Computer Support for Home-Based Health Care

by

Kartik M. Mani

Submitted to the Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2001

Author .....

Department of Electrical Engineering and Computer Science

August 31, 2001

Certified by .....

Peter Szolovits

Professor of Computer Science and Electrical Engineering

Thesis Supervisor

Certified by .....

William J. Long

Principal Research Scientist

Thesis Supervisor

Certified by .....

John Ankcorn

Research Staff

Thesis Supervisor

Accepted by .....

Arthur C. Smith

Chairman, Department Committee on Graduate Theses

Computer Support for Home-Based Health Care

by

Kartik M. Mani

Submitted to the Department of Electrical Engineering and Computer Science
on August 31, 2001, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Home-based health monitoring systems have been shown to be effective in helping patients manage chronic illness, but require significant effort from both the patient and support staff. This thesis describes the design and development of a software package that provides physicians and patients an interactive, automated system to assist with home-based health care. It discusses the requirements for effective home-based monitoring and considers different architectural approaches to providing automated acquisition and storage of patient physiological readings as well as gathering qualitative data through patient interaction. Heart-At-Home, built for Congestive Heart Failure (CHF) patients, is a prototype implementation that partially meets these requirements. When a patient stands on the scale, the system automatically reads the weight, gathers relevant contextual data by prompting the user through a touch-screen visual interface, and stores all readings in the patient's lifelong medical record.

Thesis Supervisor: Peter Szolovits
Title: Professor of Computer Science and Electrical Engineering

Thesis Supervisor: William J. Long
Title: Principal Research Scientist

Thesis Supervisor: John Ankcorn
Title: Research Staff

Acknowledgments

I would like to acknowledge several people who were instrumental in making this thesis possible.

First and foremost, I would like to thank my advisors, Dr. Peter Szolovits, PhD, Dr. Bill Long, PhD, and John Ankcorn for their guidance and mentorship.

John: I want to first thank you for building the hardware for the first version. Your technical advice on my designs and different features throughout the project was invaluable. I especially want to thank you for taking the time to help me develop my initial outline and draft for this write-up. Looks like I finally got it to sound like English.

Bill: Thank you for your help in designing the interactive framework for this project. Your insight into patient interaction and CHF was vital to making this system useful. I also want to thank you for being so available when I needed help, especially when I had a draft to be edited every day in the last two weeks. I'll let you know if I'm coming back for the Boston Marathon next year, but I'll be sure to train for it this time.

Peter: Thank you for taking me on for this project. You always had confidence in my ability and have been a great mentor to me. I also want to thank you for your advice on my career and long-term goals, and I'll let you know if I ever make that business plan a reality.

It was an honor and a pleasure working with the three of you. I couldn't have asked for a more intelligent, insightful and entertaining group of advisors. I sincerely hope we can work together again in the future.

My second set of acknowledgements goes to my parents and my sister Kalpana.

Kalpana: Your ambition, drive and impressive accomplishments have served as an inspiration to me to succeed in what I do (even if you go to Harvard now). Thank you for always being there to guide me and motivate me to do the best I can.

Mummy and Daddy: Your constant support and faith in me has helped me overcome the difficulties of MIT (and I don't just mean tuition). My accomplishments including this project could not have been achieved without your support. For this reason, I am dedicating this thesis to you both to show my appreciation for everything that you have done for me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The concept of home-based health care has shown itself to be a promising and effective method of managing chronic illness. Although several types of home-based systems exist, many still require manual operation (such as frequent patient calls or visits) and/or intrusive devices that are not only expensive but difficult to use. The real promise of home-based health care is in a system that would provide automated acquisition of physiological readings, intuitive patient interaction for qualitative assessment, and centralized and accessible patient medical records. This type of system would empower patients to take control of their own health care management.

The goal of this thesis is to design and develop such a system. The prototype program, Heart-At-Home, focuses on Congestive Heart Failure patients, a group which can benefit greatly from this type of solution.

1.1 Background and Significance

1.1.1 Congestive Heart Failure

Congestive Heart Failure (CHF) is a significant cause of death and disability in the United States and has been a frequent target for home-based monitoring programs. As many as 4.7 million Americans currently suffer from CHF and its annual incidence is estimated to be 400,000. The cost of treating these patients is over $38 billion, $23

billion of which is for hospital stays, making CHF the most costly cardiovascular illness in the United States. It is also the most frequent cause of hospitalization among the Medicare (over 65) population[8].

CHF patients have difficulty managing their disorder, as indicated by their high re-hospitalization rate, ranging from 29% to 47% within 3 to 6 months of discharge. This rate is also affected by behavioral issues such as dietary and medical non-compliance, lack of proper education about the disease, and social isolation[8].

CHF is particularly appropriate for home-based monitoring because a patient's weight change can be an accurate indication of their condition. Patient data collected at the New England Medical Center and analyzed by this group (see Figure 1-1) shows a strong correlation between a sudden weight gain in CHF patients and an increase in heart failure symptoms[5].

1.1.2 Home-Based Health Care

Home-based monitoring has been shown to be a less expensive method of treating chronic illness than repeated clinical visits and can reduce the rate of re-hospitalization and patient resource use[4]. In addition, patients prefer a home-based solution if they are constrained geographically, by work, or by physician non-referral. Studies have shown that the effectiveness of a home-based management program can be as good or better than one run on-site at a clinic or rehabilitation facility[1].

A significant benefit of home-based monitoring is that it puts a great deal of responsibility in the hands of the patient. Since the patients themselves are responsible for performing routine tasks associated with their health, they become more motivated to better manage their condition. Furthermore, because they are receiving feedback on their daily habits and activities, they become more knowledgable about their condition and more compliant about their medication and therapy routine. Home monitoring, therefore, acts as a tool to motivate and guide patients through feedback, while leveraging patients' innate concern for their own health[6].

Home-based monitoring systems have been successful in providing preventative care and improved quality of life for patients. Most of these systems, however, re-

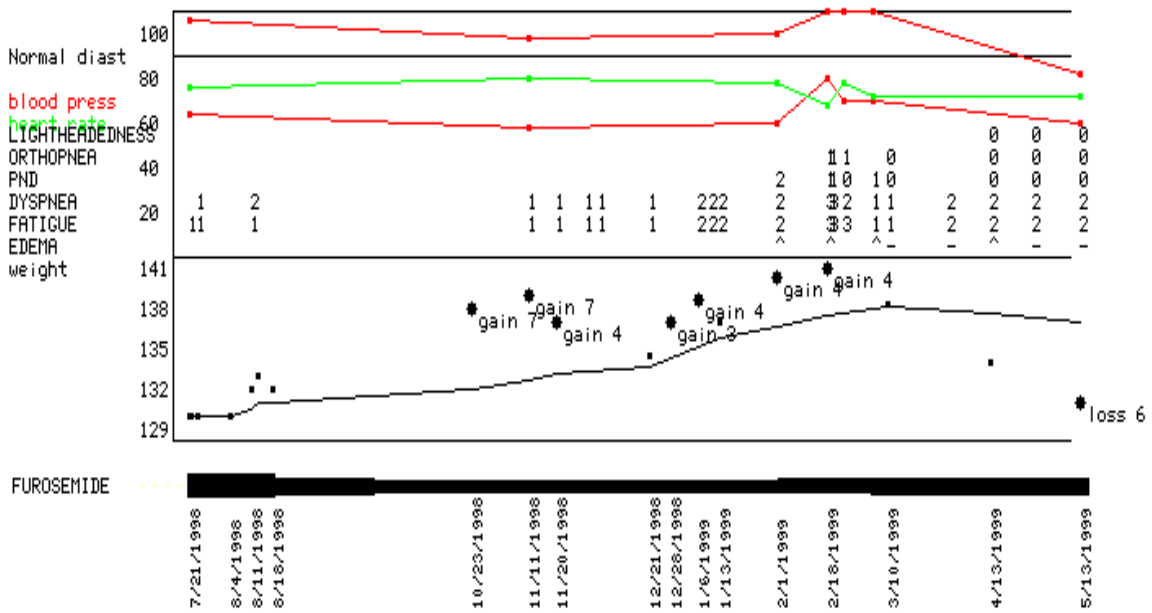


Figure 1-1: Correlation of Weight Change with Heart Failure

quire telephone support centers, a staff making routine calls to enrolled patients, or the patient to call in their daily physiological readings. Comprehensive home-based management systems for CHF have been successful at improving patient functional status and reducing medical resource usage, but all require significant manpower and continuous effort to maintain and run[9].

Heart-At-Home addresses these issues by providing an automated, interactive framework for home-based health care monitoring. It is a solution that can decrease the resources necessary to maintain a home-based health management program, provide an automated means to acquire both physiological and qualitative patient readings, and motivate the patients themselves to better manage their own health.

1.2 Design Goals

Heart-At-Home has been developed with several distinct goals in mind. These are listed below.

Automated Care Monitoring This system is intended to be a tool to provide physicians and patients a means to automate the health care monitoring process.

Extensible framework The system must be adaptable to handle different devices, input media and patient disorders.

Patient Interaction The system should provide an interaction framework that can gather contextual data from patients in addition to their physiological readings.

Accessible Medical Records The system must store patient data in a accessible, standardized repository. This feature allows physician access and automated analysis of patient data.

Secure Infrastructure The system should make use of standard encryption and authentication software to ensure patient privacy.

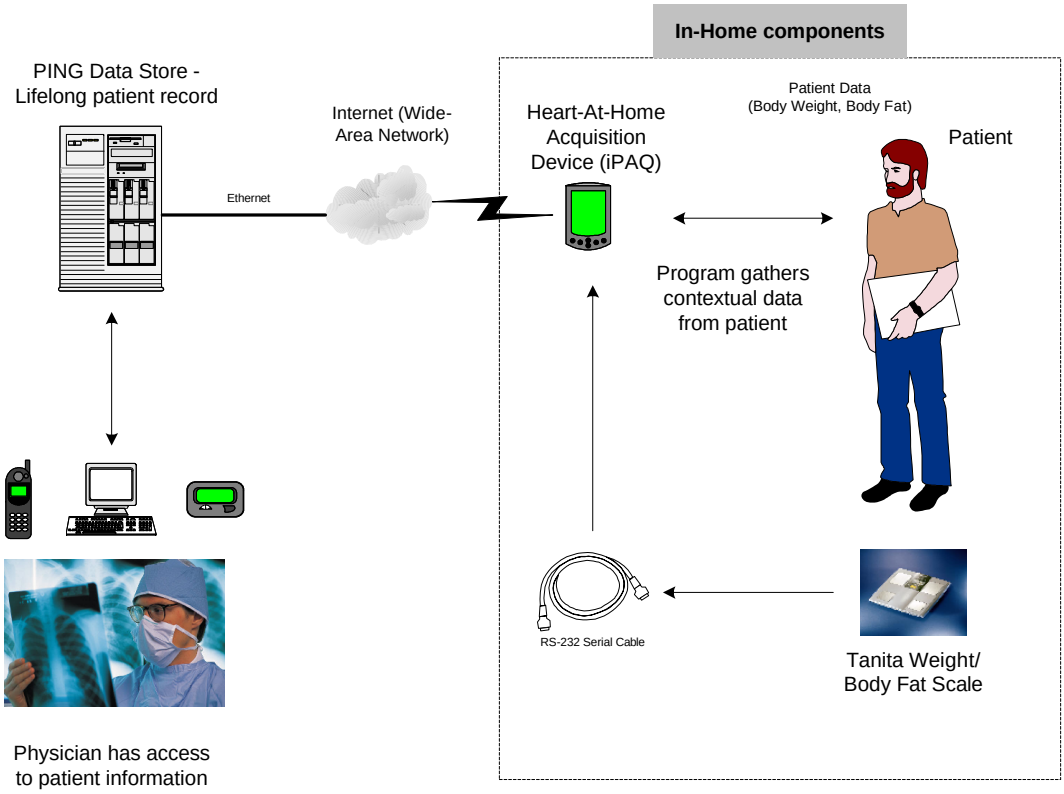


Figure 1-2: Heart-At-Home Architecture

1.3 System Architecture Overview

The Heart-At-Home architecture is shown in Figure 1-2. The system consists of the following basic components.

