A Generic Communication Protocol and User-Feedback System for Exercise Devices

ENGI 8800 Final Project Report

Ryan Green
Computer Engineering
Memorial University of Newfoundland
April 3, 2007

Table of Contents

1 Introduction

1.1 Background

The merits of physical exercise have been well documented and are common knowledge in our society. Exercise has been proven to contribute positively to maintaining a healthy weight, building and maintaining healthy bone density, muscle strength, and joint mobility, promoting physiological well-being, reducing surgical risks, and strengthening the immune system. Aerobic exercise, in particular, has been shown to help prevent or treat serious conditions such as high blood pressure, obesity, heart disease, Type 2 diabetes, insomnia, and depression [1].

Lack of physical exercise is generally attributed to a failure to adhere to an exercise program. However, research has shown that adherence to exercise programs can be greatly increased with the use of feedback systems and social support. One study has shown that computer feedback systems promote "significantly higher attendance and adherence" as well as "fewer dropouts by month, a larger number of days before dropout, and 46% less over-all dropout" [2]. Traditional social support systems have also been proven effective in increasing exercise quantity and adherence: "social support for exercise positively correlated with physical activity" [3].

Thus, a socially-oriented computer feedback system would be greatly beneficial to an individual's ability to adhere and succeed in an exercise program and would ultimately lead to increased levels of fitness. While these benefits may be sought by the average user exercising for personal health and fitness, such a system may also have important applications in training, rehabilitation, professional monitoring, etc.

The adoption of such a system is largely dependent on the system's transparency and compatibility with existing devices. As there are many forms of machine-based cardio-vascular exercise, the computer feedback system must be reasonably generic so as to accommodate the variety of equipment commonly used. Furthermore, the system should be well-documented and easily extensible so that new device implementations can be achieved with minimal effort. The system must also be reasonably transparent so that it does not interfere with the user experience. It should require minimal or no user intervention for typical use.

There a small number of computer feedback systems in use today. The industry-standard, with highest deployment figures, is the FitLinxx system. FitLinxx is based on the CSAFE (Communications Specification for Fitness Equipment) specification – the industry standard committee-designed protocol for device communication. CSAFE was designed with input from many device vendors and as such provides compatibility with a large number of devices. However, this compatibility is at the cost of a large, convoluted design. A typical implementation uses a very small subset of the specification and requires significant engineering effort. Thus, the CSAFE protocol, while offering features and compatibility, is not highly generic, and not well suited to an extensible feedback system. Similarly, the FitLinxx software system based on CSAFE is closed, outdated, and closely tied in with CSAFE and associated device vendors.

1.2 System Description

The computer feedback system is a multi-tiered software system with several components deployed on different platforms. The system presents real-time feedback to the user and facilitates socially-oriented, long-term data feedback. The software system also provides a standard means to interact with exercise devices independent of device type and communication channel.

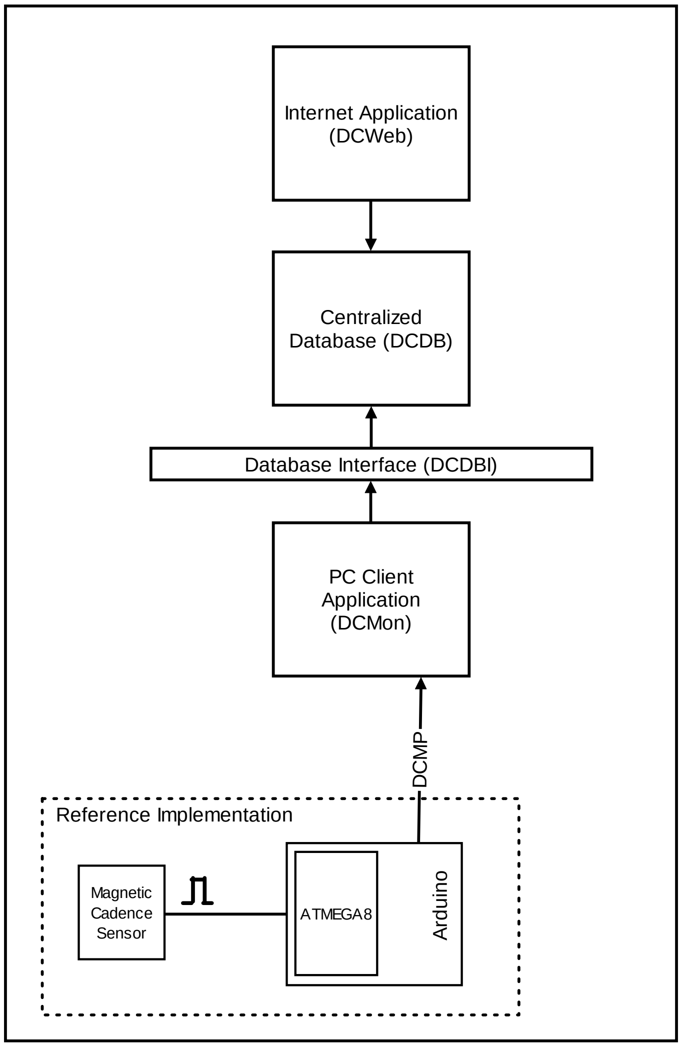

Internet Application (DCWeb)
Centralized Database (DCDB)
Database Interface (DCDBI)
PC Client Application (DCMon)
DCMP
Reference Implementation
Magnetic Cadence Sensor
ATMEGA8
Arduino

Figure 1. High-Level System Design

Central to the operation and effectiveness of the system is the *Device/Monitor Protocol* (DCMP). DCMP provides a simple, unambiguous specification for bidirectional communication between an exercise device and the software system. The *Reference Implementation* is a sample implementation of the DCMP protocol which consists of a miniature cycle device in conjunction with sensing and embedded logic to facilitate communication with the feedback system. The reference implementation is separate from the feedback system and provides a sample device implementation for demonstration, testing, and documentation purposes.

The *Client Application* (DCMon) monitors communication with the device via DCMP by means of a choice of communication channels. DCMon is a portable application (potential embedded deployment) which provides real-time feedback to the user in the form of time-series graphs and data statistics. DCMon also provides a means of configuring software/device parameters and facilitates transfer of pertinent exercise data to a central database.

The *Centralized Database* (DCDB) maintains all exercise data for all users of the system as well as secondary information such as user profiles and supplementary data required by other features of the system. Communication is achieved between the client software and DCDB via the *Client/Server Interface* (DCDBI). DCDBI provides a standard method of communication across platforms between the client software and the centrally-deployed database.

The *Internet Application* (DCWeb) is the primary user-interface to the feedback system. DCWeb is a multi-user application responsible for implementing social features of the feedback system and presenting long-term exercise data to users in a meaningful way. It is a centrally accessible application that provides facilities for account management, user profiles, "friends" management, several views for exercise data, as well as a means for sharing this data with other users.

1.3 Process Description

The system was designed and implemented according to the traditional "waterfall" process model. The inherent weaknesses of this model were negligible given the nature of the design project.

The waterfall model was well suited to this project as it is documentation-driven and self-specified. It was also more straightforward than incremental development given the hard deadlines and project time-frame.

The specifications were clearly defined once at the beginning of the process and remained generally static throughout. This approach was sufficient since there was no client to form/re-form specifications. Similarly, the entire system design was largely prepared in a single phase.

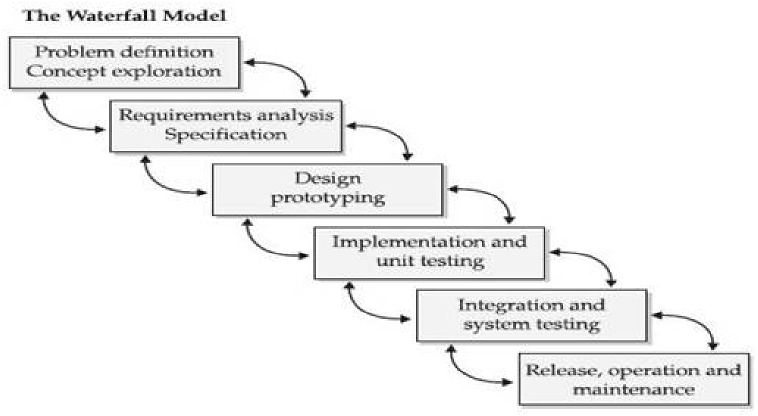

The Waterfall Model
Problem definition Concept exploration
Requirements analysis Specification
Design prototyping
Implementation and unit testing
Integration and system testing
Release, operation and maintenance

Figure 2. Waterfall Software Process Model

Since the system is an integration of several components, a complete up-front design was instrumental in the development of the components and their means of communication. Thus, the initial design was comprised of a (largely complete): reference implementation design, UML design of the client application, relational database design, internet application UML design, and a design of the communication protocols between device/client (DCMP) and client/database (DCDBI).

The problem definition, research, requirements analysis, and detailed specification were undertaken in term VII, while design, implementation, and testing were completed in term VIII. This report will be organized roughly according to the chronological order of the tasks completed.

1.4 Definitions and Terms

DCMP

Device/Monitor Protocol. The communication protocol used between the device and the client application.

DCMon

The client application used to monitor and present real-time data from the device.

DCDB

Centralized Database. The central database used to store all permanent information required by the system.

DCDBI

Cleint/Server Interface. The software interface used to provide communication between client and server software.

DCWeb

The web-based application component of the feedback system.

RPC

Remote Procedure Call. A method of interoperability between disparate software platforms.

Session

A period of device activity as defined in the client application settings. A session typically starts after a certain amount of device activity, and terminates after a certain amount of device idle time.

Tick Scale

The distance for each session tick received from the device. This value varies depending on device implementation.

Exercise Data

Data generated from the device input. Examples are distance, session time, average velocity, velocity, approximate calories, etc.

User

Someone using the feedback system, either as a device user or as a “friend” of a device user.

