|  |
| --- |
| St. Mary's University of San Antonio |
| An Expert System for Network Router Configuration |
| Software Design Descriptions |
|  |
| **Michael A. Perez** |
| **4/1/2013** |

|  |
| --- |
| This document defines the functional and non-functional requirements of this Master's project and is intended to be used by those who will implement and verify the correct functioning of the system. The project will implement an Expert System using a user-defined knowledge base and an inference engine to automate the configuration of a small home/small office computer network. |

[1.Introduction 2](#__RefHeading__2_626044097)

[Purpose 2](#__RefHeading__4_626044097)

[Overview 2](#__RefHeading__6_626044097)

[2.Data Design 2](#__RefHeading__10_626044097)

[2.1 Data Flow Diagram 2](#__RefHeading__12_626044097)

[2.2 Program Structure 3](#__RefHeading__14_626044097)

[3.System Architecture 3](#__RefHeading__16_626044097)

[3.1 Architecture Alternatives 3](#__RefHeading__18_626044097)

[4.Software Component Details 3](#__RefHeading__20_626044097)

[4.1 Wizard 3](#__RefHeading__22_626044097)

[Interface Description 3](#__RefHeading__24_626044097)

[Local Data Structures 4](#__RefHeading__26_626044097)

[Global Data Structures 4](#__RefHeading__28_626044097)

[Algorithmic Model / Pseudocode 4](#__RefHeading__30_626044097)

[4.2 Configurator 5](#__RefHeading__32_626044097)

[Narrative 5](#__RefHeading__34_626044097)

[Interface Description 5](#__RefHeading__36_626044097)

[Local Data Structures 5](#__RefHeading__38_626044097)

[Global Data Structures 5](#__RefHeading__40_626044097)

[Algorithmic Model / Pseudocode 5](#__RefHeading__42_626044097)

[4.3 File I/O 5](#__RefHeading__44_626044097)

[Narrative 5](#__RefHeading__46_626044097)

[Interface Description 5](#__RefHeading__48_626044097)

[Local Data Structures 6](#__RefHeading__50_626044097)

[Global Data Structures 6](#__RefHeading__52_626044097)

[Algorithmic Model / Pseudocode 6](#__RefHeading__54_626044097)

[4.4 Administrative Controls 6](#__RefHeading__56_626044097)

[Narrative 6](#__RefHeading__58_626044097)

[Interface Description 6](#__RefHeading__60_626044097)

[Local Data Structures 6](#__RefHeading__62_626044097)

[Global Data Structures 6](#__RefHeading__64_626044097)

[Algorithmic Model / Pseudocode 6](#__RefHeading__66_626044097)

[4.5 Text Editor 6](#__RefHeading__68_626044097)

[Narrative 6](#__RefHeading__70_626044097)

[Interface Description 6](#__RefHeading__72_626044097)

[Local Data Structures 6](#__RefHeading__74_626044097)

[Global Data Structures 7](#__RefHeading__76_626044097)

[Algorithmic Model / Pseudocode 7](#__RefHeading__78_626044097)

[4.6 CLIPS 7](#__RefHeading__80_626044097)

[Interface Description 7](#__RefHeading__82_626044097)

[Local Data Structures 7](#__RefHeading__84_626044097)

[Global Data Structures 7](#__RefHeading__86_626044097)

[Algorithmic Model / Pseudocode 7](#__RefHeading__88_626044097)

[5.Use Case Realizations 7](#__RefHeading__90_626044097)

[5.1 Interaction Diagrams 7](#__RefHeading__92_626044097)

[5.2 Class Diagrams 7](#__RefHeading__94_626044097)

[6.Glossary And Acronyms 7](#__RefHeading__96_626044097)

# Introduction

## Purpose

This document describes all software components required for successful project development. Specifications for these components are documented here with enough detail to be implemented by component developers. All software components defined here can be traced back to formal requirements in the SRS document. This document may overlap with the Analysis and Design phases and as such may contain UML diagrams from both. UML diagrams will be marked with their source phase.

## Overview

This document addresses the system's software architecture, that is, the identification and organization of all software components of the system. Because a system's software component listing consists of software modules and data entities manipulated by the system, data design will play a part in developing the architecture. In this document, a data flow diagram is developed from formal requirements in our SRS document and mapped onto a top-down architecture using transform and/or transaction mapping techniques as described in [PRESSMAN]. This generic structure can then be refined as a unique software architecture.

