Published in final edited form as:

Conf Proc IEEE Eng Med Biol Soc. 2010 ; 2010: 4160–4163. doi:10.1109/IEMBS.2010.5627397.

Real-time Experiment Interface for Biological Control Applications

Risa J. Lin,

Georgia Institute of Technology, Atlanta, GA 30332-0250 USA

Jonathan Bettencourt,

Weill Cornell Medical College, New York, NY 10065 USA

John A. White [Member, IEEE],

Department of Bioengineering University of Utah, Salt Lake City, UT 80305 USA

David J. Christini [Member, IEEE], and

Weill Cornell Medical College, New York, NY 10065 USA

Robert J. Butera [Senior Member, IEEE]

School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332-0250 USA

Risa J. Lin: risa@gatech.edu; Jonathan Bettencourt: job2033@med.cornell.edu; John A. White: john.white@utah.edu; David J. Christini: dchristi@med.cornell.edu; Robert J. Butera: rbutera@gatech.edu

Abstract

The Real-time Experiment Interface (RTXI) is a fast and versatile real-time biological experimentation system based on Real-Time Linux. RTXI is open source and free, can be used with an extensive range of experimentation hardware, and can be run on Linux or Windows computers (when using the Live CD). RTXI is currently used extensively for two experiment types: dynamic patch clamp and closed-loop stimulation pattern control in neural and cardiac single cell electrophysiology. RTXI includes standard plug-ins for implementing commonly used electrophysiology protocols with synchronized stimulation, event detection, and online analysis. These and other user-contributed plug-ins can be found on the website (<http://www.rtxi.org>).

I. INTRODUCTION

The ability to perturb biological systems has traditionally been limited to rigid pre-programmed protocols or operator-controlled reflex constrained protocols. In contrast, real-time control allows the researcher to dynamically probe a biological system with perturbations that are computed from system responses. Real-time control applications are abundant throughout biological research, including (i) integration of simulated ion-channel kinetics with intracellular electrophysiological experiments (dynamic clamp) [1], (ii) adaptive control of cardiac [2]–[4] and neuronal [5], [6] excitable systems, and (iii) brain stimulation for neurological disorders [7]. For several technical reasons, real-time control is not possible with standard computer operating systems and software. Commercial real-time systems are costly and often tailored for industrial applications. In general, real-time control systems should run with the smallest possible nominal real-time system period, a minimum amount of cycle-to-cycle variation (or jitter) in the actual real-time period, and the shortest possible system latency. The Real-time Experiment Interface (RTXI) is a fast and versatile biological experimentation system based on Real-Time Linux that allows users to quickly implement real-time, closed loop protocols.

II. SYSTEM DESIGN AND IMPLEMENTATION

A. Operating System

General purpose operating systems (OS) for desktop computers generally provide a multi-tasking environment in which many processes share the same system resources. This approach optimizes the system's average performance with the direct result that real-time accuracy cannot be guaranteed. In general, data acquisition does not require a real-time operating system and there are various solutions that allow data to accumulate in a buffer that gives the OS time to retrieve and process it. This approach is insufficient for real-time control protocols, which require immediate data processing and precise timing of pre-programmed interventions. Commercial real-time operating systems (RTOS), such as VxWorks (Wind River, Alameda, CA) and QNX (QNX Software Systems, Ottawa, Canada), are appealing because they are designed specifically for hard real-time processing. However, the design of these systems and the types of control algorithms they offer are generally intended for industrial applications. These RTOSes also lack the general purpose functionality that would make them useful for non real-time tasks such as post-experimental data analysis.

Real-time solutions exist for Microsoft Windows, the most widely used general purpose operating system. RTOSWin (KUKA Roboter GmbH, Augsburg, Germany) uses para-virtualization software to combine Windows with a RTOS so that they operate side-by-side. Another option uses a real-time subsystem realized as a Windows kernel device driver (RTX by IntervalZero, Waltham, MA). Two approaches that may be more familiar to scientists are: (i) MATLAB coupled with Simulink and Real-Time Workshop (The MathWorks, Natick, MA) and (ii) National Instruments' LabVIEW Real-Time. Both options generate executable code that is exported to an external controller such as an embedded computer, a commercial RTOS, or a field-programmable gate array. All the systems mentioned above can provide complete turnkey solutions for data acquisition (DAQ) and real-time control, but their cost and proprietary code can present a high barrier to adoption for biological research.