2 Software Requirements

2.1 DCMon

2.1.1 DCMP Implementation

DCMon will implement the DCMP protocol to allow communication with the device through whatever communication channel chosen.

2.1.2 Settings

DCMon will provide a user interface to set software and device settings. Settings will be persisted between executions of DCMon. Settings will include:

i) Username.

ii) Password.

iii) Graph update interval.

iv) Session upload.

v) Session start ticks.

vi) Session timeout.

vii) Communication timeout.

viii) Server host.

ix) Communication type.

x) Communication port.

xi) Communication rate.

xii) Velocity unit.

xiii) Distance unit.

2.1.3 Unit support

DCMon will support common distance and velocity unit types.

2.1.4 Graph view

DCMon will display and update velocity and average velocity time-series graphs for active sessions. The graphs will be updated at such a time period that they update smoothly and appear responsive to device activity. The v,t ranges will be chosen such that the graph data provides understandable feedback to the user.

2.1.5 Data view

DCMon will display and update data pertaining to the active session. Data displayed by DCMon will include:

i) Session total distance.

ii) Current velocity.

iii) Session average velocity.

iv) Approximate calories.

v) Session start time.

vi) Session elapsed time.

vii) Active username.

2.1.6 Session start

DCMon will acknowledge that a session has started when it receives as many ticks as specified in the “Session Start Ticks” setting. This allows for casual moving of the cycle pedals without unintentionally triggering a session. DCMon will go into a session state at this point and display the graph view and data view.

2.1.7 Session terminate

DCMon will acknowledge that a session has ended when it has not received any device activity for as long as specified in "Session Timeout" setting. A session termination will trigger the following actions:

i) Update DCDB with session data.

ii) Reset session data and views, put DCMon in "idle" state.

2.1.8 Memory/CPU requirements

As DCMon is intended to be an "always-on" application, it must have a relatively low memory footprint and CPU usage so as to not affect system performance.

2.1.9 Platform independence

DCMon will be executable on all major operating systems.

2.1.10 Non-obtrusive

DCMon will minimize its use of screen real-estate when in idle mode. The application can also be "hidden" for when there is no active session.

2.1.11 Fully configurable in session mode

Graph views and data views will be fully customizable in session mode. Including resize, close, float, move, minimize, tab.

2.2 DCDBI

2.2.1 Authenticate user

DCDBI will authenticate the username/password given by DCMon against that available in the database. If the user cannot be authenticated, they will not be given access to the database and DCMon will not operate.

2.2.2 Authenticate device

DCDBI will authenticate a device key against valid device keys in the database. DCMon will not operate without a valid device key.

2.2.3 Fetch user id

DCDBI will retrieve a unique identifier from the database given a username.

2.2.4 Submit session

DCDBI will provide the facility for DCMon to submit the data from a completed session to the database. This data consists of:

i) Session start date/time.

ii) Session end date/time.

iii) Session total distance.

iv) Session average velocity.

v) Data point resolution.

vi) Session device.

vii) Current authenticated user id.

viii) Data (velocity) points.

2.3 DCDB

2.3.1 Users

DCDB will maintain the following user information:

i) Last name.

ii) First name.

iii) Email address.

iv) Password.

v) Username.

vi) Age.

vii) Sex.

viii) Height.

ix) Weight.

x) Display name.

xi) Display image.

xii) Description.

2.3.2 Sessions

DCDB will maintain the following session information:

i) User identifier.

ii) Session start date/time.

iii) Session end date/time.

iv) Session total distance.

v) Session average velocity.

vi) Device identifier.

vii) Session velocity points.

viii) Velocity point resolution.

2.3.3 Friends

DCDB will maintain a list of friends (users) for each user and pending friend requests.

2.3.4 Devices

DCDB will maintain a list of registered devices.

2.3.5 Online/Offline

DCDB will maintain information specifying whether or not each user is in an active session (online).

2.4 DCWeb

2.4.1 Login

DCWeb will present the user with a secure method of authenticating their username and password before granting access to the system. DCWeb will use cookies to allow users to return to the page without having to login again.

2.4.2 Sign up

DCWeb will supply the method of registering new users to the system.

2.4.3 Navigation

All sections within DCWeb will have a navigation panel to access all features within a single click.

2.4.4 Exercise view

The exercise view will present the user with their exercise history. The exercise view will be presented in the following forms:

i) Stats view.

All-time exercise history presented as a summary of data. Including:

a) Total distance.

b) Total session time.

c) Total calories.

d) Average velocity.

ii) Month view.

Calendar-style view for the chosen month. Will display session distance overlaid on the calendar with hyperlinks to the session view. Will also display monthly statistics in the form described in i.

iii) Week view.

Calendar-style view for the chosen week. Will display session distance overlaid on the calendar with hyperlinks to the session view. Will also display weekly statistics in the form described in i.

iv) Day view.

Chronological view of session distances for the chosen day with hyperlinks to the session view.

v) Session view.

Complete session information including:

i) Session start date/time.

ii) Session end date/time.

        iii) Session duration.

        iv) Session total distance.

        v) Session average velocity.

        vi) Session total calories.

        vii) Session device.

        viii) Session velocity point graph.

vi) Recent sessions view.

Bar-chart format of last X sessions with hyperlink to the session view.

2.4.5 Edit user profile

DCWeb will provide a means for a user to modify the following information in their user profile:

i) Last name.

ii) First name.

iii) Email address.

iv) Password.

v) Postal code.

vi) Age.

vii) Sex.

viii) Height.

ix) Weight.

x) Personal description.

xi) Display name.

2.4.6 Friends view

DCWeb will support a "friends list" for each user so that users may share their exercise history with other primary and secondary users. The friends view will provide a list of all friends' display names and will display for each:

i) Online/offline indicator.

ii) Link to user profile.

iii) Link to exercise view.

iv) Option to remove.

2.4.7 View user profile

DCWeb will allow users to view their profile and that of users in their friends list.

2.4.8 Add friend

Provide facility for a user to add another user to their friends list. User will enter the email of the friend along with a message.

2.4.9 Authorize add friend

When user A attempts to add user B as a friend, user B will be notified and given the option to authorize or decline the request.

2.4.10 Remove friend

A user will have the option of removing a friend from the friends list.

2.4.11 Invite User

A user will have the option of inviting a person to sign up to the system via automated email.

2.4.12 DCMon download

DCWeb will provide the facility to download the latest version of the DCMon application, as well as any necessary support files.

2.4.13 Standards

In order to achieve present and future platform/browser compatibility, DCWeb will be developed using valid XHTML 1.0 Transitional and valid CSS2.

3 Design Details

3.1 Device/Monitor Protocol (DCMP) Design

DCMP consists of a set of messages and a set of rules enforced upon these messages. DCMP was designed to be used independent of device type and communication channel. Another major design criterion was that DCMP be easily-realizable. Thus, the protocol is based on a simple state-machine model and contains a minimal message set with no reference to low-level communication details. See the Reference Implementation design for a state-machine model of the DCMP rules. Rules are enforced on message order and parameter size. Timeouts and acknowledgements are strictly enforced. Typically, DCMP communication is meant to consist of init, session, and termination phases (See 3.1.2 for typical operation).

Significantly, the protocol does not use a master/slave relationship with the device as the slave. The protocol is sufficiently simple to allow device-initiated sessions to be implemented on any basic microcontroller devices and thus require less overhead user interaction with the software.

3.1.1 Messages

DCMP messages are fixed-length byte strings with message size depending on the message.

<u>Device to Monitor</u>

1. Init Session (0x49)

Instruct the monitor to begin a session.

Params: N/A.

Acknowledgement: Message 8.

2. Send Device Key (0x48)

Send the device key to the monitor.

Params: 20-byte device key.

Acknowledgement: Message 9.

3. Set Tick Scale (0x53)

Send the tick scale used by the device to the monitor.

Params: 4-byte tick scale. Unit cm's, zero left-padded.

Acknowledgement: Message 10.

4. Send Tick (0x54)

Send a session tick to the monitor.

Params: N/A.

Acknowledgement: Message 11.

5. Quit Session (0x51)

Instruct the monitor to terminate the session.

Params: N/A.

Acknowledgement: Message 12.

6. Communication Timeout (0x43)

Notify the monitor of a communication timeout. This is typically an acknowledgement timeout.

Params: N/A.

Acknowledgement: N/A.

7. Protocol Error (0x45)

Notify a monitor of a breach of protocol.

Params: 2-byte error code (not yet designated).

Acknowledgement: N/A.

<u>Monitor to Device</u>

8. Init Session Ack (0x69)

Acknowledge device-initiated session.

Params: 1-byte success code (0x01 or 0x00).

Acknowledgement: N/A.

9. Send Device Key Ack (0x6B)

Acknowledge device key. The device key is verified against DCDB.

Params: 1-byte success code (0x01 or 0x00).

Acknowledgement: Message 8.

10. Set Tick Scale Ack (0x73)

Acknowledge tick scale. This is the last message used in session initialization.

Params: 1-byte success code (0x01 or 0x00) + Message 14 + Message 16 + Message 15.

Acknowledgement: Message 8.

11. Tick Ack (0x74)

Acknowledge a session tick.

Params: N/A.

Acknowledgement: N/A.

12. Quit Session Ack(0x71)

Acknowledge a device-initiated session termination.

Params: N/A.

Acknowledgement: N/A

13. Protocol Error (0x65)

Notify device of a breach of protocol.

Params: 2-byte error code (not yet designated).

Acknowledgement: N/A.

14. Set Session-Start Ticks (0x72)

Set the number of ticks needed for the device to auto-init a session.

Params: 2-byte tick count. Zero left-padded.

Acknowledgement: N/A.

15. Set Communication Timeout (0x75)

Set the timeout period in milliseconds for the device to send a communication timeout.

Params: 6-byte timeout value. Zero left-padded

Acknowledgement: N/A.

