CANopen.NET – Programmingless interconnection between GUI and control application

Peter Sjödin, Mikael Nolin and Mats Kjellberg, CC Systems.

We present the novel concept CANopen.NET – In this concept we integrate Windows GUI-programming in .NET and control-applications based on CANopen. The integration is automated, thus no programming is needed.

An increasing number of CANopen-based systems are equipped with Windows-based graphical user-interfaces (GUIs). Today, the .NET framework provides the most attractive solutions for design of GUIs both for Windows and WindowsCE based nodes. However, transferring information between the CANopen-domain (which is typically unmanaged code) and the .NET-domain (managed code) is non trivial. Traditional methods require handwritten pieces of code both in the managed and unmanaged domain for each signal (object-dictionary entry). Also, binding data values to graphical controls require hand written code. This means that adding or modifying signals to the system becomes tedious, error-prone and expensive.

1 Introduction

An increasing number of CANopen-based systems are equipped with one or more nodes with Windows-based Graphical User-Interfaces (GUIs). Today, the .NET framework and the .NET development environments provide the most attractive solutions for design of GUIs both for Windows and WindowsCE. However, transferring information between the CANopen-domain (which is typically unmanaged code) and the .NET-domain (managed code) is non-trivial. Traditional methods require handwritten pieces of code both in the managed and unmanaged domain for each signal (one signal is typically one object-dictionary entry). Also, binding data values to graphical controls require hand written code. This means that adding or modifying signals to the system becomes tedious, error-prone, and expensive and requires highly skilled software engineers.

In .NET high-level abstractions like XML-documents, web-servers and databases are easily accessible, and can be automatically bound to graphical controls. Hence, in CANopen.NET we provide a .NET-database interface via the .NET type *dataset* to the data in the object dictionary. This dataset is automatically generated from a CANopen profile-specification (a so called EDS-file). Also, the CANopen-stack is automatically configured from the EDS-file. Hence, CANopen.NET provides a programming-less interconnection between CANopen-based control-applications and Windows.NET-based GUI-applications. This significantly eases the development of CANopen-systems with GUIs.

CANopen.NET and the development environment presented in this paper are currently developed by CC Systems. And the experience reported here are from using a prototype of our development environment in a couple of customer projects.

2 Microsoft .NET

Microsoft's .NET framework [4,5] provides the state of the art whiting component based development of Windows based Graphical User Interfaces (GUIs). The .NET framework is a language and platform independent set of technologies. For resource constrained systems a trimmed down edition, .NET Compact Framework, suitable for WindowsCE, exists. Thus, the .NET technologies can be used both on full scale display PCs and smaller PDA-style devices within a vehicle.

The .NET framework has been specifically designed to support development of robust applications. It also supports a highly

flexible development environment, where components easy can be deployed, customized and assembled.

The language independence means that components developed by different vendors still will interoperate. It also enables easy porting of legacy components to the .NET framework.

.NET components are not compiled to machine dependent instructions. Instead a machine independent Intermediate Language (IL) is generated by the .NET-compilers. This means that components can be deployed on any hardware platform that has the .NET framework. Hence, .NET components and application are easily moved between different platforms.

The platform portability is especially attractive in development of embedded systems. Where much of the development is performed at a working station (e.g. an Intel CPU running Windows XP) but the target platform is a much more resource constrained environment, possibly with a completely different hardware and operating system (e.g. an Xscale CPU running WindowsCE). Using the .NET framework the application can be developed, debugged and tested on the working station. The transition to the target platform is then done seamlessly.

Central to the .NET-concept is the Integrated Development Environment (IDE) provided, e.g., by the latest versions of Microsoft's Visual Studio [9]. The IDE provides an environment where components can be constructed, deployed and assembled to applications.

3 Data binding in .NET

One of the key concepts in .NET is called data binding. Data binding means that a graphical component can be bound to a data element. The graphical component will then display a representation of that data element. Traditionally, components are bound to elements in a database, thus visualizing the content of the database in the GUI [6].