In contrast, Real-time Linux [8] is freely available for download and modification under its open source license and offers a fast, predictable, real-time environment with access to the rich variety of services and applications of standard GNU/Linux. While the Linux OS has a reputation for being difficult to install and not user-friendly, its user base has been steadily increasing as various distributions have improved their stability, increased hardware compatibility, and incorporated familiar graphical desktop environments. RTXI supports two variants of Real-Time Linux: RTAI [9] and Xenomai. Real-Time Linux is comprised of a small real-time kernel that runs a guaranteed timing real-time task (RT-task) at top priority and the standard Linux kernel as a fully-preemptible low priority task. High speed and predictable timing are achieved by giving the RT-task full control over hardware interrupts, which are normally the source of non-determinism in an operating system. From a user perspective, there is no change in how Linux operates. RTXI has been tested successfully with Ubuntu, Debian, Fedora, and SUSE Linux. The current Live CD is based on Ubuntu and can be used to run a live user session capable of running real-time experiments with data acquisition or to install the complete real-time OS with RTXI. It requires no specialized hardware beyond a standard desktop computer with a DAQ device.

B. System Architecture

RTXI originated through the merging of aspects of three independent open source real-time control systems using Real-Time Linux: RTLab [10], [11], the Real-Time Linux Dynamic Clamp (RTLDC) [12], and Model Reference Current Injection (MRCI) [13], [14]. RTXI

incorporates the important features of each predecessor system in addition to many new features. At the core of RTX is a two-thread approach, originally utilized by both RTLab and RLTDC, in which a real-time thread executes all real-time tasks while a separate non-real-time thread handles asynchronous events related to the user input and the graphical user interface (GUI) (Fig. 1). Both threads are standard Linux threads but the real-time thread is assigned the highest system priority afforded by the Linux OS. Given a nominal system period specified by the user, the RT thread executes all real-time code, including accessing data from the DAQ, making the data available to other system components, and executing any real-time code from loaded plugins. Both threads share the same address space and are able to communicate with each other and share data. RTX is written in C/C++ and uses the open source Qt framework (Qt Development Frameworks, Oslo, Norway) for the GUI, also implemented in C++.

RTX uses an abstract interface for interacting with various data acquisition devices. It includes support for analog and digital inputs and outputs and provides an interface for configuring a DAQ device. Currently, this internal standard serves as a wrapper around the open source COMEDI (Linux Control and Measurement Device Interface, <http://www.comedi.org>) drivers which support a large number of DAQ devices from a variety of manufacturers, including National Instruments and Measurement Computing. RTX can support multiple DAQ devices.

Users create code for RTX by writing C++ plug-ins. Plug-ins are implemented as shared object libraries (*.so) in Linux that are dynamically linked to RTX at run-time. This mechanism allows the temporary insertion of user code into the real-time thread such that most users will never need to recompile the core RTX source code. Each plug-in is part of a Workspace in which the plug-in's named parameters and state variables are defined. The values of these variables can be modified on-the-fly by the user through the GUI without recompiling the plug-in, and in general, without affecting real-time execution. Automated real-time control is implemented by changing parameter values based on in-program logic without user intervention. RTX provides an event delivery system that allows plug-ins to signal the occurrence of events, such as changes in parameter values or a change in the nominal real-time period, and propagate events to all plug-ins sharing the same Workspace.

User-defined events and data are shared between custom plug-ins through named inputs and outputs. Data acquired from a DAQ device and output signals from plug-ins can be connected to an output channel of the DAQ device or any plug-in with a defined input slot. Plug-ins may have multiple inputs and outputs and RTX supports one-to-many and many-to-one connections. Most plug-ins will also support multiple instantiations. This modular plug-in architecture allows users to construct complex experimental protocols and interfaces with a great deal of flexibility, often eliminating the need to code all aspects of each experiment from scratch. Users will often have multiple plug-ins working in parallel during a single RTX session. In fact, all core RTX features are implemented as plug-ins.

C. Features and Plug-ins

When starting RTX, the system is bootstrapped into a usable state by loading standard system plug-ins that are defined in a settings file. These standard plug-ins include the System Control Panel, a digital Oscilloscope, a Connector for connecting user plug-ins, and a Data Recorder for streaming data to a file (Fig. 2). Users can also create their own settings files based on their current working environment, which saves the entire system state with all user options and parameter values. This allows users to recreate experimental protocols quickly, increasing efficiency and reducing error. The settings files are completely portable between computers, provided that the target RTX system has the specified plug-ins, and are currently used to distribute user tutorials.