# Data Design

## 2.1 Data Flow Diagram

Data flow diagrams (DFD) model the flow of information between processes of the system providing one way to derive a basic system architecture. DFDs are refined into greater levels of detail using functional decomposition. Before any refinement, the first "Level 0" DFD, or more commonly the fundamental system model, is derived from the statement of scope in our Vision document. The pertinent part of the statement of scope is as follows:

*"To achieve the stated goal, we will apply the conversational-mode of an expert system to gather information about a SOHO network from a user. The Expert system will read in production rules from a knowledge base along with the user's responses and infer a new status message to display to the user, a new question to ask of the user, and /or a new UCI command to append to a growing configuration script. After the conversational information gathering is complete, the system will allow the user to review, edit, save, and apply the generated configuration file. The configuration script will be applied remotely via SSH."*

Figure 1: Functional System Model and first level of refinement

Here is a two panel graphic showing the first two levels of our Data Flow Diagram (DFD). The left panel shows the fundamental system model, or Level 0 DFD, where the whole system is contained in one process "bubble" and all external entities are in boxes. Expanding the system bubble into subprocesses into the next level of detail gives us the Level 1 DFD shown in the right. The system is now decomposed into two major processes: the Wizard and the Router Configuration.

The Wizard carries out the interview process with the user using a graphical user interface and a conversational-mode expert system. The technical reader might know an expert system as a system where a three-part process called Match-Resolve-Act is carried out by an inference engine. The inference engine accepts assertions from working memory and inactive production rules from a knowledge base as inputs. An inactive production rule is one which has not been triggered by the inference engine since the wizard process started. The rules are made up of two groups of patterns: the antecedent, or Left-hand side (LHS), and a consequent, or the Right-hand side (RHS). In the DFD below, for the sake of readability, we ignore the Router Configuration sub-process and focus on the functional decomposition of the Wizard sub-process. The expert system process asserts facts from the user's response to interview questions and scans the knowledge-base, or more specifically production rule base, for rules which the Left-Hand Side (LHS), or antecedent, match the asserted facts in working memory. The inference engine resolves which of the matching rules to run and runs them. The outputs are a new interview query or status message to display to the user and/or a new shell script command to append to the growing configuration file. Input and Output boundaries are indicated by the curved dotted lines. The transform controller, the Match-Resolve-Act process, is separated from both inputs and outputs.

Figure 2: Level 2 DFD of the Wizard sub-process

In the Level 3 DFD below, we expand on the transform controller, the Match-Resolve-Act (M-R-A) process. The M-R-A process runs through the following steps described by Asheeh Goja [6]: 1) The inference engine uses a pattern matching algorithm to create or update an **agenda** of rules whose antecedents are satisfied with the asserted facts in **working memory**. 2) A **conflict resolver** selects the rule with the highest priority from the agenda to trigger. 3) An **action executor** executes the consequent of the selected rule and also removes this rule from the agenda. 4) If there are rules leftover in the agenda, repeat from step 1.

Figure 3: Level 3 DFD of Wizard sub-process

This cycle continues until there are no rules left in the agenda. In our application of expert systems, after a rule is triggered it must never be matched or triggered again until the system is reset. This is an important feature of expert systems called [refraction](http://en.wikibooks.org/wiki/Expert_Systems/Refraction) which prevents rules from firing repeatedly in the Match-Resolve-Act cycle [7].

From here we can start building our program structure. The program structures we develop below consist of a hierarchy of the processing bubbles from our DFDs above. There are many Expert Systems already developed for us so we will not bother capturing a level of detail beyond the Level 2 DFD in our program structure for the Wizard process.

Figure 4: Program structure of the Wizard sub-process

Our program structures represent a top-down control hierarchy so when building our program structure we start with the input processes as bottom-left modules, work our way up to the main "bubble" and then back down the right side with output processes. Now we focus on the next level of refinement for the Router Configuration sub-process. It is simpler than our Wizard process and is self-explanatory. It takes one of three commands from the user to load, save, or apply a configuration file. To load or save a file, only requires a filename parameter. But to apply a configuration file using SSH requires the IP address of the targeted SOHO router (TSR) as well. Additionally, an SSH username and password will be needed if a private/public key pair has not been established between the system and the TSR. When saving or loading a file, the system should also display the status of the file operation to the user using the output display. The status presented should indicate if the operation is ongoing or complete. The same goes for the status of the SSH operation when applying the configuration. If an error occurs over SSH, the return error code or message should be displayed.

Figure 5: Level 2 DFD for Router configuration sub-process

## 2.2 Program Structure



Figure 6: ERC whole system program structure

We can further refine this program structure by logically grouping the processing nodes into areas of similar responsibility. We will call these logical groupings system components and detail them and the relationships between them in the following section.

