ST. MARY'S UNIVERSITY OF SAN ANTONIO

An Expert System for Network Router Configuration

Software Design Descriptions

Michael A. Perez

Spring 2015

Table of Contents

1. Introduction 3
1.1 Purpose 3
1.2 Design Overview 3
2. System Overview 3
3. System Architecture 3
3.1 Architectural Design 3
3.2 Decomposition Description 4
3.3 Design Rationale 8
3.3.1 knowledge base design 8
4. Data Design 9
4.1 Data Description 9
4.1.1 Wizard Activity 9
4.1.2 Configuration Subsystem 10

1. Introduction

1.1 Purpose

This document's audience is is intended to be those who will implement and verify the correct functioning of the system. Descriptions of all software components required for successful project development and 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.

1.2 Design Overview

This document addresses the system's software architecture, that is, the identification and organization of all software components of the system. An Object-oriented (OO) approach is taken in this document.

2. System Overview

The project will implement an Expert System using a user-defined knowledge base and an inference engine to produce the configuration of a small home/small office computer network. The system will aditionally apply the configuration script against the router running OpenWRT 10.03.1.

3. System Architecture

3.1 Architectural Design

The Software Architecture/Subsystem diagram is presented here. Subsystems provide an interface when they implement software routines within their area of responsibility needed by other components. And they require an interface when they need to make calls to software routines outside of their area of responsibility.

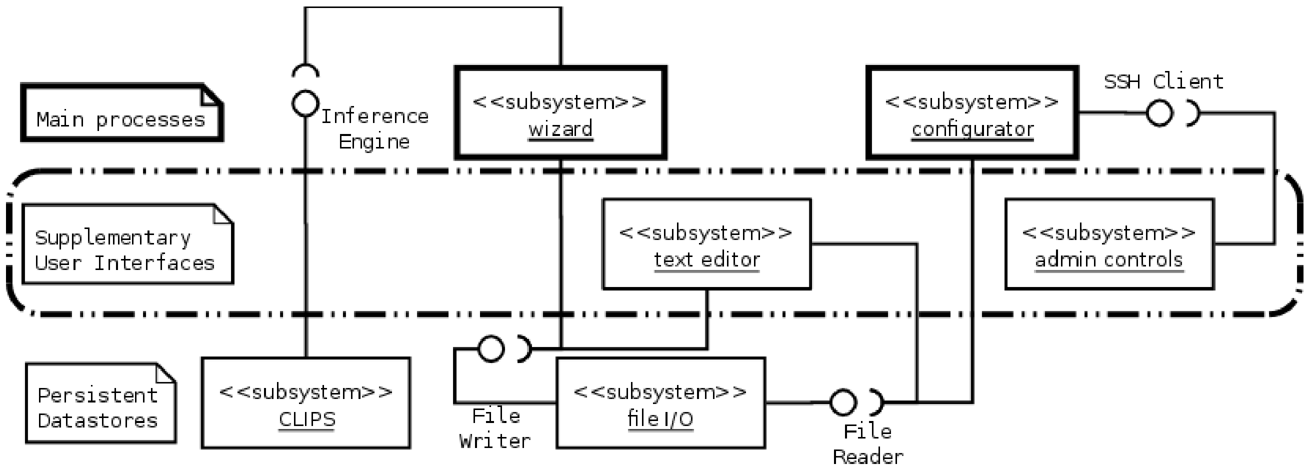

Figure 1: System Architecture

From the system architecture we see that there are three categories of subsystems: Main Processes, Supplementary User Interfaces, and Persistent data stores. A full component diagram is presented after we decompose each subsystem into classes.

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. CLIPS6.24.1 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. It also runs successfully on Linux under Wine.