16. Set Session Timeout (0x6D)

Set the timeout period in milliseconds for the device to send a session timeout.

Params: 6-byte timeout value. Zero left-padded

Acknowledgement: N/A.

3.1.2 Example Sequence

This is a trivial device-initiated DCMP session with initialization, session, and termination phases.

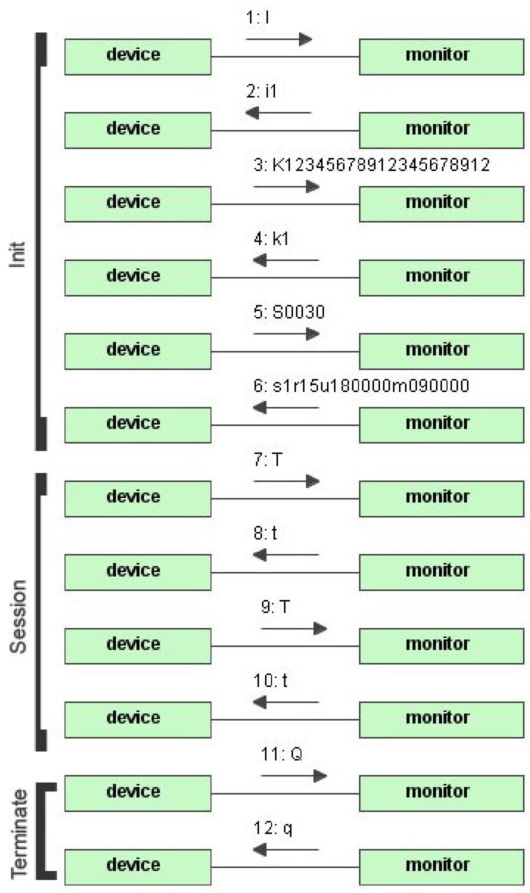
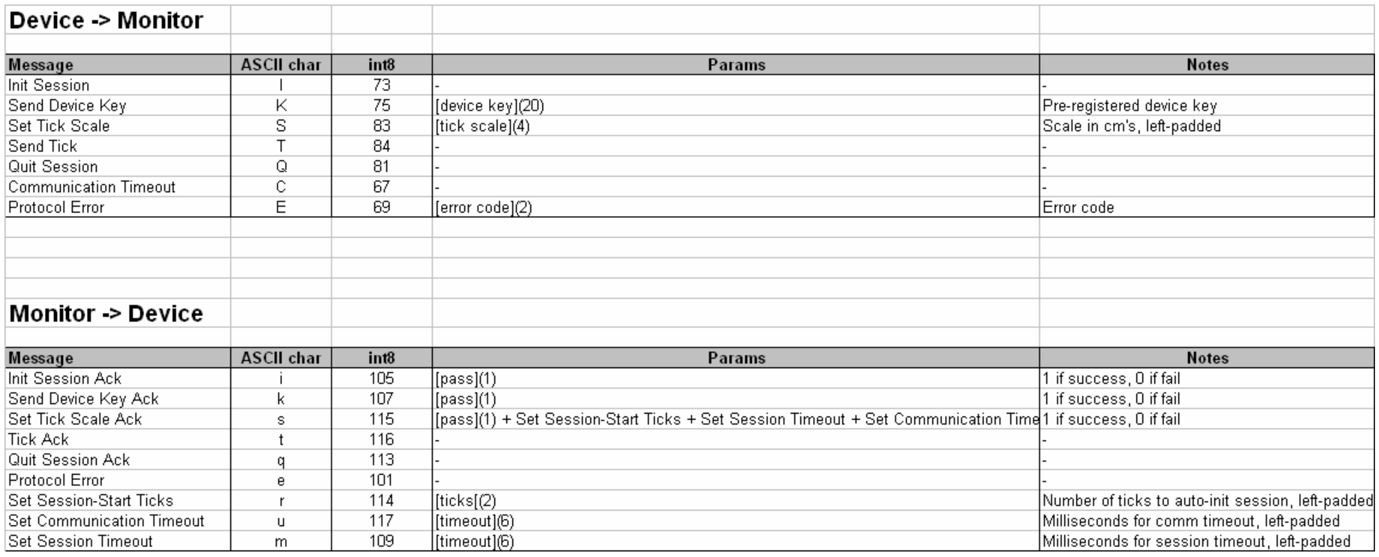


Figure 3. Trivial DCMP Session

3.1.3 Reference



Device -> Monitor

Message	ASCII char	int8	Params	Notes
Init Session	I	73	-	-
Send Device Key	K	75	[device key](20)	Pre-registered device key
Set Tick Scale	S	83	[tick scale](4)	Scale in cm's, left-padded
Send Tick	T	84	-	-
Quit Session	Q	81	-	-
Communication Timeout	C	67	-	-
Protocol Error	E	69	[error code](2)	Error code

Monitor -> Device

Message	ASCII char	int8	Params	Notes
Init Session Ack	i	105	[pass](1)	1 if success, 0 if fail
Send Device Key Ack	k	107	[pass](1)	1 if success, 0 if fail
Set Tick Scale Ack	s	115	[pass](1) + Set Session-Start Ticks + Set Session Timeout + Set Communication Time	1 if success, 0 if fail
Tick Ack	t	116	-	-
Quit Session Ack	q	113	-	-
Protocol Error	e	101	-	-
Set Session-Start Ticks	r	114	[ticks[(2)	Number of ticks to auto-init session, left-padded
Set Communication Timeout	u	117	[timeout](6)	Milliseconds for comm timeout, left-padded
Set Session Timeout	m	109	[timeout](6)	Milliseconds for session timeout, left-padded

Figure 4. DCMP Message Reference Table

3.2 Reference Implementation Design

The reference implementation provides a sample implementation of the DCMP protocol. The device used is a miniature cycle with built-in magnetic cadence sensor. The reference implementation design consists of a trivial hardware design and state-machine based microcontroller software design.

3.2.1 Interfacing Hardware

The hardware design consists of an amplification circuit to amplify the passive cadence sensor output to 5V logic levels used by the microcontroller interrupts. The output is connected to a hardware interrupt pin on an Atmega8 microcontroller. The use of digital interrupts allows asynchronous input, and thus more efficient software than using a polling method. Noise filtering is handled partially in software using typical timing characteristics such as minimum tick interval time, appropriate for the application.

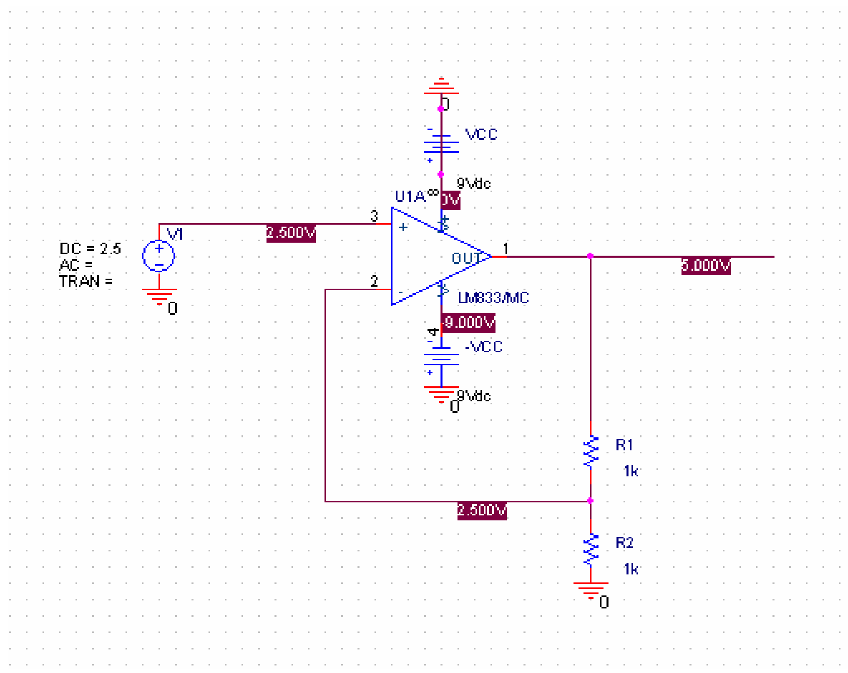


Figure 5. Sensor Amplification Circuit

3.2.2 DCMP State-Machine Software

A state-machine design was used to model the DCMP protocol for an embedded software system. A simple 4-state model was used to strictly enforce adherence to DCMP and allow for easy development. The use of a finite state model reduces the probability of error and ensures synchronization between the device and client software. The state machine design also serves as a self-documenting model of the DCMP protocol design.

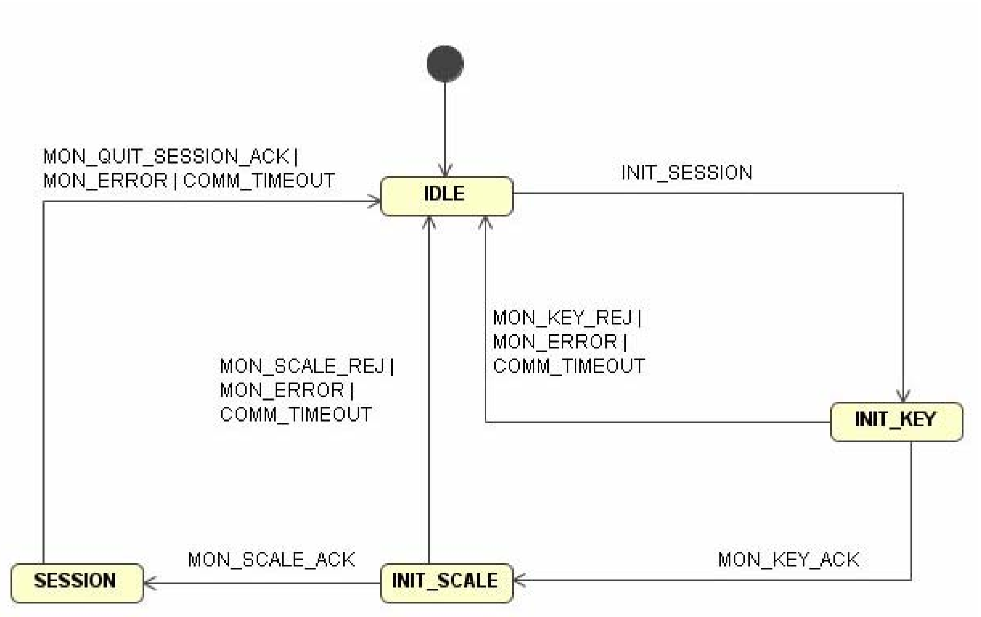


Figure 6. 4-State DCMP Model

The state transitions are inbound or outbound DCMP messages with inbound messages denoted by the "MON_" prefix. The state-machine was designed for compact and easy implementation in C for microcontroller deployment.

