Using WYSIWYG GUI Tools With UML

Ilya Lipkin
677th Aeronautical Systems Group

Martin Guldahl
309th Software Maintenance Group

This article will discuss the merging of Unified Modeling Language (UML) with "what you see is what you get" (WYSIWYG) graphical user interface (GUI) tools. The topics presented—and discussion of an example with benefits and hazards—will show that the merged solution can increase productivity and provide an improved rapid prototyping platform.

This article is based on the current work done on a project at Hill Air Force Base, which is based on the Team Software Process℠ (TSP℠) practices in the CMMI® Level 5 organization. One of the requirements of our customer is that the project should use a model-driven design UML toolset, namely Rational Rose RealTime. This project is tasked with the design and development of a real-time control system based on C++ auto-generated code. In addition, the project selected to use the Tilcon Interface Development Suite (IDS) in order to rapidly and efficiently create graphical interfaces for the real-time control system that generates stunning displays to the operators of the system at a fraction of the time or expertise otherwise needed.

Overview of UML

The focus of UML is to model systems using object-oriented concepts and methodology. UML consists of a set of model elements that standardize the design description. These elements include a number of fundamental basic model elements and modeling concepts, in addition to views that allow designers to examine a design from different perspectives and diagrams to illustrate the relationships among model elements.

Several views—such as Use Case View, Logical View, Component View, Concurrency View, and Deployment View—create a complete description of the system design. Within each view, an organized set of diagrams and other model elements are visible. Diagrams include use case diagrams, class diagrams, object diagrams, sequence diagrams, collaboration diagrams, state-chart diagrams, activity diagrams, component diagrams, and deployment diagrams. Some key primitive model elements are states, transitions, signals, classes, class roles, attributes, and operations [1].

℠ Team Software Process and TSP are service marks of Carnegie Mellon University.

® CMMI is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

The object-oriented principles (OOP) used for the UML design and development are based on the idea of creating self-contained modules that describe desired functionality and interact with others through interfaces in order to create a complete system. To achieve this goal, some of the techniques available with OOP include encapsulation and abstraction [2, 3].

"The UML language is complete enough to allow the creation of auto-generated code that implements the design. The code can be generated from the system description of the model through the use of diagrams and other model elements."

Encapsulation describes the grouping of related functionality, which separates implementation from interface. The implementation details are hidden from outside users, who can only interact with objects of the class through the interface. In this way, the implementation can be more easily changed.

Abstraction provides characteristics of the object or a class that are unique and creates specific defined boundaries with respect to the currently desired solution. Abstraction allows a way of managing system complexity by hiding irrelevant details. For example, it allows for development to continue if a class is just a placeholder for future implementation.

UML Elements for Real-Time Systems Design

Designing real-time systems is a challenge. To address this challenge, an active class model element was introduced in UML. The purpose of this element was to help simplify both the design and the implementation.

The active class model element consists of a communication structure description and a behavioral description. The communication structure is described using a collaboration diagram that shows the ports through which it sends and receives messages to and from other active classes. The behavior is described using a state transition diagram that shows how the active class acts and reacts to its environment[1]. In other words, the active class is a stand-alone capsule of software that talks to its environment through ports (specified in the structure diagram), and performs *actions* as it transitions through a sequence of states (specified by the state diagram).

The characteristics of a run-time system (RTS) object and the UML active class were determined to simplify the process of real-time software design and implementation. In addition, by encapsulating calls to the operating system of the target platform within the RTS, the auto-generated implementation of the UML design can be made largely platform-independent.

The UML language is complete enough to allow the creation of auto-generated code that implements the design. The code can be generated from the system description of the model through the use of diagrams and other model elements [4].

Overview of WYSIWYG GUI Tools

The WYSIWYG concept is a well-known technique that states that the end-product will look, act, and behave the same way as it does being designed on-screen from the developer to the end-customer look and

feel. Currently, there are many tools and products on the market that make the development of the GUIs easier for embedded applications.

One of these tools is the Tilcon IDS. Although there are other tools—such as Altia Design and the Virtual Avionics Prototyping System, which are in many ways similar to Tilcon—the choice over other similar tools was determined using a CMMI Decision Analysis and Resolution (DAR) matrix [5].

