KitchIntel
Diagnostics

Stakeholder Goals

KitchIntel Diagnostics (KID) is being created to support technicians and maintenance people. It will also serve as a proof of concept for a complete system (to be developed later).

The goal of KID is to give technicians the ability to run diagnostics on a device using a standard WWW browser and to upload/download data about normal operating ranges using a command-line application.

Needs Statement

The KID product design team interviewed 8 technicians and maintenance people during the past month to arrive at this needs statement.

System Needs

S-1 Each appliance needs to run an HTTP server.

S-2 Each appliance needs to be able to perform diagnostics.

S-3 Each appliance needs to be able to report the results of the diagnostic tests in HTML.

S-4 Each appliance needs to be able to upload updates to its "normal operating ranges".

S-5 Each appliance needs to be able to download its current "normal operating ranges".

User Needs

A user of KID needs to be able to:

U-1 Download the results of a diagnostic using a standard WWW browser.

U-2 Upload "normal operating ranges" to an appliance using an application (with a command-line interface) that runs on a networked personal computer.

U-3 Download "normal operating ranges" from an appliance using an application (with a command-line interface) that runs on a networked personal computer.

Use Case Descriptions

The use case descriptions below involve the following actors: Diagnostician, UpdateClient, and WWWBrowser.

UC1. WWWBrowser Requests Diagnostics

Actors:
WWWBrowser

Preconditions:
The HttpServer is running.

Flow:

1. The WWWBrowser transmits an HTTP GET request for `run.diagnostics`
2. The HttpServer “starts” the Diagnostician.
3. The Diagnostician performs the diagnostic tests.
4. The HttpServer constructs an HTTP response that contains the results of the tests in HTML.
5. The HttpServer transmits the HTTP response.
6. The WWWBrowser displays the HTML.

UC2. UpdateClient Requests Normal Ranges:

Actors:
UpdateClient

Preconditions:
The HttpServer is running.

Flow:

1. The UpdateClient transmits an HTTP GET request for `normal.ranges`.
2. The HttpServer reads the file `normal.ranges`.
3. The HttpServer constructs an HTTP response that contains the file `normal.ranges`.
4. The HttpServer transmits the HTTP response.
5. The UpdateClient displays the information in the file `normal.ranges` on the console in human-readable form.

UC3. UpdateClient Uploads Updated Normal Ranges:

Actors:
UpdateClient

Preconditions:
The HttpServer is running.

Flow:

1. The UpdateClient reads the file `normal.ranges`.
2. The UpdateClient constructs a POST request that contains `normal.ranges`.

3. The UpdateClient transmits the POST request.
4. The HttpServer receives the POST request.
5. The HttpServer saves the information in a file named `normal.ranges` in the appropriate directory.
6. The HttpServer sends an appropriate response to the UpdateClient in HTML (e.g., `<html><body><p>normal.ranges has been updated.</p></body></html>`).
7. The UpdateClient displays the response on the console.