1.3.1 Diagnostic Devices

The Heart-At-Home software is designed to work with devices that gather physiological data from patients. The prototype program uses a weight and body fat scale equipped with an RS-232 serial output. Heart-At-Home has been designed, however, to be device-independent and can acquire data from any device with digital output. The current version also has support for a digital blood pressure/pulse monitor manufactured by Medwave, Inc.

1.3.2 Patient Interaction

Heart-At-Home has the unique feature of interacting with patients and gathering qualitative data about their health. These interactions are dynamic and can make use of the physiological readings taken as well the patient's recent medical history. They are designed to emulate the questioning of medical professionals and provide an automated way to retrieve valuable contextual information about a patients health without significant manpower.

1.3.3 Medical Record Storage

Heart-At-Home can automatically append both physiological and contextual readings into the patient's medical record. The medical repository used is a new standard called Personal Inter-networked Notary and Guardian (PING), which is discussed in Section 2.2.

1.3.4 Security and Authentication

Heart-At-Home has support for secure transmission of patient data over the internet using Secure Sockets Layer (SSL). A simple authentication scheme for CHF patients has been built in as well. The PING software has also been designed to make use of standard authentication and cryptographic packages.

Chapter 2

Requirements for Effective Home-based Monitoring

Despite the success of home-based monitoring, there are two features that can make it far more effective. The first is automating the acquisition and storage of physiological and contextual patient data. The second is a medical record repository that is patient-controlled and accessible by home monitoring systems.

2.1 Automation

Automation allows more people to enjoy the benefits of home monitoring while requiring fewer resources and less time of both the patient and/or support staff. When the patient takes a daily reading, the system automatically appends it to their medical record instead of having them call it in or record it manually. Decreasing the amount of work the patient has to do makes a home-monitoring system more accessible and less error-prone, increasing the likelihood that patients will use it.

2.1.1 Interaction and Contextual Data

In most cases, a physiological reading is not completely sufficient to determine whether medical intervention is necessary. However, combined with contextual data from the

time of the reading (related to diet, exercise, and medication), a far more accurate judgement can be made on the condition of the patient. Some systems do exist that can automatically record physiological readings (using devices equipped with modems that dial into a support center), but lack the ability to gather valuable qualitative data on the patient's condition.

Automated interaction can be built into a home monitoring system by prompting the user for various pieces of contextual information based on their current physiological reading and reading history. Not only is this capability valuable for detecting changes in the patient's physical condition, but it provides a way to acquire data on patient compliance with their health care program (if they stop taking their medication, for instance, this will be noted immediately). Combining this interactive framework with automated updating of the patient's medical record provides a more complete and effective home-monitoring solution.

Further discussion of patient interaction can be seen in Chapter 3.

2.1.2 Warning Detection

Home-based monitoring can be improved by automated analysis of the acquired readings and alerting support staff if a problem is detected. For CHF patients, weight increase can be a strong indication of a worsening condition (see Figure 1-1). Algorithms to detect this trend could examine a patient's recent record to see if there is cause for medical intervention. In addition, these algorithms could look at the patient's qualitative readings to determine whether or not they are following their diet and medication regime[5].

This type of analysis is very useful to physicians, but can also be presented to patients as daily advice or assessments. If needed, the system could recommend that patients change their medication or diet (within bounds set by the physician). It could also give them qualitative feedback on their compliance and help them better understand warning signs they should be looking for. This type of feedback would help the patient learn more about their condition and how to better manage it[5].

2.2 Lifelong Medical Records

Providing an accessible and standardized repository for patient medical records makes home-based monitoring more effective. Readings taken at home and stored in this repository are available to physicians or automated analysis programs immediately. In addition, a home-based system that can access a patient's history is capable of more customized and relevant patient interaction.

2.2.1 PING Background

Although health care capabilities have dramatically improved in the past decade, the collection, storage, and control of access to patient medical information has not kept pace. Individual institutions have become adept at storing individual pieces of information, but it is difficult to get access to the aggregate of information collected over a patient's lifetime. Furthermore, although health care institutions's use of information technology and electronic medical records has increased significantly, patients themselves have remarkably little control over their own medical records[7].

The Personal Inter-networked Notary and Guardian (PING) patient record system is a new standard being developed collaboratively by researchers at Harvard Medical School and the MIT Laboratory for Computer Science. It is designed to serve as a secure, distributed, patient-controlled repository for personal medical records. Providing patients control over their medical records gives them the opportunity take an active role in presenting information to their physicians. Their physicians are able to accurately retrieve and store data that is relevant to the patients medical history. In addition, PING provides a means to easily aggregate patient information across different health care institutions and providers.

2.2.2 Current Problems

Current health information systems store fragmented and poorly indexed patient records. This problem is aggravated because patients often change their health care provider. Although there have been efforts to allow institutions to share electronic

records, the competitive nature of the health care industry has given individual institutions little incentive to support broad sharing of their records. Furthermore, if sharing is agreed upon by the providers involves, it is very cumbersome due to incompatible data storage formats[7].

A lifelong medical record repository has several requirements. Because of the mobility of patients both geographically and between health care providers, patient data must be portable. Patients should be able to grant access to their physician or health care institution. Data should be maintained securely, both in storage and in authentication and authorization of access operations. Several systems have been attempted meet these requirements, but none have yet achieved the level of security, portability, and protection of patient confidentiality required by a medical record system. PING is an attempt to build a system that address all of these issues in its architecture and leverages the internet and standard security and portability standards.

2.2.3 PING Architecture

PING relies on the Internet to provide a distributed repository capable of two-way communication between the storage facility and trusted external agents (human users, software programs, devices). It makes use of standard internet technologies including the HTTP protocol for transfer of commands between the client and server, leveraging the wide availability of compliant software.

The PING system consists of PING servers across the Internet and various agents who read or modify records (known as objects). These agents are assigned various privileges (set by the patient) to perform any of 5 basic operations: create a record, read a record, modify an existing record, delete a record, and annotate. A set of agents possessing certain privileges is known as a role (the patient himself is given the role of Owner).

Patient records are stored using an eXtensible Markup Language (XML) format, which can represent both directories and the information held within them. To support distributed data, a PING document may contain references to other PING objects located on other servers across the Internet. This distributed architecture also

enables versioning and mirroring schemes[7].

PING makes use of several cryptographic technologies, and uses role-based authentication to determine access rights that apply to a particular agent. Access to PING servers is also done over a Secure Socket Layer (SSL) ensuring data sent over the internet is encrypted.

2.2.4 Effect on Home Monitoring

PING is an appropriate repository to use with a home-based monitoring system. Its distributed architecture, patient-controlled access, and XML-based storage allow for automated recording of readings and retrieval of records for analysis.

Chapter 3

Patient Interaction

A key feature of Heart-At-Home is the gathering of contextual data (see section 2.1.1) from patients in addition to their physiological readings. This data pertains to their diet, exercise, adherence to medication and other important factors in their health care management. It can be used in conjunction with the patient's physiological readings to more accurately assess the patient's health. Gathering this type of data, however, requires interacting with the patients in a form they can understand and prompting them with questions relevant to their disorder and its management.

3.1 Interface Media

There are several different types of available interface media that could be used to interact with users. Several of these, such as text, console, or keypad-based interfaces, are not appropriate for the current system as they are inflexible or require too much effort from the user. Since many patients will be elderly, may have poor vision, and are not comfortable interacting with a computer, the interface used must be able to convey information and retrieve input in the simplest manner possible. In addition, the interface must be flexible because the type of content presented by the system can change depending on the patient and their disorder. A speech-based interface and a touch-screen graphical user interface (GUI) are two forms of media that would work well with patients using a home monitoring system.

To provide the user the maximum flexibility, the interactions should be able to leverage the available interface media and operate in a mixed-mode setting.

3.2 Authentication

A basic problem with having software interact with patients is determining the user's identity. Typical forms of authentication in software, such as a username and password, are too complicated to use with patients during an interaction. The simplest way to identify the user is to have the program guess their identity based on their physiological reading, prompting them for confirmation. If the guess is not correct, the program should ask them to identify themselves from a list of current users of the software (stored in a local cache).

The above method is appropriate for an in-home setting where there is most likely a single user of the system. If the same interaction software or monitoring system were used in a clinic or hospital (with a potentially large number of different users), a more robust form of user identification, such as voice-based recognition, would be required.

Visual authentication is also a possibility for interactions. However, it should be noted that it can only be used in a setting when a user is comfortable having a camera present. Patients generally keep items associated with their health in the bathroom (especially devices such as a scale), and many would not be in favor of having a camera in that location.

3.3 Data Validation

Another problem associated with interactions is verifying that the user intends to record a reading. There may be cases where the user accidentally initiates an interaction with potentially invalid data. An example of this scenario with CHF patients is when they stand on their scale while holding something significant enough to alter their weight reading (i.e a pet). A possible method to solve this problem is simply

Table 3.1: Non-linearity of Patient Interaction

Speech	**Visual**
C: Are you dressed the same?	C: Are you dressed the same?
P: Yes, but I ate a large meal for dinner	P: Yes
C: Did you eat salty food?	C: Did you eat a large meal?
P: Yes I did	P: Yes
C: Shall I record this weight?	C: Did you eat salty food?
P: Yes	P:Yes
	C: Shall I record this weight?
	P: Yes

asking the user at the end of the interaction whether they wish to store the reading in their medical record.

3.4 Non-Linear Interaction

A more natural interaction style can be achieved by avoiding fixed sets of choices offered by rigid, preconceived indexing and command hierarchies. Information is often available at different times in the dialogue, which is very likely to happen between different user interfaces (such as speech and visual). In order to be intuitive, the system should be able to recognize what information is relevant and be able to use it as soon as it becomes available.

For instance, Table 3.1 shows two different interaction possibilities, one using a speech-based interface, and the other using a touch-screen. C represents the computer program, P represents the patient's response.

The example illustrates that information may be provided out of the expected order. If the interaction were fixed, the questioning would seem redundant because the user already provided the relevant answer to the next question. Using a knowledge-based architecture, the system could identify that the user supplied an answer to another variable in the interaction and could assign it automatically. When the interaction reaches a point where a particular value is needed, it would see that it was already set and wouldn't ask the user. If the user had answered "Yes but I ate

a large salty meal," the interaction should go to the next undefined variable which is whether or not to record the weight.