3.2 Decomposition Description

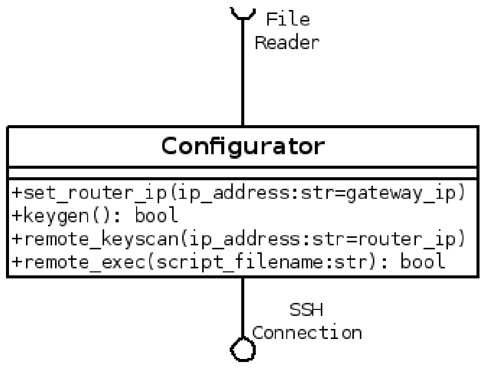
File Reader
Configurator
+set_router_ip(ip_address:str=gateway_ip)
+keygen(): bool
+remote_keyscan(ip_address:str=router_ip)
+remote_exec(script_filename:str): bool
SSH Connection

Figure 2: Configurator Subsystem: Class Diagram

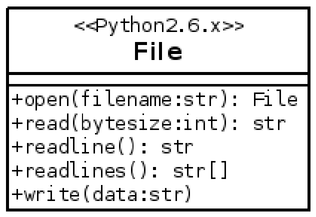

Figure 3: File IO Subsystem: Class Diagram

The configuration and File I/O subsystems are pretty straight-forward and consist on one class each. The CLIPS subsytem on the other hand consists of three main classes. The python library, pyclips, allows integration of our system with an isolated CLIPS process managed by the pyclips library. With pyclips our system can programmatically interact with CLIPS to handle control, errors, and input/output.

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 subsystem heavily relies on pyclips and the Pyramid web framework. The DeclarativeBase class provides an Object-to-Relational Database Mapper (ORM) so any classes based on that will have instant access an API to manage data in the database. The ORM implementation is called SQLAlchemy. The ViewCallable class is really a decorator for any python function or class that will handle incoming web requests. The ScopedSession class provides our system with database transaction management. Extending this session with the ZopeTransactionExtension allows our transactions to be mapped one-to-one with each web request. This means that every request-response cycle will result in a rollback or commit in the database of any data modifications during that cycle.

