Instrumentation Control and Acquisition Software (ICAS) for EPR Spectroscopy Experiments

Matthias K. Miehl, Malte H. Reitzer, Joseph J. Ratke, John D. Gassert
National Biomedical EPR Center, Medical College of Wisconsin, Milwaukee, WI, USA

Introduction

Main Issues

The instrumentation used in EPR experiments typically has the following control and data acquisition issues:

- Requires precise instrument timing and consistent settings
- Record of settings (instrument selection and parameters)
- New program for each EPR experiment
- Consistent program structure for all EPR experiments

ICAS addresses these issues.

ICAS is an in-house development of the National Biomedical EPR Center at the Medical College of Wisconsin. It is designed for optimal *usability* and *long-term maintainability*. It combines the functionality of previously developed in-house applications used to control EPR experiments and focuses on improving the overall program structure to offer the following benefits:

- Program Structure
  - Enable modularity and flexibility
  - Ease of future modification by providing a path for expansion
  - Improve maintainability

- Graphical User Interface (GUI)
  - Simplified displays
  - Common interfaces across experiments
  - Increased responsiveness to operator actions
  - Post-processing options

- Improved Execution Responsiveness
  - Take advantage of 64-bit platforms
  - Take advantage of multi-core processors
  - Take advantage of multi-threading and parallelism
  - Avoid polling wherever possible

- Data Acquisition
  - Calculation of optimum acquisition rate and record length for a high speed digitizer
  - Baseline subtraction in hardware on high speed digitizer

The Software is part of two diploma theses and is currently considered a work-in-progress. The presentation example implements a basic experiment using two instruments, a function generator and a slow speed data acquisition card.

Development System

- LabVIEW 2009 64 and 32-bit
- Windows 7
- Dual hexa-core (12 core) CPU
- 12 GB RAM

Technical Details

Program Structure

• *Large Scale Application* approach
Dynamically load the instrument drivers as external modules. Therefore the software is easily extendable for future instruments and maintains a small memory footprint. Modularizing software also reduces the overall complexity.

• Communication Framework
The main application/driver communication is realized using *Simple TCP Messaging* (STM), the *Command-Based Communication* design pattern, and an Action Engine to manage *multiple server TCP connections*.

Figure 1 shows the overall program structure. Figure 2 shows a typical STM communication structure.

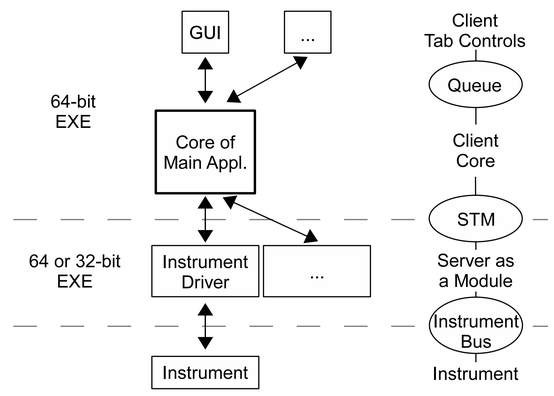

Figure 1: Overall program structure and communication paths

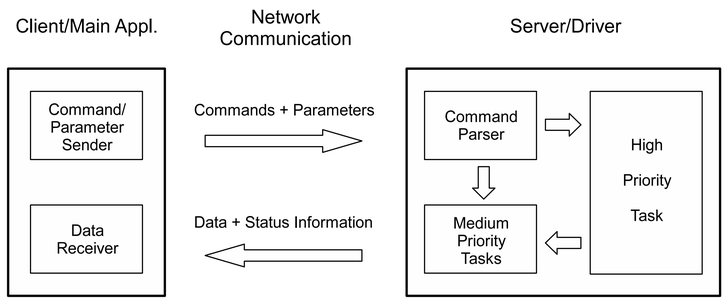

Figure 2: STM Communication Framework

It is possible to exchange instruments with equivalent functions within experiments by using standardized server commands. INI files are used to manage the application's default settings by defining which instrument to use for which task within an experiment and saving the default parameters for each instrument.

Graphical User Interface

• A *Common interface* across all experiments is achieved by incorporating the user interfaces for all experiments into a single application using Tab Controls. Maintaining the structure of only one program allows keeping acquisition software up-to-date for all experiments.

• *Responsiveness is increased* by enqueueing elements inside the user interface event handler and receiver loop using an event-driven state machine. Events are forwarded to a main loop instead of being processed immediately. Both loops require a minimum amount of time to handle incoming tasks.

• Allow the operator to perform baseline correction, FFT, and other post-processing on the acquired data.

Improved Execution Responsiveness

• Takes advantage of *64-bit* architecture. This increases the processing performance and throughput. Although the main application and most drivers are 64-bit, 32-bit drivers are supported.

• Takes advantage of *multi-threading* and *multi-core* environments by writing code with parallel threads where they make sense. These threads are then logically grouped and assigned to one of the 12 CPUs. The goal is to group them in a way that the processor usage is distributed equally over all available cores. This is accomplished by grouping threads that do not run at the same time or generate little overall CPU usage when executing.

• No significant use of polling. Polling is the repeated use of CPU time to check for the completion of a task. It is only used to check for new data from the server. By controlling it with a Timed Loop, minimal CPU time is consumed. In other program parts case structures are used. Case structures cause no CPU usage until data arrives.

Summary

Conclusions

• This software addresses *the main issues* of EPR control and acquisition applications by focusing on the program structure, GUI, execution responsiveness, and data acquisition.

• *The program structure is key*. It provides a path for future expansion and ensures high maintainability as well as optimal performance.

Acknowledgements

This work was supported by grant EB001980 and EB002052 from the National Institutes of Health. The authors would also like to thank Senior Electrical Engineer Joseph J. Ratke and Dr. John D. Gassert for their advice and mentoring.