The DAR was based on criteria such as customer service and technical support by the vendor, operating system neutrality, cost per use, development seat licenses, performance of the solution, ease of development, and training costs. Each was assigned a weight factor with respect to the importance for the project. In addition, a small prototype was implemented with some of these tools to serve as an input for the DAR. The winning solution was selected from the highest cumulative score.

The UML solution was designed with the idea of neutrality to the GUI development tool. Therefore, if the choice of Tilcon no longer becomes a best-fit selection, the impact to the UML back-end solution will be minimal when going with an alternative GUI.

The Tilcon IDS (see Figure 1) consists of three main components:

1. **The Tilcon Interface Builder.** The Tilcon Interface Builder is a WYSIWYG GUI design tool[2]. An interface is created by *drag-and-drop* of high-level GUI objects like menus, buttons, text boxes, labels, etc. Each of these objects has properties associated with it. These properties, such as color, font, size, meter range, etc., can be tailored to meet a given requirement. As interface development is applied to each object so the application programming interface (API) can manipulate it, the resulting GUI design is saved in a .twd file. The Interface Builder does not require any programming skills to construct a GUI. Non-programmers such as graphic artists can use the Interface Builder to construct a GUI. It is highly recommended that a naming convention be followed so that programmers using the API can access the objects in a consistent way.
2. **The Tilcon Embedded Vector Engine (EVE).** The EVE is a platform-specific engine that renders the graphical interface. It reads the same file as the Interface Builder and ensures that the GUI is exactly the same as seen in the Interface Builder. The EVE runs as a separate process from the application and manages all user events (button presses, mouse clicks, etc.) and handles the screen display. This engine is available for many embedded operating systems, such as VxWorks.
3. **The Tilcon API.** An API is provided to connect the EVE to the application (see Figure 1). The API starts and stops the EVE, facilitates communication between the application and the GUI objects, and allows objects to be created, displayed, modified, or deleted. No low-level, platform-specific graphic calls are required; all of that work is handled by Tilcon [6].