3.5 Patient Customization

Interactions should be able to change from patient to patient, even with the same type of disorder. Since patients will be performing routine tasks as part of their health care management, customization of the system to suit their individual needs will have a large impact on usability and compliance. It should be possible to modify the system to provide a familiar type of interaction. For instance, the user should be able to request a particular voice setting (such as the voice of a friend or relative). Small changes such as these will make the user experience much more comfortable.

3.6 Multiple Languages

Users will not necessarily speak English, so any content information (such as prompts or displays) must be separated from the actual interaction language. The interaction engine should not require re-compilation to change a language setting. Instead, there should be a configuration file specifying all the language settings for the variables (which is specific to a locale). If the language were to change, only this configuration would need to be changed and the interaction should still run the same way.

There may be cases where the actual logic of the interaction may need to change to be natural in another language. In this case, it would be necessary to program another interaction based on this new interaction pathway.

3.7 Programmatic Approaches

The actual programming of interactions brings up an important design issue. There are two possibilities for the implementation of interactions. The first is a simple scripting language that would not require a programming background to use. This

would be useful for nurses or family members who could alter the interaction without any knowledge of the code. The second option is to write the interactions in a general purpose language such as Java. In this case, it would not be as easy to program interactions, but the programmer would have access to all the code libraries giving them far more capability. This section looks at these two possibilities and their associated advantages and disadvantages.

3.7.1 Scripting Approach

A scripting language provides users a simple, easily modifiable interface to designing and developing interactions. It would be usable by nurses or patients who wished to modify existing interactions or develop completely new ones. A scripting language also supports the knowledge-based approach to software design of separating the knowledge of a program from the code itself.

For the purposes of interactions, the following list identifies the basic requirements in designing a usable scripting language.

Basic Data Types The script author must be able to use basic data types, including but are not limited to integers, floating point numbers, strings, datetimes, booleans, and choices. The language must also support sets and their manipulation.

Variables The scripting language must have support for variables including their definition as one of the above types, value assignment and value retrieval.

Branching The scripting language would have to support branching based on some expression evaluation.

Math Functions The scripting language would have to support basic mathematical operations and comparisons.

Macros or Sub-procedures The scripting language would have to have some way to define functions or macros that can be called from elsewhere in the script.

Although simple and easy to learn, one of the issues with a scripting language is that in general, it is difficult to correctly anticipate the complete set of required features. If the design doesn't include a particular library of functions that is needed, it would have to updated in the language core interpreter/compiler. As more features get added, the scripting language would start to look like a more general purpose language, but lacking the core design features of one. In the end, this may lead to a convoluted language that is, in fact, more difficult to use than a general purpose language.

3.7.2 Java Approach

Java (like other general purpose languages) has the advantage that it gives the programmer complete flexibility with coding the interaction and does not place any architectural limitations on what can be done. The programmer has access to not only the capability of the interaction software engine, but also the standard Java libraries which include several useful computational and utility classes. Interactions could also leverage the advantages of an object-oriented framework such as inheritance. Java also has the advantage that the runtime environment is much more stable than a new home grown script runtime environment.

The disadvantage of this approach is that programming interactions would be limited to those with a background in Java and basic object-oriented programming. Although Java itself is not difficult to learn, this choice would limit the number of people capable of programming effective interactions.

Chapter 4

Heart-At-Home Interactions

Chapter 3 detailed the difficulties in developing an interactive software package. This chapter outlines the specific implementation details of the Heart-At-Home package, and how the requirements are met in this project. Although some features such as speech support are not yet completely implemented, the architecture has been designed to accommodate extensions and new features without a drastic change to the core code base.

4.1 State-Machine Architecture

Heart-At-Home interactions are internally modelled as finite state machines (FSM) in order to account for the non-linearity of interactions (see section 3.4). The FSM is an appropriate framework for this type of program because it doesn't define a specific pathway through an interaction, but rather all possible pathways through its state transitions. At runtime, the pathway is dynamically generated based on the inputs of the user and variable values.

Each interaction continues to run until its state is set to a predefined complete state or if it is externally instructed to halt (in which case the state will be preserved). The Heart-At-Home engine controls the actual interaction framework, so new interactions need only to specify their states and transition logic to be plugged in and used.

The state machine provides the architecture to support non-linear usage of user input, but this feature is not implemented in the current version due to the lack of full speech support.

4.2 Java-based Authoring

Heart-At-Home interactions are programmed in Java. Based on the discussion from Section 3.7, it was determined that a Java code base would be most appropriate versus a pure scripting approach (although this was attempted and later dismissed). Interaction classes have access to all the features of the Heart-At-Home engine giving them the versatility and flexibility for all types of disorders. They also make use of inheritance allowing for code re-use and more compact interaction code. Java-based interactions are also more extensible and can take advantage of other features of the engine such as multiple device support and two-way communication (see Section 5.1.2).

Although a pure scripting framework was not chosen in this version, a scripting interface to Java-based interactions is under development for a later version. A scripting candidate for the CHF interaction can be seen in section 7.5.

The Java-based CHF interaction used with the prototype is detailed in Section 4.4 and the source code can be seen in Section C.1. There is also a guide to programming new Java interactions for this system in Section A.3.5.

4.3 Knowledge Representation

The knowledge-base of interaction variables serves two main purposes. The first is to separate content-specific information (such as displays and text the user will see and/or hear) from the state definitions and transition logic; the second is to maintain an abstraction between the user interfaces and the interaction code.

4.3.1 Variables

Each interaction is accompanied by a configuration file which is specific to a locale (a combination of a language and country, i.e U.S. English is en_us). This file contains the definitions for the variables used in the interaction. Each variable is one of six predefined interaction datatypes (boolean, integer, decimal, datetime, string, or choice) and has a name, a prompt or an initial value, and other optional attributes specific to each type (i.e a choice variable can have options associated with it). Each variable is therefore required to have either an initial value or a means to obtain that value (the prompt for the user).

The variables also have a scope inside the interaction assigned in the configuration file. They can reside as part of the patient's profile (their name, date of birth, etc.), as part of the current session (contextual data to be collected), as a transient part of the interaction (a variable used for intermediate computation), or as a constant in the interaction (variables whose value does not change). Only profile and session variables are stored into the patient's record. All variables are reset before each interaction except the constant variables which are not reset unless the engine is stopped and restarted.

The configuration file for CHF can be seen in Section C.2.

4.3.2 Internationalization and Customization

An interaction can support internationalization by simply translating all the user prompts and display information in the configuration file into a different language. The variable names must be kept constant because they are referenced directly from the interaction code, but their attributes may change to accommodate a new patient or new locale. The interaction will still run identically but with the new display/prompt information. The location of the configuration file can be specified in the engine properties so changing the configuration can be done very easily.

Interactions are also capable of placing dynamic information into user prompts and displays. This part of the code makes use of standard formatting libraries that can

internationalize well. This dynamic information is limited to dates, names, numbers and other language-independent data. More information about this can be seen in Section A.3.5.

4.3.3 Abstraction of User Interface

The interaction code itself has no concept of what interface(s) the system is using to communicate with the user. Instead, it has a variable Application Programmer Interface (API) for retrieving values of variables to be used during its execution. When a value of an interaction variable is needed, a call to this API is made. The resulting function looks at the variable and checks whether its value is defined already. If not, it then prompts the user (using the prompt from the configuration file) for the value and returns the result. This approach ensures that information can be recorded at any point in the interaction and still be used effectively at a later stage in its execution. If a variable has already been assigned a value, the interaction will not prompt the user for redundant information.

4.4 Congestive Heart Failure Example

This CHF example is a simple interaction based on the patients weight change. The following sections describe the variables used and the actual state transitions present in the interaction.

4.4.1 CHF Variables

Table 4.1 lists the variables used in the CHF interaction. The table is split into 3 sub-tables: the profile, session, and transient variables. As can be seen, the profile variables are ones that are only used when the patient is a user the system doesn't recognize. The session variables consist of the contextual questions about diet and health that would be asked in response to a particular shift in weight. The transient variables are those which are used purely for intermediate computation and should

Table 4.1: Congestive Heart Failure - Interaction Variables

Name	Type	Prompt	Scope
name	string	Hello new user. What is your name?	profile
dateofbirth	datetime	What is your birthday?	profile
gender	choice	What is your gender?	profile
dressed	boolean	Are you dressed the same as last time?	session
eatmore	boolean	Did you eat more than normal?	session
eatsalty	boolean	Did you eat salty food?	session
water	boolean	Are you drinking a lot of water?	session
diuretic	boolean	Did you take your water pill?	session
skipmeal	boolean	Did you skip a meal?	session
feelsick	boolean	Did you feel sick?	session
diuresis	boolean	Did you pee off a lot?	session
sweating	boolean	Are you sweating a lot?	session
identity	boolean	Is this {0}?	transient
record	boolean	Shall I record this weight?	transient

not be stored with the rest of the data collected.

4.4.2 Interaction Pathway

Figure 4-1 shows the states and transitions possible when this interaction is executed. Each box in the diagram represents a particular state in the interaction. Its name is in bold and the variables that are used in that state are shown in parentheses immediately after. The state transitions are described along the arrows or next to the state they go to.

When the patient stands on the scale, the interaction enters the *authenticate* state where the system attempts to identify the user based on their weight. If no match is found (the measured weight is not within a predefined range of any users recent weight), the system assumes it is a new user and enters the *newuser* state.

If there is a potential match, the interaction switches to the *query* state where it prompts the user for confirmation. If the user is in fact the one in question, the interaction moves to the *analyze* state to interpret the weight reading. If not, the system assumes it is a new user and moves to the *newuser* state.

In the *analyze* state, the system compares the current weight reading to the patient's most recent one. If the difference is below a certain threshold (which may vary from patient to patient), there is no need for contextual data so it switches to the record state. If the difference is above the threshold, however, the interaction goes either to the increase state or decrease state where it prompts the user about their diet and health through the appropriate variables. The next transition goes to the record state, where the user is prompted to verify that this is a valid reading, at which point the interaction ends.

