Experience Report: Functional Programming of mHealth Applications

Christian L Petersen Matthias Görges Dustin Dunsmuir J Mark Ansermino Guy A Dumont
University of British Columbia, Vancouver, BC, Canada
{cpetersen mgorges ddunsmuir}@cw.bc.ca anserminos@yahoo.ca guyd@ece.ubc.ca

Abstract

A modular framework for the development of medical applications that promotes deterministic, robust and correct code is presented. The system is based on the portable Gambit Scheme programming language and provides a flexible cross-platform environment for developing graphical applications on mobile devices as well as medical instrumentation interfaces running on embedded platforms. Real world applications of this framework for mobile diagnostics, telemonitoring and automated drug infusions are reported.

The source code for the core framework is open source and available at: `https://github.com/part-cw/lambdanative`.

Categories and Subject Descriptors D.1.1 [*Programming Techniques*]: Applicative (Functional) Programming; D.2.3 [*General*]: Coding Tools and Techniques; D.3.2 [*Programming Languages*]: Language Classifications - Applicative (Functional) Languages

Keywords functional programming, Scheme, application programming, medical systems, cross-platform development

1. Introduction

The Pediatric Anesthesia Research Team (PART) at the University of British Columbia is a multidisciplinary research team conducting clinical research to improve the safety and outcomes of children both in the hospital and at home. In recent years the team has increased its focus on technology innovation, and is currently developing a range of new diagnostic and monitoring solutions for low cost mobile and embedded applications. As this technology is developed in-house with very limited resources, a new software development framework promoting robust efficient code, extensive testing and debugging facilities, fast development and flexible cross-platform deployment was required.

The team has four code developers on staff; one programmer, two physicists and one engineer, with diverse coding backgrounds. The typical application development process involves a much larger diverse group of clinicians, nurses, community healthcare workers, engineers, graphic designers and research assistants.

Traditionally, biomedical research teams write graphical software applications in Matlab, Visual Basic and LabView. These prototyping languages are not a good fit for robust performance-oriented platform-independent coding. On the other hand, performance-critical applications are traditionally written in C, the ubiquitous system language that is available on virtually all platforms. C is however a very low level language and vulnerable to common null pointer dereferences.

We first discovered Scheme as an extension language to C, and the ability to automatically generate C applications from Scheme was a natural next step. Functional languages, such as Scheme, offer an attractive alternative to conventional languages, especially when combined with a C compilation backend. In this paper we describe a Scheme based software development solution using the Gambit-C Scheme to C compiler [1, 2]. Scheme is a mature language originally developed at MIT over the period 1975-1980, and is standardized on the official IEEE standard 1178-1990 [3] and the Revised Reports on the Algorithmic Language Scheme [4]. It features an exceptionally simple homoiconic syntax based on s-expressions.

The primary goal in the design of the framework was to strike a balance between productivity, correct programming and performance. The brevity and simplicity of Scheme code promotes productivity and correctness, which is essential for mission critical medical applications. At the same time, the C backend allows compilation of high-performance binaries. Many diagnostic medical devices require low-level access to relatively high bandwidth sensors, and tight integration between Scheme and C is essential to facilitate this in real time.

The framework currently contains many distinct applications ranging from medical device servers running on embedded systems, over graphical interfaces running on medical flat panel PCs, to mobile applications for iPhone and Android. In this paper we examine three examples of these applications and discuss their implementation.

2. Concept

The application model adopted herein relies on the fact that modern operating systems share many core features. For example, iOS requires applications to be developed in Objective C. However, much of the underlying core operating system (Darwin) is written in C, and can readily be addressed though the standard UNIX and POSIX application programming interfaces. The same holds true for the Android operating system, which requires applications to be developed in Java, while the underlying operating system (Linux) is C based. Another common denominator between most modern computing platforms is the presence of an accelerated OpenGL or OpenGL ES graphics engine [5]. This provides an efficient universal mechanism for graphical user interfaces to function regardless of the underlying operating system.

