Swift Fox Programming Language
Final Report

Marcin Szczodrak [1], Vasileios P. Kemerlis [2],

Xuan Linh Vu [3], and Yiwei Gu [4]

Computer Science Department
Columbia University
New York, NY

February 23, 2011

[1]msz@cs.columbia.edu
[2]vpk@cs.columbia.edu
[3]xv2103@columbia.edu
[4]yg2181@columbia.edu

Contents

Chapter 1

Introduction

1.1 Introduction

For over a decade, *wireless sensor networks* (WSNs) fostered the development of new applications and trends in the broad areas of mobile and opportunistic computing. Such networks are composed of multiple small, low-cost and low-power, multipurpose sensor nodes that communicate with each other over small distances using the wireless medium [8]. Their *self-organizing* capabilities, due to their ad hoc communication paradigm, make them suitable for deployment on demand. This saves engineering time, reduces the installation costs, and accommodates the ad hoc deployment of a wireless network during disaster emergencies or military operations. Hence, applications of WSNs are pervasive in military (*e.g.,* ammunition monitoring, battlefield surveillance, sniper localization [36], reconnaissance, damage assessment), environment (*e.g.,* animal tracking [41], chemical detection [3], pollution monitoring [18]), health (*e.g.,* patient monitoring and drug surveillance in hospitals) [30], home automation (*e.g.,* "smart home") [25], as well as in the commercial domain (*e.g.,* monitoring construction material, inventory management, building automation, vehicle tracking).

Figure 1.1 illustrates the architectural components of a typical sensor node (also known as "mote"). Each mote consists of a *micro-controller*, an outward *power source*, a *transceiver*, some amount of *external memory* (*i.e.,* external in terms of the micro-controller), and a set of *sensor devices* all connected with the micro-controller through an *analog-to-digital converter* (ADC). Each node senses its external environment and conveys the perceived data to nearby nodes using the IEEE 802.15.4 wireless communication standard. Moreover, special purpose nodes act as *base stations* and collect all sensor data in order to further deliver them to aggregation points. That is, they act as "bridges" between network boundaries, or help in establishing a multi-tier network topology [15].

In the early 2000's, WSNs were envisioned as the materialization of the "Smart Dust" concept (*i.e.,* millimeter scale sensing and communication platforms) [43]. However, due to power consumption issues we have yet to witnesses nodes of such scale; instead of "dust" sensors, we had only sensors at the size of a small PDA. Having said that, recent advances on the sensor chips exploit *ambient* power (*e.g.,* radio power from the TV and FM radio) in order to further miniaturize their size by avoiding batteries. Moreover, they offer accurate estimates of the processing that can be performed with certain amount of consumable energy, fueling even more the development of new services of *ubiquitous* manner. As a characteristic example consider the Central Nervous System for the Earth (CeNSE) project, which has been

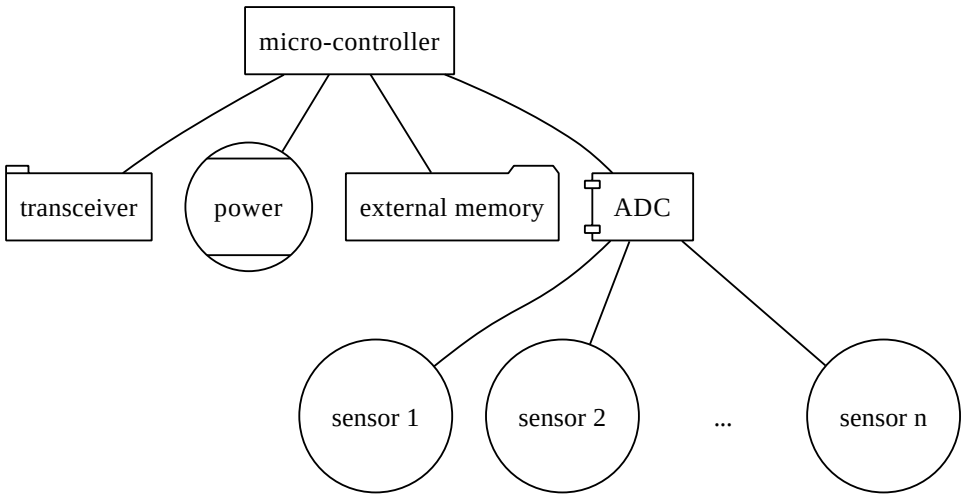

Figure 1.1: Sensor node (*mote*) architecture.

announced by HP and aims at the development of trillion nano-scale sensors and actuators that will all be interconnected with computing systems and provide new services [22].

1.2 Problem Statement

Akyildiz *et al.,* [2] surveyed a large number of different WSN applications and identified a set of factors that affect the design of a sensor network. Such factors include fault tolerance, scalability, operating environment, network topology, hardware constraints, transmission media, and various others. This variety of aspects that deeply affect the design, and therefore the implementation and operation, of a WSN, gave rise to an enormous number of WSN applications that heavily re-implement basic system facilities (*e.g.,* network protocols, routing, data dissemination and collection) in order to provide optimal results. This style of development although it offers many degrees of freedom for the design and development of new and elaborate facilities, it has also serious drawbacks; it delays the adoption of the WSN technology by non-experts (*i.e.,* a typical vendor that provides WSN software and services should have deep understanding and expertise in various fields, ranging from OS development to routing protocol design and implementation).

Fennec Fox is a platform that facilitates policy-based, self-controlling,

dynamic network *reconfiguration* of low-power and lossy networks (*e.g.,* WSNs). It aims at maximizing the efficiency of the system by managing the provided resources, appropriately, so as to meed application requirements at the lowest possible cost. The WSN operation is constrained by power, memory, and computational resources. Sensor nodes are typically battery powered, with few kB to few MB of memory, and have limited processing power (*e.g.,* their clock speed ranges from kHz to few MHz). Using these limited resources, WSNs are expected to support various applications. The Fennec Fox platform is the apparatus of a philosophy, which assumes that WSN applications consist of a *chain of network tasks*.

A task is defined as a simple computational operation, optimized to perform some functionality given certain constraints (*e.g.,* power consumption, delay, security). During their life-cycle, Fennec Fox-based sensor nodes *transition* between various states, where each state performs a simple task. In this context, applications running on Fennec Fox identify conditions for transitioning between various tasks, and controlling the behavior of the tasks, by specifying task parameters. However, to perform their job, tasks require support from other parts of the WSN that provide various services, such as routing or sensing. Therefore, sensor states are not only represented by the task which they perform, but also by a set of other WSN systems that help this task to achieve its expected results.

Fennec Fox was motivated by the need to *auto-configure*, *reprogram*, and *manage* wireless sensor networks. Since such tasks are already complicated by themselves, Fennec Fox attempts also to simplify the modeling and programming of sensornets. That is, sensornets are required to auto-configure themselves during their lifetime. [1]

Fennec Fox is built out of three logical components: a *dynamic network stack*, a *policy control unit*, and a *user programmable application layer*. The dynamic network stack is a library, consisting of protocols and utilities for providing a high level abstraction to the TinyOS [19] system, which supports adaptive applications. Policy control unit is the component responsible for guiding the transitioning between different stack configurations. The transitions are expressed in form of policies, which in response to an event, reconfigure the stack. Finally, the application layer provides a simplified API; it allows the programmer to focus on the application logic, instead of

[1] Ali El Kateeb has stated that "because wireless sensor network applications are new and might require abilities not fully anticipated during design and development, supporting sensor networks with design-modification capabilities is vital to making sensor-network platforms capable of adjusting to their surrounding environment after initial network deployments." [24]

reprogramming communication protocols or mathematical functions. Swift Fox language is a high-level "handle" for programming Fennec Fox. The following example demonstrates the usage of Fennec Fox and Swift Fox and defines their relationship. Suppose that we have a WSN for home monitoring that is also capable of handling emergency situations. In this setup, our network application supports two scenarios; the first one is focusing on monitoring the home environment by collecting sample data from all sensor nodes, and the second one is focusing on monitoring areas with the potential of fire. For the needs of the second scenario, the WSN is capable of taking and streaming pictures. Yet, collecting sensor data, such as light intensity, humidity, and temperature, requires network protocols that can support the type of network traffic generated by sensor nodes, as well as a network topology that can efficiently disseminate the collected data. A simple representation of data collection scenario is illustrated on Figure 1.2. In the second scenario, the sensors detect a fire, based on multi-modal correlation of sensor readings that represent increased probability of fire occurrence. In that case, when fire is detected, we want the system to capture pictures of the fire area to verify the correctness of the fire alert and also as estimate if there are still people around the fire area who can be in danger.

From the network communication perspective, this scenario requires establishing *point-to-point* communication between a sensor node, which can capture a picture of the area in fire, and the sink node, for providing direct high bandwidth communication for data transmission in an emergency situation such as this one. A representation of network configuration supporting this scenario is shown in Figure 1.3. Both figures show scenarios with different data flow objectives: *network energy conservation* versus *low-delay direct data transmission*.