3.3 Client Monitor Application (DCMon)

DCMon is the client application responsible mainly for device monitoring with real-time feedback, software/device customization, and session data transfer. The DCMon design was approached as a typical object-oriented software design. The application architecture is generally modeled around the Model-View-Controller pattern.

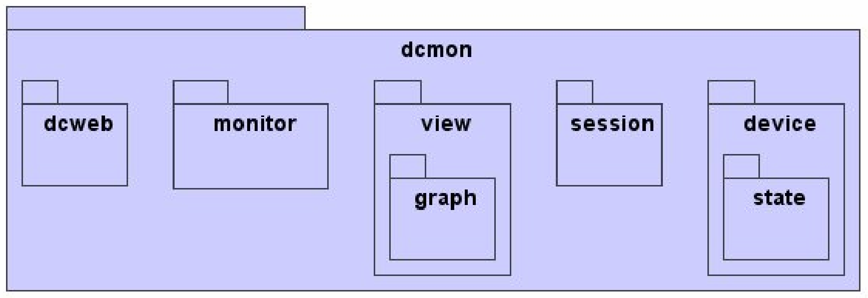


Figure 7. DCMon Package Hierarchy

The software is logically divided into 7 packages within the org.rg.dcmon root package:

i. **device:** Components required to effectively communicate with a device.
   a. **state:** All states and associated behavior for the DCMP state machine.

ii. **monitor:** Components central to the application operation such as the main application loop and the settings system.

iii. **session:** Components required to model an exercise session and utilities beneficial to the application domain.

iv. **view:** User-interface related components such as data views, graph views, and options views.
   a. **graph:** All graph types used in the system.

v. **dcweb:** Components required for communication with the central server i.e. DCDBI.

3.3.1 Design Goals

Several primary design goals were considered during the DCMon design process. In particular, special attention was paid to extensibility, portability, and flexibility.

The system should be extensible in several key areas:

- Graphs
  - Creation of new graph types should be easy and quick. Software doesn't need to worry about any particular graph implementation.
- Device communication channel
  - Use of different device communication channels should be easily adapted. Software should operate independently of comm. channel.
- Server interface
  - Use of different client/server RPC methods should be easily adapted. Software should operate independently of server interface implementation.

DCMon should also be flexible in terms of its user-interface. The application should be non-obtrusive, and feature fully-customizable UI components.

Portability is also a major design goal due to the range of potential OS/platform deployment opportunities for such software.

Furthermore, DCMon should leverage "state of the art" technologies and benefit from use of free/open-source software components.

The DCMon design makes extensive use of established software design patterns to satisfy these goals.

3.3.2 Device Support

DCMP is implemented in software using a state-machine model implemented with the state design pattern. Each state has its own behavior and operates independently allowing a simplified controller.

By forcing devices to adhere to the DCMP protocol, the client software need only worry about the communication channel used. The comm. type is a user-defined setting and new types can be easily added (though not at runtime) to ensure device compatibility.

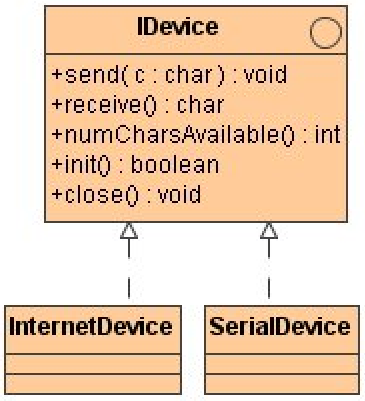


Figure 8. Simplified Example of Device Extensibility

New device links, such as RS232, TCP/IP, USB, Bluetooth, etc. can be easily developed by implementing the IDevice interface.

3.3.3 Graphing System

The DCMon design employs a similar technique as in the device support design for the graphing system. The default graph set includes velocity, average velocity, and a distance graph. However, the design allows new graph types to be rapidly developed - usually only involving a few lines of code.

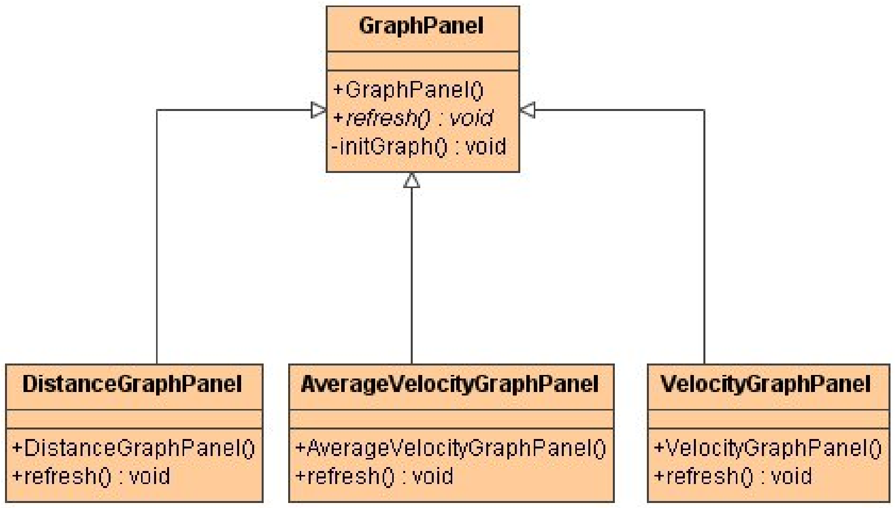


Figure 9. Simplified Example of the Graphing System

The inheritance hierarchy allows a new graph type to be easily developed by inheriting from the GraphPanel base class and implementing its abstract "refresh" method. The refresh method simply populates the graph with particular data and forces a redraw. Thus, new graph types require very little code to develop. Each graph inherits a default behavior and appearance, but this can be overridden in the child class if desired. For example, a basic distance graph can be implemented with the following refresh method:

```java
public void refresh(Session session){
      double d = session.calculateTotalDistance();
      timeseries.addOrUpdate(new Millisecond(), d);
}
```

3.3.4 User-interface System

The requirements specify a "non-obtrusive", "fully-configurable" user-interface system for DCMon. The DCMon design allows for a non-obtrusive UI, with minimal or no user-intervention required. This is achieved via DCMP design and software implementation details more so than the DCMon software design.

Furthermore, all UI "panel" elements are fully customizable and can be scaled, moved, docked, floated, hidden, minimized, tabbed, etc. The view design relies on an open-source package "Infonode Docking Windows" to supply basic functionality for UI customization. Thus, all UI containers in the design, such as graph panels and data panels, inherit from the corresponding InfoNode class.

3.3.5 Settings

The DCMon design allows for a comprehensive settings management system. User settings are made globally available throughout the application via the SettingsManager class implemented with the singleton design pattern. Use of the singleton pattern allows settings to be accessed throughout the application while avoiding multiple instantiations of the class.

Settings are persisted between application executions using a custom XML format. The XML schema and value formats are best documented with an example.

DCMonSettings.xml:

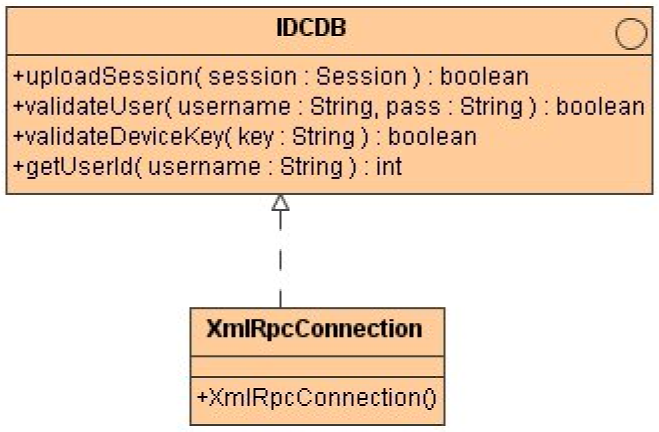
```xml
<?xml version="1.0" ?>

<DCMonSettings>
  <UserName>demo</UserName>
  <Pass> 89e495e7941cf9e40e6980d14a16bf023ccd4c91</Pass>
  <UpdateInterval>100</UpdateInterval>
  <UploadSession>true</UploadSession>
  <SessionStartTicks>10</SessionStartTicks>
  <SessionTimeout>15000</SessionTimeout>
  <CommTimeout>90000</CommTimeout>
  <ServerHost>http://ryangreen.ca/dcweb/server.php</ServerHost>
  <DeviceCommType>DEVICE_SERIAL</DeviceCommType>
  <CommPort>COM4</CommPort>
  <CommBaud>19200</CommBaud>
  <VelocityUnit>KM_H</VelocityUnit>
  <DistanceUnit>M</DistanceUnit>
</DCMonSettings>
```

The tags are self-documenting. Native units are cm for distance and ms for time. Settings are read and changed by the user via a user-interface component accessible from the application menu. Since settings values are enforced via the UI, it is an impossible for a user to supply an invalid value without manually editing the xml file. It is worthy of note that passwords are stored as SHA-1 hashes of the plaintext password. The hashed password can be verified against the stored password in DCDB. The UploadSession setting is significant in that a false setting forces DCMon to store sessions locally as XML-RPC strings (discussed later) for later upload.