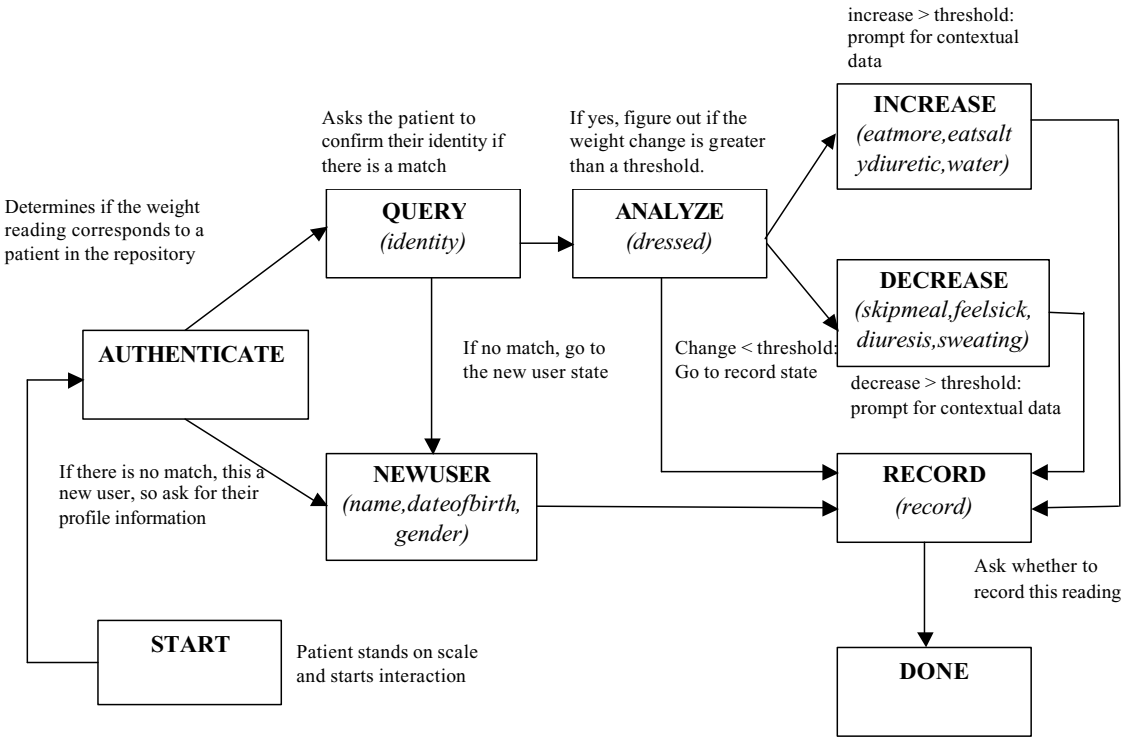


Figure 4-1: Congestive Heart Failure - Interaction Pathway

4.4.3 Interaction Walk-through

Table 4.2 runs through a typical interaction a patient would have. In this scenario, P represents the patient, John Doe, whose current weight reading has decreased 4 pounds since his most recent one. Actions are shown in parentheses. C represents

the Heart-At-Home software. The right column shows the state transitions that the interaction is taking.

Table 4.2: Congestive Heart Failure Example

Interaction Dialogue	**Interaction State**
(John Doe stands on scale, triggers CHF Interaction to start. The initial state is the one which identifies the user	START
(Heart-At-Home looks in internal cache of users and finds John Doe, a match to the current reading within 4 pounds)	AUTHENTICATE
(Heart-At-Home is unsure of its guess at the user's identity and prompts them for confirmation) C: Is this John Doe? P: Yes (John confirms his identity, so Heart-At-Home begins to analyze his current weight reading)	QUERY
(The CHF Interaction understands that some changes in weight are related to non-physiological reasons, and verifies that the user is keeping parameters consistent) C: Your weight has decreased since my reading from yesterday. Are you dressed the same? P: Yes (Since John's weight has decreased, the interaction begins to ask questions about possible reasons for this change)	ANALYZE
C: Did you skip a meal? P: Yes C: Did you feel sick or nauseous? P: No (Heart-At-Home now needs to verify that this reading is a valid one and should be recorded in John's medical record)	DECREASE
C: Shall I record this weight? P: Yes (Heart-At-Home communicates with the PING Server and adds the current weight and contextual readings answers to the to John's record. It also updates its local cache with these readings	RECORD
	DONE

Chapter 5

Design Details

The last chapter discussed how interactions are implemented in the Heart-At-Home system. This chapter will provide some implementation details for the other components of Heart-At-Home, including the devices, user interfaces and integration with the PING software

5.1 Device Support

Heart-At-Home makes use of at least one diagnostic device, which is represented by a Device class in the engine (see Section B.1). The Device class is an abstraction for any object that provides data input. The CHF module uses a modified weight and body fat scale with a digital output.

5.1.1 Using Devices

Devices make use of a DataSource object which is an abstraction for any type of communication channel (serial communication ports, TCP sockets), allowing for reuse of code when devices use the same form of digital transmission. Adding support for a new device requires programming a new class (extending the base Device class) that understands the format of the incoming data stream and can parse it into usable values (see Section A.3.3). Currently DataSources exist that support reading from

serial ports, network sockets and the system console (for debugging). Interaction programmers have direct access to the Device object and can retrieve any typed value that was read in.

5.1.2 Two-way Communication

Although this version of Heart-At-Home uses a scale with one-way digital output, the Device API supports two-way communication for new devices and interactions that may require more complicated data acquisition. This communication can occur at any point during the interaction. This feature could be useful when more sophisticated devices emerge that may require using specific two-way protocols. The scale used with the CHF interaction is capable of measuring weight and body fat percentage, but requires configuration for the user including height and gender in order to get both readings. Two-way communication could be used to send the scale the user information once he or she is identified and retrieve the body fat reading.

5.2 User Interfaces

Heart-At-Home can support different user interfaces to allow users to more easily customize their experience. The engine has a base interface UserInterface that has an implementing class for each supported media type. The implementing class deals only with interaction variables and handles converting the user information into the usable typed information based on the variable type.

5.2.1 Visual/Touch-screen

The current version of Heart-At-Home has a swing-based GUI that can be used as a touch screen interface on a Compaq iPaq handheld device.

5.2.2 Speech Interface

A speech-based interface is in development using the Speech engine from Spoken Language Systems at the MIT Laboratory for Computer Science. Although not complete, this feature was considered while designing Heart-At-Home and the user interface and interaction architecture have been built to accommodate it in future releases.

5.2.3 Mixed-Mode Operation

Heart-At-Home can use multiple interfaces at the same time. Many interactions consist of sets of questions that are suited to a particular interface type. For instance, choice variables with many options are not conducive to speech, while boolean questions with yes/no answers are easily answered by speech. For this reason, the user can leverage both interface types during an interaction.

5.3 Patient Data Repository

Heart-At-Home has a Repository abstraction representing any storage system for patient data. Extension classes for PING and a local repository for serialized objects (for caching and debugging) have been built into the system. It should be noted that the PING effort is ongoing and the integration may change in the future pending a stable release of the PING software.

5.3.1 PING Integration

A test environment for PING was set up in at MIT LCS (heimlich.lcs.mit.edu:3000). Heart-At-Home acts as a software agent on this server and makes use of the PING client API.

5.3.2 Security and Authentication

Heart-At-Home supports secure communication with the PING Server, although this has not yet been built into the version of PING currently in use. User Identification for the CHF interaction is performed by recognizing the patient based on weight locality to their last reading and prompting them for confirmation.

Chapter 6

System Evaluation

This chapter describes how the prototype met the requirements and design goals outlined in Section 1.2. Some features were not implemented in this version and could not be properly evaluated or tested.

6.1 Automated Care Monitoring

Heart-At-Home was successful in adding automation to the home monitoring process. Not only can it interface with diagnostic devices, but was also successful at appending this data to the patient's PING record without any manual intervention by the patient. Automated warning detection (see Section 2.1.2) for disorders was not built into this version of Heart-At-Home and thus could not be tested in the scope of this project.

6.2 Extensible Framework

Heart-At-Home was designed and built with sufficient abstraction and generality that extending it to accommodate new disorders, devices, and interactions was possible without significant effort. This characteristic was tested when programming support for Hypertension patients and a Medwave Vasotrac digital blood pressure (BP) and pulse monitor.

6.2.1 New Devices

Support for the Medwave Vasotrac was successfully added to Heart-At-Home without any changes to the engine code. This extension was able to make use of Heart-At-Home's default support for serial port communication, which simplified the development process. More information on adding new devices can be seen in Section A.3.3.

6.2.2 New Interactions

A simple interaction for Hypertension was programmed into the Heart-At-Home system. This also required no modification to the engine code. The interaction authenticated the patient by a prompt for identity (because blood pressure is not stable enough to identify a user with), then asked simple questions about their diet (caffeine intake) and activity (exercise, stress level, etc.). A guide to programming new interactions can be seen in Section A.3.5.

6.3 Patient Interaction

Heart-At-Home does have a functional interaction component that uses a graphical user interface (touch screen). This component has been successful in prompting users for contextual, qualitative data and storing it in the patient's PING record. Interactions have been built for both CHF and Hypertension, but could not be tested as to their intuitive usability because of the lack of a complete speech-based interface and a patient base to pilot the system.

6.4 Medical Record Storage

Heart-At-Home has support for the PING repository system and has successfully updated records after acquiring a weight reading and contextual data from a user. Heart-At-Home also has support for adding new repositories if new standards/technologies

emerge that prove to be appropriate.

6.5 Secure Infrastructure

Although architecturally supported, robust security and authentication have not been implemented in this version of Heart-At-Home. They are vital features in this type of system and careful consideration has been taken to account for them in the software design process.

Chapter 7

Future Work and Conclusions

There are several features which could not be added in the time-frame of this project.

7.1 Automated Analysis

The data entered into the PING record could be automatically analyzed for early warning signs of a patient entering a more critical condition. This type of feature is not a direct extension of Heart-At-Home but is an important aspect of home-based monitoring.

7.2 Device Recognition

There are some communication protocols, such as Bluetooth, that have automatic resource recognition built into them. These technologies could be used to automatically know when a device is being used or is in close proximity.

7.3 Authentication

Improved authentication is required for situations involving multiple users and for general security requirements. Different approaches include a security proxy model

(where users authenticate themselves against a security proxy), voice-based authentication or smart-card based identification.

7.4 Full Speech Support

The true usability of Heart-At-Home will not be seen until interactions can be as intuitive as possible. Speech support is under development, and is an essential piece of an effective, interactive home monitoring system.

7.5 Interaction Scripting

A scripting language that is translated into Java before execution could be useful in extending the usability of Heart-At-Home to a larger user-base. Family members and nurses could alter the interactions allowing further customization for patients. Some design work has gone into prototyping a scripting language candidate for the CHF interaction. One such example is shown below.

```
decimal: diff
set: patients
set: matches
patient: p

state authenticate {
    matches = match(patients,"weight",5)
    if (size(matches) == 0)
        goto(newuser)
    else
        goto(query)
}

state newuser {
    getValue("name")
    getValue("birthdate")
    getValue("gender")
    goto(record)
}
```

```
state query {
    if (getValue("identity",p) == true)
        goto(analyze)
    else
        goto(newuser) }

state analyze {
    diff = getDeviceReading("weight") - getReading(p,"weight")
    if (abs(diff < 2)
        goto(record)
    if (diff > 0)
        goto(increase)
    else
        goto(decrease)
}

state increase {
    if (diff > 5)
        getValue("drinkwater")
    else
        getValue("eatsalty")
    getValue("eatmore")
    goto(record)
}

state decrease {
    if (abs(diff) > 5) (
        getValue("diuresis")
        getValue("sweating")
    else (
        getValue("skipmeal")
        getValue("feelsick")
    )
}

state record {
    if (getValue("record") == true)
        save()
    goto(done)
}
```

This script is simple and exposes a straight-forward method to make use of vari-

ables without knowledge of object-oriented design or Java. It follows the same state transitions as the Java version but is significantly shorter and much easier to understand and modify.