The simplest components just give a text-based representation of the data element. However, a rich set of components exists to represent data in more graphical forms. Figure 1 shows examples of four components: the two leftmost components are bound to numerical data items and can visualize the state of the controlled system in a natural and intuitive way. In the top right of the figure, a text field that can be used for both input and output of data values is shows. In bottom right, a graphical "warning light" is shown. This component is bound to Boolean data items, and different colors can be selected for both possible value of the data item.

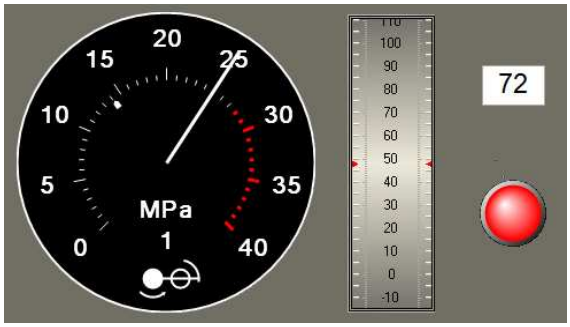

Figure 1: Example graphical components

Graphical components can be bound to various types of data sources. In CANopen.NET we use the .NET type *data set* to hold data item that should be bound to components. A data set is a Microsoft specific technology which enables storing data from a data source in RAM. The data is stored in any number of tables and it is possible to have relations between the tables. One may view a dataset as an instance of a database in a lighter version (for instance, database features like persistent storage is not available). A dataset may be typed or non-typed. In a typed dataset it is possible to address data using "names" that the compiler may verify.

4 Managed vs. unmanaged code

.NET has high emphasis on development of robust applications. Hence, it has built in run-time support for key techniques such as strong typing, array and variable bounds checking, garbage collection, and exception handling. These functions are provided by the Common Language Runtime (CLR).

Components compiled to IL that executes using the CLR are executing in the

managed domain (also called *managed code*). Managed code is protected from many common sources of run-time errors such as erroneous type casts, memory leakage, pointer errors, etc.

Any new development in .NET should preferably be done using managed code. However, often it is necessary to integrate existing, *legacy*, code in new projects. In CANopen.NET, for instance, we use a legacy implementation of CANopen. Hence, we need to call this code which is not executing in the managed domain. Such legacy code is in .NET called *unmanaged code*, and is typically written in C or C++.

Luckily, .NET provides mechanisms to integrate unmanaged code and managed code. When calling unmanaged code from managed code, the managed domain is left. The unmanaged code is, like all traditional code, susceptible to all the errors the CLR is supposed to protect your application from. And unfortunately, CLR cannot help you if you unmanaged code screws things up. This means that crossing the managed boundary introduces a risk for your application.

5 Architecture of CANopen.NET

In CANopen.NET we use the data-binding facility of .NET to bind graphical components to signals in a CANopen network. Specifically, CANopen defines an *object dictionary*, which (among other things) hold the data values sent and received over the CANopen network. CANopen.NET. We provides a run-time coupling between the object dictionary and a .NET data set, thus exposing the data values in the object dictionary to the .NET environment.

Figure 2 illustrates the building blocks of CANopen.NET. At the lowest level the CAN hardware and device drivers provide access the CAN bus [1]. The CANopen protocol provides the protocol logic needed to integrate the node on the CANopen network and to encode and decode data values to and from CAN frames [2].

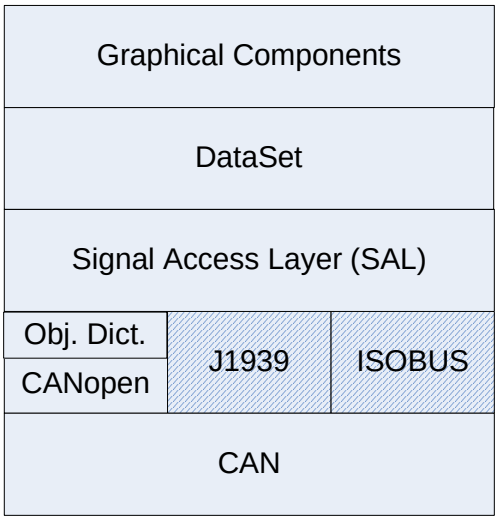

Figure 2: CANopen.NET architecture

The CANopen protocol retrieves data values to be sent from, and stores data values received to, the object dictionary. Hence, in essence the object dictionary is a database containing the current values for all data items.