3.3.6 DCDBI for DCMon

As DCDBI is intended as an interface between the client and server software, a design and implementation must exist on both sides. Similar to device communication channel, DCMon contains an interface for DCDBI so that implementation details can vary. Note the naming is different due to Java naming conventions.



IDCDB
+uploadSession(session : Session) : boolean
+validateUser(username : String, pass : String) : boolean
+validateDeviceKey(key : String) : boolean
+getUserId(username : String) : int
XmlRpcConnection
+XmlRpcConnection()

Figure 10. Simplified Example of DCDBI Design for DCMon

This precludes the DCDBI design which will be discussed later.

3.3.7 Full UML Class Diagram

(See Insert)

3.4 Centralized Database (DCDB)

The DCDB design is a straightforward relational database design. DCDB is designed around the feature specification to contain all data required to implement the features of the feedback system. DCDB is based on a centralized server-based 3rd-party database engine. This provides the benefits of:

- Efficiency.
    - Optimal use of disk space.
    - Extremely efficient searching, retrieving, and updating.
- Security.
    - Security features are inherent in the system.
- Accessibility.
    - Server-based so can be accessible from anywhere (via DCDBI).
- Concurrency.
    - Built-in concurrency features for many users reading/writing.

The DCDB design can be described exhaustively by the Data Definition Language (DDL) diagram. Data types are not included in the design as they are dependent on the implementation details. Each table provides a unique identifier (primary key). Some tables contain fields which provide a relationship to other tables' primary keys (foreign key) in order to create associations within the data model.

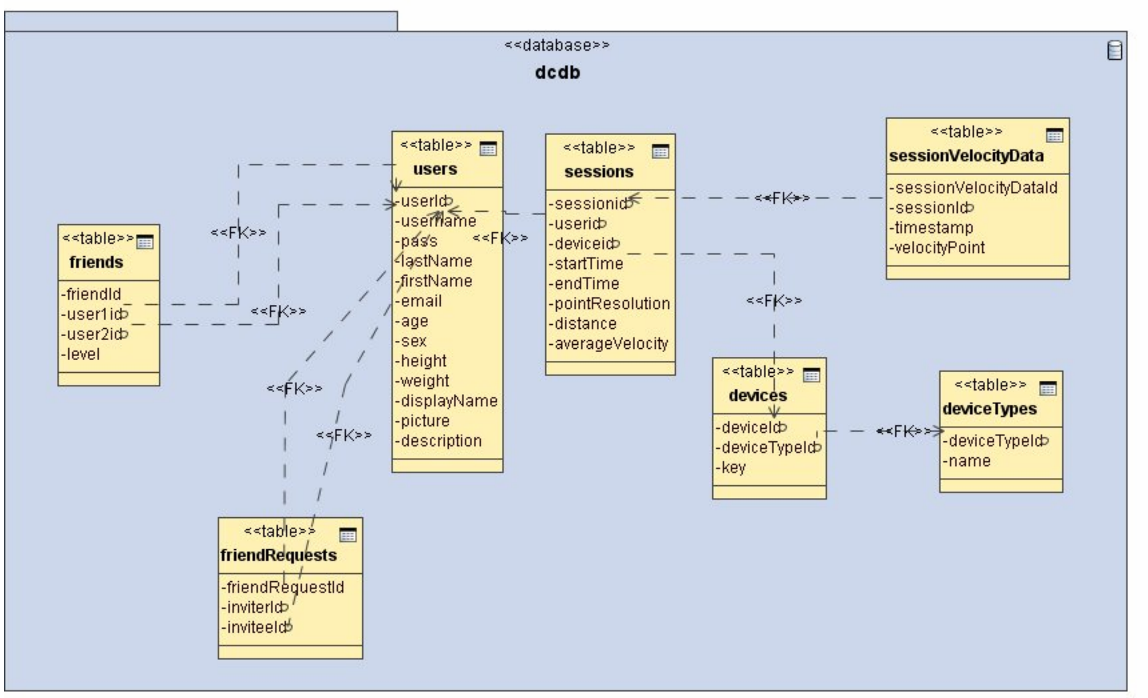


Figure 11. DCDB Design DDL Diagram

3.5 Client/Server Interface (DCDBI)

DCDBI provides a standard interface between DCMon and DCDB. DCDBI is designed around the XML-RPC specification. XML-RPC allows communication via TCP/IP by sending messages as HTTP headers encoded with XML strings sent through port 80 (web server). Thus, XML-RPC provides a commonly available, standardized means to communicate while remaining impervious to networking issues such as firewalls, policies, etc. Furthermore, XML-RPC provides specifications for encoding data types (i.e. dates, strings, etc.) to allow cross-platform compatibility. Common software libraries are available for a variety of platforms to exploit this, allowing easy implementations of XML-RPC systems.

3.5.1 Messages

DCDBI contains a small set of messages (or operations) which are implemented on both client and server side platforms. This set of operations is consistent between the two platforms while the implementations vary.

The current DCDBI message set design includes four messages:

1. uploadSession

> Upload all session data for the completed session. This session data is outlined in 2.3.2.
>
> Returns: Boolean. True if successful.

2. validateUser

> Check validity of the attempted username and password for the active user.
>
> Returns: Boolean. True if valid.

3. validateDeviceKey

> Check validity of the 20-byte device key for the active device.
>
> Returns: Boolean. True if valid.

4. getUserId

> Get the unique user identifier associated with a given username.
>
> Returns: Integer. User id. -1 if not found.

3.6 Internet Application (DCWeb)

DCWeb is the main presentation element of the feedback system. It organizes and presents all exercise data to the user and allows social-interaction and data sharing. DCWeb also provides account registration and user profile management. Since DCWeb is intended to be a multi-user, easily accessible application, it was a natural choice to design DCMon as a web-based application.

Often an important design consideration for web-based applications is how to logically divide the design such that the presentation elements (view) of the design remain separate from the application logic. Thus, a MVC architecture was chosen with the view elements strictly separated from the controller by the use of a “template engine”.

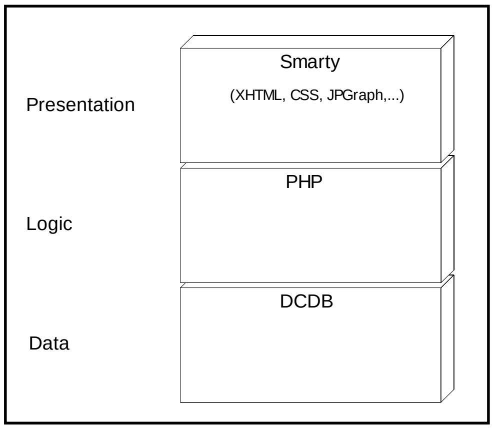
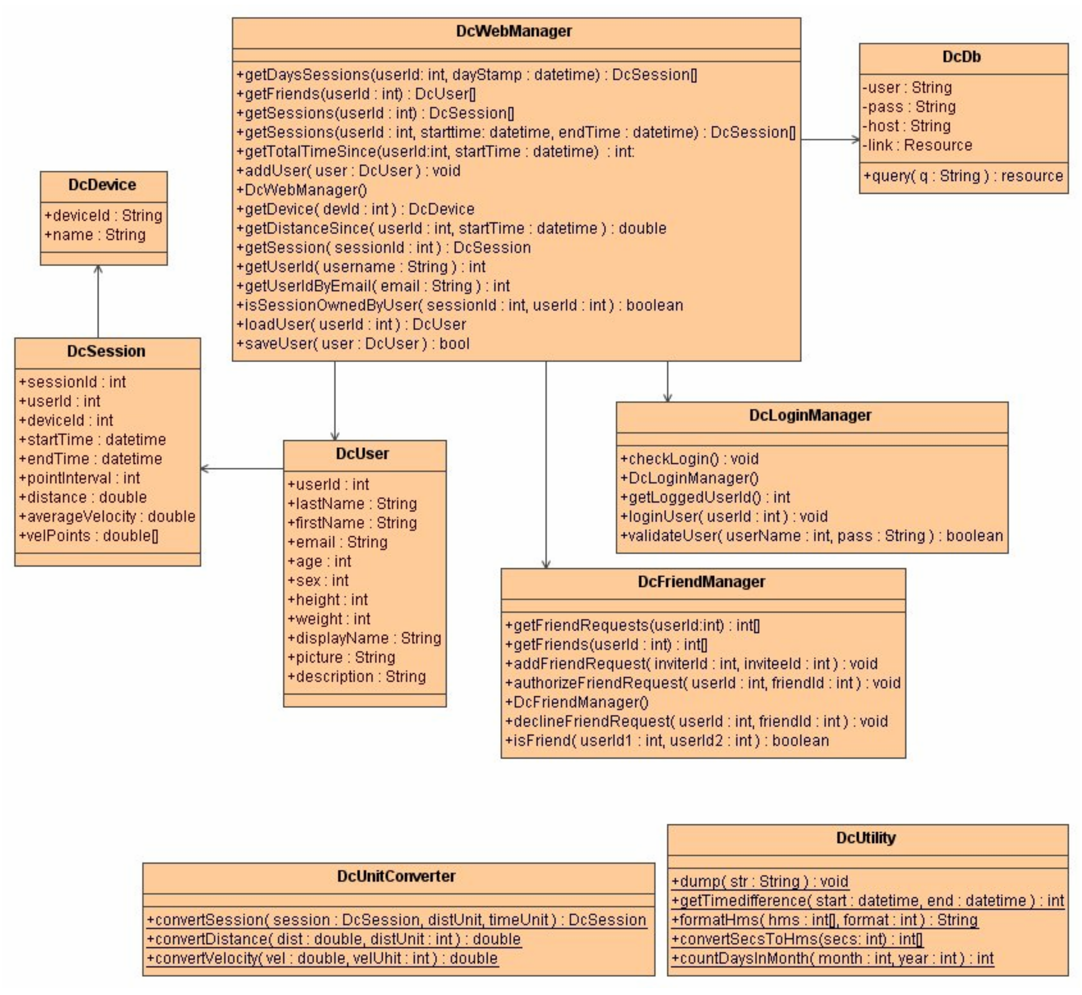


Figure 12. DCWeb Architecture

The use of a template engine or “Presentation Framework” allows a distinct line between logic and presentation elements of the design, thus improving design reusability and scalability, and greatly aiding development and partitioning of development tasks.