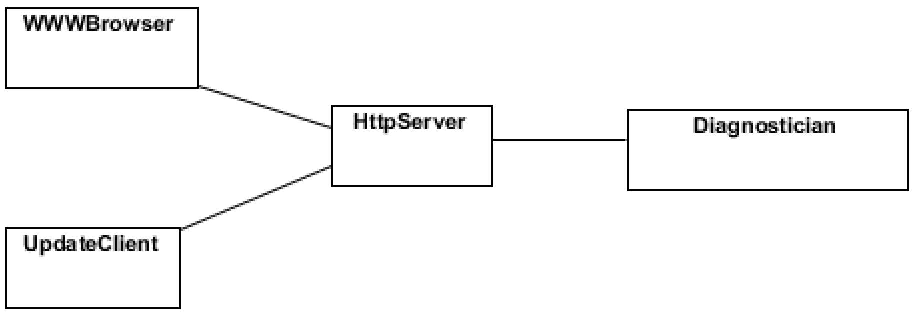
Requirements Specification

Abbreviations and Acronyms

HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol

Conceptual Model

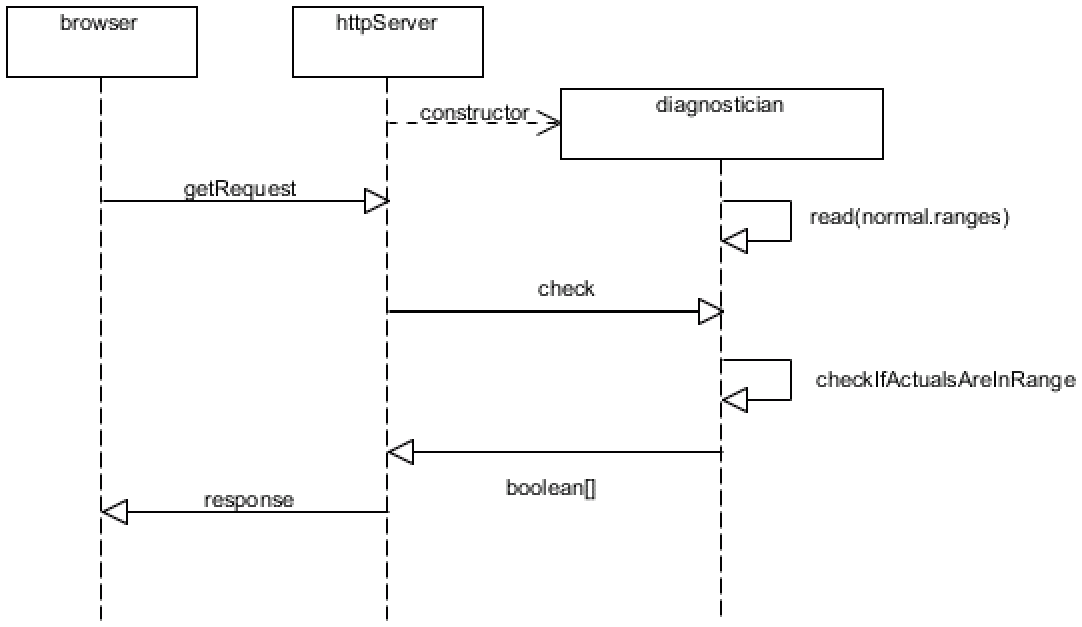
The following conceptual model was developed early in the design process:



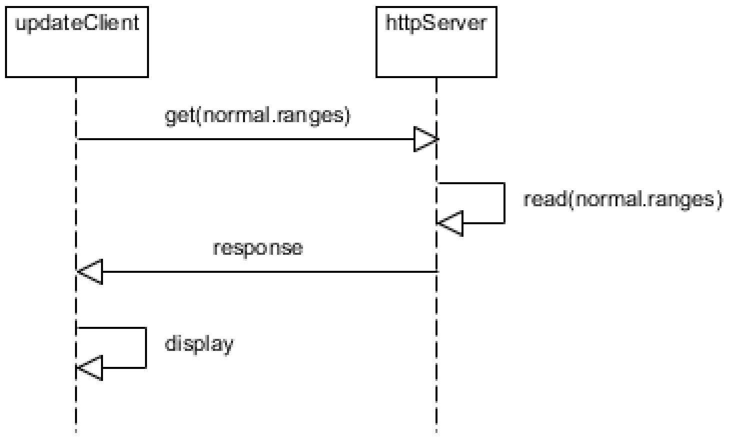
It illustrates, at a fairly high level, the important concepts in the system. It is not intended as an engineering design. That is, the software components that need to be designed and developed may or may not correspond to these concepts.