It is thus possible to construct a cross-platform program that, while meeting the minimum requirements of the host operating system, performs the bulk of its operations through a universal C

ICFP'13 September 25–27 2013, Boston, MA, USA

ACM 978-1-4503-2326-0/13/09. . . $15.00
http://dx.doi.org/10.1145/2500365.2500615

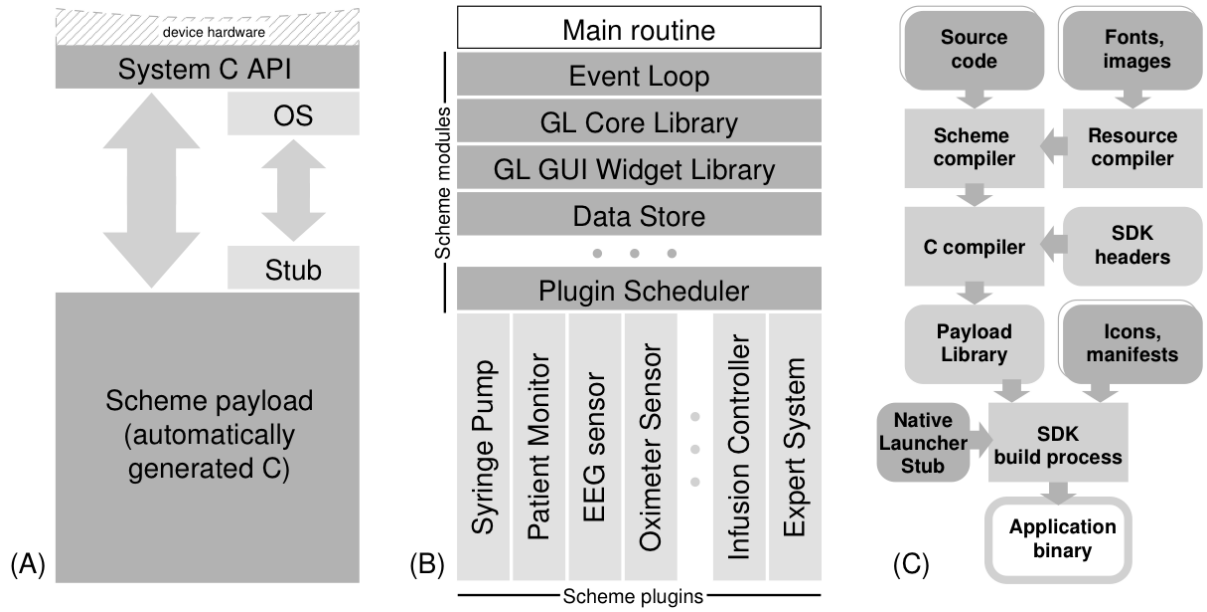

Figure 1. (A) Schematic of the portable application concept. (B) Application structure. (C) Compilation flow diagram.

application interface. This is illustrated in Fig 1(A). The application consists of a payload written in Scheme which is compiled to a native binary using the Gambit Scheme-to-C compiler and the C (cross-)compiler available for the target platform. A small stub shared between all applications in the framework manages the operating system specific application requirements, launches the program, and relays events to the payload.

The advantage of this approach is that a single code base can be deployed unchanged to a wide range of platforms, and run with a high level of performance suitable for IO and computationally intensive applications. Support for a new platform is simply a matter of writing another stub to launch the application on the platform and tie into its operating system event system. The framework has successfully been used to build applications for iOS, Android, Mac OS X, Linux, Windows, OpenBSD, and NetBSD.

The disadvantage of the approach is that a Native Development Kit with a C cross-compiler is required to build the payload. While all modern operating systems have this compiler available, some vendors may choose to withhold access to the compiler. In addition, user interface elements integrated into to the host operating system are not readily available to the payload. To overcome the interface limitations we have developed a widget based graphics engine entirely in Scheme, which provides a flexible platform-independent front end to the host OpenGL or OpenGL ES Graphics Processing Unit (GPU).

3. Structure

The applications are structured in a modular fashion to maximize code reuse between applications. This is illustrated in Fig 1 (B). Each application consists of a main routine that references modules in a common code repository shared by all applications in the framework. The four main modules shared by most applications are `eventloop`, `glcore`, `glgui` and `store`.

The `eventloop` module is responsible for feeding user and operating system events such as touch screen presses and redraw triggers to the payload application. It also relays sensor information such as battery level and device orientation to the application.