The logical layer of DCWeb occupies most of the design since the presentation layer consists of trivial XHTML/CSS data output and the data layer closely mirrors the DCDB design.

The DCWeb design is an OO design targeted for the web platform. The application itself is logically subdivided into a series of separate scripts or "actions" whereby objects typically have shorter lifetimes than a typical OO application. It was thus decided that the design would focus around singleton classes, utility classes, and proxy classes to delegate and perform tasks common to these actions.



DcWebManager
+getDaysSessions(userId: int, dayStamp : datetime) : DcSession[]
+getFriends(userId : int) : DcUser[]
+getSessions(userId : int) : DcSession[]
+getSessions(userId : int, starttime: datetime, endTime : datetime) : DcSession[]
+getTotalTimeSince(userId:int, startTime : datetime) : int
+addUser(user : DcUser) : void
+DcWebManager()
+getDevice(devId : int) : DcDevice
+getDistanceSince(userId : int, startTime : datetime) : double
+getSession(sessionId : int) : DcSession
+getUserId(username : String) : int
+getUserIdByEmail(email : String) : int
+isSessionOwnedByUser(sessionId : int, userId : int) : boolean
+loadUser(userId : int) : DcUser
+saveUser(user : DcUser) : bool
DcDb
-user : String
-pass : String
-host : String
-link : Resource
+query(q : String) : resource
DcDevice
+deviceId : String
+name : String
DcSession
+sessionId : int
+userId : int
+deviceId : int
+startTime : datetime
+endTime : datetime
+pointInterval : int
+distance : double
+averageVelocity : double
+velPoints : double[]
DcUser
+userId : int
+lastName : String
+firstName : String
+email : String
+age : int
+sex : int
+height : int
+weight : int
+displayName : String
+picture : String
+description : String
DcLoginManager
+checkLogin() : void
+DcLoginManager()
+getLoggedUserId() : int
+loginUser(userId : int) : void
+validateUser(userName : int, pass : String) : boolean
DcFriendManager
+getFriendRequests(userId:int) : int[]
+getFriends(userId : int) : int[]
+addFriendRequest(inviterId : int, inviteeId : int) : void
+authorizeFriendRequest(userId : int, friendId : int) : void
+DcFriendManager()
+declineFriendRequest(userId : int, friendId : int) : void
+isFriend(userId1 : int, userId2 : int) : boolean
DcUnitConverter
+convertSession(session : DcSession, distUnit, timeUnit) : DcSession
+convertDistance(dist : double, distUnit : int) : double
+convertVelocity(vel : double, velUnit : int) : double
DcUtility
+dump(str : String) : void
+getTimedifference(start : datetime, end : datetime) : int
+formatHms(hms : int[], format : int) : String
+convertSecsToHms(secs: int) : int[]
+countDaysInMonth(month : int, year : int) : int

Figure 13. DCWeb UML Class Diagram

The central class DcWebManager, provides a high-level interface for the scripts to access the core of the application. All access to DCDB is proxied through DcWebManager, thus use of DCDB is transparent to most of the application, allowing all database queries, for example, to be contained within the same module (this is one of many database-access design patterns). Similarly DcLoginManager, and DcFriendManager provide interfaces to simplify common tasks related to the login system, and friend management, respectively. Other classes provide utilities used throughout the application such as unit conversion, date conversion, debug utilities, etc. Classes such as DcUser, DcSession, and DcDevice are instantiated directly from data contained in DCDB and are directly reflective of the systems data model (DCDB).

These classes are used through user-invoked "scripts" of which the application is comprised. They can be thought of as actions or use-cases of the application though objects do not persist between script executions. The only data (besides DCDB) persisted between script executions is data pertaining to the login system handled by DcLoginManager.

Action List:

1. **registerDevice**

   Register a new device.

2. **registerUser**

   Register a new user account.

3. **login**

   Login to DCWeb.

4. **profile**

View a user profile.

5. **recentSessions**

   View last X sessions.

6. **month**

   Calendar-view of month's sessions.

7. **week**

   Calendar-view of week's sessions.

8. **day**

   Listing of days sessions.

9. **viewSession**

   View the data for a particular session.

10. **sessionGraph**

    Generate a graph of velocity points for a particular session.

11. **addFriend**

    Add a user to friends list.

12. **friendRequests**

    View pending friend requests for a user.

13. **authorizeFriend**

    Authorize a friend request.

14. **friends**

    View all friends for a user.

4 Implementation Details

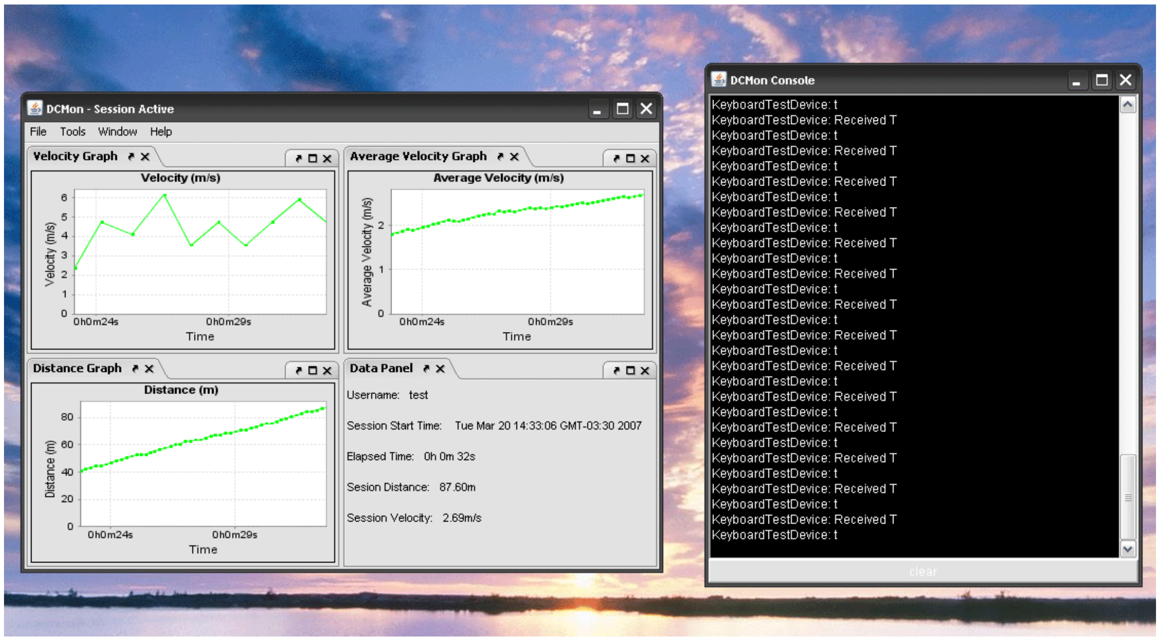
All features included in the software requirements and design have been fully implemented and verified. Some of the implementation details for each component will be discussed with a brief feature introduction from a user-standpoint.

4.1 Client Monitor Application (DCMon)

DCMon was implemented in the Java programming language for platform independence. Several open-source tools were used in the application:

- Charting: JFreeChart
- Communication: Javax Comm API, Apache XML-RPC
- GUI: InfoNode Docking Windows

