# Rendering Control System Applications

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

*Abstract*

One of the main goals of the control system engineer is to provide functional, intuitive, easy-to-use-and-maintain application programs to the operators, machine physicists, and hardware engineers. One way to do this is to avoid writing code entirely and systematically generate applications via databases and configuration files, where servers are database driven and clients are wide-interface or widget driven. The resulting applications tend to be 'primitive', to extent that client applications can do little more than get data and display it, or issue commands. That is, it is difficult if not impossible in such schemes for widget A to react to an event (a mouse click) and tell widget B to do something, simple because there is no opportunity to program. On the other hand programming is itself a daunting task for many potential developers. A framework is needed where the same fundamental application can be generated but where it is nonetheless possible to expand the application via simple programming. We propose here the generation of applications by control system wizards, which react to input criteria and render fully functional code to the desired platform. This code can be run as is and/or be used as a starting point for further development. Specifically, 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). In the case of the client wizard, an application exists as a 'meta'-application in XML format, specifically UIML (User Interface Markup Language).

## 1 INTRODUCTION

Many control systems packages offer 'generators' of one form or another, the goal being to provide applications (server-side or client-side) with minimal effort. To date, most efforts involve 'tailoring', where configuration files provide input to collections of widgets and components, or other interfaces, whose behavior and functionality are pre-determined. In other words, there is no possibility of programming, if additional behavior or other embellishments are desired. In this paper, so-generated applications will be referred to as 'non-extensible' as opposed to applications where the user can extend its capabilities via programming. Such applications are termed 'extensible'. While 'non-extensible' applications might cover, say, 90 percent of the control system needs, there are a good many cases where 'extensible' applications are desired by users of the control system. Indeed, everyone knows this, and those control system packages which provide primitive generators also provide APIs for making extensible applications where necessary.

Generating extensible applications is on the other hand a good deal more involved than tailoring non-extensible ones. In this case, we need to generate code, projects, and make files, and make sure that the generated products compile, link and run. With platforms such as java, we could also consider generating a running application on the fly. This technique is called 'rendering'. Of course, if we simply stop here, the product is apt to have the same order of capabilities as tailored applications. However, for one thing, as we now have code at out disposal, we don't have to stop here. For another, many of the rendering techniques used in, say, UIML[3] offer methods of rendering dynamic cross-link behavior among a collection of components, something that is very difficult if not impossible to realize with standard tailoring. In other words, if I want to, for instance, click on chart, determine which array element I clicked on, use this as input in another chart, etc., etc., I stand a chance of achieving this if I can generate code. If on the other hand I can only push input into the components at my disposal, and must rely on the component to do something reasonable when I click on it (because I have no influence on its 'click'-behavior), I will be left with only the basic behavior of the individual widgets themselves. Server applications will necessarily require programming logic targeted to the specific behavior of the hardware represented.

Consider the following two cases.

*One (Server-side)*: An existing front-end data acquisition system needs to be integrated into the control system. If the control system is of the 'primitive' variety, then the data-acquisition system either needs to be a "perfect fit" or the control system engineers might have to invest considerable time retro-fitting device drivers, IO addresses, control algorithms, command structures, alarm information, archiving information, etc.. 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 (Client-Side)*: 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 has 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 XML 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"), a property could be simply a trigger. 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.

Note that this frequently follows what actually happens in real situations. Namely, an engineer thumbs through a catalog, chooses a hardware interface card which does what he wants and has the characteristics he needs, and implements it. As it comes with an ActiveX control, he easily builds a stand-alone data acquisition station, and then offers it to the beam diagnostics group, which wants it integrated into the control system as soon as possible. By making use of the TINE server wizard, this turns out to be an easy task .

As an example consider the input parameters shown in figure 1 below.

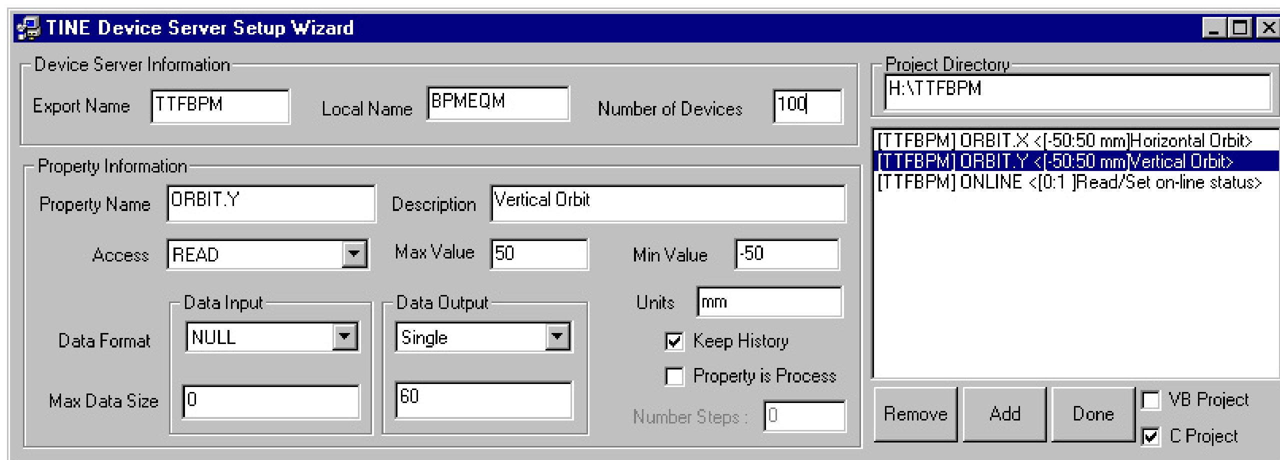


Figure 1: TINE Device Server Setup Wizard with example input.

