Are Fit / FitNesse Appropriate for Biomedical Engineering Research?

Jingwen Chen1, Michael Smith1, Adam Geras1,2, James Miller3

1 Electrical and Computer Engineering, University of Calgary, Calgary, Alberta, Canada T2N 1N4
{JinChen, SmithMR} @ ucalgary.ca
2 Ideca, Calgary, Alberta, Canada
AGeras@ucalgary.ca
3 Electrical and Computer Engineering, University of Alberta, Edmonton, Alberta, Canada T6G 2V4
JM@ece.ualberta.ca

Abstract. The Agile development of small real-time systems has many aspects not found with standard business desktop systems. A prototype test driven development tool for embedded systems (E-TDDunit) has been developed around several hardware-oriented extensions to CPPUnitLite. However xUnit tests are written in the language of the solution which, at the embedded level, often implies “with extensive knowledge of the domain”. This is problematic in the development of biomedical engineering research applications. The customer (the “doctor” or other “professional medical researcher”) just does not have that knowledge. The biomedical application is often prototyped within MATLAB before movement down to the “plumbing level” on a high-speed, highly parallel, processor to meet the requirement for real-time application in a safe and secure manner “in the surgical theatre” or “on the ward”. We discuss issues around using Fit and FitNesse as communication tools under these circumstances.

1 Introduction

Imagine you are developing in a biomedical engineering environment where a reliable, high performance, medical instrument must be produced “for the surgical theatre” or “on the ward”! Signal processing algorithms will be needed to monitor, analyse and report on patient life signs. You know that some of the required algorithms have already been developed in a research laboratory (using MATLAB). These algorithms need to be migrated to, and then validated on, an embedded platform using a combination of C++ and assembly code in order to meet strict time constraints.

Even in such an environment, you will still want to undertake unit testing, but now the “testing requirements” differ significantly from the standard “desktop” business problem. But a really key issue is “How are you going to communicate with the customer (the “doctor” or “professional medical researcher”) while you apply a test driven development approach to explore this medical instrument’s design?” There have been a number of notable efforts in migrating Agile ideas into the embedded
environment [e.g. 1]. However using xUnit embedded tools, such as E-TDDunit [2], to support Agile development is problematic. The xUnit concept, by design, requires tests written in the language of the solution. But now this language involves MATLAB, C++ and assembly code. As in the business area, the bio-medical engineering customer just does not have that sort of experience level. Another mode for communication is required!

The purpose of this experience report is to discuss issues surrounding the application of Fit [3] and FitNesse [4] in a biomedical engineering research and / or application development environment. To meet the space requirements of this experience report, it is taken as a “given” that, in the long run, we will be able to show that all of the advantages of the Fit table format as a tool for communicating with the customer will move from the business to the biomedical research environment.

2 Fit and FitNesse in a Biomedical Engineering Environment

Our long term goal is to analyze (in real-time) a series of images coming from a magnetic resonance imaging scanner used to determine cerebral perfusion parameters (blood flow) for patients suffering from stroke. Stroke is a major disabler and killer across the world; and its financial impact is staggering. Algorithms from such a study [5] involve deconvolution, signal aliasing issues and modeling techniques, combined with issues arising from distortions from noise removal and image segmentation, with everything running at full speed on highly parallel processors; far beyond the scope of a five page report. We will therefore, for illustrative purposes, choose a greatly simplified device to illustrate the concept of communicating with the customer through Fit with the product development occurring in MATLAB, C++ and assembly code.

Assume that the biomedical application requires the determination of the temperature of a doctor’s stethoscope. Cold stethoscopes are a common complaint received from patients! The basic hardware involves using a TMP03 thermal sensor [6] which produces a voltage pulse (voltWidth) whose width is proportional to temperature. Conversion from pulse width to actual temperature is to be performed using a function (CalculateTemperature( )) running on an embedded system powered by a BlackFin (ADSP-BF533) processor using the VDK real-time operating system [6].

Fig.1 provides an overview of the biomedical Fit / Fitnesse test and development system. The customer tests are described through tables stored on a Wiki-page. The Runner takes the test data and fixture name, and passes these to the FitServer for execution. Customized fixtures are available to use the MATLAB API to start and run the MATLAB engine to validate the MATLAB code associated with the code (ConvertTemperature.m). Additional customized fixtures are available for running the code on the embedded platform from the host computer. For this example, this required the development of fixtures to use the Visual DSP development environment (VDSP IDDE) [6] to compile code, download and run that code (ConvertTemperature.dxe) on the embedded platform over a communication link.
3 Using *Fit* within a biomedical research development environment

There are a number of stages the developer must work through to use the proposed *Fit* approach to designing, developing and testing a biomedical engineering product.