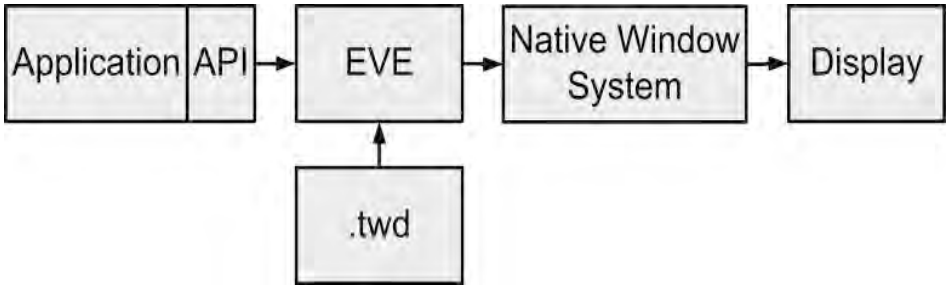


Figure 1: *Tilcon IDS*

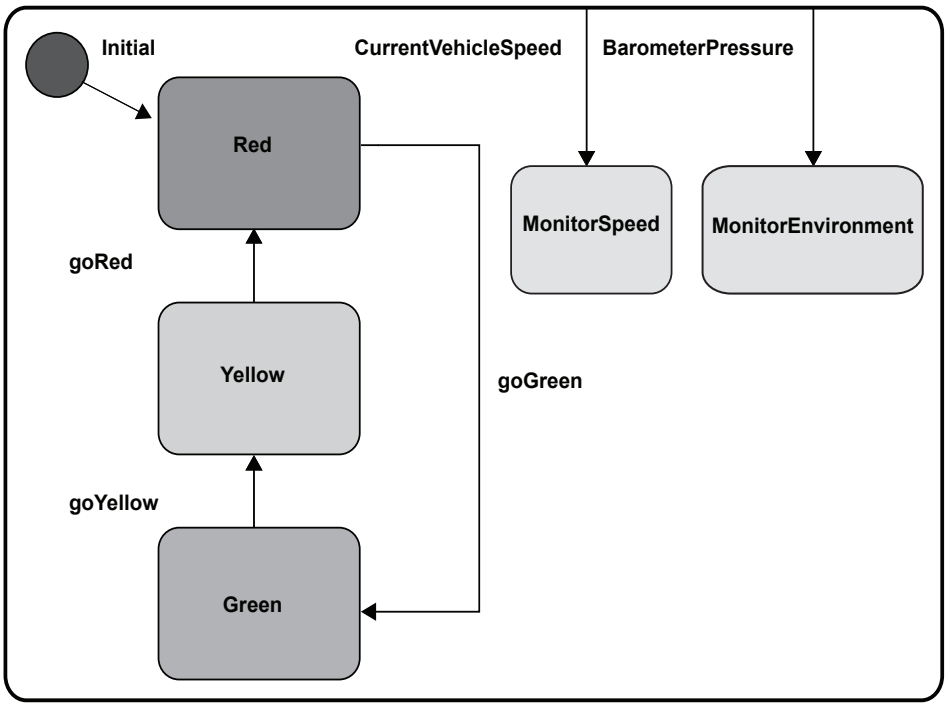
State	Description	Attributes
Initial	System not running	Timer event set to 30 seconds
Green	On timer time out event go to (goYellow) Yellow	Timer event set to 45 seconds
Yellow	On timer time out event go to (goRed) Red	Timer event set to 10 seconds
Red	On timer time out event go to (goGreen) Green	Timer event set to 30 seconds

Table 1: *State Specification Template for the Sequence of Event for the Traffic Light (SEI)* [4]

Merging of UML and a WYSIWYG Interface

Now that the tools have been presented, it is time to discuss the benefits and com-

Figure 2: *Traffic Light Advanced Monitoring System States*



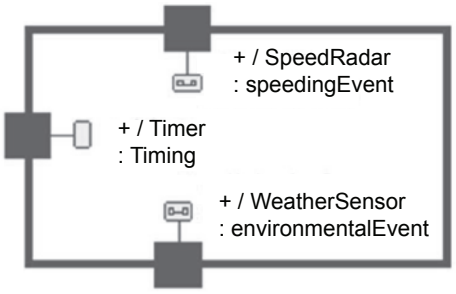


Figure 3: *Traffic Light Input Structure*

mon pitfalls of the approach. To this end, we will demonstrate a simple example using a standard traffic light with some basic gauges for the GUI presentation.

For example, consider Table 1 (on the previous page), which is an SEI State Specification Template [7]. The requirements for the traffic light state that in addition to correctly changing the lights from green to yellow to red and back to green, it will also monitor speed and pressure.

The UML solution shown in Figure 2 (on the previous page) is completely language- and platform-neutral; because the traffic light is drawn in UML, there is no code associated with it (with the exception of event descriptions). Therefore, it is left for the auto-code generation engine to translate UML to a destination language or platform of choice; in this case, C++ [8]. As a result, the amount of source code written is less than 20 lines of code for both the functionality of the traffic light and monitoring systems. The source code written in C++ consists of timer commands in the form "timer.informIn(seconds)" and Tilcon API calls to the graphical objects in the form of "TRT_SetValue(objectID, value)." The Tilcon API calls can be replaced one-to-one with other GUI API calls such as from Altia's "AtSendEvent" if one chooses a different front-end solution.