The graphics engine consists of a minimal Scheme wrapper around the OpenGL and OpenGL ES functions in `glcore` and a Scheme based graphical user interface in `glgui`. `glgui` implements a full widget system with buttons, labels, boxes, scrollable lists, keypads, images, containers, modal dialogs etc. and is easily expandable with more widget types. Each widget type consists of a hash table holding widget properties, a procedure for drawing the current representation of the widget and a procedure for handling input events. Widgets are associated with gui hash tables, and multiple gui tables can be displayed on screen at the same time. This allows simple grouping of widgets to support different screen configurations. The system also supports rotation and scaling to accommodate different screen geometries, resolutions and device orientation events.

`store` is a generic data store implemented as a hash table with support for data timeouts and categories. Data stores are used throughout the applications as a convenient way to organize data. For example in the telemonitoring application described below, patient data is grouped in individual data stores based on the network address of the patient monitors from which the data originates.

The framework also includes support for a plugin structure built around the `scheduler` module. The scheduler drives the sequential execution of plugins based on their type (four types are supported; input, algorithm, decision support and output). Plugins are instantiated against a data store, and multiple instances are allowed. For example, this allows multiple syringe pump drivers and infusion controllers to be instantiated for different drugs. This functionality is used in the closed-loop anesthesia controller described later.

The stub (in Fig. 1 (A) and (C)) which ties the Scheme payload into the operating system is a small piece of code in the native language of each platform, and is responsible for opening an OpenGL window and relaying rendering and user events to the payload event loop. The rendering itself is accomplished through direct calls to OpenGL in `glcore`.

4. Development and debugging

The framework is designed to be independent of an Integrated Development Environment (IDE), allowing developers to choose their

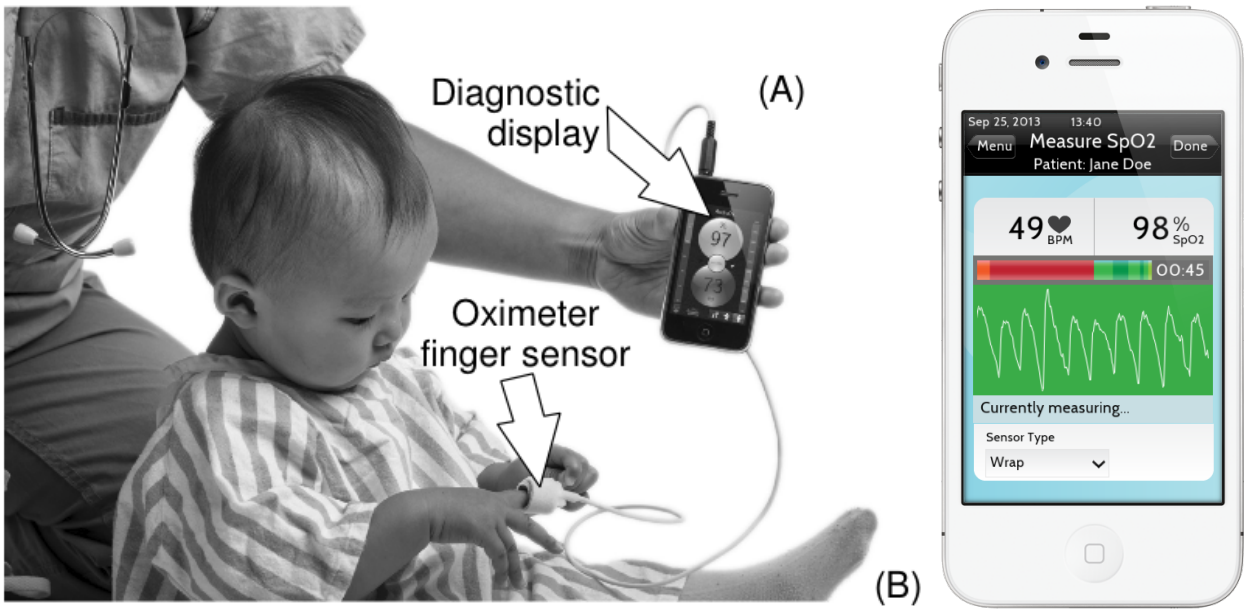

Figure 2. (A) Elements of the phone oximeter. (B) Screen shot showing a PIERS phone oximeter measurement in progress.

most productive environment. Developers are currently working with diverse software development environments ranging from the Eclipse IDE to the Vi editor on both Mac OS X and Linux.