Fennec Fox is a platform that provides the ability to have a reconfigurable WSN that adapts to different application requirements. Through Swift Fox we want to be able to program the Fennec Fox platform, seamlessly, in a high-level and abstract fashion. Hence, Swift Fox is a simple policy-based programming language that specifies conditions that have to be meet for a network to be reconfigured. Using the fire emergency application, we want Swift Fox to reconfigure the network from the monitoring scenario to the emergency scenario whenever light sensors show increased light intensity, humidity sensors show decreased humidity, and temperature sensors show an increase in temperature. Let `monitor` and `fire` represent network state configurations that correspond to the two aforementioned scenarios (*i.e.,* the home monitoring environment and the fire emergency response). Then, using Swift Fox language we would like to be able to express a condition,

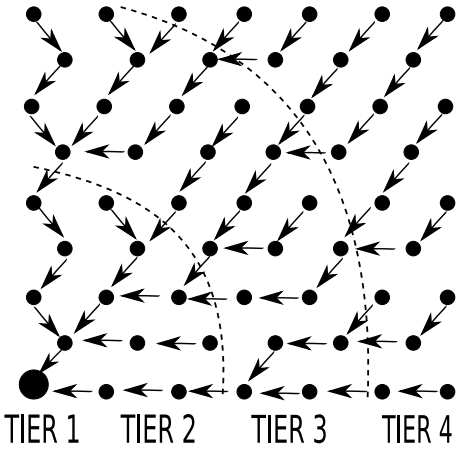

Figure 1.2: Sensor network that establishes a multi-tiered tree topology, where data are propagated toward higher tiers (lower in id), and eventually reach a sink node. To optimize network performance, for example to lower power consumption, data may be aggregated or compressed to decrease the size of the transmitted messages.

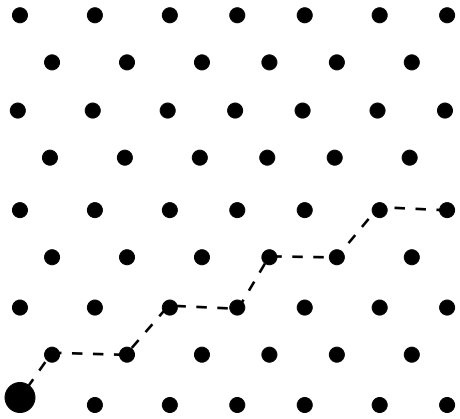

Figure 1.3: In a result of some event, a point-to-point communication between a node and the sink node is established. Network topology is flattened, no data aggregation is performed, and low delays are imported.

represented by an event e captured by the sensors, which would trigger a transition on the network configuration from `monitor` state to `fire` state.

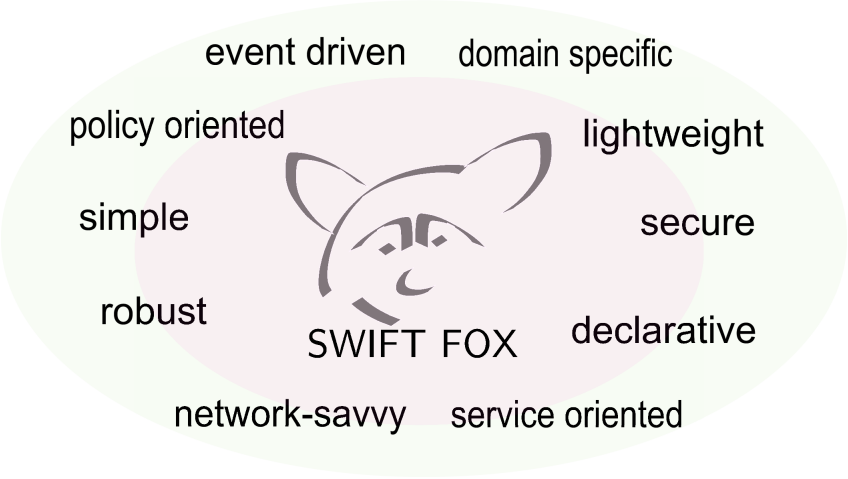

Figure 1.4: Swift Fox buzzwords.

We define eleven objectives that Swift Fox should provide in order to become a successful programming language and we specify them in a form of "buzzwords". These buzzwords were depicted on Figure 1.4, and are defined as follows:

1. **Policy oriented**
   Policies describe precisely the behavior of entities that reside within the bounds of a *policy domain*. Swift Fox allows, implicitly, the definition of a policy domain by giving the ability to describe, seamlessly and accurately, reconfiguration scenarios in an intuitive way. In general, each entity (*i.e.,* a sensor node) is capable of responding differently to certain stimuli, based on the actual facilities provided by the underlying reconfiguration platform (*i.e.,* Fennec Fox). For example, given the sensation of a certain event that indicates a critical incident, a node can cause the reconfiguration of the whole network in order to best disseminate critical data (*e.g.,* proof of destruction or imminent hazard). Evidently, not all possible responses are permissible, either for security reasons (*e.g.,* a node that does not have a vibration sensor should not be able to reconfigure the network for an earthquake response) or for application-specific purposes (*e.g.,* there might be a strict priority among different applications based on their resource

consumption). Through Swift Fox the task of defining a *deterministic* and *authoritative* behavior for the WSN, as a whole, is largely simplified and becomes less error-prone. This is accomplished by adopting a policy-based notation that captures the essence of the *environment semantics* as well as the entities' *rights*, *prohibitions* and *obligations*.

2. **Secure**
   WSNs communicate and provide their services over the wireless medium. The open nature and the omnipresence of wireless signals, along with the lack of physical barriers in most WSN installations, leave a window of opportunity open for the proliferation of malicious behavior (*i.e.,* either because WSNs provide an infiltration channel to a certain domain or because the WSN is the target itself). Therefore, protection against malevolent adversaries will most likely be mandatory in the near future. Authentication protocols, access control mechanisms, and encryption schemes have been studied in the past (in the WSN context) and have already lead to established techniques that offer specific levels of *assurance*. However, most of the available solutions are highly optimized for certain scenarios. Hence, it is of paramount importance to allow dynamic configurations that enable the reuse of *trusted*, mature, and provable-working security components. Swift Fox offers the potential for expressing, naturally, specific security guarantees as well as behavioral requirements for the WSN nodes. Moreover, it allows the combination of different security primitives (*i.e.,* distributed authentication, key exchange, encryption) in a composable fashion that meet the needs of the environment and serve the applications to the best extent possible.

3. **Robust**
   In Section 1.1, we outlined some of the possible WSN usages and application scenarios. However, many of these tasks (*e.g.,* battlefield surveillance, patient monitoring, inventory management) are critical because they might either affect human lives or hinder assets that are essential for the efficient operation of other systems. In particular, for such tasks the WSN is not allowed to exhibit *best-effort* behavior and it is considered a *critical infrastructure*. Swift Fox provides *error-resistant*, *reliable* WSN configurations by allowing the deterministic expression of WSN states in a high-level fashion.

4. **Simple**
   Swift Fox is designed to fuel the development of commercial WSN ap-

plications. Thus, we assume that the audience of the language, people that deploy WSNs or simple contractors, may not be aware of every underlying aspect of a WSN. Yet, Swift Fox should be able to provide a simple way of programming a WSN by defining reconfiguration policies.

5. **Service Oriented**
   Swift Fox provides a way to express stages in the lifetime of a WSN. Its service oriented architecture (SOA) is inherited from Fennec Fox platform, which provides network services for multiple applications. Swift Fox loosely integrates Fennec Fox services with their applications into a network program. This network program is a chain of network states, during which one application with its supporting network protocols is active, and transitions between network states occur in response to events sensed by the network.

6. **Declarative**
   The management of WSNs requires well-defined declarative languages [21]. The programmer should be able to describe *what* should the network configuration be in every case, via event handling actions. In contrast to a procedural language (like C), where the program is a set of step-by-step instructions for execution, a declarative language aims at providing assertions on beginning conditions and destination targets [20]. Nevertheless, like any programming language, it must be *expressive* enough so both the programmer and the compiler to be able to grasp how the program performs over time.

7. **Domain specific**
   Swift Fox is designed specifically for programming and managing WSNs. Its usage scope is well defined for the middle layer of network management. Our language belongs to the same range of domain specific languages (DSLs), in the WSN context, like nesC and Verilog. In fact, at the moment of writing, nesC is the underlying language that Swift Fox compiles into. As a DSL, Swift Fox is developed to address the needs of a given domain and therefore it can only cater to a limited number of concepts. This leads to a higher level language that improves the developers' *productivity* and *communication* with domain experts [9]. The typical characteristics of a domain specific language are the following: it is designed for a particular problem domain (in our case WSNs), it is intended to glue together with other languages to complete a system (*e.g.,* nesC), and finally it resembles configuration

code (*i.e.,* policy specifications) [13]. Moreover, keeping the language confined allows for use of libraries, which are pre-compiled in whatever languages they need, in order to complete the desired underlying tasks. The configuration part of the language can be *expressive* and easily *readable*.