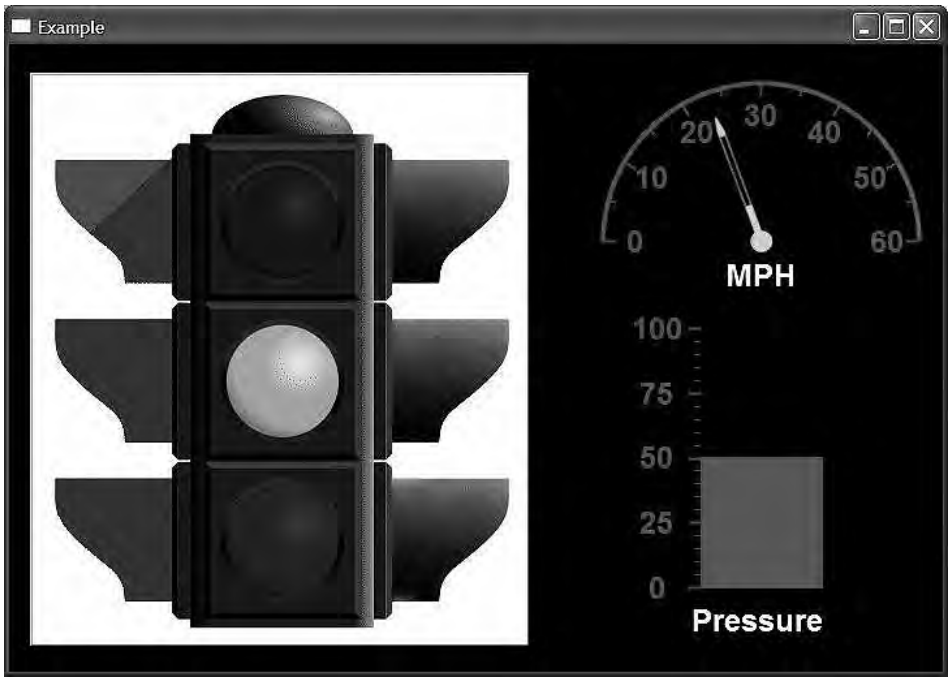
The structure diagram in Figure 3 represents system events that are used to trigger actions on the state diagram. As a result, the stimulus to these events is provided by the system itself in the case of the timer port; as well, external entities such as a radar detector or an environmental sensor feed back data updates for the other ports.

When merging WYSIWYG GUI tools with UML on a complex development project, it is of the utmost importance to exercise good OOP and spend effort to simplify and abstract the WYSIWYG GUI as much as possible from the rest of the UML solution. Abstraction will allow for better unit testing as it is possible to create wrappers that can simulate operator display and input.

The GUI in Figure 4 was quickly created with the Tilcon Interface Builder. This tool supports a drag-and-drop development methodology to create an interface, which consists of several graphical objects listed in Figure 5. The figure lists the object structure with the type and identifier of each object. The entire interface was created and tested without any UML or programming effort or an application back end.

In this example, the description of the mechanics of the speed gauge, pressure gauge, and traffic light animation can be used to demonstrate the effectiveness of this approach at the front end. Therefore, it is of interest to discuss how these objects were created in the Tilcon editor.

Figure 4: *Example GUI*



Traffic Light

The image for the traffic light in Figure 4 (created in Adobe Photoshop) was imported into the Tilcon Interface Builder and placed into a state object. One of many types available in the Interface Builder, this object type can display a different image depending on its state, which can be changed with messages sent to it through the API.

Using Adobe Photoshop with the Tilcon Interface Builder allows for an improved visual experience for the end customer, as graphics generated are generally more visually appealing at a fraction of the cost otherwise incurred if this was done in any other way (e.g., using C/C++).

In order to effectively identify the object for the UML application back end, the use of an API-unique ID such as "StateTrafficLight" needs to be assigned for the screen name. It is important to note: As more complex GUIs with hundreds of graphical objects are created in the Tilcon editor, a strict adherence to a naming convention will be required.

Speed Gauge

The speed gauge is a meter object that was created directly in the Interface Builder. This object is a standard development component that requires a minimum effort of customization. A meter object has many attributes—the range, alarm regions (green and red areas in the scale), tick marks, fonts, and colors—that were all entered in the Interface Builder. For this example, the previously mentioned attributes were slightly adjusted for the visual presentation.

Speed Gauge Needle

The needle for the speed gauge is a needle object that was also created from the standard Interface Builder object type. The needle selected is a predefined object type. Several predefined styles are also available or a custom style can be imported.

Pressure Gauge

Like the speed gauge, the pressure gauge is a standard object available in the Tilcon Interface Builder. It is an object of type "FillMeter" that represents meter position with a fill amount. The modifications for visual effects were primarily the adjustment of the visual width, font color, tick marks, and range.

Speed and Pressure Label

The speed and pressure labels are standard label-type objects. The ID, text, font, font size, and color are all attributes that were entered into the Interface Builder. The primary effort in the case of the labels was to ensure their proper alignment with their respective objects.