The application framework incorporates a simple file based logging system that can be used for fault finding. Exceptions triggered in the Scheme payload will result in a message about the fault written to the log. Applications built in debug mode include source code file and line number information, allowing easy identification of the problem code. Faults on the C side can similarly be tracked down by compiling in debug mode and using a platform specific stack analysis tool like `ndk-stack` for Android.

A simple unit test system is used to test integrity of algorithmic components such as the encryption and compression communication routines, and a `lint`-like static analyzer [6] is available to find structural problems within the code, such as shadowed declarations and use of reserved keywords.

The built-in Scheme interpreter and equivalence of data and code provide unique ways to investigate and debug the state of the live running system. For example, a console application can be used to test a new code module interactively, prior to deployment in the final applications.

Profiling is accomplished by using Gambit-C's internal ability to capture thread continuations and provide information about file names and line numbers to build statistical information about bottle necks in the source code.

An automated smoke test script is used to periodically build and run all applications in the framework and verify that the binaries are runnable. This catches compatibility problems when making changes to the common code base of the framework.

For application verification we have developed systematic test suites. For example, an automated test including 100 simulated patients is used to verify the integrity of the the drug delivery controller described later. These tests run in real-time, using the built-in interpreter to script the inputs to the applications.

The everyday development and training within the programming team takes place in a flat structure, where code is reviewed and bugs resolved by drawing on the individual knowledge of each developer about particular portions of the code. The somewhat simplistic debugging tools have not limited us so far. The most elusive bugs were unexpected number types at the foreign function interface which caused type conversion problems in the C API calls.

5. Compilation

The compilation flow diagram is shown in Fig 1 (C). The basic idea is to build a single static library containing all of the Scheme payload, and link this library to a small launcher stub in the native language of the target platform (Objective C on iOS and OS X, Java on Android etc).

The payload is compiled using the standard automake tool chain. First the Scheme source files are converted to C using the Gambit-C `gsc` compiler. Next, resources such as images and fonts are processed into Scheme data with custom resource compilation tools and likewise compiled to C with `gsc`. Finally, the resulting intermediate C source files are compiled to object files using the target platform (cross-)compiler against the SDK provided header files. The resulting object files are assembled into a single static payload library.

Compilation of the application binary is complicated by the platform vendors requirement for specific tools. For example, iPhone applications must be built with XCode to satisfy the stringent code signing requirements for app store submissions. For this reason a CMake script is used to automatically generate an XCode project and `xcodebuild` is used to link the payload to the application stub and generate a valid application binary. Similarly the unconventional linking done in the Android build chain requires automating the use of the Android NDK `ant` based build system. The framework also automatically handles the manifests and icon graphics needed for the SDK driven builds on the different platforms.

6. Applications

The following sections describe three representative examples of the framework applications. Most of the applications are research tools used for human trials, and have been used to collect clinical data from more than 10,000 subjects in Canada, France, India, Uganda, Bangladesh, and South Africa, in more than 10 separate clinical studies.