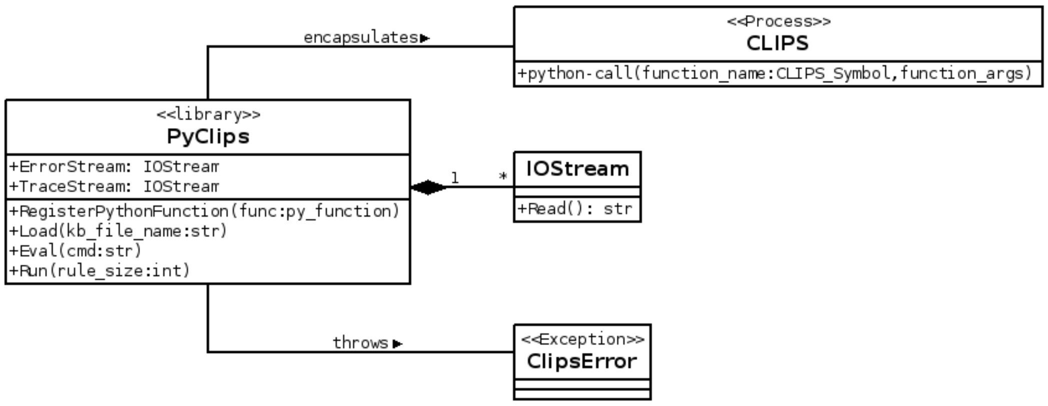

FIGURE 4: CLIPS SUBSYSTEM: CLASS DIAGRAM

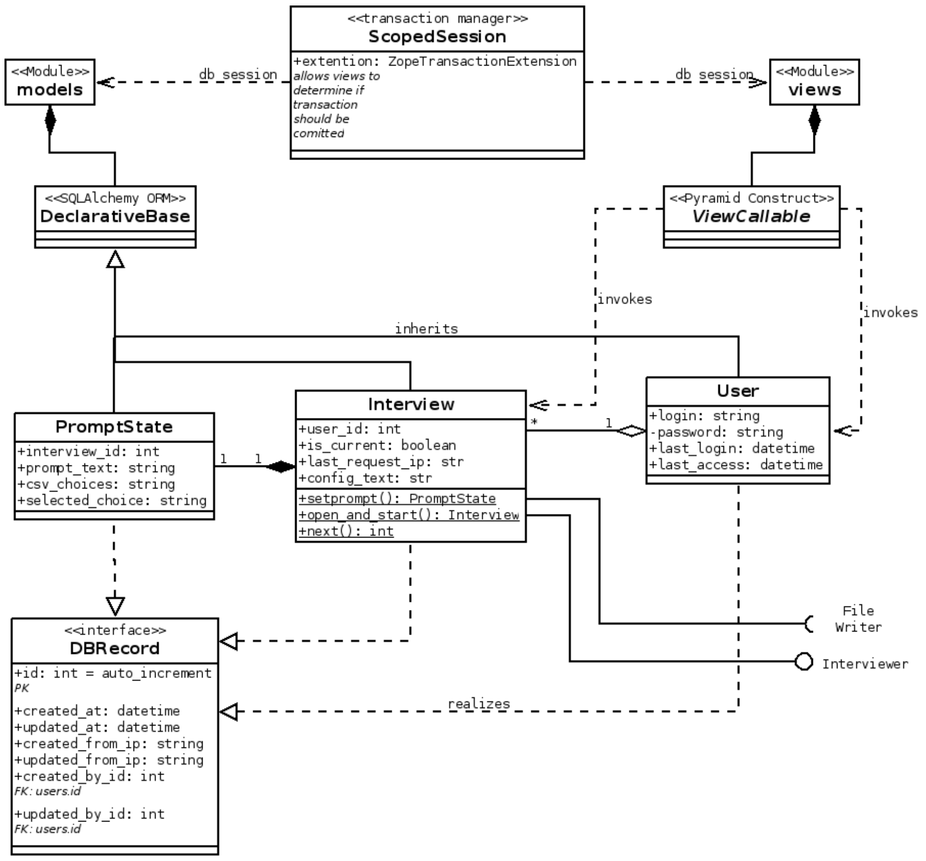

FIGURE 5: WIZARD SUBSYSTEM: CLASS DIAGRAM

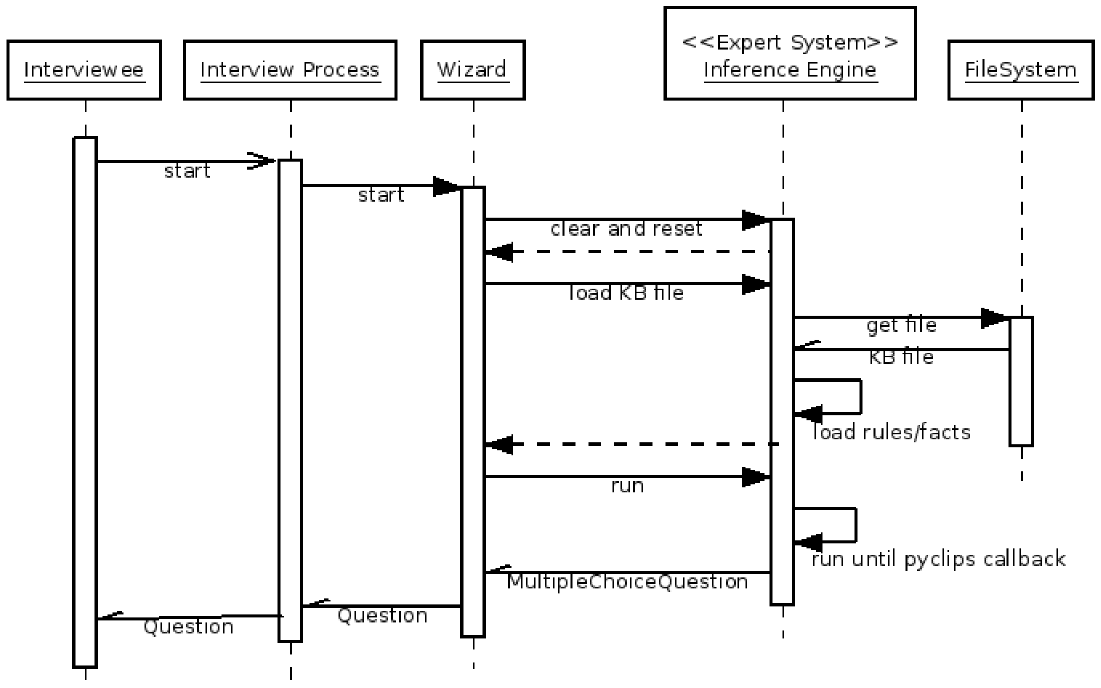

FIGURE 6: START WIZARD: SEQUENCE DIAGRAM