Once the graphical objects are entered into the Interface Builder, the GUI functionality can be verified with the Interface Builder operation "Run Test... ." This mode lets the GUI designer verify that the objects operate properly by animating their behavior without the need for a back-end solution; in this case, UML.

What Have We Gained?

The merged solution of a UML and WYSIWYG GUI development tool allows for several advanced flexibilities for the software creation effort. Most of those flexibilities are geared toward rapid development and prototyping. The separation between the front-end WYSIWYG GUI and a back-end UML provides the kind of platform development combination that can bring together technical and non-technical development efforts seamlessly.

Non-Programmers Collaboration

Using a WYSIWYG GUI design tool, it is possible to outsource the generation, prototyping, and user interaction analysis effort to usability experts, graphic artists, and other non-programmers. They no longer need to know anything about UML or any programming language. As well, a GUI object-naming convention should be followed for the project. This will allow programmers to access the objects in a consistent way [9].

Quick Prototyping

With WYSIWYG GUI tools, it is possible to adjust the user interfaces in a matter of minutes—even in the field—rather than hours or days with a comparable native programming solution. It also provides a way to easily evaluate different approaches that otherwise would have taken too long to develop.

Platform Neutrality

The development effort for the GUI is identical regardless of the deployment platform. Whether the target platform is Windows, Linux, VxWorks, or another operating system, the same solution is available and executes identically on the operating systems previously mentioned.

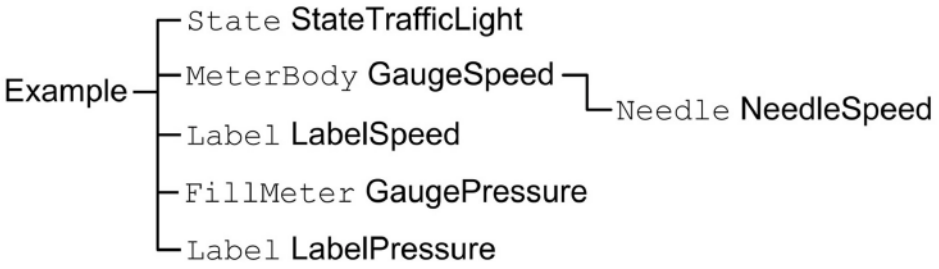


Figure 5: *Example Object Hierarchy*

Therefore, it is possible to deploy the same solution to various other platforms.

Ease of Unit Testing

WYSIWYG GUI development tools allow for automatic editor-based debugging of the GUI designs such that the prototype solution is debugged in terms of visual actions and presentations to the operator. This allows for software developers to concentrate more on the UML part of the solution and spend more time enhancing functionality and quality.

What Have We Risked?

As one might expect, there are always drawbacks to any solution. Over the course of the project, several pitfalls of this merged tools approach have been identified. Here are some of the key issues:

Merging of Design Files

One of the features that has been sorely missed was the ability to merge GUI design files. Because the design resides in a binary file format, the source control tool could not merge them. Therefore, when multiple developers work on the GUI, developers have to be extra vigilant when checking in files to avoid overwriting each other's changes.

WYSIWYG GUI Editing vs. UML Development

There is a fine line between what is considered GUI interface functionality and the UML back-end functionality. Therefore, it is the call of the system architect to identify the separation criteria. When developing in a WYSIWYG GUI tool, it is easy to get carried away with point-and-click, advance animation, and presentation development. While these are great options for some projects, they might not be for others. Some of the functionality that belongs in the UML part of the project should not be moved over to the WYSIWYG GUI development tool.

Let's say someone needs to rapidly change more than 100 objects or text references at the same time on the screen. One approach is to use the WYSIWYG GUI development tool, which results in 100 point-and-click activities; another is to implement the whole thing in the UML back-end for a loop, which can perform the same task in three lines of code. The risk of misplaced functionality can easily wipe out the gains of the merged solution.

Limited Development Language Support

Most of the WYSIWYG GUI tools have a predefined set of software languages that they support. The selection of the WYSIWYG GUI front-end might force the choice of a software language that is not in the best interest of the project, or same language translation must occur. For example, if someone wants to use Visual Basic .NET with a WYSIWYG GUI tool (such as Tilcon or Altia), they will find that it will not be supported and, therefore, be forced to reconsider going with C/C++. The implication of code generated from UML is that it forces a restriction to whatever language the UML tool generates and that this language must be compatible with the API supported by the WYSIWYG tool.