4.1.1 Basic Feature Tour



Username: test

Session Start Time: Tue Mar 20 14:33:06 GMT-03:30 2007

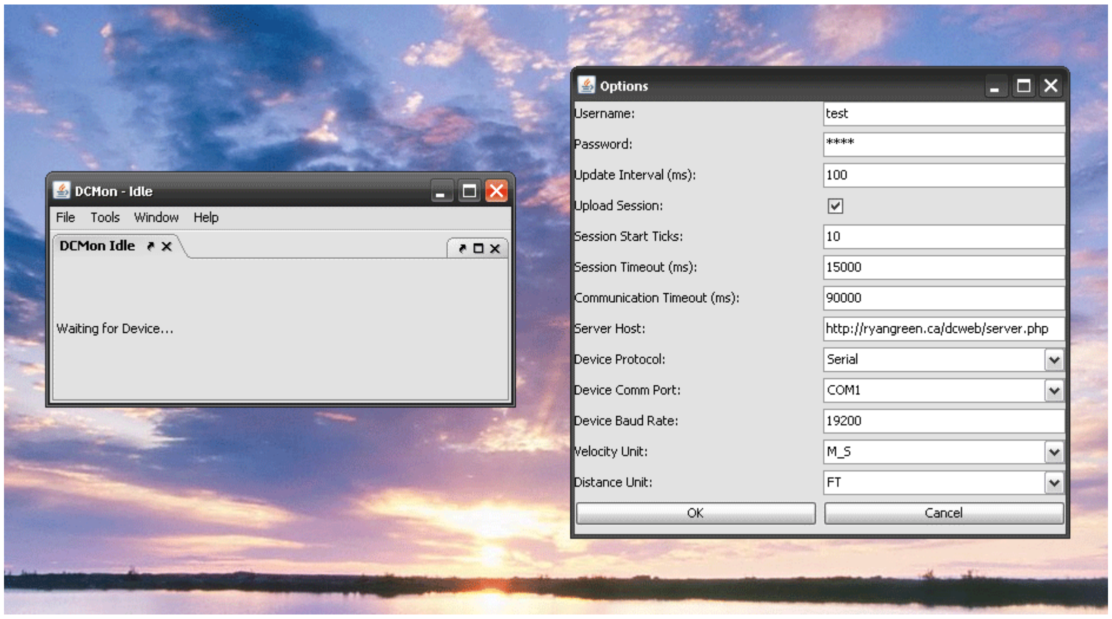
Elapsed Time: 0h 0m 32s

Sesion Distance: 87.60m

Session Velocity: 2.69m/s

```
KeyboardTestDevice: t
KeyboardTestDevice: Received T
KeyboardTestDevice: t
KeyboardTestDevice: Received T
KeyboardTestDevice: t
KeyboardTestDevice: Received T
KeyboardTestDevice: t
KeyboardTestDevice: Received T
KeyboardTestDevice: t
KeyboardTestDevice: Received T
KeyboardTestDevice: t
KeyboardTestDevice: Received T
KeyboardTestDevice: t
KeyboardTestDevice: Received T
KeyboardTestDevice: t
KeyboardTestDevice: Received T
KeyboardTestDevice: t
KeyboardTestDevice: Received T
KeyboardTestDevice: t
KeyboardTestDevice: Received T
KeyboardTestDevice: t
KeyboardTestDevice: Received T
KeyboardTestDevice: t
KeyboardTestDevice: Received T
KeyboardTestDevice: t
KeyboardTestDevice: Received T
KeyboardTestDevice: t
KeyboardTestDevice: Received T
KeyboardTestDevice: t
```

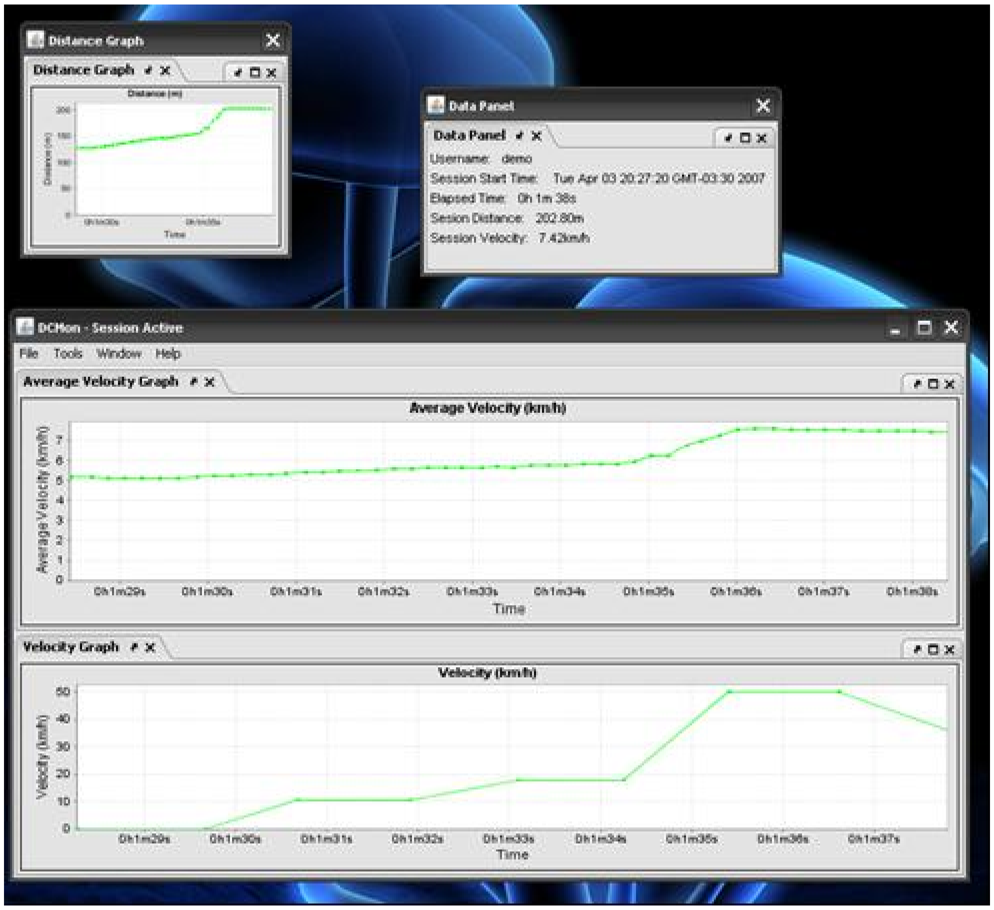
Figure 14. DCMon with Default Graph Set and Console



Waiting for Device...

Username:	test
Password:	****
Update Interval (ms):	100
Upload Session:	✓
Session Start Ticks:	10
Session Timeout (ms):	15000
Communication Timeout (ms):	90000
Server Host:	http://ryangreen.ca/dcweb/server.php
Device Protocol:	Serial
Device Comm Port:	COM1
Device Baud Rate:	19200
Velocity Unit:	M_S
Distance Unit:	FT

Figure 15. DCMon Options Panel



Username: demo
Session Start Time: Tue Apr 03 20:27:20 GMT-03:30 2007
Elapsed Time: 0h 1m 38s
Sesion Distance: 202.80m
Session Velocity: 7.42km/h

Figure 16. DCMon with Custom UI Configuration

4.2 Centralized Database (DCDB)

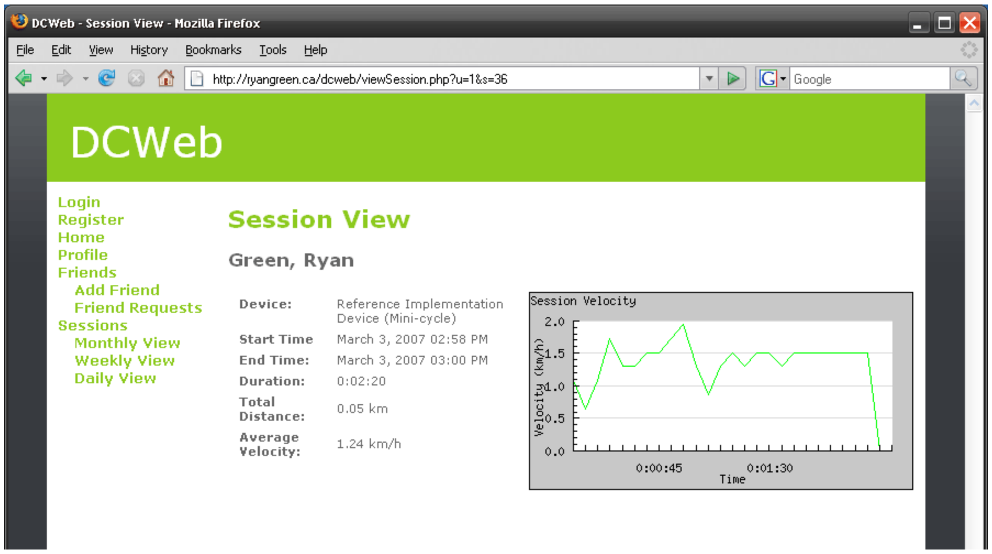
DCDB was implemented for the open-source MySQL database system. The database is located on a central database server accessible via TCP/IP.

4.3 Internet Application (DCWeb)

DCWeb was developed in the PHP language on the Linux/Apache environment. The Smarty template engine was used heavily in the implementation and the view was developed in XHTML and CSS.

4.3.1 Basic Feature Tour

The features shown are not comprehensive. Login, register, add friend, friend requests, weekly view, and daily view not shown.

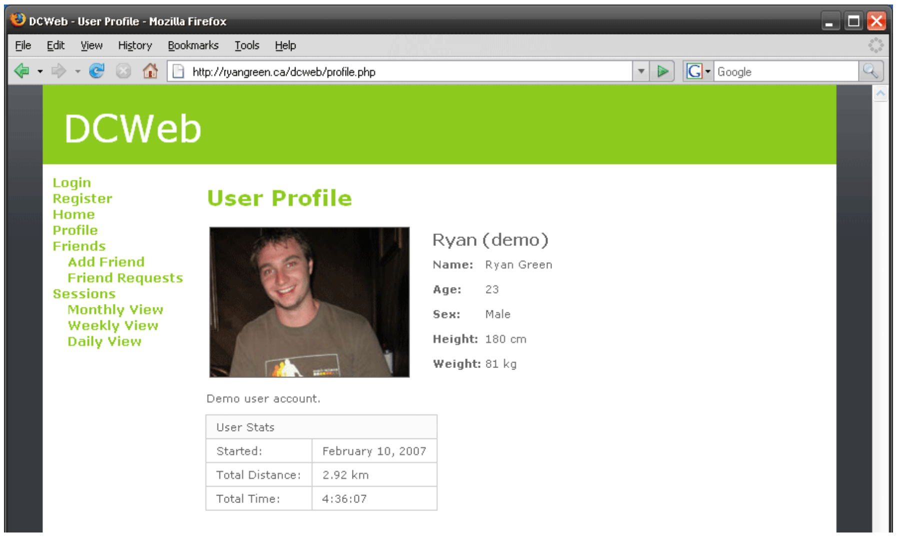


Session View

Green, Ryan

Device:	Reference Implementation Device (Mini-cycle)
Start Time	March 3, 2007 02:58 PM
End Time:	March 3, 2007 03:00 PM
Duration:	0:02:20
Total Distance:	0.05 km
Average Velocity:	1.24 km/h

Figure 17. DCWeb Detailed Session View


DCWeb - User Profile - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://ryangreen.ca/dcweb/profile.php
Google
DCWeb
Login
Register
Home
Profile
Friends
Add Friend
Friend Requests
Sessions
Monthly View
Weekly View
Daily View

User Profile

Ryan (demo)

Name: Ryan Green

Age: 23

Sex: Male

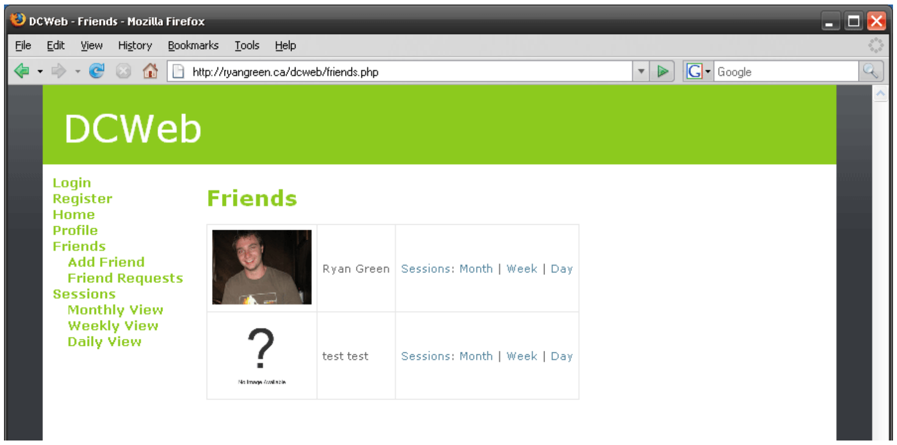
Height: 180 cm

Weight: 81 kg

Demo user account.