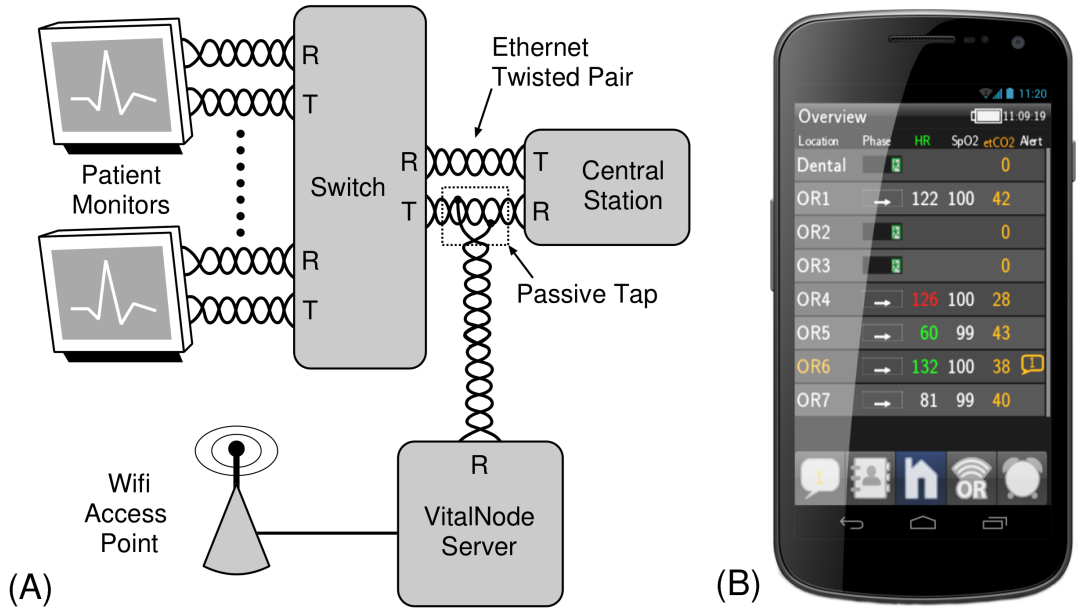

Figure 3. (A) Schematic of the VitalNode server accessing a patient monitor network through a passive tap. (B) telePORT application overview screen showing real time vital signs from a hospitals operating room theatres.

Almost all our application code is functional style Scheme (R5RS) without macros. The code repository contains 317,829 lines of Scheme, including comments and auto-generated code. There are 58 modules, 30 plugins and 77 applications. The only other languages used are C and shell script for supporting tools.

Current applications are deployed on embedded platforms, medical grade flat panel displays, netbooks, and smartphones running iOS and Android. The applications have no noticeable performance problems with respect to garbage collection, memory leaks, footprint etc. Our telemonitoring server described below has uptimes of over a year running on embedded hardware, and our mobile applications run on devices with only 128Mb RAM.

Any device used for diagnostic purposes in humans requires regulatory approval. Health Canada and regional ethics boards have approved the use of our devices for clinical research. The use of Scheme did not directly impact the regulatory process, which is focused on documentation and hazard analysis/mitigation.

7. Use case: The Phone Oximeter

A pulse oximeter is a noninvasive sensor that measures the oxygen concentration of blood. This is a critically important tool for continuous monitoring of oxygen saturation (SpO_2) in the operating room (OR), and is also a good predictor of disease severity and treatment response in a wide range of diseases such as pneumonia in children [7]. The Phone Oximeter is an inexpensive and portable pulse oximeter that runs on a mobile phone [8], as shown in Fig 2 (A). This device is targeted at users in low and middle income countries with little or no availability of conventional pulse oximetry equipment.

For research, we are designing mobile phone applications to record and store accurate spot-checks of a patients SpO_2 using the Phone Oximeter. Fig 2 (B) shows the measurement screen of one of our newest such applications, Pre-Eclampsia Integrated Estimate of RiSk (PIERS), which is described later in this section. This screen displays a photoplethysmogram, as well as heart rate (HR) and SpO_2 trends.

PhoneOxR [9] and its sequel PhoneOxR2 are Phone Oximeter applications used to collect accurate SpO_2 and HR values for patients admitted to studies on post discharge mortality, newborn and early childhood sepsis, and respiratory distress in Uganda, Bangladesh, and India respectively.

Our first goal with these applications was to ensure the accuracy of data used within the research studies. Changes in data quality reflected in a Signal Quality Index (SQI) result in prominent colour changes on the display. A horizontal progress bar shows the SQI colour over time, to promote the collection of good data, as seen in Fig. 2 (B). The percent of the recording that was good quality, as well as the median HR and SpO_2 trend values calculated from only green (high quality) sections of the recording are displayed at the end of the recording.

For each study subject, these applications create patient records, which include the median HR and SpO_2 as calculated above. A survey component allows entry of patient details such as demographics and medical history for each patient. The data is stored locally in an encrypted database and whenever network access is available the data is sent to a secure Research Electronic Data Capture (REDCap) [10] server via the REDCap application programming interface.

The PIERS application Fig 2 (B) represents the next step in our Phone Oximeter applications as it provides an interface for collecting not just SpO_2 but also the details of signs, symptoms, medical history, and medications taken over multiple visits with a patient.