7.6 Conclusions

Heart-At-Home is an extensible software package that is intended to assist patients and physicians with home-based health care. Its automated and extensible framework address several of the current problems with home-based solutions. By automatically interfacing with different diagnostic devices, gathering contextual data through patient interaction, and storing readings in the patient's lifelong medical record, Heart-At-Home is a comprehensive solution that can significantly improve the effectiveness of home-based health care and the quality of life of patients with chronic illness.

Appendix A

System Usage

A.1 Heart-At-Home Software

This chapter is a user guide for the Heart-At-Home system. It contains information on how to use it and customize it for a particular user scenario.

A.2 Using Heart-At-Home

A.2.1 CVS Repository

All source code, configuration files, and documentation (including the source for this document) for Heart-At-Home are stored in a Concurrent Version System (CVS) repository. The CVS root for this server is

:pserver:<user>@bebe.lcs.mit.edu:d:\cvsroot

The root of the repository is the *medg* module. To request an account on the CVS server, please look at the contact information listed on http://medg.lcs.mit.edu/. For more information on CVS, please look at http://www.cvshome.org/.

The medg directory is arranged as follows:

+ –medg	The root of the project
+ –class	Directory for output class files
+ –docs	Directory for project documentation
+ –thesis	Source files for this thesis document
+ –api	Javadoc API Documentation
+ –etc	Properties files for Heart-At-Home
+ –include	Directory for configuration files for interactions
+ –lib	Directory for external libraries/JAR files
+ –src	Directory for Java source code
+ –var	Directory for log files

A.2.2 Source Hierarchy

The Heart-At-Home source code is written in Java and has a following package hierarchy associated with it.

+ –org	The root of the source tree
+ –medg	
+ –console	Heart-At-Home Control Panel files
+ –devices	Default package for new devices
+ –engine	Heart-At-Home Engine Core
+ –interaction	Default package for new interactions

+ –interfaces	Default package for new user interfaces
+ –patient	Heart-At-Home Patient API
+ –repository	Default package for new repositories
+ –sources	Default package for new DataSources
+ –test	Package for test scripts
+ –util	Package for utility object

A.2.3 Javadoc Documentation

Javadoc API Documentation is available through the CVS repository or off of the Clinical Decision Making Group's project website.

A.2.4 Compilation

Heart-At-Home uses the Ant compile tool from Apache. The makefile (build.xml) is found in the *medg* folder at the root of the project. The Ant binaries can be found at http://jakarta.apache.org/ant/index.html.

The project can be compiled in the different modes listed below. All builds must be performed while in the root directory. Ant looks at all .java files in the source tree so any new additions will be compiled with the project without special configuration.

Command	Result
ant	Builds new source files not yet compiled
ant clean	Deletes the entire output tree
ant prepare	Creates the output directory
ant debug	Builds new source files with debug information
ant release	Builds new source files with no debug information
ant javadoc	Generates Javadoc API Documentation

A.2.5 Properties File

Heart-At-Home uses a single properties file (medg.properties) located in the *etc* folder of the project. The only required properties are the class files for the four components of the engine (Interaction, UserInterface, Device, and Repository) specified by *(Component Name).Class*. An example is shown below.

Interaction.Class=org.medg.interaction.CHFInteraction

All additional properties specified must be prefixed with the class name they apply to. For instance, in order to set a property for the specific CHF Interaction, it must be specified as

CHFInteraction.Threshold=10

An example properties file is shown below.
#MEDG PROPERTIES FILE

#Interaction Properties
Interaction.Class=org.medg.interaction.CHFInteraction
Interaction.File=/medg/include/CHFInteraction.xml
CHFInteraction.Threshold=10

#UserInterface Properties
UserInterface.Class=org.medg.interfaces.ConsoleInterface

#Device Properties
Device.Class=org.medg.devices.ConsoleDevice

#Repository Properties
Repository.Class=org.medg.repository.ObjectRepository
Respository.ObjectPath=/medg/obj/

A.2.6 Classpath Settings

There are several third party libraries in the Heart-At-Home package which are located in the *lib* folder in the project tree. Each one must be referenced in the system classpath in order to run properly. In addition, the output directory for Heart-At-Home, medg/class/, must also be in the system classpath.

A.2.7 Running the Engine

The Heart-At-Home engine requires the Java Virtual Machine version 1.3 or later (binaries available at http://java.sun.com). The Engine has been tested with version 1.3 but Sun Microsystems claims backward-compatibility with new versions of the runtime environment.

The Heart-At-Home software can be run in two different modes. The standard mode is command line, but there is a graphical Control Panel that is useful for debugging.

Command Line

To run the engine, the class references in the medg.properties file must be in the system classpath. The command to run the engine is

java org.medg.engine.Server

The following debug output should appear on the console.

INFO Tue Aug 21 16:04:03 EDT 2001
Acquisition Server STARTING

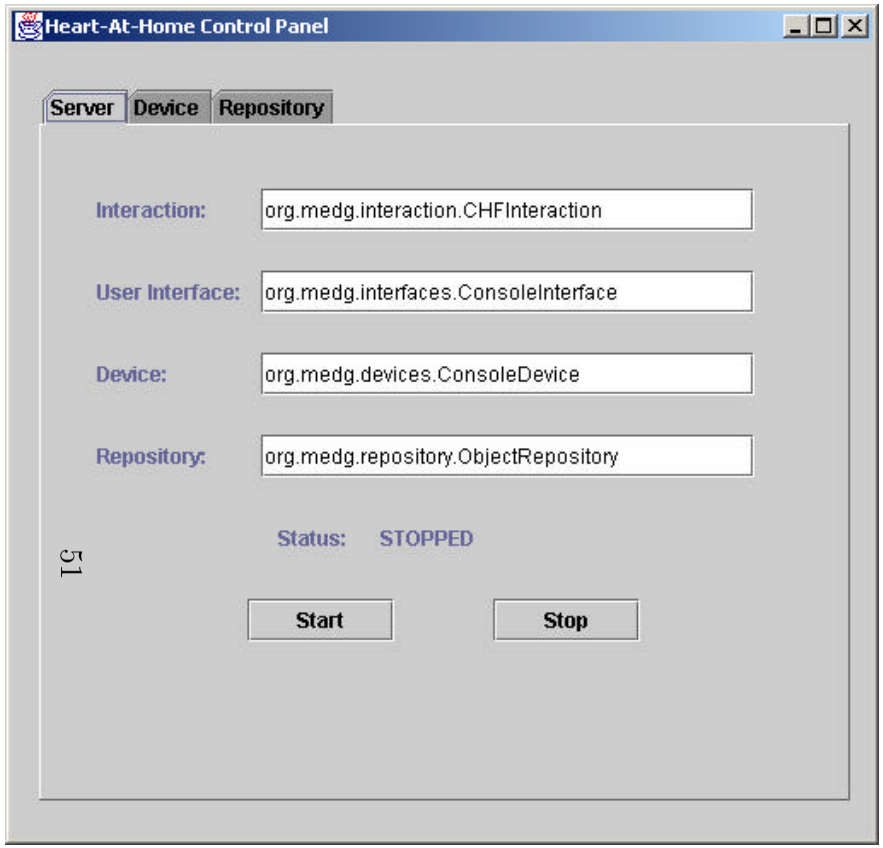
Control Panel

There is a graphical control panel in the org.medg.console package that can be useful for debugging the engine and seeing what interim data values are being acquired and used. It is also useful for browsing the patient repository to see if the system is properly recording values.

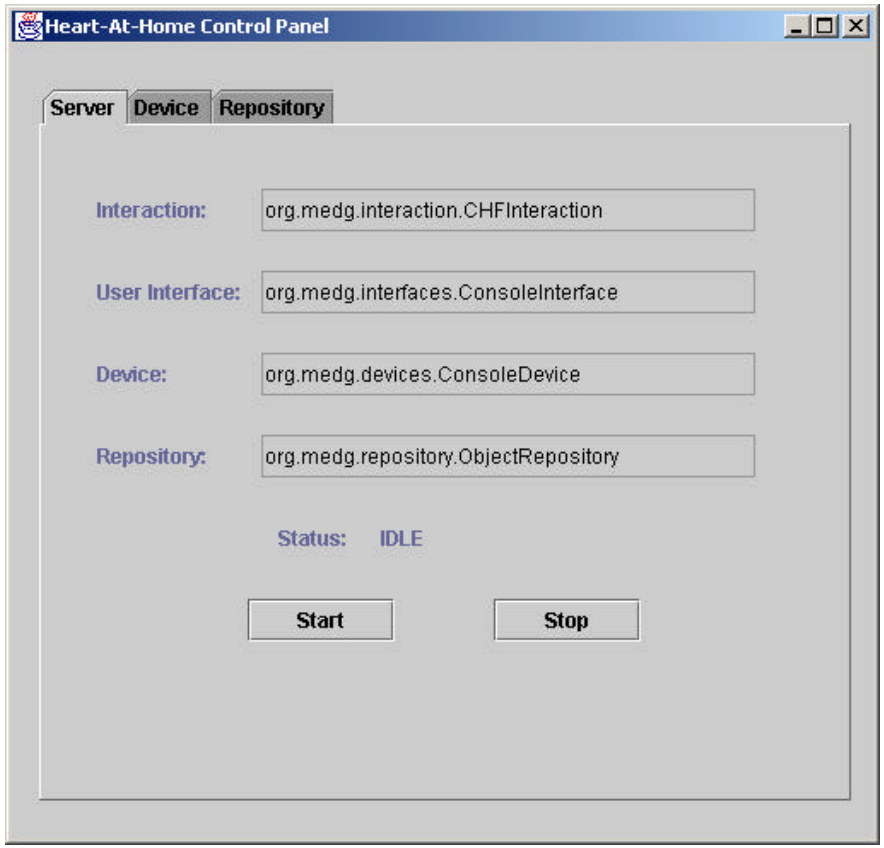
The command to run the control panel is

java org.medg.console.ControlPanel

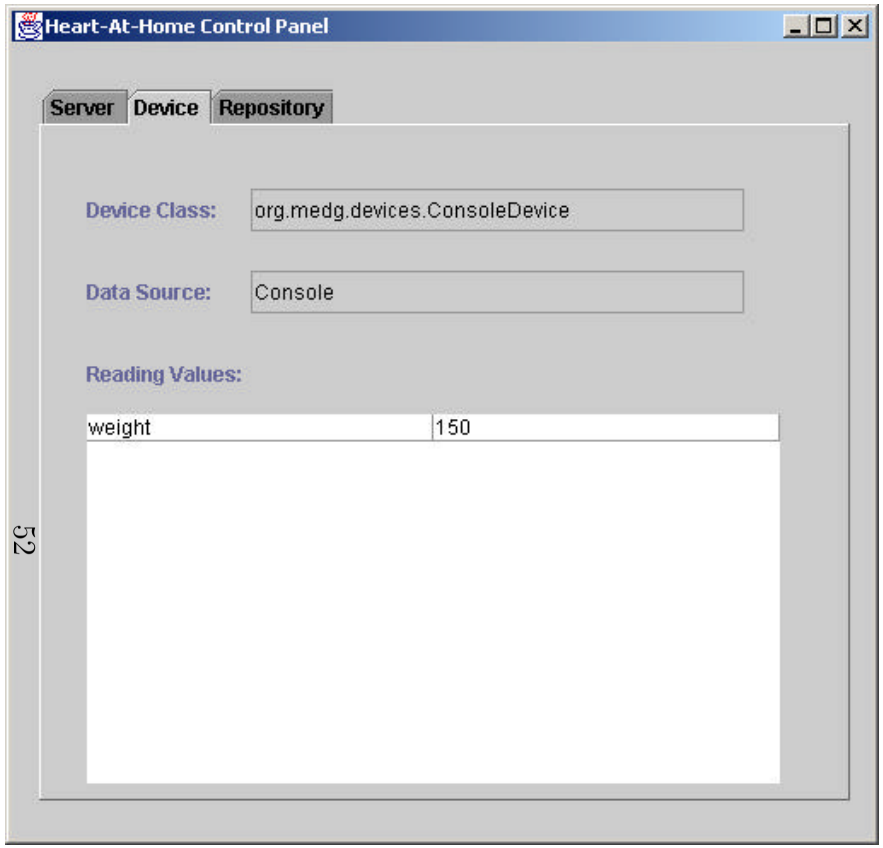
The screen that comes up should be the server status panel as can be seen in the Figure A-1. The Server can be started and stopped from this window as well as configured to use different components. The control panel also has informational tabs (see Figure A-2) for viewing current device and repository information.