8. **Lightweight**
   Software subsystems can often be designed and implemented in a *clear*, *succinct*, and *aesthetically pleasing* way, by using specialized linguistic formalisms. In cases where such formalisms are not compatible with the principal implementation language (*e.g.,* nesC), lightweight languages [38] are necessary so as to provide the required primitives as meta-constructs. The minimalistic syntax of Swift Fox not only allows memory-efficient objective code, but also provides the ability to exploit the primitives of the underlying platform in an clear, straightforward, and coherent manner.

9. **Network-savvy**
   Swift Fox provides a computing abstraction for the TinyOS system and Fennec Fox platform, which contain libraries of various protocols, such as medium access protocols (MAC), network protocols (*e.g.,* TCP/IP [10] and IPv6 [12]), data dissemination protocols (*e.g.,* Trickle[28]), quality of information (QoI) protocols, reconfiguration schemes (*e.g.,* TENET [15] and Mate [27]), and on demand sensor addressing [35]. Swift Fox allows the programmer to adequately match the services provided by various protocols to quality of service (QoS) expectations imposed by applications.

10. **Platform independent**
    Swift Fox compiles into nesC [14] that is suported by the Fennec Fox platform running on top of TinyOS. In some way, this approach resembles Java; Java code is compiled into byte code for Java Virtual Machine (JVM). The same way as Java becomes platform independent through JVM, Swift Fox becomes independent through Fennec Fox. Currently, Fennec Fox supports only TinyOS; however, in future other popular embedded systems are expected to be supported, particularly Linux, and Contiki [11], which have strong commercial support. Other operating systems, such as MANTIS [4], LiteOS [5], SOS [17], and Pixie OS [30] will be supported as time will allow. Today, wireless sensor nodes running all these operating systems create separate networks, with different programming environments, and with various

programming interfaces. Swift Fox establishes a single application programming interface and unifies various, currently, disjointed network environments. Moreover, it greatly simplifies WSN deployment and management, and encourages WSN production and deployment from various vendors.

11. **Event-driven**
    Swift Fox belongs to a family of languages that express system policies, and compiles into the nesC language [14], a component-based event-driven programming language. The event-driven nature of sensors becomes more evident as hardware reconfiguration mechanisms get to be incorporated in the sensor nodes [24]. The event-driven nature of Swift Fox stems from the *event-condition-action* (ECA) information model of polices, which for the first time proposed and implemented in Bell Labs in 1999, and called Policy Description Language (PLD) [29]. As shown in PLD, this information model of policies allows the creation of a programing language that does not have to be based on any graphical representation (*e.g.,* in Unified Modeling Language (UML)). Finally, Swift-Fox establishes a relation between the event-driven specification of an embedded system and the event-driven policy that describes WSNs reconfiguration.

1.3 Language Overview

Swift-Fox like many other languages, such as AWK, "was born from necessity to meet a need" [16]. Similarly to AWK, we are aiming for a simple language that we can easily use to express transitions between the states of a WSN platform. These transitions occur in response to events detected by the sensor nodes. Here we propose a simple *event-condition-action* language – Swift Fox – that we hope will meet our expectations. Because we believe that we are the first one to address the wireless sensor network reconfiguration problem by designing a new programming language, we hope this language will become successful outside the walls of the classroom. To make this a successful case, we defined eleven characteristics which we want our language to posses: policy oriented, secure, robust, simple, declarative, service oriented, domain specific, light weight, network savvy, platform independent, and event driven.

Chapter 2

Language Tutorial

2.1 Introduction

Swift Fox is an easy to learn programming language. It was designed to allow the quick and easy deployment of a wireless sensor network (WSN) running on top of the Fennec Fox platform [40]. This tutorial consists of two parts. The first part is titled "*Dive Into Swift Fox*"[1] and it has been written for novice Swift Fox programmers, mostly contractors on the field who deploy a WSN and seek to customize its performance so as to meet their clients' needs. It describes tools that are necessary to setup the network, and shows how to install and configure them. Next, the Swift Fox language is taught step-by-step through solving real life WSN deployment issues. In this part of the tutorial we will follow Mike who is a contractor deploying WSNs. Based on Mike's work we will demonstrate how a network is programmed using Swift Fox.

The second part of the tutorial is titled "*Tune Up with Swift Fox*". This is the more advanced part of the tutorial written for expert Swift Fox programmers who want to understand all the details of the Swift Fox language. We will talk about typical errors, control flow, and the compilation process. This part of the tutorial is a supplement to the first part, which will allow a Swift Fox programmer to write correct code, understand errors, and debug programs.

[1]The title *Dive Into Swift Fox* was inspired after the book *Dive Into Python* by Mark Pilgrim [34].

2.2 Dive Into Swift Fox

2.2.1 Installation

Swift Fox is a programming language that allows for the quick and easy setup of self-reconfiguring WSNs. In order to be able to understand how to setup a network with Swift Fox, we first discuss the relationship between the operating system (OS) running on the sensor nodes, Fennec Fox, and Swift Fox.

WSNs are composed of sensor nodes, also known as motes, which are severely restricted embedded devices [8]. Similarly to other embedded devices, sensor nodes require an OS to manage their resources and provide a hardware abstraction layer. Though some sensor networks may use Linux, there is another special and lightweight OS designed for all sensor motes, which we shall use. TinyOS is an operating system for WSNs in which we can execute simple applications that can run on the sensor network. On top of the TinyOS we place the Fennec Fox platform. Fennec Fox is essentially a middleware that allows sensor nodes to adapt the network performance on the fly and also facilitates the development of adaptable multi-tasking applications [40]. Swift Fox is a programming language to configure the services provided by Fennec Fox. The relationship between the underlying OS (*e.g.,* TinyOS), Fennec Fox, and Swift Fox is illustrated in Figure 2.1. Using Swift Fox we can easily program Fennec Fox to meet various system objectives; Fennec Fox receives the blue-print of the expected network performance, written in Swift Fox, and by using various protocols defined in its libraries, it instructs the operating system to perform in a way that will meet the programmers' expectations.

Evidently, to program a WSN with Swift Fox we will need to have a Swift Fox compiler. However, because Swift Fox runs on top of the Fennec Fox, which in turn runs on top of TinyOS, we will need to have these parts also in place.

The following components are necessary in order to successfully program a WSN with Swift Fox:

1. a laptop or a PC
2. an OS for the WSN
3. a running Fennec Fox platform
4. a Swift Fox compiler

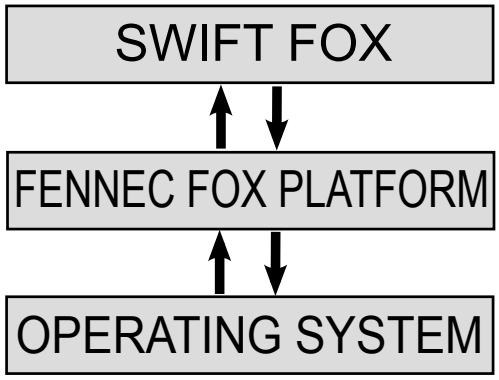

Figure 2.1: Swift Fox running on top of Fennec Fox

Start by selecting some host machine, either a laptop or a PC, where you can reserve some space to set up the Swift Fox programming environment. This machine can either run Windows, Linux, or Mac OS X. However, in this tutorial we recommend to use the VMware image prepared by the authors; any machine with an OS that can run a VMware hypervisor should be fine (*e.g.,* VMware Workstation, VMware Player, or VMware Fusion).

TinyOS is the WSN OS that we will use. As we said before, Linux is another alternative for embedded devices such as wireless sensor motes, but it is not yet supported by Fennec Fox. To prepare your machine to compile programs written in Swift Fox, you will need to get TinyOS. One way of installing TinyOS on your machine is by following installation steps as explained on the TinyOS website: `http://www.tinyos.net`. It is important to install correctly the TinyOS and nesC compiler, and setup the system variables so as to compile Swift Fox programs. In the provided VMware image, the TinyOS, nesC, C cross-compiler for ARM processors, and many other tools are already installed for you. This image can be downloaded from: `http://www.cs.columbia.edu/~msz/wsn/`.

At this point we assume that you have either correctly installed TinyOS as described on the TinyOS website, or you have downloaded the VMware image from author's website. The next step is to install the Fennec Fox platform. Fennec Fox can be downloaded from `http://www.cs.columbia.edu/~msz/fennec/`. The current version is 0.03. After downloading, please install Fennec Fox into `/opt/fennec-0.03`.

