/>
     <d-param id="Timeout" type="string"/>
     <d-param id="DrawMode" type="string"/>
    </d-method>
   ...
```

UIML renderers, which parse the UIML and generate the necessary code, are commercially available (Harmonia [5] offers renderers for Java, HTML, and VoiceXML for instance). However they are not generic enough to offer graphical display and data access on the one hand or to offer cross over renderering to other language groups such as Visual Basic on the other. We are therefore now in the process of writing renderers for VB, Java, and DDD [6]. These will be presented in full detail in the upcoming PCaPAC'02 workshop.

The ideal situation would be to have applications live as UIML repositories, which can be rendered not only to the platform of choice, but to the control system of choice. This requires adherence to a common UIML control system specification as well as the corresponding platform specific/control system specific renderers. As the first task of the wizard is to generate the UIML repository, we imagine a setup wizard similar to the server wizard, where the developer provides information as to what device servers need to be accessed and what should be displayed, etc. We could also imagine the ability to scan an existing, say, VB project and produce the UIML. This UIML could then either regenerate the VB project or, more interestingly, a Java project, thereby offering a way to convert VB code to Java code.

4 CONCLUSION

The TINE server wizard has been in use for the better part of this year and has been a welcome addition to the set of development tools. This is primarily because it generates default code covering the operational setup and functionality of a server. Developers should of course learn and be familiar with these aspects of a TINE server, however the wizard not only offers a tremendous head start but also immediate successful feedback.

The TINE client is still a work in progress. We expect it also to become a welcome tool for the client-side developers. Primitive applications (those which basically get data and display it, or perhaps change device settings) will be able to be generated and run immediately. More complicated applications (requiring logic) will be generated in skeletal form, i.e. as a project which compiles and runs but will have empty routines which need to be filled in with code by the developer.

REFERENCES

[1] Philip Duval, "The TINE Control System Protocol: Status Report", Proceedings PCaPAC 2000, 2000. *and* http://desyntwww.desy.de/tine
[2] http://glade.gnome.org
[3] http://www.uiml.org
[4] I.Deloose, P.Duval, H.Wu, "The Use of ACOP Tools in Writing Control System Software", Proceedings ICALEPCS'97, 1997.
[5] http://www.harmonia.com
[6] K.Rehlich, "An Object Oriented Data Display for TESLA Test Facility," Proceedings ICALEPCS'97, 1997.