Sequence Diagrams

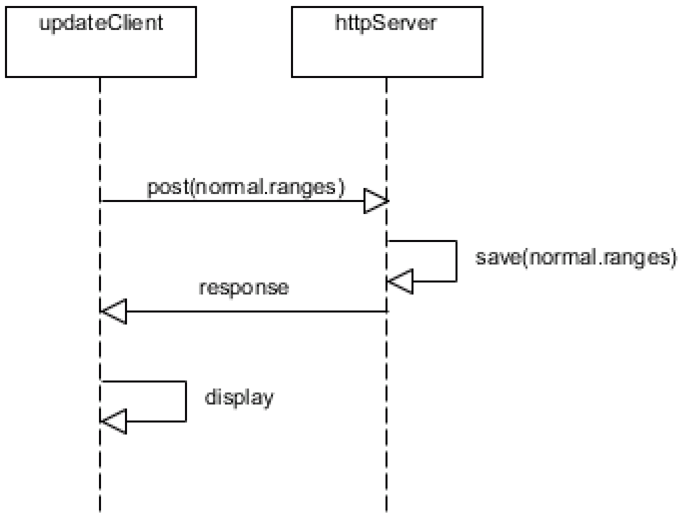
Based on the use-case descriptions, the product design team created three sequence diagrams that illustrate, using the concepts above, how the system interacts with the outside environment. The first illustrates a user instructing an appliance to perform a diagnostic:



The second illustrates a user downloading the normal operating ranges from an appliance:



The third illustrates a user uploading the normal operating ranges to an appliance:



Physical Requirements

The system must satisfy the following physical requirements:

PR-1 The `normal.ranges` file for all devices must contain ten pairs of integers.
PR-2 The `normal.ranges` file must not contain "text"; it must contain big-endian representations of the integers.
PR-3 The integers in the `normal.ranges` file must each be 4 bytes long. (Hence, in total, the `normal.ranges` file must contain 80 bytes.)
PR-4 Each pair in the `normal.ranges` file must correspond to a particular diagnostic test (i.e., pair 0 corresponds to test 0, pair 1 corresponds to test 1, etc...).
PR-5 Each pair in the `normal.ranges` file must contain two `int` values, the first of which is the lowest normal value for the test and the second of which is the highest normal value for the test.
PR-6 Each value in `normal.ranges` must be in the interval [0, 8191].

Operational Requirements

The system must satisfy the following operational requirements:

OR-1 Each appliance must have an HTTP server. (see Need S-1)
OR-2 Each appliance must be able to perform diagnostics. (see Need S-2)
OR-3 The HTTP server must be able to handle POST requests containing the "normal ranges". (see Needs U-2, S-4)
OR-4 The HTTP server must be able to handle GET requests. (see Needs U-1, U-3, S-5)
OR-5 The HTTP server must be able to start diagnostic routines as a result of some GET requests. (see Need U-1)
OR-6 The results of the diagnostic routines must be transmitted in HTML. (see Needs U-1, S-3)
OR-7 The HTTP server must be able to transmit the "normal operating ranges" in response to some GET requests. (see Needs U-3, S-5)

Engineering Design Specifications

Engineering design specifications have been created for the following classes.

The `Diagnostician` Class

Attributes

The `Diagnostician` class must have the following private attributes.

```
/**
 * The normal operating ranges.
 *
 * Currently, ranges.length will always be 10 (i.e., one for each
 * test) and ranges[i].length will always be 2 (i.e., for the low
 * and high values in the range).
 */
private int[][]    ranges;
```

It may have other private attributes as well.

Methods

The `Diagnostician` class must have the following public methods.

```
/**
 * Default Constructor
 *
 * This method reads the file normal.ranges into ranges[][].
 * If a line contains any errors (e.g., if either value is not in
 * the interval [0, 8191] or if the low is greater than the high)
 * then both values are set to 9999.
 */
public Diagnostician()
```

```
/**
 * Checks to see whether each of the 10 tests is in the normal range.
 *
 * For testing purposes, this method generates 10 random int values
 * in the interval [0, 8191] and checks to see if each is in the
 * normal range.
 *
 * result[i] is true if test i is in the normal range and is false
 * otherwise.
 */
public boolean[] check()
```

```
/**
 * Returns the ranges[][] array.
 */
public int[][] getRanges()
```

It may have other private methods but must not have any other public/protected/package methods.