Can't Decide? Use them all!

Gary Johnsey[1]

Abstract

Students often study programming languages in a limited context, totally focusing on one language at a time. However, large, distributed software systems often rely upon diverse programming languages This is particularly true of Internet applications where a single client session might use products written in 5 or more languages. Interfacing and integration issues can create significant problems. An exercise was designed for the Computer Engineering Technology Program at the University of Southern Mississippi highlighting some of these issues . The exercise involved the use of a large number of languages across three hardware platforms, e.g. Perl, Visual Basic, C++, 68hc11 Assembly Language, Small C. It specifically focused upon data delivery and control over an interactive network, the Internet. The student's task was to design a software system that would allow a browser-active client to display data collected by a remote microcontroller. The client was also to provide control inputs, i.e. setpoints, for the microcontroller. The paper focuses on the academic exercise rather than the technical details.

Introduction

A capstone exercise was designed after observing that senior and other projects dealt primarily with hardware and hardware interfacing but rarely with issues deriving from diverse software or software interfacing. The exercise was conducted in the summer of 2000 with Computer Engineering Technology (CET) students at the University of Southern Mississippi (USM). Its purpose was to create a complex, network-based, software-systems by combining diverse language components, i.e. by throwing a lot of languages at a problem to see how well they work together. How effectively each software component participated in the overall system was more important than the strengths and weaknesses of the individual programming languages. The goal of the exercise was to focus on the interfaces/integration of the software components and not upon their individual complexity. This paper might have been entitled "Interactive Control of Remote Data

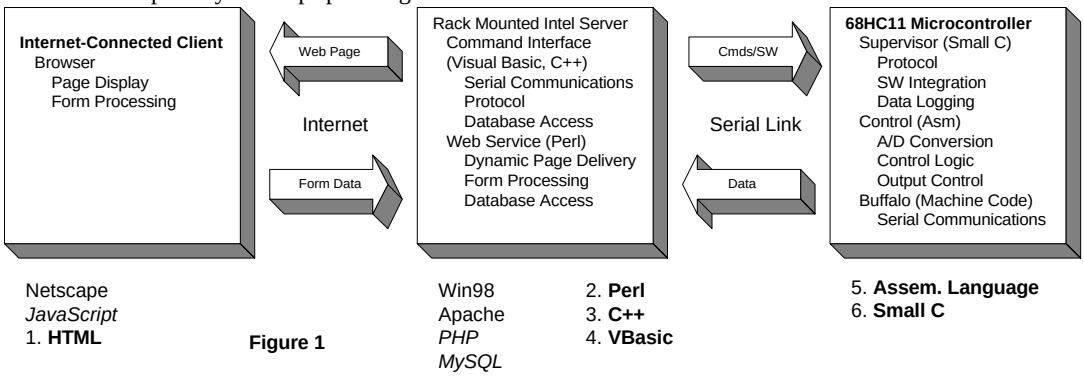

Figure 1

[1] Assistant Professor of Computer Engineering Technology, University of Southern Mississippi, P.O. Box 5137, Hattiesburg, MS, 39406-5137.

Logging" but that title would have been misleading, i.e. highlighting the hardware functions and not the software interfacing.

Figure 1 illustrates the complete networked control loop along with the relationships between the hardware and software. The students' task was to collect temperature data using a microcontroller single-board-computer (sbc). This sbc was to be connected by a serial link to a network server, providing the data on request (and accepting commands and/or control data). The server-based software was to store the data in either a flat file or relational database file. This data was to be subsequently included in a web page. The task provided to the students was in fact only guidance. Student teams were to evaluate different solutions and select individual approaches and programming languages. The only real requirement was that each major link, local-serial or Internet, be interactive. Figure 1 represents one choice of a hardware configuration. Other choices included industrial, wall-mounted cases for the server and i386SX sbc's for the data logger. The exercise also included hardware installation and configuration, e.g. motherboards and interface cards, along with operating system installation, e.g. Windows 98 or Linux. However, this paper does not address the installation and configuration portion of the exercise.

Background

The capstone project in the CET program at USM does a good job of incorporating electronic and computer-hardware knowledge from earlier courses. However, the individual projects generally use no more than one programming language, and practically never interface different language products. The required Computer Science courses focus on individual programming languages with perhaps a discussion of weaknesses and strengths but no integration or interfacing. This academic experience does not adequately represent the environment of modern, network-based applications. Each team of 2 to 4 students was given general guidance, but few restrictions. Their hardware and software solutions varied, but certain design elements were popular. The following description represents a typical example.