After the Fennec Fox installation ensure that the `FENNEC_FOX_LIB` system variable is set correctly. If the library path of the Fennec Fox was not

changed during the installation process, after running:

```
$ echo $FENNEC_FOX_LIB
```

from command line, one should get:

```
/opt/fennec-0.03
```

Finally, the last thing you need is to get the Swift Fox compiler. You can download the Swift Fox compiler from `https://nslvm2.cs.columbia.edu/trac/attachment/wiki/WikiStart/swift0.01.tar.gz`. Make sure that your Swift Fox compiler is on your system path, by running **sfc**, which should print the Swift Fox welcome banner and a description of how to use the compiler.

```
                    Swift Fox Compiler!

For example if you have a code in sample.sfc and library in sample.sfl,
then run:

$ ./sfc sample
```

Great! You have setup your computer to work with Swift Fox. Now, let us introduce you to Mike.

2.2.2 This is How We Do (step-by-step)

Mike lives in Brooklyn and works as a network engineer for various companies around New York City. Mike also has its own small business that specializes in deployment, installation, and management of wireless sensor networks. Recently, he learned about Swift Fox. In his opinion, Swift Fox can speed up and improve the quality of his work. We will spend a day with Mike to see how his typical work day looks like and why Swift Fox is beneficial for him.

Blinking

Today Mike has a couple of WSNs to deploy. However, before deploying them he is going to check if the hardware works correctly. It is possible that during the shipment, some of the motes were broken. The simplest way to verify that nothing is wrong with the motes is to order them to do a simple task. Each mote has few light-emitting diodes (LEDs); Mike tests his motes by writing a program that orders them to start blinking one of their LEDs.

Mike's favorite text editor is *vi*. Therefore, he invokes it and starts programming the sensor motes as follows:

```
$ vi blink.sfp
```

Of course Mike is free to use any editor he wants, apart from vi, in order to create, edit, and save files. Inside the new file, Mike enters:

```
configuration hello {Blink nothing}
start hello
```

and then saves and closes the file. This simple program stored in the *blink.sfp* file can be compiled, resulting in code for Fennec Fox. To compile blink.sfp program Mike enters in the shell:

```
$ sfc blink.sfp
```

By calling **sfc**, Mike invokes the Swift Fox compiler. This compiler takes as the input the source code of a Swift Fox program. Once the program is compiled, the generated Fennec Fox code together with the operating system running on sensor motes (*i.e.,* TinyOS) are compiled to hardware dependent machine code, creating a new system image. Mike wants to use the TinyOS operating system and to test blink.sfp on the telosB motes (*i.e.,* a specific wireless mote brand). To compile and install he should enter into a shell the following:

```
$ make telosb install
```

This command will compile the code, load it into a telosb mote attached to Mike's laptop (or PC) through a USB port, and start the sensor mote. Mike takes each mote, one by one, and loads the code into them. If the mote LED does not start blinking, then the mote is broken.

While Mike is testing his motes, let us discuss the Swift Fox program that he wrote. First, notice that this program is very short; it contains only two lines. Mike's job is very simple: to check that motes can blink, therefore his program should also be short and simple. Let us now dig into the code to understand what Mike has written there. The first line of the program is:

```
configuration hello {Blink nothing}
```

In this short statement, Mike defines the configuration of the wireless sensor network mote. To do this, he is using the keyword **configuration**, which tells Swift Fox that whatever comes on the right of the `configuration` keyword defines a system configuration. The next word, which in Mike's program is `hello`, allows Mike to give a name to the configuration that he defined. We do not know yet what is the configuration, but we do know that Mike is defining one, and he is calling this configuration `hello`. What comes next in the brackets is the definition of the configuration. The first word specifies the job that the sensor mote should do, whereas the second word specifies the communication mechanism that the sensor mote should use in order to communicate with other motes. In this program, Mike tells the sensor motes to start blinking, by using the word `Blink`. He is also telling the sensor motes not to use any kind of communication mechanism; he is expressing this by using the word `nothing`. He does not want sensor motes to send any messages. All he wants is to make sure that they are not broken, and he can check that by making their LEDs blink.

The second line of Mike's program is:

```
start hello
```

In this statement Mike specifies the default configuration of the sensor mote, right after it starts (boots up). To do this, Mike is using the keyword **start** followed by the name of the configuration that he has defined in the first line of the blink.spf program. In other words, the second line of the program tells the motes to start running a configuration called `hello`. Based on the definition of the `hello` configuration, motes start executing the task, which is called `Blink`, and they do not attempt to communicate since `hello` configuration specifies `nothing` in the second field of the definition.

Mike has finished testing all sensor motes and he puts them into his car. Now we are ready to drive with Mike to the first site, where Mike will demonstrate the commercial deployment of a WSN using Swift Fox.

Exercise

The two line blinking test program written by Mike is actually not the simplest code you can write. Write a simpler two line Swift Fox program that will tell the sensor motes not to use any application or any communication mechanism.

It's hot here, not!

Mike's first client is Tony. Tony owns few buildings in Brooklyn and he is leasing apartments. Most of his buildings are around the Bay Ridge area where he lives, but he also has a four-family building in Sunset Park. Last winter was very cold and windy, so the temperature inside the buildings was very low and required constant heating. Whenever it was cold, Tony had to visit all his buildings to turn on the boilers. However, one day there was so much snow on the street that he could only walk to nearby buildings, but he could not get on Sunset Park. People living in Tony's Sunset Park building were freezing in their apartments, and Tony was violating The City Housing Maintenance Code and State Multiple Dwelling Law that requires building owners to provide heat and hot water to all tenants [31]. Tony decides to invest in WSN technology, hoping that Mike can setup a system that will automatically adjust the temperature according to the New York City (NYC) regulations.

The NYC Department of Housing and Preservation Development specifies the temperature requirements that should be met by the owners of NYC buildings [31]. According to these rules, between October 1st and May 31st, temperature inside an apartment is required to be above 68 and 55 degrees Fahrenheit, between 6 AM till 10 PM and 10 PM till 6 AM respectively. To make sure that temperature is never falling below the value required by the city, and to make tenants feel warm inside their apartments, Tony tells Mike to set the threshold values to 70 and 60 degrees Fahrenheit, for day and night respectively.

Mike starts designing his network with the specification of policies, which will govern a network behavior that meets Tony's expectations. The wireless sensor network, which Mike will deploy inside Tony's building, should monitor the temperature all day and night, and send messages to the boiler whenever temperature falls below the threshold values. On a piece of paper, Mike draws a finite state machine, which will represent the various states of the network. His drawing is shown in Figure 2.2.

Based on the drawing, shown on Figure 2.2, Mike comes to the conclu-

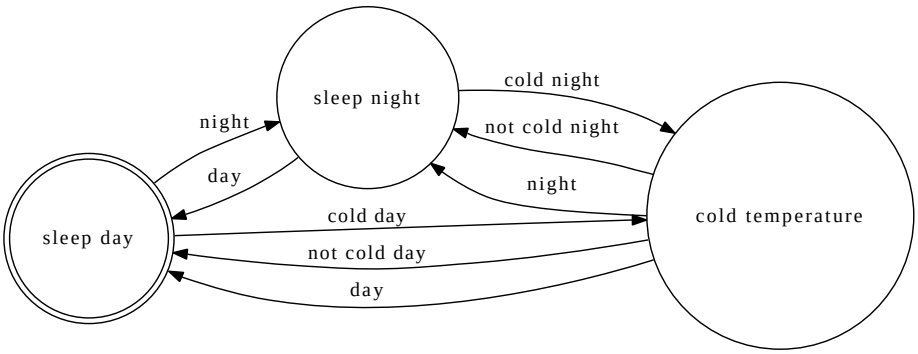

Figure 2.2: Mike's drawing of the WSN states for Tony's building.

sion that his network has to operate in three different configurations:

1. sleep during the day and monitor temperature
2. sleep during the night and monitor temperature
3. send temperature alerts when is too cold

Mike opens a new file, called *tony-bld.sfp*, and programs the following configurations in Swift Fox:

```
configuration sleep-day {nothing CTP}
configuration sleep-night {nothing CTP}
configuration too-cold {Send-Temp CTP}
```

Let us look at Mike's configuration definitions in more detail. He defines three configurations, which he calls according to the states of his drawing (see Figure 2.2). The names of these configurations are: *sleep-day*, *sleep-night*, and *too-cold*. In each of these states there is network protocol activated, called *CTP*. CTP, which stands for Collection Tree Protocol, is the wireless sensor network protocol that establishes a tree network topology, and can efficiently collect data from sensor motes. Mike enables this protocol in all configurations, even those that represent sensor motes in sleeping states (*i.e.,* motes that will not generate any alarm message). He wants all motes to be able to send messages. In case one node senses low temperature and starts sending temperature messages to the boiler, then these messages

would have to be forwarded through other sensor motes that are sleeping and not sensing the cold temperature.