In Figure 2 we also illustrate the generality of our approach, in that other CAN-protocols such as J1939 [3] and ISOBUS [7] could easily be added to our architecture. Thus the approach is not CANopen specific. Today however, we have only implemented the binding to CANopen.

Above the specific higher-level CAN-protocols, we have implemented a *Signal Access Layer* (SAL). SAL provides a uniform interface towards the data representation for each protocol. This enable us to implement support for new protocols with minimum interference to existing components in CANopen.NET.

SAL also provides the boundary between managed and unmanaged code in CANopen.NET. By having this single and simple layer as the interface towards the unmanaged code we reduce the risk that the legacy protocol implementations will interfere adversely with the graphical application.

6 Using CANopen.NET

Before using CANopen.NET to build a GUI for a CANopen-system, the CANopen network needs to be designed. This is typically done by constructing an

Electronic Data Sheet (EDS) for each node. The EDS for the node with CANopen.NET must specify all messages and data values (so called Process Data Objects, PDOs) that should be received and transmitted from the node. Since EDS is a standardized format, many tools that can aid in the construction of an EDS-file exist. For instance, the PLC tool-system from 3S [8] automatically generates EDS-files for nodes in systems designed with this tool.

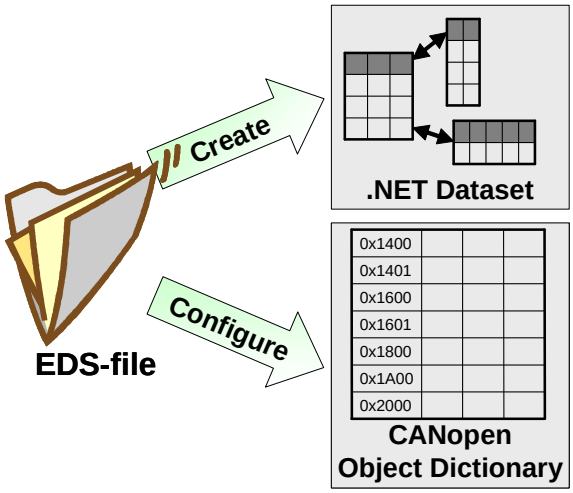

Figure 3: Configuration of CANopen.NET

In Figure 3 the process of automatically configuring CANopen.NET from an EDS-file is shown.

The next step is for a designer to build the application specific GUI interface. Figure 4 shows the Integrated Development Environment (IDE), Microsoft's Visual Studio, that we have used when developing CANopen.NET applications. This IDE has *design-time capabilities*, thus components may execute both in run time and in design time. Design time execution is done while a developer constructs an application in the IDE. These functions may include customization of component behavior and user interfaces. In an IDE this could be done in a graphical way using a GUI, thus with no actual coding needed to deploy the components. Design-time code and run-time code might or might not be identical. For example, design-time code could be a graphic layout of a control or an execution of a connection to a database.