System Description

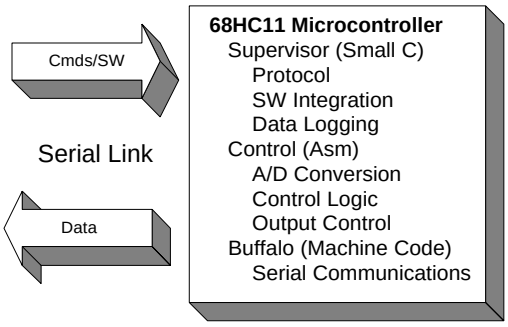

Figure 2

The system description begins with the microcontroller software components. Although the teams tended to develop the different software components concurrently, installation and testing progressed from the microcontroller to the server, then to the client browser, and back along the same path to the microcontroller. The following discussion uses this same sequence, starting with the microcontroller-based components.

Microcontroller - Small C, Machine Code, and Assembly Language

Consider the 68hc11 sbc, Figure 2, and its three software components, Buffalo, Control, and Supervisor. Buffalo comes pre-installed on the Axiom sbc the students use, and provides a development environment with typical monitor-type features, memory dump, in-line assembly, single-stepping, etc. It is normally used in an interactive terminal mode, but works just as well receiving commands from software over the serial link. Buffalo was used as a stand-in for Supervisor during the prototyping stage. In other words a Visual Basic (VB) program ran on the server using timer and communication controls. It provided a steady flow of data/commands,from a script to aid prototyping and development. Buffalo was not needed after the Supervisor was installed but portions were called as subroutines.

The Supervisor, written in Small C, was to replace Buffalo with a higher level protocol, interfacing with the communications link on one hand and the control functions on the other. However, as indicated above, the Buffalo machine code would never be totally abandoned. Supervisor would rely upon Buffalo's interrupt-driven, communication subroutines. This reliance actually enhanced the integration goals. These subroutines are not well documented, and their use required reverse engineering and interfacing between machine code and the Small C based Supervisor objects. This Small C compiler provides a limited subset of the C language, and has two specific advantages for this application. First, it produces reasonably small object code. Second, it produces Assembly Language objects, not machine code. This made it very easy to study the parameter passing conventions. The students could easily mimic these conventions when using Assembly Language to write the Control component as discussed next. In other words the compiled objects from Small C are in Assembly Language and are, thus, easy to study. The inefficiencies of this non-optimizing compiler were obvious, especially in such a small implementation of C. Both the 68hc11 Assembler and the Small C compiler were used in an earlier course, so there was little surprise over these observations. The Supervisor code was never developed beyond the rudimentary prototype (possibly due to distractions). However, even in this rudimentary form it provided the necessary interfacing between the communications, data collection, and control functions.

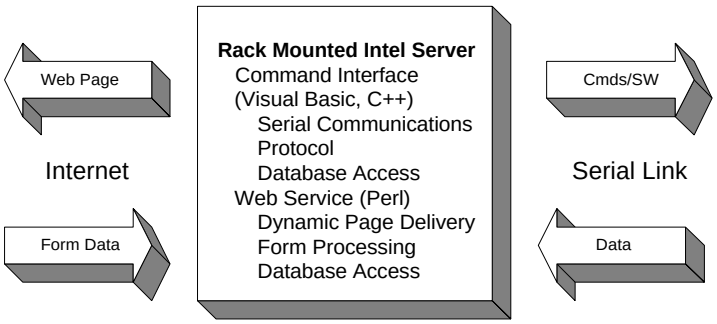

Figure 3

The Control software was almost as simple as the rudimentary Supervisor, primarily because the emphasis was on communication and interfacing, and not on control algorithms. When installed, the Control software responded to a timed interrupt, performing analog to digital conversions on 4 channels and storing the values. The Supervisor reported these microsecond-old values when requested. The intent was to provide the control engine with a set point (temperature) to be maintained with local feedback and control. In this first effort potentiometers (in place of thermistors) and light-emitting diodes (in place of heating elements) were used. The students concentrated on the more risky tasks, i.e. the Internet interactivity.

Internet Server - Visual Basic, C++, Perl

The microcontroller, as discussed above, hosted two primary software functions, communications (Supervisor) and productive input/output (Control). The server hosted similar functions, serial communications (Command Interface) and productive Internet input/output (Web Service). See Figure 3.

However, the server-based, software design was considerably different from the microcontroller component interfaces. The interfaces between the microcontroller components were "tight," i.e. direct subroutine calls or shared memory cells. In the server-side design, data was exchanged using the operating system's file-handling facilities, a loose coupling. Data from the microcontroller-based Supervisor was stored in a simple file, a flat file, by the Visual Basic Command Interface. Data from this file was read by the Perl interpreter (Web Service) for inclusion in a Web page. Data types in the two languages, VB and Perl, do not necessarily have compatible structures. The design of the shared file required considerations supporting the purpose of the exercise, i.e. language integration and interfacing. I allowed a simple solution in this exercise, a text file, but will probably require popular data structures in the future.