This information is the input to a decision tree, which includes the miniPIERS predictive model, to assess the risk of pre-eclampsia and provide a recommendation, such as urgent transport to the hospital. The miniPIERS model is similar to the fullPIERS model [11], but it excludes the use of laboratory tests so that it can be used in community and primary health care facilities where these test are not possible. This application is designed to be used by midwives in local clinics and a simplified version of the application is being constructed to be used by community health care workers in their regular visits to pregnant women's homes. This application has undergone multiple iterations of usability testing and the simplified

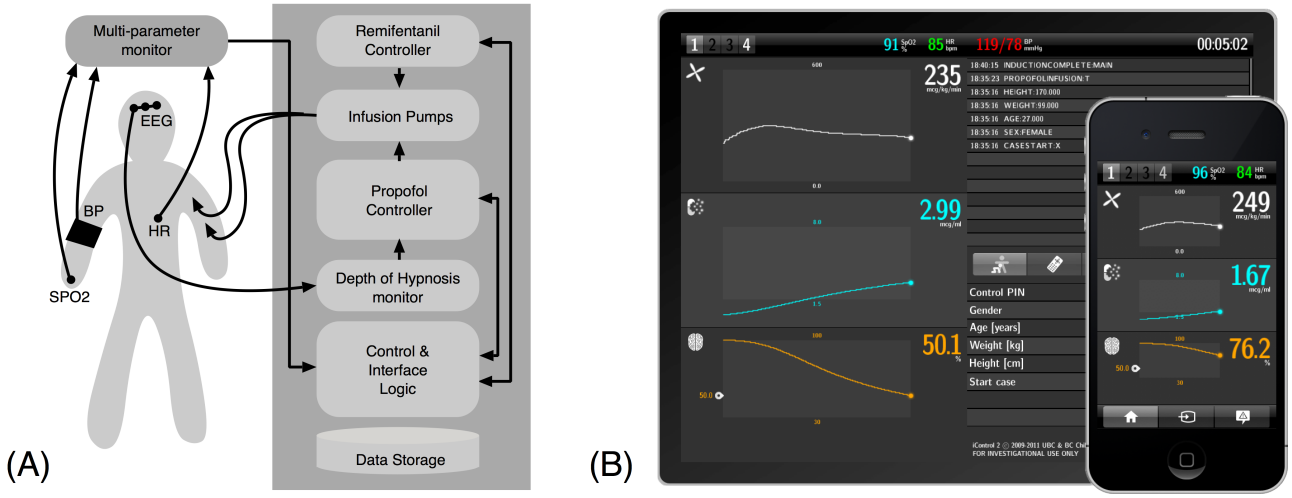

Figure 4. (A) Schematic of the iControl system for automated drug administration during general intravenous anesthesia. (B) The iControl interface running on platforms with different screen sizes.

version will be used in a multi-site study in India, Nigeria, Pakistan and Mozambique in the near future.

8. Use case: The telePORT monitoring and messaging device

Conventional physiological monitors in multi-bed environments such as operating rooms (OR) and intensive care units (ICU) are connected to central monitoring stations. However, these central stations are immobile and have limited proprietary interfaces. In order to make better use of collected patient data, we have developed a data access node and accompanying wireless real-time display of multi-bed patient data for mobile phones, which operates independently of the commercial patient monitor network.

We use an intrinsically safe method for extracting physiological trends and waveforms from patient monitoring networks by means of a passive network tap [12]. The tap is inserted into the central monitor network, Fig. 3 (A), and an exact copy of the network traffic is analyzed in real time with no impact on the network itself. Data collected from the tap is analyzed by a VitalNode server, developed in the application framework. This server runs on a Soekris 5501 embedded computer, and uses the PCAP (Packet CAPture) [13] application programming interface interfaced to a Scheme based parser plugin. Our parser understands raw network packages from Philips IntelliVue MP70 and Datex-Ohmeda S/5 monitors and stores continuous waveforms and vital sign trends from the monitors. This data is then fed through a forward chaining inference engine decision support plugin in the framework to generate trend based alerts. This approach allows wireless and wired access to encrypted physiological data, from all beds of the entire OR, ICU or ward to which the system is connected, as well as secure message exchange between users of the system.