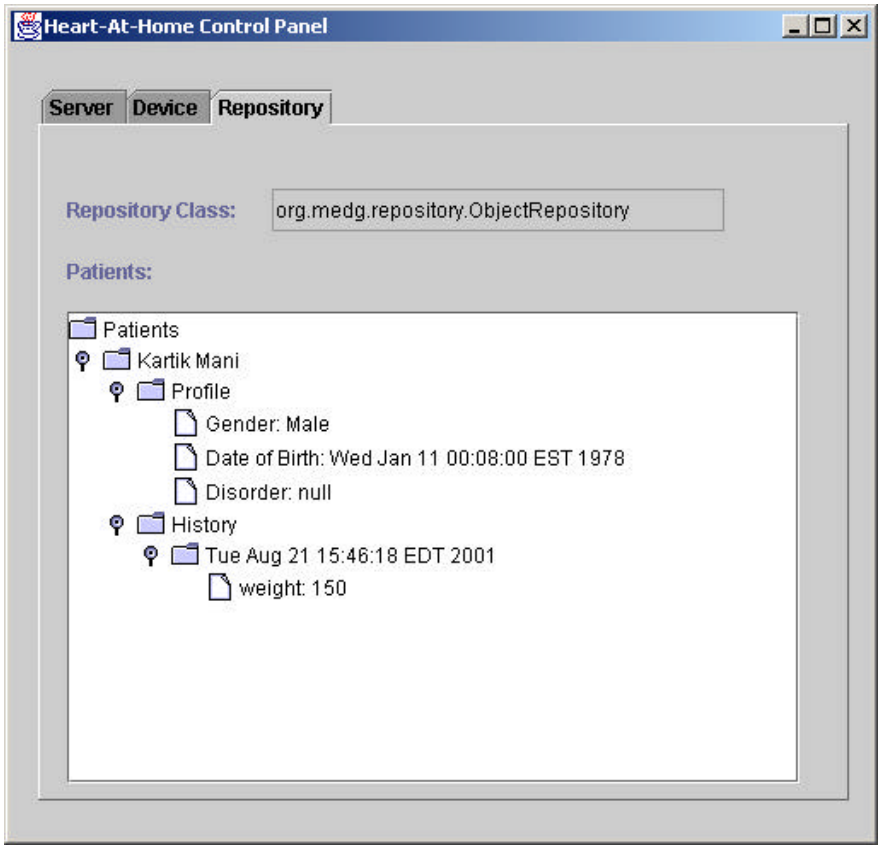
Heart-At-Home Engine STOPPED



Heart-At-Home Engine IDLE



Heart-At-Home Engine
Device Details



Heart-At-Home Patient
Repository Details

Table A.1: Heart-At-Home Supported DataSources

DataSource Class	Description
ConsoleSource	Supports I/O from the System Console
SocketSource	Acts as a TCP Server for I/O from network sockets
SerialPortSource	Supports I/O over serial communication ports

A.3 Extending Heart-At-Home

This section details the extension capabilities of Heart-At-Home, and is a programming guide for new devices, interfaces and interactions.

A.3.1 Utility Classes

The Heart-At-Home includes some useful utility classes for use by those programming extensions. One of these is the XMLUtil class in the org.medg.util package which includes several static methods for manipulating XML documents. The org.medg.util.Logger class also provides a convenient API for logging informational data, debug information and errors within the platform.

A.3.2 DataSources

The DataSource interface in the engine is an abstraction for a data communication channel. DataSources are used by Devices for retrieving and sending data.

Supported DataSources

Heart-At-Home has support for basic DataSources shown in Table A.1

DataSource Conventions

All DataSource objects should follow the naming convention of the type of source followed by word "Source". They should also be placed in the org.medg.sources package.

Adding New DataSources

The DataSource interface is shown below. A new class needs to implement this interface in order to work with Heart-At-Home.

```
public void open();
public boolean isOpen();
public void waitForData();
```

public InputStream getInputStream();
public OutputStream getOutputStream();
public void reset();
public abstract void close();

The function *waitForData()* should block the calling thread until data is available. The *reset()* function changes the state of the object back to what it was when the call to *open()* was made. Full source code and comments can be seen in Appendix B.

A.3.3 Adding Devices

In order to add a new device, a new class must be written that extends the base Device class of the engine.

Package Location

New devices are generally kept in the org.medg.devices package in the source tree, although this is not required.

Implementation

Extending the Device class requires implementation of three abstract functions.

public String getDataSourceType() {}
public int getDefaultBufferSize() {}
public void parseInput(byte[] inputData) {}

The function *getDataSourceType()* returns the name of the device's associated *DataSource* object as a String (Console, Socket, etc.). The engine will use this to generate a new *DataSource* object by introspection.

The function *getDefaultBufferSize()* returns the number of bytes to read when data becomes available. If this value is less than zero, the engine will continue to read until the input stream is done. Otherwise it will read exactly the specified number of bytes.

The function *parseInput(byte[])* reads in the byte array and parses it into usable name value pairs. New values can be added to the Device's internal variable map by a call to

super.addValue(String,Object).

While parsing, the values read should be converted to typed Objects (Integer, Date, etc.) before storing them in the Device's map.

A.3.4 Adding User Interfaces

The Interface UserInterface is shown below and only has one function to implement.

public void promptUser(Variable variable)

Implementing classes are responsible for prompting the user for the specific variable and setting the value when the user responds. The type of the variable can be obtained by a call to *variable.getType()*. All type constants are static members of the Variable class (Variable.INTEGER, Variable.CHOICE, etc.). Since Variable values are typed, there are two possibilities for setting the value. The first is a call to

variable.setValue(Object)

which checks that the Object passed in is of the correct type (if the implementing class knows enough to set the typed value directly). Otherwise, a call to

variable.assignValue(String)

can be made which takes in a String representation of the value and uses the variable parsing utility to set the correct typed value.

A.3.5 Programming New Interactions

To create a new interaction, a new class must be created that extends the Interaction class of the engine. Three abstract functions must be implemented.

public String getPromptsFile();
public String getName();
public void changeState();

Defining States

All states in a new interaction should be declared as integer constants at the beginning of the class. The following is an excerpt from the CHFInteraction class.

public static final int AUTHENTICATE = 2;
public static final int QUERY = 3;
public static final int NEWUSER = 4;
...

The values, -1, 0 and 1 are reserved for the ERROR state, the DONE state and the START state respectively. Each state should have an associated private member function with no arguments.

private void authenticate()
private void query()
...

In a particular function, the next state can be set with the following line.
super.currentState = (NEW STATE VALUE);

The changeState() Function

States are declared at the beginning as integers so that a switch statement can be used to call the correct function. Observe the example code from the CHFInteraction class.

```
public void changeState() {
    switch(this.currentState) {
        case Interaction.START :
            this.authenticate();
            break;
        case CHFInteraction.AUTHENTICATE :
            this.authenticate();
            break;
        case CHFInteraction.QUERY :
            this.query();
            break;
        case CHFInteraction.NEWUSER :
            this.newUser();
            break;
...
```

Accessing Variables

The Interaction code has no access to the UserInterface, only to the variables declared in the configuration file. A Variable object can be retrieved by a call to

Variable myVar = super.getVariable(String name);

This function retrieves the variable object, not the value. The variable object is accessible to allow the programmer to put dynamic information into the prompts or other variable properties. In order to retrieve the value of the variable, one of three functions can be used.

super.getValue(String name);
super.getValue(String name,Object[] fillins); (see section A.3.5
super.getValue(Variable variable);

Configuration File

The CHFInteraction configuration file with all variable declarations can be seen in Section C.2.

The interaction configuration file is an XML file with four main sub-nodes. XML was used in order to take advantage of features such as schema validation and native tree structure.

The root node is the interaction node which contains name and locale (a combination of country and language separated by an '_' character, i.e en_us) attributes. The node has four sub-nodes: profile, session, transient, and constant. Each of these four sub-nodes contain an unlimited number of variable child nodes. Only profile and session variables are stored in the patient's record, and constant variables are not reset with the interaction until the engine is restarted and the file is reloaded. Transient variables are used only for intermediate computation.

<interaction name="CHFInteraction" locale="EN_US">
    <profile>
        (var nodes)
    </profle>
    <session>
        (var nodes)
    </session>
    <transient>
        (var nodes)
    </transient>
    <constant>
        (var nodes)
    </constant>
</interaction>

The variables are specified as nodes that correspond to the 6 basic variable data types (boolean, integer, decimal, datetime, string, choice). The node name is the data type and the attributes are its name, either a prompt or a value, and other type-specific attributes (such as the options attribute for choice types). An example is shown below.

<boolean name="dressed" prompt="Are you dressed the same?"/>

With choice types, the options attribute is a comma-delimited string of different options for the variable value.

The prompt attribute can be configured to accept dynamic inputs during interaction runtime. This is accomplished by placing array indexes in {} characters in the prompt value. When the call to get the variable's value is made, it will pass in an array of Objects to use to fill in the prompt array references. For example, if the prompt value were

Your weight has decreased by {0} since my reading from {1}.

the call could be made as follows.

super.getValue("myVar",{new Integer(5),reading.getDate()})

which would output

Your weight has decreased 5 pounds since
my reading from 08/27/2001 8:35:00 AM.

This functionality makes use of the java.text package which is designed to provide internationalization. Dates, numbers and currencies can also be formatted to be consistent with the local setting. Additionally, the user can specify a format attribute for the variable output (i.e MM/dd/YYYY for datetimes).

