DEVice **I**nterface **L**anguage (**DEVIL**)

Boklyn Wong (bw2007@cs.columbia.edu)
Pranay Wilson Tigga/Team Leader (pt2116@cs.columbia.edu)
Vishal Kumar Singh (vs2140@columbia.edu)
Hye Seon Yi (hsy2105@cs.columbia.edu)

Chapter 1: Introduction

1.1 **Introduction**:

DEVIL is a device configuration language that would generate a script or an executable which when run can perform the actual configuration on the device as specified by DEVIL.

Devil is an Easy to Use, Platform independent, Reusable and Extensible.

1.2 **Background**:

Different devices/servers from different vendors are configured in different ways which are generally specific to each individual device. Network administrators need to know every specific detail about the device before they can configure it properly. A network admin capable of configuring a Cisco router might not be able to properly configure a NEC router and need to learn different configuration interfaces for configuring devices from different vendors. This increases the possibility of error and increases the time it takes for deployment and configuration of network. There is currently no network application that would be able to configure devices across vendors in a generic way. Moreover, different devices might follow different configuration protocols, which make it difficult to develop a common utility to work across different platforms. For example, a device running Linux operating system would be configured differently than devices running windows.

1.3 **Motivation**:

The DEVIL has far-reaching implications and applications in today's world of diverse platforms and devices. One such application would be in the field of Network Management for device configuration. Future generations of Network Administrators would not longer have to be familiar intuitively with all aspects of devices in his/her network. With the creation of DEVIL, A Linux network admin would be able to manage as well as maintain a Window's based network. Further applications of DEVIL could be in the field of personal computing. DEVIL could possibly grow into a scheduling language.

1.4 **Related Work**:

There has been much attempt to standardize the configuration mechanism which resulted in different interfaces and protocols for configuration e.g. SNMP, SOAP and Web services are being used for device configuration. This has not helped to alleviate the problem but wound up aggravating it, by creating a need to learn more interfaces and support them. Other work has been done by Netconf Inetrnet Engineering Task force (IETF) working group which proposes a request response based mechanism to configure devices. Our group found out that the main issue of different ways of configuration can be solved by separating the user from device specifics and providing a vendor specific

compiler which generates the output code whose target is the vendor's device. The compiler is provided by vendor and he understands the internals of device. This disassociates user from multiplicity of device configuration techniques and provides him with a single and easy to use interface in the form of our language.

1.5 **Goals**:

High Level: Users who uses DEVIL would never have to know or understand the lines of code that went into producing the output that performs his/her desired task.

Syntactically Intuitive: One of the Goals of the DEVIL language is that users who use DEVIL would not need any prior knowledge of the device to be configured but simply an understanding of what he would like the device to perform. In other words DEVIL must be Easy to Use.

Portable: Since DEVIL is text based and written in XML, it is capable of generating the target script for any device. This fact makes DEVIL a highly portable language. DEVIL is analogous to Java in the sense that compilers for either language generate platform specific code.

1.6 **Features**:

Template Based: To handle device idiosyncrasies i.e. diverse commands across different devices, we will provide a template based mechanism to generate commands and codes for those commands on devices which are not directly supported by language. Moreover, a device provider can provide many more templates.

Context Less and Context Aware: There are certain kind of configurations which are state-full and others are stateless. A state full configuration has a notion of context whereas a stateless configuration can be context less. An example of a context less configuration command is setting IP address and hostname in Linux where there is no dependency on the order of commands being executed, whereas a context aware case of configuration is a router configuration where global commands override local commands e.g. Blocking ICMP packet can be global and Allowing can be local to interface so the context of command being invoked is important.

1.7 **Sample Code:**

Sample Input program

```
<?xml version="1.0" encoding="UTF-8"?>
<Device >
    <informational>
        <type>        Router    <type/>
        <platform>    Linux     <platform/>
        <version>     2.6       <version/>
```

```
        <family>        Core     <device/n>
    <informational/>
   <configuration>

      <hostname > PLTLabTestMachine  </hostname>

       <command template="1">ifconfig eth0:0 ipaddress 10.2.2.14 netmask 255.255.255.0
broadcast 192.168.10.55 </command>
      <createcommand>
          <command>ifconfig</command>
            <args>
               <eth0:1>
                  <ipaddress> 10.12.12.14   <ipaddress/>
                  <netmask>  255.255.255.0 <netmask/>
               <eth0:1/>
             <args/>
      </createcommand>
     <!-- WE CAN DEFINE COMMAND TEMPLATES  ALSO and refer to replace
them-->
     <!-- route add -host XXX.XXX.XXX.XXX dev devicename-->

       <createcommand>
          <command>route add</command>
           <args>
               <host> 192.168.10.12</host>
               <dev> eth0</dev>
           </args>
           <args>
               <host> 192.168.11.12</host>
               <dev> eth0</dev>
            </args>
     <createcwouldommand/>
  </configuration>
 <Device />
```

INTERPRETATION

The above input program is parsed and interpreted to generate the following output. For example inside the informational tag <informational> the user provides information about the platform and other details which will be used by translator to make a decision in order to generate the target program. E.g. if the user specifies Linux as platform the output generated can be shell script as the one given below. The configuration tag holds configuration details and contains information which enables the translator to generate code automatically.

OUTPUT

Output (Target Script)=> output.sh

```
hostname  PLTLabTestMachine
ifconfig eth0:0 ipaddress 10.2.2.14 netmask 2
hostname mycomputer.mynetwork
92.168.10.12 55.255.255.0 broadcast 192.168.10.55
I  fconfig  eth0:1  10.12.12.14  netmask  255.255.255.0  broadcast
192.168.10.255
route add -host 192.168.10.12 dev eth0
route add -host 192.168.11.11 dev eth0
```