Figure 7: Program structure grouped in areas of responsibility

# System Architecture

Figure 8: UML diagram showing system components and the interfaces between them

Taking the program structure from section 2 and grouping the processing nodes into unique areas of responsibility (see Figure 7) we have developed the above Software Architecture/Subsystem diagram using UML component and interface notations. Note that we have kept the same associations between the groupings as in Figure 7 except for the Text Editor component which was not present in the original graphic. These associations have been refined into interfaces by which the components communicate with each other. Components provide an interface when they implement software routines within their area of responsibility needed by other components. Components require an interface when they make need to make calls to software routines outside of their area of responsibility.

This architecture uses an Expert System and an SSH client for its main processing duties. An SSH Client can be used to manage SSH credentials such as a public/private key pair and to remotely execute configuration scripts on the targeted SOHO router (TSR). In this case, the TSR should be running OpenWRT and the dropbear SSH server. An Expert System can be used to interview the user regarding the SOHO network to be configured. CLIPS is a freely available public domain Expert System developed by NASA in the late 1980's and early 1990's. As of this writing, it is actively maintained by the public on sourceforge.net. Its basic user interface is command-line-based and its command syntax resembles the LISP programming language. A graphical user interface is available for CLIPS in Microsoft Windows and Apple Mac OS X. For details on each subsystem including the interfaces between them, refer to section 4 Software Component Details.

## 3.1 Architecture Alternatives

One obvious alternative to our chosen architecture is to fore-go the Expert System and use a collection of IF-THEN-ELSE or SWITCH-CASE statements. The advantage of the Expert System in this case is its inherit handling of the problem known as Combinatorial Explosion with pattern-matching algorithms such as Rete. Consider a system with n variables. Each variable has Z possible values. This system then has Z^n possible end-states. A system with n=4 variables and Z=2 possible outcomes per variable has 4^2=16 possible outcomes. Adding just one more variable increases the number of outcomes to 5^2=25. The high number of variables involved in developing a network configuration and the exponential nature of the Combinatorial Explosion problem means that adding one new rule to an already high number can be a major maintenance headache. So instead of trying to hard-code every possible combination, we rely on the pattern-matching ability of an Expert System and the knowledge representation syntax of its knowledge-base. On the other hand, expert systems are not meant to handle systems with too few variables since a lot of effort goes into knowledge engineering. This comes to the main disadvantage of Expert Systems, the quality of the knowledge gathered.

A feasible alternative might include a real-time, online human expert in place of the Expert System. However, consistent application of heuristics would have to be questionable, as would the trust-worthiness of the online expert.

# Software Component Details

## 4.1 Wizard

The Wizard component provides a simple, easy to use graphical user interface (GUI) to input computer network usage information. It should conform to good design heuristics as described by [8]: 1) Visibility of system status where clear instructions are presented and feedback for any action is featured prominently, 2) a Match between the system and the real world by using the vocabulary of the user in place of the technical vocabulary of the network configuration and the system, 3) Flexibility of use is satisfied by allowing the user to skip input prompts which he or she cannot answer, and 4) Aesthetic and minimalist design where each input prompt presents only information needed to respond to the input prompt.

### Interface Description

The CLIPS subsystem and our Wizard subsystem will communicate using an Interviewer interface. This interface will provide our Wizard subsystem with a means to gather information from the user in a conversational format and allow the CLIPS expert system to reason towards a network router configuration based on the collected data.

The Wizard component initiates system calls to a File Writer implementation to create a new configuration script file, append new commands as they are generated by the inference engine one by one, and possibly cancel the script file.

The Wizard software component provides an interface for a client process to pose a multiple-choice question to the user and get a response and to begin or end an interview transaction.



Figure 9: Wizard interfaces

### Local Data Structures

Structures passed between other components

### Global Data Structures

Structures passed between external entities

### Algorithmic Model / Pseudocode

import clips

if \_\_name\_\_ == "\_\_main\_\_":

env = clips.Environment()

clips.Load("erc.clp")

clips.Reset()

clips.DebugConfig.WatchAll()

clips.DebugConfig.DribbleOn("erc\_clips.log")

clips.SetExternalTraceback(True)

clips.RegisterPythonFunction(pose, 'pose')

#in CLIPS: (return (python-call pose "how are you?" "fine" "not so well" "I do not know"))

clips.RegisterPythonFunction(gain\_insight, 'gain-insight')

#in CLIPS: (return (python-call gain-insight "uci set httpd.@httpd[0].port=80"))

clips.RegisterPythonFunction(begin, 'begin')

#in CLIPS: (return (python-call begin))

clips.RegisterPythonFunction(end, 'end')

#in CLIPS: (return (python-call end))

#cycle of questioning begins here!

clips.Run()

clips.PrintFacts()

## 4.2 Configurator

Software structure

### Narrative

### Interface Description

The Script File Data component provides two interfaces. The File Writer and File Reader interfaces.