The Command Interface on the server performed a similar function to the microcontroller's Supervisor. It provided serial communications, and a command protocol. It also delivered a script that directed the operation of the microcontroller. This script, a command list, was simple and was stored within the Visual Basic program. However, it could have been more complex and stored in a separate file. Visual Basic proved to be well suited for prototyping, e.g. it was easy to make quick modifications, and to immediately observe the results.

The design of the Web Server component was more varied. The primary function of this component was dynamic Web-page creation. The most common approach in industry, and the one used by the students, is to use the Common Gateway Interface (CGI) facilities available on Web servers such as Apache. There is a trend for modular expansion of the server to directly interpret such languages as PHP (Originally named "Personal Home Page" but its use has grown beyond personal applications) resulting in faster page delivery. When the Web Server software detects that a CGI compatible program is to be used it delivers the output from that program directly across the Internet to the client browser. Essentially the only requirement is that the CGI program be capable of using standard output and standard input (if data is to flow in both directions). You can use Perl, Visual Basic, C++, and other languages to create a CGI program. At least one team in this exercise used each of these languages for this purpose. Perl proved to be a powerful text processor. Visual Basic proved the easiest to use. C++ proved to be reasonable at this task. The CGI technology may be replaced in the near future if current trends continue. Candidates include Active Server Pages (ASP) and PHP (PHP uses a Web-page embedded script that is activated server side). PHP development is extremely easy and fast. It even emulates some of the text processing features of Perl. There is amply opportunity to reflect new software technology in this type of exercise.

Internet Client - HTML

The pages delivered by the Web Service component were delivered across the network and displayed using the Netscape and/or Internet Explorer browsers at the client computer. These pages contained simple HTML tags, graphics, data, and a form for providing commands and setpoints for the microcontroller. Client-side languages, e.g. Javascript, Vbscript, could have been used but were not in this first effort. It is not entirely clear how their use would have exemplified software interfacing.

The HTML code was generated at the server, but the data was displayed and the input collected at the client computer. The cross-platform diagnostics and troubleshooting provided a final interfacing challenge for the students.

Conclusions

By relying upon previous course work it was possible to design a practical exercise emphasizing the interfacing and integration problems of using a large number of programming languages for problem solutions. The interfaces requiring design decisions by the students are identified in Figure 4.

There are several changes/enhancements under consideration for the next exercise.

1. The microcontroller's output should be interfaced to functional hardware, but not as to divert too much attention from the software issues.
2. The Control software should contain a feedback loop, perhaps imported from an earlier class.

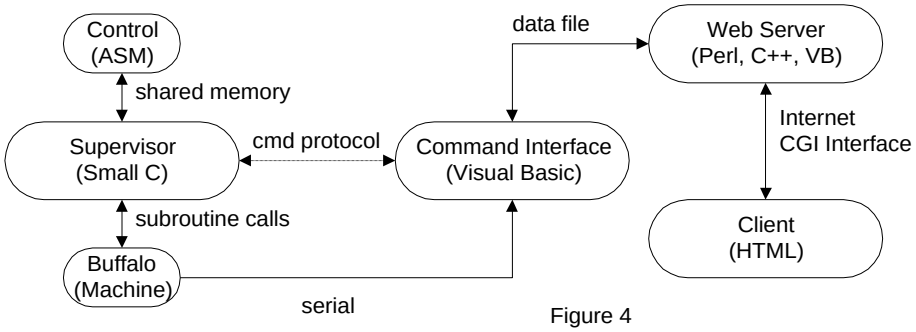

Figure 4

3. The Supervisor protocol should be expanded, e.g. software downloads, individual output pin controls.
4. The server-side data file should mimic more typical data files with various data types.
5. Alternative languages should be considered, e.g. Java, PHP, ASP, XML.
6. The HTML form should include options to display the data, e.g. a listing, a graphic, a histogram.

References

Laura Parker Roerden (1997) *Net Lessons: Web-Based Projects For Your Classroom*, O'Reilly, Sebastopol, CA.

Eric F. Johnson (1996) *Cross-Platform Perl*, M&T Books/Henry Holt and Company, Inc, New York, New York.

Walter Savitch (1999) *Problem Solving with C++*, Addison Wesley, Reading, Massachusetts.

Chris Loosley, Frank Douglas ((1998) *High-Performance Client/Server,* John Wiley & Sons, New York, New York.

Gary Nutt (1999) Operating System Projects Using Windows NT, Addison Wesley, Reading Massachusetts

7. Gary Johnsey

Gary Johnsey is an Assistant Professor of Computer Engineering Technology at the University of Southern Mississippi. He received his BS in electrical engineering from Auburn University, his MS in telecommunications from the University of Southern Mississippi and his MS in electronics engineering from the University of Missoury at Columbia. His interests lie in the areas of embedded computer hardware and software. He is a member of ASEE and often serves as division officers for the Southeastern Section of the ASEE.