Conclusion

The example presented in this article shows that using UML for the back-end, run-time engine development and a WYSIWYG GUI builder tool for the front-end graphics development can result in overall gains in productivity and ease of prototyping. The event-driven nature of real-time UML facilitates straightforward integration with an event-driven GUI; to some extent, both solutions are platform-neutral. The example demonstrates the ease of these concepts and the integration and simplification of the problem.

One of the key benefits of this approach is that non-programmers can utilize the WYSIWYG GUI design tool to create the GUI. Requirements can be expressed in UML, and these descriptions can be used in the design and implementation of the system. As a result, the development of a complex system is simplified,

in turn minimizing risk, reducing development costs, and shortening schedules.◆

References

1. Sanderfer, Lynn. "How and Why to Use the Unified Modeling Language." CROSSTALK June 2005.
2. Bohn, Christopher A., and John Reisner. "A Gentle Introduction to Object-Oriented Software Principles." CROSSTALK Oct. 2006.
3. Dennis, Alan, Barbara Haley Wixom, and David Tegarden. Systems Analysis and Design With UML Version 2.0: An Object-Oriented Approach. 2nd ed. New York: John Wiley & Sons, Incorporated, 2004.
4. Lipkin, Ilya, and A. Kris Huber. "UML Design and Auto-Generated Code: Issues and Practical Solutions." CROSSTALK Nov. 2005.
5. Chrissis, Mary Beth, Mike Konrad, and Sandy Schrum. CMMI® Guidelines for Process Integration and Product Improvement. 2nd ed. New York: Addison-Wesley Professional, 2006.
6. Tilcon Software Limited. "Tilcon Interface Development Suite White Paper." May 2008 <www.tilcon.com/manual/Tilcon_WhitePaper.pdf>.
7. Humphrey, Watts S. A Discipline for Software Engineering. New York: Addison-Wesley Longman, Limited, 1995.
8. Webb, David R., Ilya Lipkin, and Evgeniy Samurin-Shraer. "Designing in UML With the Team Software Process." CROSSTALK Mar. 2006.
9. Altia, Inc. "How Medtronic Used Altia to Prototype and Deploy Custom User Interfaces for Medical Devices." 12 June 2008 <www.altia.com/downloads/case_studies/Medtronic_Case_Study.pdf>.

Notes

1. In the Rational Rose RealTime tool, active classes are called *capsules*, and the associated collaboration diagrams are called *structure diagrams*.
2. The Tilcon Interface Builder (see <www.tilcon.com/products/interface-development-suite/tilcon-interface-builder> to learn more) is not to be confused with the Interface Builder application for the Apple Mac OS X.

About the Authors

Ilya Lipkin is a project engineer for the 677th AESG/EN Global Hawk Simulations at Wright Patterson AFB. His current research interests include artificial intelligence, human knowledge capture and analysis, neural networks, fuzzy logic, user interface design, software engineering, UML, supply chain control, and customer relations management. Lipkin has a bachelor's degree in computer engineering, an MBA in operations management, and a master's degree in computer engineering. He is currently a doctoral student at the University of Toledo's College of Business Administration.

77th AES Wing
677th AES Group
2300 D ST, BLDG 32
Wright-Patterson AFB, OH
45433-7249
Phone: (419) 290-6017
E-mail: BookWormUT @yahoo.com

Martin Guldahl is an electrical engineer for the Common Aircraft Portable Reprogramming Equipment program, part of the 520th Software Maintenance Squadron, 309th Software Maintenance Group at Hill Air Force Base, Utah. He has more than 15 years of experience in a variety of industry and government positions. Guldahl has a bachelor's degree in electrical engineering and has taken graduate level computer science coursework from the University of Utah. His areas of interest include UML, C++, Verilog, and Perl.

520 SMXS/MXDED
7278 4th ST, BLDG 100
Hill AFB, UT 84056
Phone: (801) 775-4397
E-mail: martin.guldahl@hill.af.mil