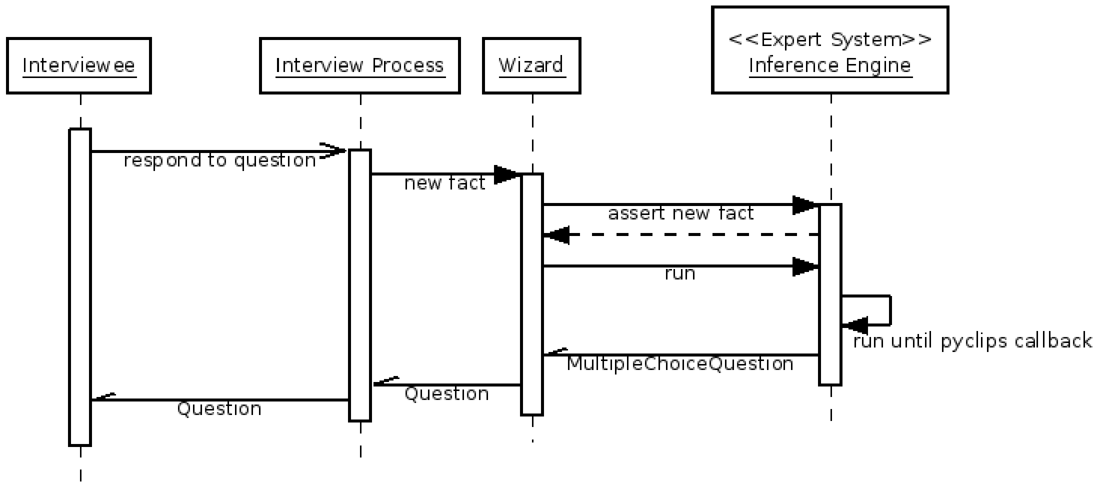

FIGURE 7: ONGOING INTERVIEW: SEQUENCE DIAGRAM

We have grouped these classes into six main components of the system and have deployed our components in the deployment diagram below.