telePORTs purpose is to improve information exchange, and simplify communication between anesthesia team members [14]. The telePORT interface is shown in Fig 3 (B). It runs on a mobile phone and polls user- and system generated messages, vital signs trends, and waveforms securely over the hospitals wifi network from the VitalNode server. It then supports the users work though five components: A) an overview screen, which provides the user with basic information about their subscribed monitoring locations, such as the anesthetic phase and three vital sign values. If additional detail is required access to waveforms and trends is provided. B) a messaging screen, which works as a combination of person-to-person chat system, and a system to receive requests for help through a button pressed on the patient monitor in an OR, C) a phonebook screen, which lists frequently used phone extension and pager numbers; and provides editing capabilities on the device, D) a room subscription screen, where room monitoring locations are subscribed to and where subscriptions can be delegated to other users, and E) a reminder screen, which allow the user to set up time- and anesthetic phase-based reminders. telePORT has been used in our department continuously for the last year with highly favourable reviews by its users. Based on this success, we are currently in the progress of developing a mobile monitoring, information exchange, and team communication solution for nurses, respiratory therapists and physicians in the ICU.

9. Use case: The iControl anesthesia controller

Today, intravenous anesthesia is typically delivered by a manually adjusted continuous infusion; a syringe pump is programmed to deliver a fixed volume per unit time, usually calculated on a weight basis with dosage based on a historical population average. Due to the high inter-patient variability, it is difficult to predict a manual infusion rate that will accurately produce the desired target brain concentration of the drug. Too little drug will mean that the patient will move or wake up, and too much drug will result in prolonged recovery and an increase in side effects such as cardiovascular or respiratory depression.

An automated anesthetic drug delivery system that offers a continuously adjusted rate of intravenous drug administration has advantages over traditional static (weight based) infusion, including greater hemodynamic and respiratory stability, more stable depth of anesthesia, the ability to predict recovery, and a lower total dose of drug administered.

iControl is such a closed-loop solution that delivers drugs based on feedback from electroencephalography (EEG) derived measurements of depth of hypnosis, as shown schematically in Fig 4 (A). Closed-loop control technology is used extensively in many fields such as the aviation and nuclear industry, but has yet to make a significant impact in anesthesia (or medicine in general). One of the main reasons for this is the natural concern about safety of closed-loop systems in medical applications.

The iControl system was designed and implemented using rigorous validation requirements of high-risk medical instrumentation, including a detailed user and system requirements specification, hazard analysis, and system design specification. The system underwent usability studies and extensive manual and automated system tests prior to being approved for clinical trials by Health Canada.

iControl is implemented as a client-server system. The server resides on a Soekris 5501 embedded computer running Linux or OpenBSD. The server has several sub-processes that use the framework plugin structure to feed data from syringe pumps and patient monitors to a patient specific data store. A controller plugin instantiated on the patient store calculates the new infusion rates and instructs the pump plugins to update the devices. The server communicates with a client interface through the same encrypted communication scheme as used by the telePORT application.

The iControl client interface is shown in Fig 4 (B). It features a real time display of current infusion rates, estimated brain concentrations of the delivered drugs, measured depth of hypnosis, and target depth of hypnosis.

The system is controlled through a simple touch screen based input system that is designed to provide a consistent and safe interface. The user interface screen layout automatically reconfigures based on the size of the display. The primary system console client runs on a medical grade flat panel PC (Windows XP embedded), mounted on a mobile medical cart together with pumps, EEG monitor and the embedded controller. Remote monitoring is possible over wifi by running the iControl interface application on a mobile phone or tablet. Multiple clients can run against the server simultaneously.

iControl has successfully been used to automate hypnotic drug (propofol) delivery in adults [15] and children [16, 17] , and is currently used in a study on adults controlling both hypnotic and analgesic drug (remifentanil) delivery concurrently.

10. Conclusions

A modular framework for cross-platform development of robust medical applications has been developed and used for deployment of diverse applications ranging from mission-critical embedded drug delivery systems to diagnostic and monitoring apps on iPhone and Android smart phones.

The use of Scheme and the extensive reuse of code between applications have enabled efficient development cycles with minimal resource requirements and a very small code base. The ability to deploy applications on a wide range of different platforms has proven to provide great flexibility and offers a way to easily port the system to new platforms if the need arises. The possibility of developing on a platform other than the target platform also provides greater productivity as it allows developers to work with the platform and development tools they prefer.