Once the configurations are defined, Mike starts defining events that will trigger transitions between different configurations. Mike starts with two simple events: night and day, which will keep track of when the day and the night start. To program this in Swift Fox he writes:

```
event-condition day {Timer = 16hr}
event-condition night {Timer = 8hr}
```

These two lines define two event-conditions. Each line starts with the keyword **event-condition**, which says that what will follow is the definition of a new event that is triggered when some condition becomes true. In particular, the first line defines an event that is called *day*. This event uses a timer to measure time and is triggered after 16 hours. Mike sets this event to 16 hours, because this is how long the sensor motes should measure temperature inside the building and test it against the 70 degrees temperature value, which is the minimum temperature value that Tony is allowed to have inside the building during day hours. Similarly, the second line defines an event called *night*, which sets a timer for 8 hours.

By looking Figure 2.2 again, we notice that Mike has already defined two events *day* and *night* that trigger network reconfiguration and transition from one configuration to another. However, there are two more transitions related two the temperature; they are called *cold-day* and *cold-night*. Mike programs them as follows:

```
event-condition cold-day {Temperature < 70F }
event-condition cold-night {Temperature < 60F }
```

As before, these two new events have names: *cold-day* and *cold-night* respectively. Both of these events are tested against temperature sensing values, and their conditions are set to less than 70 and less than 60 degrees Fahrenheit. As in case of the two time related events before, these two definitions of temperature related events have similar constructs: the keyword **event-condition** is followed by a chosen name of a new event definition, followed by a specification of the source of event and a value condition applied to the data received from the source of event.

Once configurations and events are defined, Mike specifies the policies that will define the transitions between different configurations. First, Mike notices that there are events, which they always trigger the same configu-

ration. These events are *day* and *night*. As shown on Figure 2.2, whenever *day* occurs, the network is reconfigured to the configuration called *sleep-day*, and whenever *night* occurs, the network is reconfigured to the configuration called *sleep-night*. Mike programs these policies in Swift Fox as follows:

```
from any goto sleep-day when day
from any goto sleep-night when night
```

Let us look at the first line of the code. This line creates a policy which says whenever (the network can be at any configuration, as indicated by keyword **any**) an event called *day* occurs, reconfigure the network (expressed by keyword **goto**) to a configuration called *sleep-day*. Similarly, the second line creates a policy, which says that whenever an event *night* occurs, reconfigure the network to a configuration called *sleep-night*. Policies in Swift Fox match configurations and events together.

All policy statements are specified by defining a transition from one, or more, configurations to another configuration in the presence of an event. Each policy starts with keyword **from**, which is followed by the name of a configuration that is already defined. Hence, in Mike's case he can start policies by writing the keyword **from** followed by *sleep-day*, *sleep-night*, or *too-cold*, which are all configurations that he defined. The keyword **from** followed by the name of the configuration says that this policy can only be considered when the network is currently configured according to that configuration name. Therefore, if the network is currently in configuration *sleep-day*, none of the policies starting with **from** *sleep-night* or **from** *too-cold* can be applied. The exception to this rule is a policy starting with **from any**, which always has to be applied, no matter what is the current configuration of the network. Next, we have a policy statement that is followed by the keyword **goto** and another configuration name; one of those already defined. The **goto** $< configuration - name >$ part of the policy statement specifies the configuration in which the network should be after the policy is applied. For example, Mike may say **from** *sleep-day* **goto** *sleep-night*, which starts a policy that is considered when the network is in configuration *sleep-day* and when it is applied, leaves the network in configuration *sleep-night*. Finally, the policy statement ends with the event that must be fired to trigger the network reconfiguration. This last part of the policy statement starts with the keyword **when** and it is followed by an event name that is already defined. For example, Mike can say **when** *day* or **when** *cold-night*. All defined events can also be used in negation by preceding the event name with keyword **not**. For example, Mike is using negation on his transition

events in Figure 2.2, where he says *not cold-day* and *not cold-night*. Let us actually see how does he program the rest of the events, that are related to the temperature conditions.

```
from sleep-day goto too-cold when cold-day
from sleep-night goto too-cold when cold-night
from too-cold goto sleep-day when not cold-day
from too-cold goto sleep-night when not cold-night
```

Up to this point it should be straightforward to understand Mike's policies, but let's go once more over the last policy. The last policy is only considered when the network is in configuration *too-cold*. If that is true, the **not** occurrence of the event called *cold-night* is checked. Note that if it is actually cold, and the event *cold-night* is triggered, this policy is skipped. However, if it is not cold, meaning that the event *cold-night* is not triggered, the policy is applied, which means that the network is reconfigured to the *sleep-night* configuration.

After defining all configurations, the events, and then specifying the corresponding policies, Mike has to write one more statement; the initial configuration statement. For this program Mike writes:

```
start sleep-day
```

Mike adds comments inside the program code and saves his program in a *tony-bld.sfp*. Comments start with the hash (#) symbol. Everything that Mike writes on a line that starts with # is not a part of the program, and the Swift Fox compiler will skip it. The comment lines are useful for documenting the code and for noting important information.

Exercise

Before we move to the other examples of the programs written by Mike, for one more time consider the tony-bld.sfp program and the associated Figure 2.2. Based of the figure and the code, can you tell when (at what hour) Mike will start (boot up) the sensor network to collect the temperature readings inside of the Tony's building on the Sunset Park?

The complete Swift Fox code for the temperature monitoring system that Mike will install in Tony's building is the following:

```
# tony-bld.sfp
# Author:  Mike
# Date:  03/12/2010
# Job:  Tony's building on Sunset Part
# Program:  Monitor temperature and send data to boiler

# define configurations

configuration sleep_day {nothing nothing}
configuration sleep_night {nothing nothing}
configuration too_cold {sendTemp ctp}

# define time passing events

event-condition day {timer = 16hr}
event-condition night {timer = 8hr}

# define temperature sensing events

event-condition cold_day {temperature < 70F }
event-condition cold_night {temperature < 60F }

# reconfiguration policies

from any goto sleep_day when day
from any goto sleep_night when night
from sleep_day goto too_cold when cold_day
from sleep_night goto too_cold when cold_night
from too_cold goto sleep_day when not cold_day
from too_cold goto sleep_night when not cold_night

# and finally, the initial configuration

start sleep_day
```

The library file for Tony's building would be saved in tony-bld.sfl file and would contain the following libraries:

```
# Standard Fennec Fox Applications

use application blink $(FENNEC_FOX_LIB)/Application/SimpleBlink
use application collector $(FENNEC_FOX_LIB)/Application/DataCollector
use application sendTemp $(FENNEC_FOX_LIB)/Application/SendTemp

# Standard Fennec Fox Network Protocols

use network broadcast $(FENNEC_FOX_LIB)/Network/broadcast
use network ctp $(FENNEC_FOX_LIB)/Network/ctp

# Events

use source temperature $(FENNEC_FOX_LIB)/Events/Temperature
use source timer $(FENNEC_FOX_LIB)/Events/Timer
```

2.3 Tune Up with Swift Fox

2.3.1 Errors and Exceptions

Beginners writing in a new language often make errors and mistakes. These errors may confuse or carry an incorrect message. Although Swift Fox is a relatively simple programming language, with limited number of keywords and constrained structure, initial programming in Swift Fox can be error prone. To minimize the number of errors, which in Swift Fox can be typographical or syntactical, the Swift Fox compiler has a mechanism to catch language errors and exceptions.

Swift Fox errors and exceptions mechanisms are designed to provide readable and intuitive feedback to the programmer and to ensure the correct execution of the underlying Fennec Fox platform. Programs written by novice Swift Fox programmers may contain a lot of errors. During compilation, Swift Fox recognizes an error, and provides a descriptive error message to the programmer, along with a hint about its possible source and a way of recovering. Swift Fox reports errors related to one statement at the time only, and aborts. If there are more errors in the following program statements, these errors will be reported after the first error line is corrected and accepted by the compiler during the next compilation process.

For example, we may have a program that starts with two configuration statements, where the second statement is missing left bracket '{':

```
configuration report { collector ctp }
configuration sleep nothing nothing }
```

When this program is compiled Swift Fox reports the following error message which displays the error line and indicates the position of the error within the line. Swift Fox compiler will also specify if the source of the syntax error is in the library or in the program file.

```
$ sfc example

sfc in program:  example.sfp
syntax error at 2 in this line:
configuration sleep nothing nothing }
                    ^
```

2.3.2 Control Flow

The control-flow statements of the Swift Fox language specify the order in which configuration and events are defined, and the priority and order in which the execution of policies is performed. In the first part of the tutorial all types of statements were presented and the flow of the program was discussed based on Swift Fox examples. Now, the control flow will be presented in a more formal way.