The System Control Panel allows users to configure their DAQ devices. RTXl automatically detects the number and type (analog or digital) of inputs and outputs and users can set the gain, range, and whether the channel should be configured for a common ground reference or differential input. The digital Oscilloscope provides a real-time plot of any signal within the RTXl session, including acquired data and all the parameters, state variables, external inputs, and outputs of any plug-in. Signals are plotted on the same time scale but may have different vertical scaling as well as different line colors and styles. Similar to an external hardware oscilloscope, this plug-in also features a trigger that detects threshold crossings. The Connector plug-in provides a GUI that allows users to create connections between plug-ins and the DAQ device. This plug-in determines the available inputs and outputs from currently loaded plug-ins and maintains a list all current connections. All loaded plug-ins are assigned a unique ID number that differentiates multiple instantiations of the same plug-in.

The Data Recorder allows users to record the parameters, state variables, inputs, and outputs of any plug-in that is currently loaded in RTXl. The data is streamed to an HDF (Hierarchical Data Format) file [15], a portable, open source scientific data format that allows the definition of hierarchical data objects for organizing large and complex datasets. The Data Recorder automatically records all parameter values as well as changes to those values. Thus, all the metadata about an experiment is captured along with the experimental data itself. Users can also easily create custom plug-ins that serve as a log of other experimental metadata and notes. Open source applications are available for viewing data stored in HDF files and they are currently readable by several popular software for data analysis, including MATLAB, Octave [16], and Igor Pro (WaveMetrics, Inc., Portland, OR).

RTXI also contains benchmarking tools for evaluating the system's real-time performance. The Performance Measurement plug-in measures the average and worst-case timesteps as well as the average and worst-case running times of the real-time component of a particular algorithm. The worst-case timestep sets an application-dependent upper-bound on timing related errors and indicates whether a given computer has the computational power to implement a particular algorithm [17]. On a relatively new computer with a dependable DAQ board (such as a National Instruments M series board), RTXl is capable of acquiring two analog inputs at a 40–50 kHz sampling rate, with only a 5–10 μ s jitter, and 10–15 μ s latency. Other useful plug-ins include inline FIR filters, power spectrum estimation, and plug-ins for emulating previously recorded data or generating synthetic data for testing algorithms in real-time. Custom plug-ins can be written in C++ or the DYNAMO scripting language incorporated from MRCI. DYNAMO provides an intuitive equation-oriented language for describing a dynamical system through declarations of parameters, state variables, and differential equations [14]. Users can easily incorporate other open source code such as libraries for numerical methods.

III. SUMMARY AND FUTURE WORK

Real-time and/or closed-loop protocols are of increasing interest in biological experiments but there are few tools that allow researchers to quickly implement them without requiring expertise in numerical algorithms or computer programming. RTXl fills this void, enabling the scientific community to take advantage of the many strengths of Real-Time Linux. Existing plans for increasing the range of experimental applications include implementing algorithms for online optimization of model parameters, plug-ins for multichannel in vivo recording and stimulation, and implementation of optical mapping techniques with support for image acquisition. The usability of the software will be improved by visual plug-in design tools and the incorporation of other open source software. For example, the NEURON [18] simulator uses a powerful scripting language, NMODL, to describe individual ion-channel mechanisms and in some instances, specialized numerical methods

for solving the channel equations. A translator for NMODL into RTXI plug-ins would provide neurophysiologists with a tool to quickly leverage existing NEURON code in their experiments.

Acknowledgments

This work was supported by the National Institutes of Health under Grant 2-R01RR020115-05.