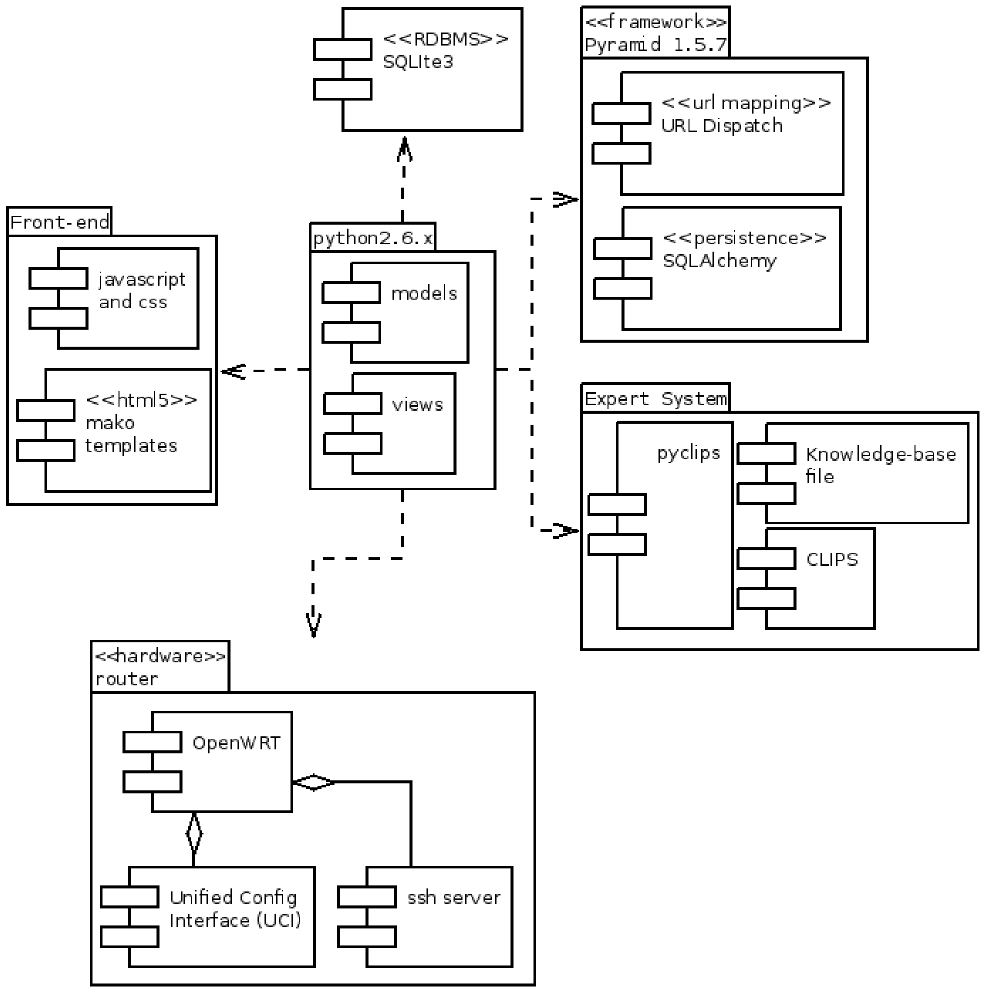

FIGURE 8: COMPONENT DIAGRAM

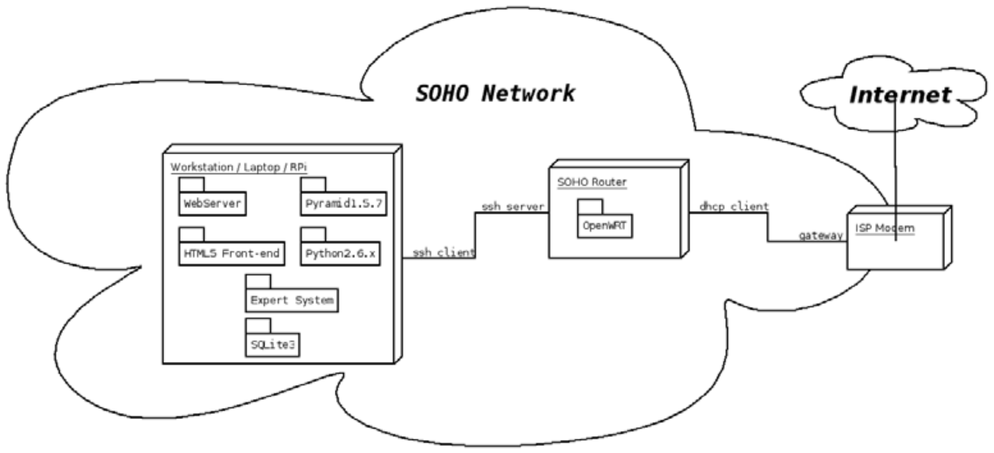

FIGURE 9: DEPLOYMENT DIAGRAM

3.3 Design Rationale

An alternative to our chosen architecture is to forego 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 ability to efficiently handle a large knowledge base. Specifically, Expert Systems handle the problem known as Combinatorial Explosion with pattern-matching algorithms such as Rete.   Consider a system with n variables. Each variable as Z possible values. This system then has Z^n possible end-states. A system with 4 variables and just 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 with an IF-THEN-ELSE or SWITCH-CASE, we instead can 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 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. The Knowledge-base itself cannot configure a router. This system will need to integrate with the Expert System as it runs the interview process.