All Swift Fox programs have a fixed and well defined structure. First, a Swift Fox program starts with at least one or more configuration definition statements. Formally, the syntax of a configuration statement is:

`configuration` *`configuration-name`* { *`list-of-component-names`* }

where the *list-of-component-names* in the current version of the Swift Fox is a list of two components, one for the application task and one for the network protocol, which together construct a definition of a new Fennec platform configuration. Second, following the configuration statements, Swift Fox programs have zero or more event definition statements. Formally, the syntax of the event statement is:

`event-configuration` *`event-name`* { *`event-condition`* }

where the *event-condition* consists of the source of the event followed by a relational operator and the constant value that specifies the condition to be met. Third, after the event statements, Swift Fox programs have zero or more policy definitions. Formally, the syntax of the policy definition is:

`from` *`configuration-name`* `goto` *`configuration-name`* `when` *`event-name`*

where the first *configuration-name* is a name of already defined configuration according to which Fennec Fox has to be configured to consider this policy valid. A policy that is universally valid, meaning a policy that should always be considered no matter what is the name of the current configuration, should use a special keyword **any**. The second *configuration-name* should be an already defined configuration to which the system should be reconfigured in the presence of the *event-name* event. Policies are applied from top to bottom, and once the policy condition becomes true, the system is reconfigured according to the configuration specified in the second

configuration-name. Finally, Swift Fox programs have a one-line initial configuration statement. Formally, the syntax of this statement is:

```
start configuration-name
```

where the *configuration-name* is the name of the defined configuration in which Fennec Fox starts after system is booted.

Concluding, control flow in Swift Fox is top to bottom, without any statements that would alter it. Comparing to other languages like C, Java, and Python, there are no if-else statements, while loops, gotos, and labels. Because of this, Swift Fox programs are readable, from top to bottom, allowing for a intuitive and easy to understand programming style, with the declarations of configurations and events preceding the specification of polices.

2.3.3 Run-Time Environment

The Swift Fox compiler generates code that corresponds to a Swift Fox program and runs it on the Fennec Fox platform. Once the system environment is created, it stays fixed afterwards. From the perspective of the Swift Fox programmer, everything that is defined is globally accessible, so once a configuration, or an event is defined, it can be used in the rest of the program. The Fennec Fox run-time environment needs to be fixed for various reasons, but more importantly for assisting the memory and storage allocation, access to variables, and linkage between the various Fennec Fox components, which are actually taken care by the run-time environment of the system in which Fennec Fox is executed. For example, the issues previously listed, are taken care by TinyOS and nesC compiler, or Linux and C compiler, depending on the OS that hosts Fennec Fox.

Swift Fox uses libraries, as well as configuration and event declarations to generate a run-time environment in Fennec Fox. The library source definitions are used in order to locate libraries with platform components that provide various functionalities, such as applications, protocols, and so forth. The source code stored in such libraries provides the services requested in the configuration and event declarations. Notice that during compilation, only services that are utilized are actually included in Fennec Fox. For example, Swift Fox has a library with applications, and one of them, Blink, provides a service with a simple task that blinks LEDs on the sensor mote. If this task, Blink, is used in any of the configuration statements, the code of the Blink is included in the Fennec Fox, and Swift Fox can refer to this

task from various statements in the code. Otherwise, although Swift Fox knows about Blink, it will not include its code inside the Fennec Fox, since this code is never used, and the Blink task will not be accessible.

2.3.4 Libraries

Libraries contain source code that provides some service classified according to the components of the Fennec Fox platform. For example, libraries can provide application services, where simple tasks can be called and executed (*e.g.,* Blink). Libraries also contain code related to services of other components, such as network protocols, security, and quality of information. Currently, only libraries related to application tasks and network protocols are supported in Swift Fox. These libraries can be part of the Swift Fox standard library or can be a third party libraries.

2.3.5 Swift Fox Standard Library

Swift Fox standard library is directly related to the Fennec Fox library, which provides applications, network protocols, and other services provided by the Fennec Fox platform. At the current state Swift Fox contains two simple libraries with applications and network protocols. The application library contains three simple applications: blink that blinks LEDs, sendTemp that senses temperature and broadcasts the sensed temperature to a base station, and sendHum that senses humidity and broadcasts the sensed humidity to a base station. The network library contains two protocols: the Collection Tree Protocol, known as CTP, and a simple broadcast protocol. Fennec Fox library also provides handle functions for event management. At the current state timer, temperature, and humidity events have been ported to the Swift Fox. Timer event can be requested to fire after some period of time, whereas temperature and humidity events fire when sensor's reading reaches specified threshold value. Eventually, more applications, network protocols, and sources of events will be supported by Swift Fox standard library.

2.3.6 Compilation

Swift Fox programs are compiled as follows:

```
sfc <program_name>
```

where `sfc` is the command to start the Swift Fox compiler, and $< program\_name >$

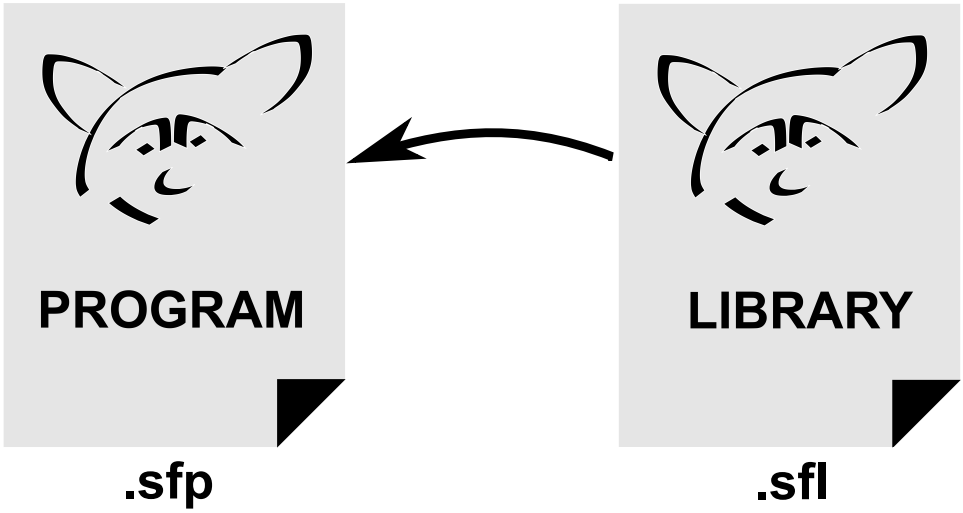

Figure 2.3: Swift Fox source files (.sfp) and libraries (.sfl)

is the path of the file containing the source code of the program written in Swift Fox. Source code files must have the suffix `.sfp`, which stands for Swift Fox Program. Before compilation, the compiler looks for a special file specifying where libraries of various Fennec Fox components are available. This allows the compiler to check if all libraries used in policies written in the Swift Fox program exist and are accessible to the compiler (*e.g.,* where are all the applications and network protocols specified inside the Swift Fox program). Swift Fox looks for the library file in the current directory under the name $< program\_name > .sfl$. Therefore, Swift Fox library files must have the suffix `.sfl`, which stands for Swift Fox Library.

Figure 2.3 illustrates the relationship between a Swift Fox program and its corresponding library file. First Swift Fox compiler processes the library file, saving the path to the source code of each library and labeling the library with an appropriate type. For example, a library with *blink* application has type *application*. Next, Swift Fox compiles the program file. During the compilation, Swift Fox uses information from library to ensure that all types are correct and that program code refers only to Fennec Fox services which path is known.

Chapter 3

Language Reference Manual

3.1 Introduction

This manual describes the first draft of Swift Fox language as specified by the authors on February 23, 2011. In the following sections we define in detail the basic constructs used in Swift Fox. Special consideration have been given in making this manual reader-friendly and comprehensible, but at the same time formal enough so as to capture the the basic concepts that Swift Fox employs.

> Commentary material is indented and written in smaller type, similar to this excerpt.

3.2 Lexical Conventions

In the Swift Fox language, a program consists of one or more *translation units* stored into different files. The translation process consists of multiple phases, which are described in great detail in [39]. In the following section, we present the primitive constructs that are used throughout the first phase of the translation, namely the *lexical analysis*. During that phase, the source code of a Swift Fox program is considered to be a stream of characters that is fed into the lexical analyzer (lexer). Subsequently, the lexer[1] groups sequences of characters together and identifies *tokens*. Each token is a pair of a `name` and a `value`; the value corresponds to a particular *lexeme* that is identified by the regular expression for the corresponding token, whereas the name is merely an identifier for abstracting the set of lexemes (*i.e.,* referring to the language of lexemes that are described by the same regular expression) [1].

token: $<$ `token_name`, `token_value` $>$ (*e.g.,* $<$ `identifier`, `blink` $>$)

> Appendix .2 includes the definition of a Swift Fox lexical analyzer for the `flex` lexer-generator [26, 33]. The reference implementation of the Swift Fox compiler uses the `flex` lexer-generator and the `YACC` parser-generator, [23, 32], however this manuscript provides (hopefully) all possible details for implementing a Swift Fox compiler in every available compiler-compiler tool.