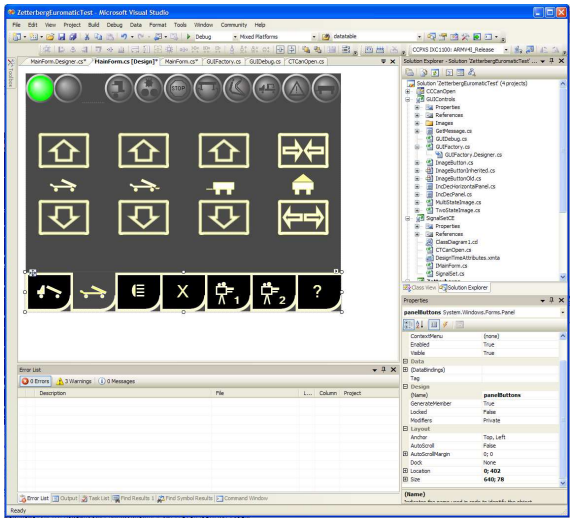

Figure 4: IDE for development of GUI

In Visual Studio, controls that have design-time behavior are collected in a toolbox which enables them to be dragged into the project you are working with. Figure 5 shows the toolbox of Visual Studio with all the available controls.

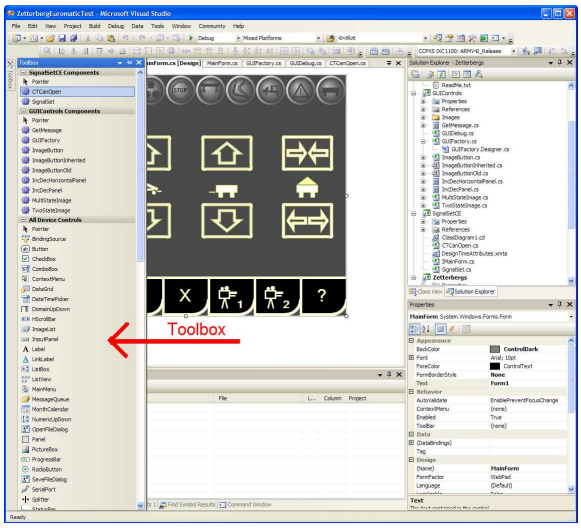

Figure 5: Toolbox with e.g. CANopen.NET control

When the control is dragged into the project from the toolbox, the control executes its design-time code. In our case when we drag our CTCanOpen control into our form, it reads the EDS-file and builds a dataset representation of signals in the file. Having a dataset allows the user to make databindings from a graphic control to an element in the dataset. It is possible to make databindings in design time using Visual Studio's databinding wizard. This wizard is show in Figure 6.

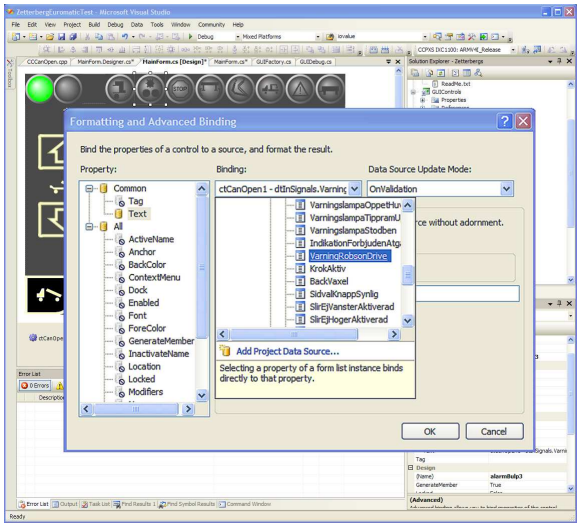

Figure 6: Databinding wizard

After constructing the GUI and binding all graphical components to the dataset we can build an executable application which may act as a CAN-slave, and it is fully capable of receiving and transmitting signals on the CAN network (using CANopen PDOs). Figure 7 shows an example GUI from a real application. The application is designed to execute on CC Systems' vehicular PC "CCP XS" which is a thin, portable PC, using PDA technology providing touch screen and possibility to use battery operation [10].