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 such as

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.

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.

3.3.1 Knowledge-base Design

Several approaches to knowledge base design are known such as the use of certainty factors, backward-chaining, and decision trees. When using certainty factors, the "expert advice" given by the system can be shown to have a quantitative confidence level. For example, it may calculate all factors used to derive the end solution and state that the system has a confidence level of some percentage on the solution presented. This is useful for systems only providing advise to the user who will ultimately make the final decision. However, the goal of our system is determine a configuration script for OpenWRT in a transparent and useful way for the user who knows nothing about OpenWRT configuration. The use of backward-chaining presumes that an ideal or otherwise optimal goal is known ahead of time. Such a design approach is one that tries to determine if the various criteria for the goal are met. A decision-tree, or forward-chaining, is an approach that works towards a goal by gathering facts and inferring the implications of those facts. In the context of propositional logic, this is a logical argument form known as modus ponens. For example, if P implies Q, and P is asserted, then Q must be true as well. Backward-chaining uses the same principle but infers the assertions from the implied goal. Forward-chaining gathers data from the user and works toward finding any solution, optimal or not.

Forward-chaining would be adequate for a small decision tree knowledge-base but we anticipate this knowledge base to grow and to be maintained across a rotating queue of domain experts. Backward-chaining would allow a much larger knowledge-base without having the user to traverse the much larger tree. With backward-chaining the user would only need to traverse some path much closer to the optimal one. CLIPS supports only forward-chaining but we can simulate backward-chaining using rules.

4. Data Design

4.1 Data Description

In order to derive the data points needed for the system, we should examine the desired behavior as described in the statement of scope in our Vision document.

"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."

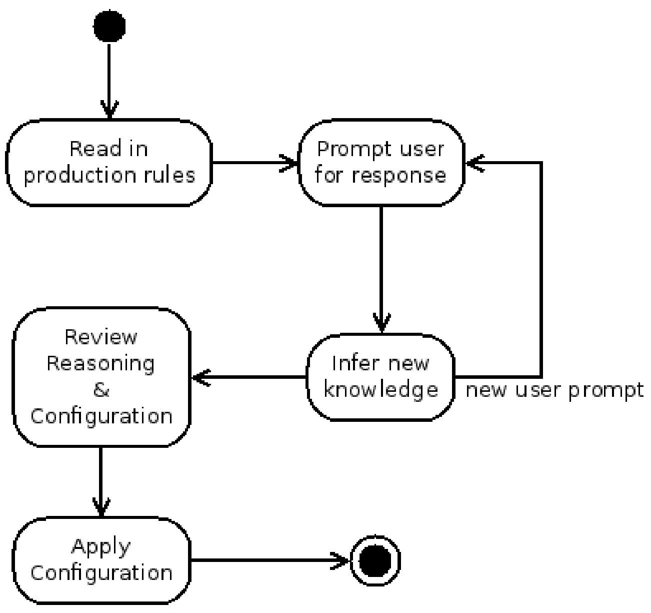

Figure 10: Statement of Scope: Activity Diagram

The above Activity diagram models this statement of scope at a high level. In the following sections we will present details at a lower level to decompose the system into two major activities: the Wizard and the Configuration.

4.1.1 Wizard Activity

The Wizard subsystem carries out an interview-style process using a graphical user interface and a conversational-mode Expert System. An Expert System can be described as a three-part process called Match-Resolve-Act (MRA) carried out by an inference engine. The MRA 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 production rules whose antecedents are satisfied with the asserted facts in **working memory**. 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).

2) A **<u>conflict resolver</u>** selects the rule with the highest priority from the agenda to trigger.