[1] Also known as *scanner*.

3.2.1 Tokens

Tokens in Swift Fox, similarly to every other language, are composed by sequences of characters. The *lexer* (*i.e.,* lexical analyzer) produces four classes of tokens:

- `identifiers`
- `keywords`
- `constants`
- `operators`

Additionally, the *whitespace* character class is used in order to separate the various tokens. The following characters are considered whitespace and along with comments are ignored throughout the rest phases of the translation.

- `space`
- `horizontal tab`
- `vertical tab`
- `form feed`
- `carriage return`

Notice, however, that `newlines` are not considered whitespace and they have a special meaning in Swift Fox (see Section 3.3.3).

> Swift Fox is currently under development. Therefore, there might be some inconsistency between the formal description of the tokens and the corresponding `Lex` definition in Appendix .2.

3.2.2 Comments

Comments are introduced by character `#` and terminate with a newline. Everything between the comment character and the newline character is considered to be a comment, and therefore it is ignored. Comments must start on a new line, and can only be preceded by a delimiter, but not a line of a Swift Fox code.

3.2.3 Identifiers

Identifiers in Swift Fox are sequences of letters, digits, and character "_". They are used in order to *name* specific instances of the primitive types that the language provides (see Section 3.3.1) and, hence, they are considered to be variables. Identifiers can be of arbitrary length but not zero (*i.e.,* there is no such thing as the "empty" identifier). Moreover, the first character of an identifier cannot be a number; it must be either a letter, or character "_". Finally, identifiers are case sensitive and have a number of characters that are considered to be significant.

> Notice that the number of significant characters in identifiers depends on the compiler implementation. The reference implementation provided by the authors supports up to 127 significant characters.

Identifiers in Swift Fox are only used for naming objects (*e.g.,* applications, networks, sources, configurations, event-conditions) and referring to them. The binding between a name and the corresponding object is performed during the declaration of the identifier and remains the same for the rest of the program. For example, Swift Fox does not allow a user to change a configuration variable and assign a different configuration to it. The *scope* of all identifiers is, therefore, the same and it global (for a particular program). Similarly, the *lifetime* of all identifiers is the same and it is equal to the lifetime of the Swift Fox program.

> Objects in Swift Fox are not created and disposed *dynamically* (or *automatically*) at runtime.

3.2.4 Keywords

Swift Fox uses a set of special-meaning identifiers, namely *keywords*, which cannot be used otherwise. Table 3.1, below, illustrates all the "reserved" words that are used in Swift Fox.

> All keywords have special meaning for the Swift Fox translation procedure and therefore cannot be used as identifiers; one cannot name a configuration or an event-condition as "network", or "goto".

use	application	network
source	configuration	event-condition
from	goto	when
nothing	any	once
and	or	not
	start	

Table 3.1: Swift Fox keywords.

3.2.5 Constants

Currently only four different kinds of *constants* are supported. Temperature, time, integer constants, and string literals. Temperature and time constants are comprised of digit sequences that resemble a decimal value, followed by the corresponding suffix that denotes the scale. The allowed suffixes for temperature are "`C`" and "`F`" and they denote the Celsius and Fahrenheit scale respectively. Similarly, the time-allowed suffixes are "`msec`", "`sec`", "`min`", "`hr`". On the other hand, integer constants are not allowed to have a suffix; they are merely comprised of digit sequences that resemble a decimal value. Finally, Swift Fox also supports string literals, which are essentially sequences of characters.

> Typically, string constants are used in order to denote path elements on the definition of applications, network protocols, and sources. They are semantically compatible with the definition of C strings.

3.3 Syntax Notation

In this section we define the syntactic constructs that are used in Swift Fox language. The *syntactic analysis* is the bulk part of the second phase of the translation procedure [1]. During that phase, a stream of tokens, which is provided by the lexer, is checked for adherence to Swift Fox rules. In other words, the source code is tested for conformity to the syntactic rules that are defined by the formal grammar of the language. The result of this procedure is an *annotated parse tree* of the program along with a special data structure that keeps information about each identifier (*i.e.,* the *symbol table*).

Appendix .3 includes the definition of a Swift Fox parser. The reference implementation of the Swift Fox compiler uses the YACC (Bison) parser-generator [23, 32]. However, this manuscript provides all possible details for implementing a Swift Fox compiler with every available compiler-compiler.

3.3.1 Types

Swift Fox is a *strongly typed* language [6] where type checking occurs only at compile time. The simplistic nature of Swift Fox syntax prevents type intermixing without loss of expressibility. The benefits of this approach are manifold. First, the language does not need to involve additional complexity with runtime checks. Notice that the execution environment of Swift Fox is severely constrained and any runtime type check will not only impact the performance of the system, but also drain the available power supply [40]. Second, it allows strong guarantees about the runtime behavior of the program since there are no explicit or implicit type conversions. Third, it guards against evasions of the type system that can lead to unpredictable behavior.

Swift Fox is also a *static typed* language. This allows the optimal selection of the storage needed for the various Swift Fox objects. This is, again, of paramount importance for the over constrained application domain of Swift Fox [40]. The *primitive* types that define the building blocks of the language are the following:

- `application`. This primitive type is used in order to define application components (*e.g.,* Blink, Collector, or Picture) as introduced in [39]
- `network`. Similarly to the previous type, network is used in order to define network components (*e.g.,* CTP or P2P-MultiHop)
- `source`. Defines event sources (*e.g.,* temperature readings, timeouts)
- `configuration`. Configuration defines a specific binding among instances of the different classes of components that are provided by Fennec Fox [39])
- `event-condition`. Event-condition associates events with specific conditions (*e.g.,* temperature $>$ 90 F)

Swift Fox does not have composite types nor does it allow the programmer to define and use its own types. Moreover, it provides a set of predefined values for application, network, and source that correspond to the basic applications, network protocols, and sensors that are always available to a system. This feature is essentially similar to the notion of C libraries that provide additional functionality in user programs when needed.

3.3.2 Operators

The operators used in Swift Fox can be classified as *relational*, *logical*, or *enumerative*. Relational operators are used in order to define an event-condition and they evaluate to `true` or `false`. Infix operators `<` (less), `>` (greater), `<=` (less or equal), `>=` (greater or equal), and `=` (equal) yield 0, if the relation that corresponds to a specific condition is false, and 1 if it is true. The infix logical operators `and` and `or` are used in order to combine two or more event-conditions conjunctively or disjunctively, whereas the infix *comma* (`,`) operator is used among two or more configurations in order to create a set from them. The `not` operator works as negation operator and takes one argument of type `event`. The return value of the `not` operator is an event which becomes true when the argument event is false. The `and` operator takes two arguments of type `event` and returns an event which is true when both argument events are true. The `or` operator takes two arguments of type `event` and returns two events that are true if either one of the argument events is true. To improve readability, in every statement we allow to use maximum one `or` operator. The number of `not` and `and` operators is unlimited. Finally, the `not` operator has higher precedence than the operator `and`, and operator `or` has the lowest precedence.

3.3.3 Separators

Separators are special characters that are used in Swift Fox in order to segregate various statements. Currently, Swift Fox uses only one separator: the `newline` (`LF`). Hence, library declaration statements, configuration and event-condition declaration statements, and policy and initial-configuration statements (Section 3.3.4 and Appendix .3) are separated from each other using newline characters.

3.3.4 Statements

Swift Fox programs consist of sequences of *statements*. There are three different types of *statements*. The first type includes *declaration statements* for applications, network protocols, and event sources, as well as configurations and event-conditions (see Section 3.3.1). The lines 154 – 231 and 394 – 453 in Appendix .3 illustrate the syntax for that particular type. The second type of statements is about *policy definitions* (see lines 252 – 380 in Appendix .3). The bulk part of a Swift Fox program is typically made of such statements, since they capture the reconfiguration strategy of the system. Finally, the last type of statement (actually it is only one statement) is about declaring the initial configuration of the system; similar to the main entry point of a C program, there is an initial configuration for a Swift Fox policy.

> Appendix .3 illustrates the syntax of the Swift Fox language through a YACC grammar-definition. Without loss of generality, this should give an indication of the syntactic rules of the language so as to implement those in other compiler-compilers.

Notice that the order of the previous statements is important and fixed. A valid Swift Fox program cannot intermix different types of statements and the first statements of a valid program should be application, network, or source declarations followed by configuration declarations. The application, network, and source declarations are optional (*i.e.,* they can be omitted) and are typically provided in the form of a "library". Subsequently, there might be some event-condition declaration statements followed by policy statements. Again, the declaration of event-conditions and the definition of policies is not mandatory. The final statement is always the initial configuration statement and it is necessary in every valid Swift Fox program.