1. Consulting with the customer to produce *Fit* test tables.
2. Developing standard fixtures to use the test data from the test tables.
3. Validating the linkage between the code under test and *Fit* through a method stub running in the same environment as *Fit* / *Fitnesse*.
4. Extending the method stub to use the *MATLAB* API to pass the test table data to the *MATLAB* development environment.
5. In *MATLAB*, developing the code to meet the tests.
6. Extending the method stub to use the embedded development environment’s API to pass the test table data (over a communications link) to the external device.
7. Using the *MATLAB* code as a template, develop the necessary C++ and assembly code to pass the known functional tests, and any additional non-functional tests. Key test validation issue – *double precision floats* *MATLAB* variables become *fixed length integer* for speed on the embedded system.

**Stage 1:** Assume that the developer and customer have worked together to provide a two column test table (**DSPFixture.CalculateTemperatureTestFixture**) that will be used to validate the function *ConvertTemperature*(). The columns are *voltWidth*, detailing input values from the temperature device, and *Temperature*, the expected results.
Stage 2: Standard fixture components can be used for all three platforms

```cpp
#include “necessary_Fit.includes.h”
class CalculateTemperatureTestFixture: public ColumnFixture 
{
  // Make voltWidth variable and outputTemperature() function known to Fitserver
  public:  explicit CalculateTemperatureTestFixture (void){
    PUBLISH(CalculateTemperatureTestFixture, unsigned, voltWidth);
    PUBLISH(CalculateTemperatureTestFixture, float, outputTemperature);
  }
  private: unsigned voltWidth;
  float outputTemperature (void){ // Call the function to calculate temperature
    return(CalculateTemperature (voltWidth));
  }
};
```

Stage 3: Validation of the fixture code through a method stub

```cpp
float CalculateTemperature (unsigned voltWidth){
  // Necessary code to simulate calculation of temperature using voltWidth
  return dummy_temperature;
}
```

Stage 4: Extending the method stub to activate the MATLAB Engine requires a series of calls through the MATLAB API to first activate, then to transfer data to global data within the MATLAB environment, run the code and finally retrieve the result.

```cpp
#include “API_interface.h”  // API Environment
float CalculateTemperature (unsigned voltWidth) {
  // Create an API application project, then build the code.
  API_Interface  API
  API.CreateADSPApplicationProject ( );
  API.BuildAndLoadProgram ( );
  // Transfer voltWidth value. Use a communication link on embedded system
  API.PUBLISH (voltWidth,”API_voltWidth”);
  API.RunProgram ( ); // Run the code
  // Read (transfer) the test result (From embedded platform back over the COM link)
  float outputTemperature = API.PUBLISH (”API_Temperature”);
  return outputTemperature;
}
```

Stage 5: The reader is referred to [7] for a comprehensive example of developing MATLAB algorithms for an assisted hearing device using Fit.

Stage 6: Extending the method stub to activate the embedded platform is essentially equivalent to Stage 4 except this stage uses the VDSP API.
Stage 7: The communication between the fixture (on the host machine) and the external embedded platform (Blackfin ADSP-BF533) is through global variables.

```c
.byte4 _Embed_voltWidth, _Embed_Temperature; // global variables changed by Fit
_main:
  // PUBLISHed over COM link
  // Pass global variable as a parameter to CalculateTemperature()
  P0.L = lo(_Embed_voltWidth);     P0.H = hi(_Embed_voltWidth);
  R0 = [P0];
  CALL _CalculateTemperature;   // Perform required function
  // Prepare result from CalculateTemperature( ) for Fit access over COM link
  P0.L = lo(_Embed_Temperature); P0.H = hi(_Embed_Temperature);
  [P0] = R0;
  RTS;
```

4 Discussion and Conclusion

We have demonstrated, proof of concept level, Fit fixtures modifications that could permit improved communications between the customer and technical developer in a research environment where MATLAB and embedded system interfacing are de rigueur. However usability issues based on this initial experience are extensive. The development of customized column fixtures is one possible solution. An alternative is to take a different approach where, instead of direct interfacing of Fit and FitNesse into these environments, fixtures are developed to allow use of the xUnit tools MATLABUnit [8] and E-TDDUnit [2]. Financial support was provided by University of Calgary, Analog Devices and the Natural Sciences and Engineering Council of Canada (NSERC) through a Collaborative Research and Development grant (CRD 299423-03). MRS is Analog Devices University Ambassador.

References

Submitted XP2006, January 2006