Figure 10: Configurator Interfaces

### Local Data Structures

Structures passed between other components

### Global Data Structures

Structures passed between external entities

### Algorithmic Model / Pseudocode

## 4.3 File I/O

Software or Data structure?

### Narrative

### Interface Description



Figure 11: File I/O interfaces

### Local Data Structures

Structures passed between other components

### Global Data Structures

Structures passed between external entities

### Algorithmic Model / Pseudocode

## 4.4 Administrative Controls

This is a software interface component.

### Narrative

### Interface Description

### Local Data Structures

Structures passed between other components

### Global Data Structures

Structures passed between external entities

### Algorithmic Model / Pseudocode

## 4.5 Text Editor

Software or Data structure?

### Narrative

### Interface Description

### Local Data Structures

Structures passed between other components

### Global Data Structures

Structures passed between external entities

### Algorithmic Model / Pseudocode

## 4.6 CLIPS

The CLIPS subsystem is a conversational-mode Expert System including the knowledge-base written . Expanding on the aesthetic and minimalist design heuristic first discussed in section 4.1 Wizard, the Wizard interview questions provided by the knowledge base do not force users to come up with an answer. Instead the knowledge base gives the user an out by including an "I do not know", "Skip this question", or similar response choice in the multiple choice questions sent along with the question itself to the Wizard subsystem.

### Interface Description

The CLIPS subsystem requires an Interviewer interface as described in section 4.1.

### Local Data Structures

Structures passed between other components

### Global Data Structures

Structures passed between external entities

### Algorithmic Model / Pseudocode

# Use Case Realizations

## 5.1 Interaction Diagrams

## 5.2 Class Diagrams

# Glossary And Acronyms

* AS - see Autonomous System
* Autonomous System - Originally, the definition required control by a single entity, typically an Internet service provider or a very large organization with independent connections to multiple networks, that adhere to a single and clearly defined routing policy, as originally defined in RFC 1771.The newer definition in RFC 1930 came into use because multiple organizations can run BGP using private AS numbers to an ISP that connects all those organizations to the Internet. Even though there are multiple Autonomous Systems supported by the ISP, the Internet only sees the routing policy of the ISP. That ISP must have an officially registered Autonomous System Number (ASN).
* BGP or BGP4 - Border Gateway Protocol (version 4 is the current version)
* Border Gateway Protocol - Core routing protocol of the Internet. It maintains a table of IP networks or 'prefixes' which designate network reachability among autonomous systems (AS). It is described as a path vector protocol. BGP does not use traditional Interior Gateway Protocol (IGP) metrics, but makes routing decisions based on path, network policies and/or rulesets.
* DHCP - Dynamic Host Control Protocol
* Gateway -
* IP - Internet Protocol (version 4 is common, version 6 is reluctantly being adopted)
* ISP - Internet Service Provider
* LAN - Local Area Network
* Local Area Network - The defining characteristics of LANs, in contrast to wide-area networks (WANs), include their usually higher data-transfer rates, smaller geographic place, and lack of a need for leased telecommunication lines. ARCNET, Token Ring, and many other technologies have been used in the past, and G.hn may be used in the future, but Ethernet over twisted pair cabling, and Wi-Fi are the two most common technologies currently in use.
* Router -
* Shannon Entropy - A quantification of the information contained in a message usually in units such as bits. Equivalently, the Shannon entropy is a measure of the average information content one is missing when one does not know the value of the random variable. Shannon's entropy represents an absolute limit on the best possible lossless compression of any communication, under certain constraints: treating messages to be encoded as a sequence of independent and identically-distributed random variables, Shannon's source coding theorem shows that, in the limit, the average length of the shortest possible representation to encode the messages in a given alphabet is their entropy divided by the logarithm of the number of symbols in the target alphabet.
* SOHO - Small Office/Small Home. Consumer-grade. Not enterprise level.
* TCP - Transmission Control Protocol
* TCP/IP - Suite of protocols used in computer IP networks
* VPN - Virtual Private Network
* WAN - Wide Area Network
* Wide Area Network - WANs, in contrast with personal area networks (PANs), local area networks(LANs),campus area networks(CANs), or metropolitan area networks (MANs) are not limited to a room, building, campus or specific metropolitan area (e.g., a city) respectively. The largest and most well-known example of a WAN is the Internet. WANs are used to connect LANs and other types of networks together, so that users and computers in one location can communicate with users and computers in other locations. Many WANs are built for one particular organization and are private. Others, built by Internet service providers, provide connections from an organization's LAN to the Internet. WANs are often built using leased lines. At each end of the leased line, a router connects to the LAN on one side and a hub within the WAN on the other.