3) An **<u>action executor</u>** 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.

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 <u>refraction</u> which prevents rules from firing repeatedly in the Match-Resolve-Act cycle [7].

The interview process builds on the MRA sub-process by prompting the user for dynamic feedback, asserting the facts from the user's response and fed back into the MRA sub-process. The outputs from the MRA sub-process 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.

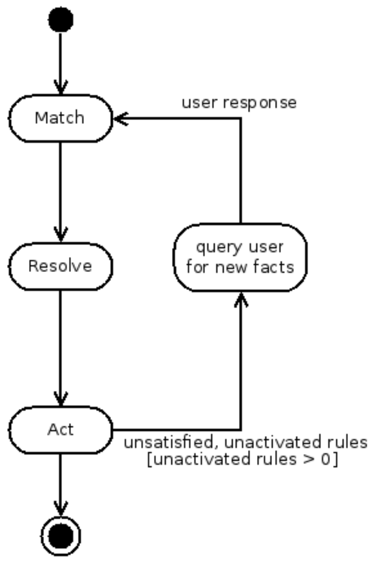

FIGURE 11: WIZARD: ACTIVITY DIAGRAM

CLIPS will handle the dynamic data such as the Agenda and our system should be responsible for keeping track of the interviews conducted and the configuration script origins and reasoning. The rules and initial facts should be stored for access outside of the interview process. Flat file storage will allow portability and human-readability of the knowledge-base. It is vital to this system that there be no barriers for the domain-experts in the public to pick up on this project and contribute to the knowledge-base.

4.1.2 CONFIGURATION SUBSYSTEM

The Configuration subsystem is simpler than the Wizard subsystem. It takes one of three commands from the user to load, save, or apply a UCI 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.

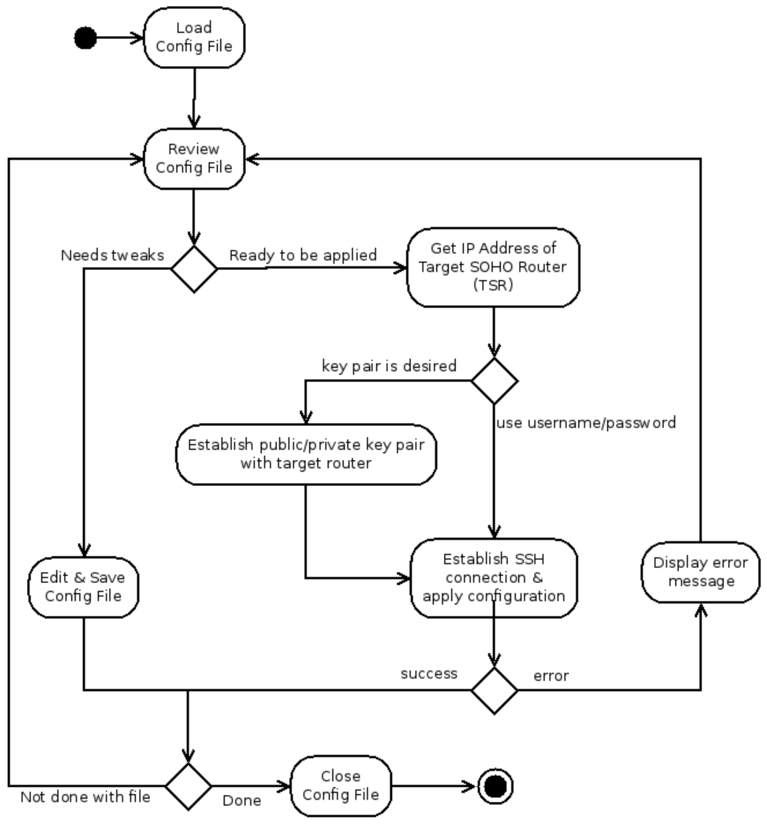

FIGURE 12: CONFIGURATION: ACTIVITY DIAGRAM