User Stats	
Started:	February 10, 2007
Total Distance:	2.92 km
Total Time:	4:36:07

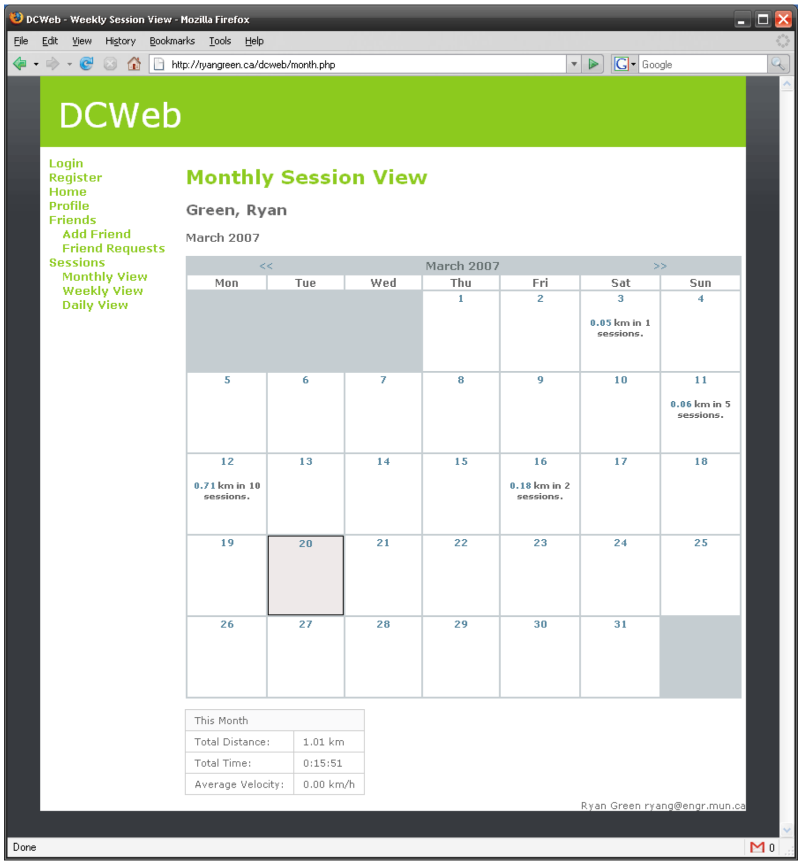
Figure 18. DCWeb Profile View


DCWeb - Friends - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://ryangreen.ca/dcweb/friends.php
Google
DCWeb
Login
Register
Home
Profile
Friends
Add Friend
Friend Requests
Sessions
Monthly View
Weekly View
Daily View

Friends

	Ryan Green	Sessions: Month | Week | Day
?	test test	Sessions: Month | Week | Day

Figure 19. DCWeb Friends View


DCWeb - Weekly Session View - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://ryangreen.ca/dcweb/month.php
Google
Done

DCWeb

Login
Register
Home
Profile
Friends
  Add Friend
  Friend Requests
Sessions
  Monthly View
  Weekly View
  Daily View

Monthly Session View

Green, Ryan

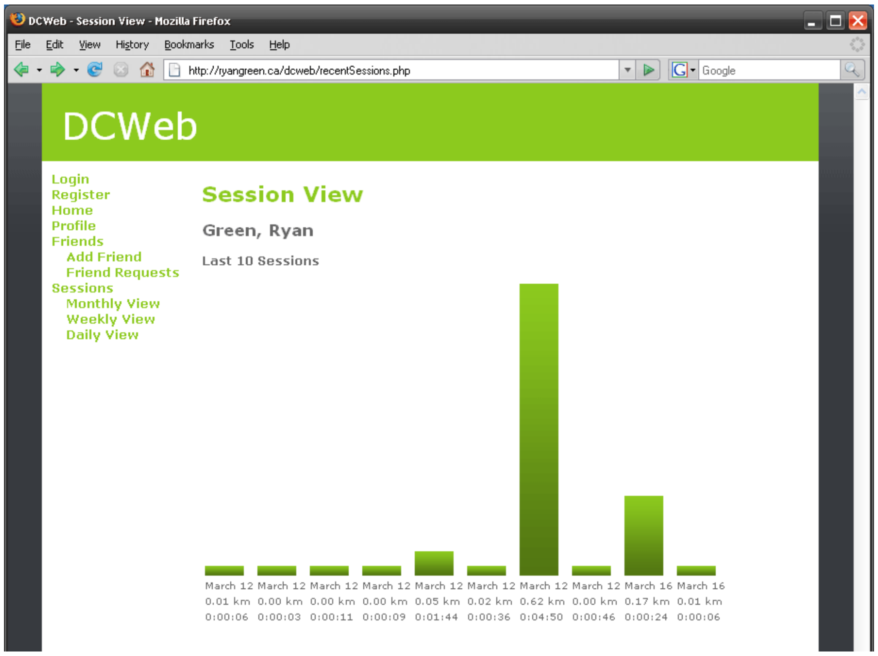
March 2007

<<			March 2007		>>	
Mon	Tue	Wed	Thu	Fri	Sat	Sun
			1	2	3 0.05 km in 1 sessions.	4
5	6	7	8	9	10	11 0.06 km in 5 sessions.
12 0.71 km in 10 sessions.	13	14	15	16 0.18 km in 2 sessions.	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

This Month	
Total Distance:	1.01 km
Total Time:	0:15:51
Average Velocity:	0.00 km/h

Ryan Green ryang@engr.mun.ca

Figure 20. DCWeb Monthly View


DCWeb - Session View - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://ryangreen.ca/dcweb/recentSessions.php
Google

DCWeb

Login
Register
Home
Profile
Friends
  Add Friend
  Friend Requests
Sessions
  Monthly View
  Weekly View
  Daily View

Session View

Green, Ryan

Last 10 Sessions

March 12	March 12	March 12	March 12	March 12	March 12	March 12	March 12	March 16	March 16
0.01 km	0.00 km	0.00 km	0.00 km	0.05 km	0.02 km	0.62 km	0.00 km	0.17 km	0.01 km
0:00:06	0:00:03	0:00:11	0:00:09	0:01:44	0:00:36	0:04:50	0:00:46	0:00:24	0:00:06

Figure 21. DCWeb Recent Sessions View

5 Testing Strategies

Testing was performed on each component individually during development and on the system as a whole following integration. Ad-hoc testing was performed on a per-component basis. Several test-harnesses were developed in order to automate and simplify testing activities. Test harnesses were developed for:

- Device emulation
- Server emulation (DCDBI)
- Client emulation

End-to-end system testing was performed with multiple user accounts using both test harnesses and the reference implementation device.

Testing and bug-fixes were generally performed during the same phase so there are no known critical bugs to date.

6 Issues and Limitations

There are several acknowledged issues and limitations in the system. Due to the relatively short development time there was not adequate time to develop comprehensive use cases and automated testing procedures. Thus, there may be robustness issues in the software that could be solved with more rigorous code and use-case analysis. Closely related to this is the issue of security. A multi-user system with server side software such as this requires extensive security measures. Security should have been more tightly integrated into testing and a rigorous post-implementation should be performed post-implementation. However, security was a major consideration throughout design and development. This is evident in the use of password hashing, XML-RPC encryption, device/user authentication, etc.

A trivial, though significant improvement to the system would be the improvement of user-interface presentation. More visual polish and look and feel improvements would have a positive impact on user experience.

Lastly, the DCMP message set could be expanded to support more features. While the basic message set is adequate, features such as heart rate, inclination, and resistance would be beneficial to the protocol. This could be a straightforward addition in the future.

7 Conclusion

The project outcome was a definite success with a stable implementation of attempted requirements. The end result is a highly- usable computer feedback system with a solid base for future expansion. Ultimately, the effectiveness of the system will be judged by the user experience. Robustness issues would certainly improve with increased user involvement and have a positive impact on the systems effectiveness. The main determining factor in the use and success of such a system however lies in the quality and availability of device implementations. The simple, open DCMP protocol should allow for device implementations to be attempted by people with varying levels of expertise. A means of exercise coupled with the software feedback and social support system will ideally lead to positive exercise trends.

References

1 Anon, *Physical Exercise.* http://en.wikipedia.org/wiki/Exercise#Exercise_benefits

2. J.J. Annesi, *Effects of computer feedback on adherence to exercise*, Percept Mot Skills. 1998 Oct;87(2):723-30.

3. Treiber FA, *Social support for exercise: relationship to physical activity in young adults*. Prev Med. 1991 Nov;20(6):737-50.