The biggest challenge encountered has been the learning curve associated with the Scheme language, and the lack of knowledge of functional programming in the engineering and the medical community from which we normally recruit. Unfamiliar syntax appears to be the biggest barrier. Use of a more familiar C or Java like in-fix notation, for example as provided by the Gambit-C internal "six" notation, may overcome this problem, and provide a robust cross-platform development system that can benefit from the strength of functional programming while being disseminated to the programming community at large.

A. Source code

The source code for the core framework, including simple demonstrator applications, is open source and available for download at: `https://github.com/part-cw/lambdanative`.

References

[1] Gambit Scheme programming language, `http://gambitscheme.org`

[2] Feeley M, Miller JS, Guillermo JR, Wilson JA. Compiling higher-order languages into fully tail-recursive portable C. Technical Report 1078, département d'informatique et r.o., Université de Montréal (1997)

[3] IEEE Standard for the Scheme Programming Language, IEEE 1178-1990, ISBN: 1559371250

[4] The Revised^6 Report on the Algorithmic Language Scheme, `http://www.r6rs.org`

[5] Kilgard MJ, Akeley K. Modern OpenGL: its design and evolution. SIGGRAPH Asia '08, p. 13 (2008)

[6] Johnson S. Lint, a C program checker. Computer Science Technical Report 65, Bell Laboratories, December 1977

[7] Neuman MI, Monuteaux MC, Scully KJ, Bachur RG. Prediction of Pneumonia in a Pediatric Emergency Department. Pediatrics, 2011; 128:2 p. 246-53

[8] Karlen W, Hudson J, Lim J, Petersen C, Anand R, Dumont GA, Ansermino JM. The Phone Oximeter. IEEE Engineering in Medicine and Biology Society Unconference, Boston, USA, August 30, 2011

[9] Dunsmuir D, Petersen C, Karlen W, Lim J, Dumont GA, Ansermino JM. The Phone Oximeter for Mobile Spot-Check. Society for Technology in Anesthesia 2012 Annual Meeting, Palm Beach, FL, USA, January 18-21, 2012

[10] Harris PA, Taylor R, Thielke R, Payne J, Gonzalez N, Conde JG. Research electronic data capture (REDCap) - A metadata-driven methodology and workflow process for providing translational research informatics support. J Biomed Inform. 2009, 42(2), p. 377-81

[11] von Dadelszen P, Payne B, Li J, Ansermino JM, Broughton-Pipkin F, Cote AM, Douglas JM, Gruslin A, Hutcheon JA, Joseph KS, Kyle PM, Lee T, Loughna P, Menzies JM, Merialdi M, Millman AL, Moore MP, Moutquin JM, Ouellet AB, Smith GN, Walker JJ, Walley KR, Walters BN, Widmer M, Lee SK, Russell JA, Magee LA. Predicting adverse maternal outcomes in pre-eclampsia: the fullPIERS (Pre-eclampsia Integrated Estimate of RiSk) model - development and validation. Lancet, 2011, 377 (9761), p. 219-227

[12] Görges M, Petersen C, Ansermino JM, Capturing vital signs for research in a multi-bed monitoring environment. Anesth Analg, 2011, 113 (2S Suppl), p. 42

[13] McCanne S, Jacobson V. The BSD Packet Filter: A New Architecture for User-level Packet Capture. Proc. of the USENIX Winter 1993 Conference p. 2

[14] Görges M, Ansermino JM. Development of a Mobile Monitoring and Communications Solution for Anesthesia Team Members. Anesth Analg, 2011, 112 (5S Suppl), p. S207

[15] Dumont G, Liu N, Petersen C, Chazot T, Fischler M. Closed-Loop Administration of Propofol Guided by the NeuroSense: Clinical Evaluation Using Robust Proportional-Integral-Derivative Design. American Society of Anesthesiologists (ASA) Annual Meeting 2011

[16] van Heusden K, Dumont GA, Soltesz K, Petersen C, West N, Ansermino JM. Clinical evaluation of closed-loop controlled propofol infusion in children. World Congress of Anesthesiologists, Buenos Aries, Argentina, March 25-30, 2012

[17] West N, Dumont GA, van Heusden K, Khosravi S, Petersen C, Ansermino JM. The administration of closed-loop control of anesthesia for gastrointestinal endoscopic investigations in children. Society for Pediatric Anesthesia AAP Pediatric Anesthesiology Annual Meeting, Tampa, FL, USA, February 23-26, 2012