References

1. Destexhe, A.; Bal, T. *Dynamic Clamp: From Principles to Applications*. New York, NY: Springer; 2009.
2. Gaeta SA, Bub G, Abbott GW, Christini DJ. Dynamical Mechanism for Subcellular Alternans in Cardiac Myocytes. *Circ. Res.* 2009; 105:335–342. [PubMed: 19628792]
3. Ahrens-Nicklas RC, Christini DJ. Anthropomorphizing the mouse cardiac action potential via a novel dynamic clamp method. *Biophysical Journal*. 2009; 97:2684–2692. [PubMed: 19917221]
4. Christini DJ, Riccio ML, Culianu CA, Fox JJ, Karma A, Gilmour RF Jr. Control of electrical alternans in canine cardiac Purkinje fibers. *Physical Review Letters*. 2006; 96:104101–104101. [PubMed: 16605736]
5. Schiff SJ, Jerger K, Duong DH, Chang T, Spano ML, Ditto WL. Controlling chaos in the brain. *Nature*. 1994; 370:615–620. [PubMed: 8065447]
6. Potter, SM.; Wagenaar, DA.; DeMarse, TB. Closing the Loop: Stimulation Feedback Systems for Embodied MEA Cultures. In: Taketani, M.; Baudry, M., editors. in *Advances in Network Electrophysiology Using Multi-Electrode Arrays*. New York: Springer; 2006.
7. Schiff ND, Giacino JT, Kalmar K, Victor JD, Baker K, Gerber M, Fritz B, Eisenberg B, O'Connor J, Kobylarz EJ, Farris S, Machado A, McCagg C, Plum F, Fins JJ, Rezai AR. Behavioural improvements with thalamic stimulation after severe traumatic brain injury. *Nature*. 2007; 448:600–603. [PubMed: 17671503]
8. Barabanov M, Yodaiken V. Real-Time Linux. *Linux Journal*. 1996 Mar.
9. Mantegazza P, Bianchi E, Dozio L, Papacharalambous S, Hughes S, Beal D. RTAI: Real-time Application Interface. *Linux Journal*. 2000 Apr.
10. Culianu, CA.; Christini, DJ. 3rd Real-Time Linux Workshop. Milan, Italy: 2001 Nov. Real-time experiment interface system: RTLab; p. 26-29.
11. Culianu, CA.; Christini, DJ. Real-time Linux experiment interface system: RTLab; Proceedings of the 29th IEEE Northeast Bioengineering Conference; 2003.
12. Dorval AD, Christini DJ, White JA. Real-time Linux dynamic clamp: A fast and flexible way to construct virtual ion channels in living cells. *Annals of Biomedical Engineering*. 2001; 29:897–907. [PubMed: 11764320]
13. Butera RJ, Wilson CG, DelNegro CA, Smith JC. A methodology for achieving high-speed rates for artificial conductance injection in electrically excitable cells. *IEEE Trans. Biomed. Eng.* 2001; 48:1460–1470. [PubMed: 11759927]
14. Raikov I, Preyer A, Butera RJ. MRCI: a flexible real-time dynamic clamp system for electrophysiology experiments. *J. Neurosci. Methods*. 2004; vol. 132:109–23. [PubMed: 14706709]
15. The HDF Group. HDF Group homepage on HDF5. [Online]. 2010 Available: <http://www.hdfgroup.org/HDF5/>.
16. Eaton, JW. *GNU Octave Manual*. Network Theory Limited; 2002.
17. Bettencourt JC, Lillis KP, Stupin LR, White JA. Effects of imperfect dynamic clamp: Computational and experimental results. *J. Neurosci. Methods*. 2008; 169:282–289. [PubMed: 18076999]
18. Hines ML, Carnevale NT. The NEURON simulation environment. *Neural Comput.* 1997; 9:1179–1209. [PubMed: 9248061]

Fig. 1.

RTXI system architecture with two-thread process structure. All standard system features, such as the Oscilloscope, are implemented as plug-ins. The RT thread executes the RT section of all of the plug-ins currently loaded in the system. Each plug-in will typically have a sub-window embedded in the main window through which users can control the plug-in and its various parameters.

Fig. 2.

Screenshot of a dynamic clamp experiment conducted on a model Connor-Stevens neuron simulated within RTX. The Connor-Stevens model (upper left) is an example of a GUI class provided for users' convenience. In the Connector plug-in (lower left), the neuron model is connected to a plug-in that detects spikes based on threshold crossings of the membrane voltage (Vm). The neuron model is also connected to a dynamic clamp plug-in (upper right), which injects back into the neuron a current computed from Vm, a stimulus waveform saved in an external file, and other stimulus parameters. This plug-in also receives input from the Spike Detector and generates a raster plot of detected spikes over time for repeated trials of the same stimulus. Here, the loop was closed on trial 11 with a clear change in the spike pattern. The Oscilloscope (lower right) plots the voltage of the neuron model.