Appendix B

Modular Dependency Diagrams

B.1 Heart-At-Home Engine API

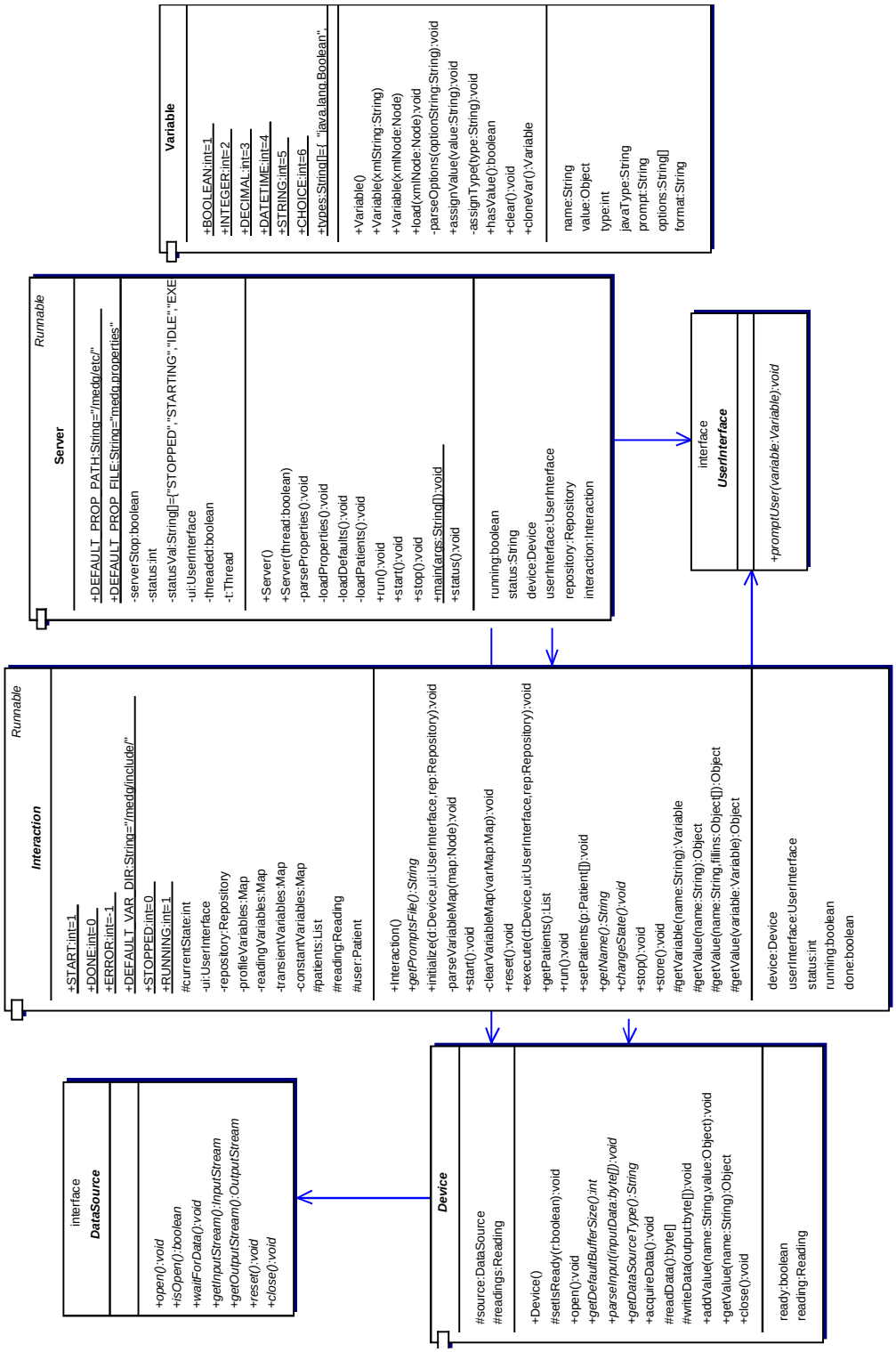


Figure B-1: Heart-At-Home Engine Package - Modular Dependency Diagram

B.2 Heart-At-Home Patient API

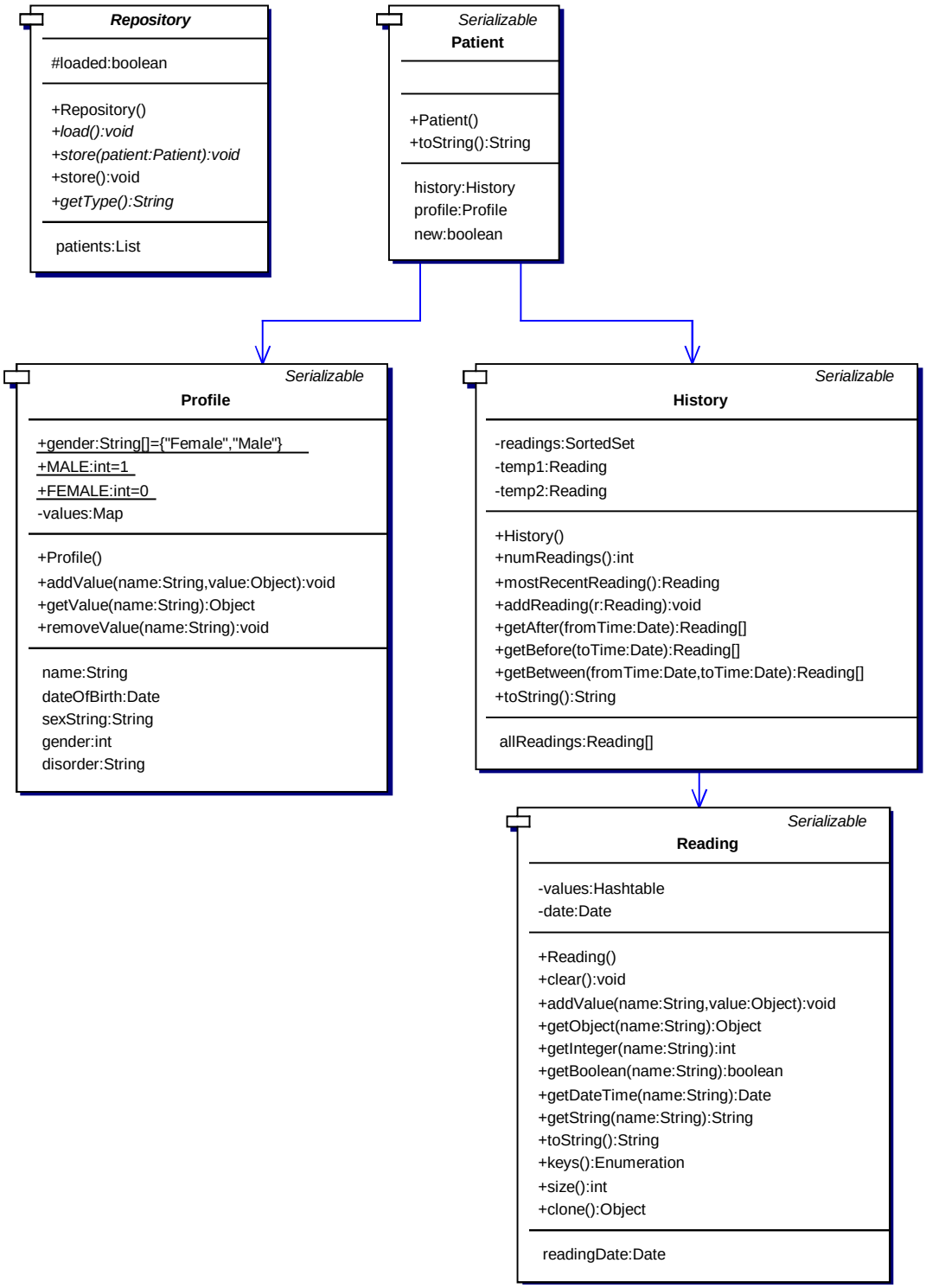


Figure B-2: Heart-At-Home Patient Package - Modular Dependency Diagram

Appendix C

Congestive Heart Failure - Interaction Files

C.1 CHFInteraction Source Code

```
package org.medg.interaction;

import org.medg.engine.*;
import org.medg.patient.*;
import org.medg.util.*;
import java.util.*;

/**
 * The CHFInteraction class provides the logic for handling a patient with
 * Congestive Heart Failure.  This interaction is based primarily on the
 * weight reading of the patient.
 *
 * author Kartik Mani
 * version 1.5
 */
public class CHFInteraction extends Interaction
{
    //state constants
    public static final int AUTHENTICATE    = 2;
    public static final int QUERY           = 3;
    public static final int NEWUSER         = 4;
    public static final int RECORD          = 5;
    public static final int ANALYZE         = 6;
    public static final int INCREASE        = 7;
    public static final int DECREASE        = 8;

    //utility constants
    public static final int DEFAULT_THRESHOLD = 10;

    Patient[] matches;
    Patient[] mismatches;
    double weight;
    double diff;
    int threshold;

    public CHFInteraction() {
        super();
        String strThreshold = System.getProperty("CHFInteraction.Threshold");
        if (strThreshold != null) {
            threshold = CHFInteraction.DEFAULT_THRESHOLD;
        }
        else {
            try {
                threshold = Integer.parseInt(strThreshold);
            }
            catch (NumberFormatException e) {
                Logger.log(e);
                threshold = CHFInteraction.DEFAULT_THRESHOLD;
            }
        }
    }
```

```
/**********************************************************************
STATES AND STATE TRANSITION LOGIC
**********************************************************************/
public void changeState()
{
    switch(this.currentState) {
        case Interaction.START :
            this.authenticate();                                          60
            break;
        case CHFInteraction.AUTHENTICATE :
            this.authenticate();
            break;
        case CHFInteraction.QUERY :
            this.query();
            break;
        case CHFInteraction.NEWUSER :
            this.newUser();
            break;                                                        70
        case CHFInteraction.ANALYZE :
            this.analyze();
            break;
        case CHFInteraction.INCREASE :
            this.increase();
            break;
        case CHFInteraction.DECREASE :
            this.decrease();
            break;
        case CHFInteraction.RECORD :                                      80
            this.record();
            break;
    }
}

/**
 * Authentication for CHF patients is performed by comparing the weight
 * reading to the most recent reading of all the patients, and picking
 * the best guesses.
 */                                                                       90
public void authenticate()
{
    weight = this.reading.getDecimal("weight");
    Patient currPatient;

    this.matchPatient(weight);

    if (matches.length == 0) {
        this.currentState = CHFInteraction.NEWUSER;
    }                                                                     100
    else {
        this.currentState = CHFInteraction.QUERY;
    }
}
```

```
/**
 * This interaction queries the patient to confirm if the guess at
 * at the user's identity was correct
 */
public void query() {
    boolean correct = false;

    if (matches.length == 1) {
        String name = matches[0].getProfile().getName();
        String[] fillins = {name};

        correct = ((Boolean)this.getValue("single-identity",fillins)).booleanValue();
        this.user = matches[0];
    }
    else {
        Variable variable = this.getVariable("mult-identity");
        String[] choices = new String[matches.length + 1];

        //form a list of choices for the user
        for (int i=0;i<matches.length;i++) {
            choices[i] = matches[i].getProfile().getName();
        }

        //add the option to not be on the list
        choices[matches.length] = (String)this.getValue("wrong");
        variable.setOptions(choices);

        int index = ((Integer)this.getValue(variable)).intValue();

        if (index < matches.length) {
            correct = true;
            this.user = matches[index];
        }
    }
    if (correct) {
        this.currentState = CHFInteraction.ANALYZE;
    }
    else {
        this.currentState = CHFInteraction.NEWUSER;
    }
}
/**
 * This function is called when a new user is standing on the scale
 */
public void newUser() {
    this.user = new Patient();
    this.getValue("name");
    this.getValue("birthdate");
    this.getValue("gender");
    this.getValue("disorder");

    this.currentState = CHFInteraction.RECORD;
}
```

```
/**
 * This function records the readings into the patient's record if
 * they confirm this was a valid reading.
 */
public void record() {
    boolean record = ((Boolean)this.getValue("record")).booleanValue();

    if (record) {
        this.store();
    }

    this.currentState = Interaction.DONE;
}

/**
 * This function determines whether the weight has increased or decreased
 */
public void analyze() {
    diff = weight - this.getRecentWeight(user);
    String change;

    if (Math.abs(diff) < 2) {
        this.currentState = CHFInteraction.RECORD;
        return;
    }

    if (diff > 0) {
        change = "higher";
        this.currentState = CHFInteraction.INCREASE;

    }
    else {
        change = "lower";
        this.currentState = CHFInteraction.DECREASE;
    }

    Date lastDate = user.getHistory().mostRecentReading().getReadingDate();
    Object[] fillins = {new Integer(Math.abs((int)diff)),change,lastDate};
    this.getValue("dressed",fillins);
}

/**
 * This function gathers information relevant to a weight increase.
 */
public void increase() {
    if (diff >= 5) {
        this.getValue("drinkwater");
    }
    else {
        this.getValue("eatsalty");
    }
    this.getValue("diuretic");
    this.getValue("eatmore");
    this.currentState = CHFInteraction.RECORD;
```

```
}

/**
 * This function gathers information relevant to a weight decrease.
 */
public void decrease() {
    if (Math.abs(diff) >= 5) {                                          220
        this.getValue("diuresis");
        this.getValue("sweating");
    }
    else {
        this.getValue("skipmeal");
        this.getValue("feelsick");
    }

    this.currentState = CHFInteraction.RECORD;
}                                                                       230

/**********************************************************************
UTILITY FUNCTIONS
**********************************************************************/
public String getName() {
    return "CHFInteraction";
}

private double getRecentWeight(Patient p) {
    if (p != null) {                                                    240
        Reading r  = p.getHistory().mostRecentReading();
        if (r != null) {
            return r.getDecimal("weight");
        }
    }

    return 0;
}

private void matchPatient(double currWeight) {                          250
    double tempWeight,diff;
    Patient tempPatient;
    Vector matchesList = new Vector();
    Vector mismatchesList = new Vector();

    for (int i=0;i<patients.size();i++) {
        tempPatient = (Patient)patients.get(i);
        tempWeight = this.getRecentWeight(tempPatient);

        //construct a listing of possible matches                       260
        diff = Math.abs(tempWeight - currWeight);

        //check if the weight is within a threshold for a match
        if (diff <= threshold) {
            matchesList.add(tempPatient);
        }
        else {
```

```
                mismatchesList.add(tempPatient);
            }
        }

        matches = new Patient[matchesList.size()];
        mismatches = new Patient[mismatchesList.size()];

        matchesList.toArray(matches);
        mismatchesList.toArray(mismatches);
    }


}
```