The wizard selections above will generate either a C project and/or a Visual Basic project. Note: One could imagine generating, for instance, a LabView project as another alternative. Although the TINE ActiveX server control works fine with LabView, LabView uses a proprietary binary storage format which makes the rendering task difficult if not impossible. for project information 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 "tags" 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 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. We use the UIML specification 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]). Note that UIML is XML. What we need to standardize on is a CSML, i.e. a Control-System Markup Language.

Currently we are using a data access specification exemplified by the following UIML snippet, which specifies that a data link to obtain the horizontal orbit should be started when a button is pressed:

```
<rule>
 <condition>
  <event part-name="button1" class="actionPerformed"/>
 </condition>
 <action>
  <call name="TineLink" return-type="String">
   <param name="output" type="ACOP">TChart1</param>
   <param name="DevContext" type="String">HERA</param>
   <param name="DevServer" type="String">HEPBPM</param>
   <param name="DevName" type="String">WL197 MX</param>
   <param name="DevProperty" type="String">ORBIT.X</param>
   <param name="DataFormat" type="String">float</param>
   <param name="DataSize" type="int32">141</param>
   <param name="LinkMethod" type="String">poll</param>
   <param name="LinkTimeout" type="String">1000</param>
  </call>
 </action>
</rule>
```

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. Therefore, we have written our own TINE renderers for VB and Java. The case of DOOCS DDD[6] is also an interesting one, since the plan there is to use a CSML scheme to hold the DDD configuration information. Thus DDD would become a rendering tool in itself, or rather a tailoring tool, since DDD does not produce extensible applications. The DDD GUI builder can then also be used as an application wizard, where one can trivially generate those applications not requiring display logic (the afore-mentioned 90 percent) and generate viable VB or Java projects where more complicated display logic is required.

A sample UIML description hardly longer than the above snippet produces the rendered VB application or Java application shown below in Figure 2.

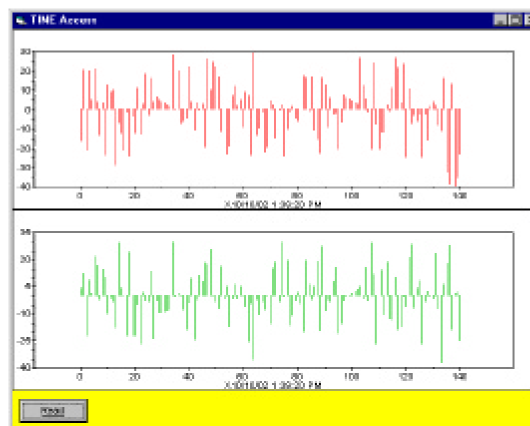


Figure 2. A trivial rendered application.

The application shown in Figure 2 is of course ridiculously trivial. Indeed the current TINE renderers are by no means restricted to such simple applications, whose functionality would by covered by primitive tailoring applications anyway. However it is instructive to have brief look at the generated code, as that is the whole point: The application is extensible. In VB for instance, the rendered event handler responsible for processing the incoming data looks like:

```
Sub TChart2_Receive(ByVal LinkIndex As Long, …
```

```
  Dim result As Integer
  If StatusCode <> 0 Then
    TChart2.Caption = "Poll fails: Err " + Str$(StatusCode)
    Exit Sub
  End If
  TChart2.ClearScreen
  TChart2.ClearText
  result = TChart2.Draw(TChart2y)
  TChart2.AutoScale True, True, True, True
  TChart2.XAxisLabel = "X" & Now
End Sub
```

With this as a starting point, one is free to remove the 'AutoScale' functionality if not desired, or to plot a reference orbit (using the ACOP ReferenceFunction() method), and so on.

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 the past two years 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. It also helps to eliminate "optimistic-programming traps" where a developer might forget to check a return code for success, etc. 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 wizard is still a work in progress but has now reached a useable state, primarily regarding development in Java. Many control system developers at DESY are familiar enough with using TINE in a VB context so that the modest head-start a wizard-generated application might give is for them frequently not worth the trouble. Those new to the system, on the other hand might have a different opinion. However, as Java is new enough to most developers at DESY, rendered applications are much welcome, since they eliminate a considerable amount of groping in the dark. Furthermore, as such a wonderful open-source java development such as Eclipse[7] does not yet provide a GUI builder, a generated GUI application is also a welcome beast.

The development and wide acceptance of a CSML will also be welcome. This could in fact provide a real basis for software sharing, at the behavioral (or maybe 'meta') level. Institute A could be using an entirely different platform, control system, display tools, etc. as Institute B. However, if both are capable of rendering CSML to their target platforms and control systems, then a CSML meta-application can be trivially shared.

## 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.
[7] http://www.eclipse.org