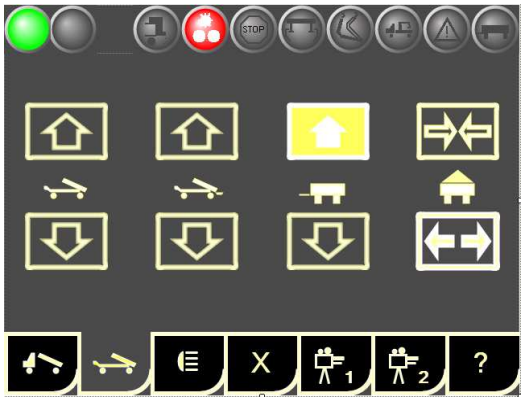

Figure 7: GUI for CCP XS application

Using CANopen.NET such an application can be designed and constructed by engineers specializing in graphical design and usability. No specialized knowledge in either CANopen or .NET-programming is required.

7 Discussion

Using CANopen.NET allows developers to use a highly domain-driven design approach for creating CANopen GUI nodes. This is due to the fact that CANopen.NET yields a clear separation of logic and GUI. One part is to program all logic, e.g., in a PLC program, and the other is to design (but not program) a GUI.

This means development costs are cut since the time to debug and maintain the application is very much decreased. Having CANopen.NET we work with a high level abstraction of signals that are transferred from CAN into the managed domain, and we need not to test the framework of the signal handling since this handling is done automatically.

The GUI designer needs only to work at the managed side which simplifies the developing process. One need not to care about CAN protocol or interoperability issues concerned having unmanaged and managed components. Further, there is no need for the GUI designer to write code, if he/she has access to graphic components.

All this means we generally get better quality in our applications since designing and maintaining CAN-networks often is tedious and error prone due to changing of signals to use only effects one well defined part of the project, and not various parts as is used to do.

8 Conclusion

We have presented CANopen.NET, a technology for separate design and implementation of CANopen-based control systems and their Graphical User Interfaces (GUIs). Using CANopen.NET a GUI-designer can construct a GUI for the CANopen-based control system (possibly) without writing a single line of code.

Specifically, no coding is needed to get access to the CANopen data values in the graphical .NET environment. Data values appear in .NET datasets, which can be directly bound to .NET graphical components without any programming.

The dataset and the configuration of the CANopen protocol is done automatically by CANopen.NET tools from an Electronic Data Sheet (EDS) specification of all the data values that should be sent and received by the GUI-node.

The result of using CANopen.NET is significant reduction in cost and complexity of constructing CANopen systems with GUIs. By separation of concerns, engineers can focus on their main task (e.g. designing a control system or designing a user interface), significantly simplifying project management. Further, engineers need not be specialized in both CANopen and .NET development; simplifying the process of staffing the project.

Also, the quality of resulting application is enhanced since error prone, manually written; code for handling signals at different levels of abstraction is eliminated by automatically generated configurations from the CANopen.NET tools.

References

[1] Road Vehicles - Interchange of Digital Information - Controller Area Network (CAN) for High-Speed Communication, International Standards Organisation (ISO), vol. ISO Standard-11898, Nov 1993.

[2] CANopen - Application Layer and Communication Profile, CAN in Automation DS301, EN 50325-4, http://www.canopen.org, 2003.

[3] SAE J1939, Joint SAE/TMC Electronic Data Interchange Between Microcomputer Systems In Heavy-Duty Vehicle Applications, http://www.sae.org. CAN

[4] Microsoft .Net Compact Framework, Andy Wigley and Stephen Wheelwright, 2003

[5] .NET Common Language Runtime Unleashed, Kevin Burton, Sams Publishing, April 04, 2002

[6] ADO.NET Cookbook, Bill Hamilton, O'Reilly, September 2003

[7] Stone Marvin L,, ISO 11783: An electronic communications protocol for agricultural equipment, ASAE, Feb 1999

[8] CoDeSys home page, 3S – Smart Software Solutions. http://www.3s-software.com/index.shtml?oem1

[9] Visual Studio 2005. Microsoft. http://msdn.microsoft.com/vstudio/

[10] CC Pilot XS - a versatile and robust on-board computer for Win CE and Linux. CC Systems. http://www.cc-systems.com/en/index.php?option=content&task=view&id=43