C.2 XML Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<interaction xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
             xsi:noNamespaceSchemaLocation="/medg/include/interaction.xsd"
             name="CHFInteraction" locale="EN_US">
    <profile>
        <string name="name" prompt="Hello new user. What is your name?"/>
        <datetime name="birthdate" prompt="What is your birthday?"
                  format="mm/DD/yyyy"/>
        <choice name="gender" prompt="What is your gender?" options="Male,Female"/>
        <boolean name="CHF" prompt="Do you suffer from Congestive Heart Failure?"/>
    </profile>
    <session>
        <boolean name="dressed" prompt="Are you dressed the same?"/>
        <boolean name="eatmore" prompt="Did you eat more than normal?"/>
        <boolean name="eatsalty" prompt="Did you eat salty food?"/>
        <boolean name="water" prompt="Are you drinking a lot of water?"/>
        <boolean name="diuretic" prompt="Did you take your water pill?"/>
        <boolean name="skipmeal" prompt="Did you skip a meal?"/>
        <boolean name="feelsick" prompt="Did you feel sick?"/>
        <boolean name="diuresis" prompt="Did you pee off a lot?"/>
        <boolean name="sweating" prompt="Are you sweating a lot?"/>
    </session>
    <transient>
        <boolean name="record" prompt="Shall I record this weight?"/>
        <choice name="mult-identity" prompt="Are you one of the following people?"/>
        <boolean name="single-identity" prompt="Is this 0?"/>
    </transient>
    <constant>
        <string name="wrong" prompt="" value="No, I'm not"/>
    </constant>
</interaction>
```

C.3 Interaction Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
            elementFormDefault="qualified">
    <xsd:element name="interaction">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="profile"/>
                <xsd:element ref="session"/>
                <xsd:element ref="transient"/>
                <xsd:element ref="constant"/>
            </xsd:sequence>
            <xsd:attribute name="name" type="xsd:string" use="required"/>
            <xsd:attribute name="locale" type="xsd:string" use="required"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="profile">
        <xsd:complexType>
            <xsd:choice maxOccurs="unbounded">
                <xsd:element ref="boolean"/>
                <xsd:element ref="datetime"/>
                <xsd:element ref="integer"/>
                <xsd:element ref="decimal"/>
                <xsd:element ref="string"/>
                <xsd:element ref="choice"/>
            </xsd:choice>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="session">
        <xsd:complexType>
            <xsd:choice maxOccurs="unbounded">
                <xsd:element ref="boolean"/>
                <xsd:element ref="datetime"/>
                <xsd:element ref="integer"/>
                <xsd:element ref="decimal"/>
                <xsd:element ref="string"/>
                <xsd:element ref="choice"/>
            </xsd:choice>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="transient">
        <xsd:complexType>
            <xsd:choice maxOccurs="unbounded">
                <xsd:element ref="boolean"/>
                <xsd:element ref="datetime"/>
                <xsd:element ref="integer"/>
                <xsd:element ref="decimal"/>
```

```
                <xsd:element ref="string"/>
                <xsd:element ref="choice"/>
            </xsd:choice>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="constant">
        <xsd:complexType>
            <xsd:choice maxOccurs="unbounded">
                <xsd:element ref="boolean"/>
                <xsd:element ref="datetime"/>
                <xsd:element ref="integer"/>
                <xsd:element ref="decimal"/>
                <xsd:element ref="string"/>
                <xsd:element ref="choice"/>
            </xsd:choice>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="choice">
        <xsd:complexType>
            <xsd:attribute name="name" type="xsd:string" use="required"/>
            <xsd:attribute name="prompt" type="xsd:string" use="required"/>
            <xsd:attribute name="options" type="xsd:string" use="optional"/>
            <xsd:attribute name="value" type="xsd:boolean" use="optional"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="boolean">
        <xsd:complexType>
            <xsd:attribute name="name" type="xsd:string" use="required"/>
            <xsd:attribute name="prompt" type="xsd:string" use="required"/>
            <xsd:attribute name="value" type="xsd:boolean" use="optional"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="integer">
        <xsd:complexType>
            <xsd:attribute name="name" type="xsd:string" use="required"/>
            <xsd:attribute name="prompt" type="xsd:string" use="required"/>
            <xsd:attribute name="value" type="xsd:integer" use="optional"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="decimal">
        <xsd:complexType>
            <xsd:attribute name="name" type="xsd:string" use="required"/>
            <xsd:attribute name="prompt" type="xsd:string" use="required"/>
            <xsd:attribute name="value" type="xsd:decimal" use="optional"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="datetime">
```

```
        <xsd:complexType>
            <xsd:attribute name="name" type="xsd:string" use="required"/>
            <xsd:attribute name="prompt" type="xsd:string" use="required"/>
            <xsd:attribute name="format" type="xsd:string" use="required"/>
            <xsd:attribute name="value" type="xsd:date" use="optional"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="string">
        <xsd:complexType>
            <xsd:attribute name="name" type="xsd:string" use="required"/>
            <xsd:attribute name="prompt" type="xsd:string" use="required"/>
            <xsd:attribute name="value" type="xsd:boolean" use="optional"/>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

References

[1] Philip A. Ades, Fredric J. Pashkow, Ileana L. Pina, Lenore R. Zohman, and James R. Nestor. A Controlled Trial of Cardiac Rehabilitation in the Home Setting Using Electrocardiographic and Voice Transtelephonic Monitoring. *American Heart Journal*, 139(3):543–548, 2000.

[2] Marie Elena Cordisco, Ainat Beniaminovitz, Kim Hammond, and Donna Mancini. Use of Telemonitoring to Decrease the Rate of Hospitalization in Patients With Severe Congestive Heart Failure. *American Journal of Cardiology*, 84(7):860–862, 1999.

[3] Gregg C. Fonarow, Lynne W. Stevenson, Julie A. Walden, Nancy A. Livingston, Anthony E. Steimle, Michele A. Hamilton, Jaime Moriguchi, Jan H. Tillisch, and Mary A. Woo. Impact of a Comprehensive Heart Failure Management Program on Hospital Readmission and Functional Status of Patients With Advanced Heart Failure. *Journal of the American College of Cardiology*, 30(3):725–32, 1997.

[4] Paul A. Heidenreich, Christine M. Ruggerio, and Barry M. Massie. Effect of a Home Monitoring System on Hospitalization and Resource Use for Patients with Heart Failure. *American Heart Journal*, 138(4):633–640, 1999.

[5] Long W. J., Fraser H., Naimi S., Perry K., Cotter L., and Konstam M. A. Developing a Program for Tracking Heart Failure. In *Proc AMIA Symposium*, page 1070, 2000.

[6] Friedman RH, Jette A, Smith MB, Stollerman J, Torgerson J, and Carey K. A telecommunications system for monitirng and counseling patients with hyper-

tension. impact on medication adherence and blood pressure control. *American Journal of Hypertension*, 9(4):285–292, 1996.

[7] Alberto Riva, Kenneth D. Mandl, Do Hoon Oh, Daniel J. Nigrin, Atul Butte, Peter Szolovits, and Isaac S. Kohane. The Personal Internetworked Notary and Guardian. *International Journal of Medical Informatics*, 62(1):27–40, 2001.

[8] Nihir B. Shah, Elaine Der, Chris Ruggerio, Paul A. Heidenreich, and Barry M. Massie. Prevention of Hospitalizations for Heart Failure with an Interactive Home Monitoring Program. *American Heart Journal*, 135(3):373–8, 1998.

[9] Jeffrey A. West, Nancy H. Miller, Kathleen M. Parker, Deborah Senneca, Ghassan Ghandour, Mia Clark, George Greenwald, Robert S. Heller, Michael B. Fowler, and Robert F. DeBusk. A Comprehensive Management System for Heart Failure Improves Clinical Outcomes and Reduces Medical Resource Utilization. *American Journal of Cardiology*, 79(1):58–63, 1997.