Chapter 4

Project Plan

4.1 Project Plan (*Marcin*)

4.1.1 Development Process

To develop the language and its translator we were meeting at least once a week and were using software / project management tools to enhance the development process. As a group we have met 13 times, and our team has also met 7 times with the group's mentor, Prof. Alfred Aho. During the meetings we were brainstorming how Swift Fox should be designed and implemented. Moreover, we were ensuring that we are on the right track to meet the milestone submissions, and if necessary we were rescheduling the roles and assignments to meet the deadlines. The rule of thumb which was kept by the project manager was to have draft / alpha version of a document, or a code, to be ready a week before the milestone date set by the instructor. Although this last week before submission was expected to be used for polishing the final work, corrections, and debugging, it was also used to finish the work which was missed by other members of the group.

To improve work assignment, keep work schedule on time, and improve software engineering practice, a dedicated server with a web-based project management and bug-tracking tool was setup on the Network Security Research Lab (NSL) [42]. Vasileios has dedicated a server and installed Trac [37]. Trac is a project management tool with wiki interface, ticket scheduling system, and source code repository. All group members received accounts on the server and have access to the Trac management portal through the address `http://nslvm2.cs.columbia.edu/trac/wiki`. After every meeting, job tickets were assigned and all members could track the progress of the team work. Unfortunately, after a month the excitement from using Trac has fallen down, and some members stopped using the tool.

4.1.2 Roles and Responsibilities

The group was started with three members: Marcin, Vasileios, and Linh. After Yiwei joined us, we assigned the roles, which in practice were often reassigned to meet various milestones.

- Project manager - Marcin (Vasileios was the second project manager)
  Marcin kept the project schedule, hold weekly meetings with the entire team, maintained the project log, and made sure the project deliverables get done on time
- Language and tools guru - Vasileios
  Vasileios defined the baseline process to track language changes and

maintained the intellectual integrity of the language. Moreover he was teaching the team how to utilize the various tools used to build the compiler

- System architect - Marcin
  Based on Fennec Fox platform, Marcin defined the compiler architecture, modules, and interfaces

- System integrator - Yiwei
  For system integration, Yiwei designed the syntax tree traversal subsystem, which was used in the semantic check and code generation steps

- Tester and validator - Linh (Vasileios was actually designing and coding the test suit)
  Linh was attempting to create the testing suit

4.1.3 Implementation

Swift Fox documentation was implemented using LaTeXand the code was implemented in C. Both documentation and code were stored in the svn [7] repository and managed using Trac [37] project management tool.

1. Swift Fox Whitepaper
   - Introduction and Problem Statement
     Marcin and Vasileios
   - Buzzwords: simple, declarative, domain specific, lightweight
     Linh
   - Buzzwords: network-savvy, platform independent, event-driven
     Marcin
   - Buzzwords: robust, secure, policy-oriented
     Vasileios
   - Draft Review
     Marcin, Vasileios, Yiwei
2. Swift Fox Tutorial & Manual
   - Tutorial
     Marcin

- Manual
  Vasileios
- Draft Review
  Marcin and Vasileios

3. Class Presentation

   - Graphics and Text
     Marcin and Vasileios
   - Implementing Presentation in LaTeX
     Vasileios
   - Practice Presentation Meeting
     Marcin, Vasileios, Linh, Yiwei

4. Swift Fox Compiler Code

   - Lexer
     Marcin and Vasileios
   - Parser
     Marcin
   - Code Generation
     Marcin
   - Test Suit
     Vasileios
   - Semantic Check
     Vasileios
   - Tree Traversal
     Yiwei

5. Final Report

   - Chapter 4
     Marcin
   - Chapter 5
     Vasileios
   - Chapter 6
     Marcin
   - Chapter 7
     Yiwei

- Chapter 8
  Vasileios
- Chapter 9
  Marcin, Vasileios, Linh, Yiwei
- Draft Review
  Marcin, Vasileios, Linh

4.1.4 Timeline and Project Log

The timeline of the project is listed below.

- January 28 - team formed by Marcin and Vasileios
- February 9 - roles assigned for Marcin, Vasileios, and Linh (one member is missing)
- February 22 - first meeting with all members present (Yiwei joins the team)
- February 24 - Swift Fox white-paper was submitted
- March 12 - grammar, tokens, and types are finished
- March 24 - Swift Fox tutorial and Swift Fox manual are submitted
- April 12 - presentation of a simple swift fox example; blink demo works
- April 25 - practice for the class presentation
- April 26 - project presentations in class
- May 6 - Swift Fox compiles more sophisticated programs than blink
- May 9 - report and code are submitted
- May 10 - demo presentation

A more detailed, day-by-day timeline can be found at `http://nslvm2.cs.columbia.edu/trac/timeline` The day-by-day timeline generated by Track shows every member contributions to the project, including coding and writing text. In the Trac we were also storing the meeting log. The meeting log showing all meetings with each meeting's agenda can be found at `http://nslvm2.cs.columbia.edu/trac/wiki/MeetingsLog`. Overall, our group had 20 meetings, out of which 7 meetings were in the presence of the

Prof. Alfred Aho. Marcin and Vasilis were present on all meetings, and Linh and Yiwei both missed 6 meetings. Finally, the Trac system was generating reports from all jobs done based on the assignment tickets assigned after each meeting. Detailed tickets assignment log can be found at `http://nslvm2.cs.columbia.edu/trac/report/6`.

Chapter 5

Language Evolution

5.1 Language Evolution (*Vasileios*)

Swift Fox, like most programming languages, was created from the necessity to meet a particular need. However, identifying a problem, suggesting a prominent solution, and finally implementing the prescribed logic is not always a straightforward procedure. Oftentimes, various parameters affect not only the design and the characteristics of the final outcome, but also result in the reconsideration of the original proposal. In this section, we shall see how we managed to maintain compatibility with our original plan, as our language was evolved and expanded throughout its development, and how we kept track of that process.

5.1.1 Evolution Throughout Development

The design of a programming language pretty much touches upon computer architecture, algorithms, language theory, and software engineering. Since Swift Fox is a special-purpose language, it also incorporates a considerable amount of OS-based concepts, networking and communications issues, as well as low-level characteristics regarding sensors (*e.g.,* dealing with sensor readings). Such parameters not only had a major impact on the design of Swift Fox, but oftentimes they shaped completely the systemic approach that was followed on our implementation.

Simplicity has been a first class citizen in Swift Fox and one of the most pervasive characteristics of the language. After all, one of the goals of our language was to be used by non-expert users of the field for allowing them to concentrate on a particular problem and "avoid plumbing" (*e.g.,* dealing with platform peculiarities, low-level details, and various system, but problem-irrelevant, issues). Thus, the basic constructs of the language (*i.e.,* configurations, event-conditions, and policies) were defined in a rudimentary manner just for enabling our language to capture the essence of the corresponding concepts from the sensor domain (see Section 3.1). As a result, they did not evolve considerably throughout the development of the language. The only considerations were regarding the readability of the language and the structure of the analogous statements. A typical example of such a dispute was whether the definition of a new `configuration` type (see Section 3.3.1) should include a comma between the application and the network components, or not. Another example of that nature was the selection of various suffixes in the constant definitions (*e.g.,* `msec`, `min`, `hr`). In any such case, the underlying structure of our translator was not affected significantly. That is, we isolated the corresponding changes on the lexical/syntax

analysis part and in particular on the definitions of the related constructs.

The most important changes on the specification of the language were imposed by the run-time environment and the underlying platform that Swift Fox utilizes, namely Fennec Fox. Fennec Fox evolved and extended along with Swift Fox. Therefore, oftentimes we found necessary to redefine various aspects of the original proposal due to the nature and the properties of Fennec Fox. Briefly speaking, the deviations of our language from the original specifications were mostly due to the limitations, or specific idiosyncrasies, of the target platform. In particular, we re-established the definitions of `application`, `network`, and `source` types since the concurrent development of Fennex Fox regularly affected the instantiation of those types (*i.e.,* how we can refer to an application, or a particular network protocol).

Overall, we tried to stay in line with the original proposal as much as possible. Surprisingly, the complexity of a programming language design made this task non-trivial even for a language like Swift Fox with a tiny set of basic types and allowable constructs. In order to maintain the attributes of the original proposal, we followed two approaches: *i.* simplicity on the language definition, and *ii.* layered design. As far as the former is concerned, we made the language types and constructs relatively simple and manageable. This reduced the degrees of freedom regarding their specification and kept the complexity of having them in agreement with the original specification relatively low. The latter approach enabled us to isolate changes and make local extensions. All the adjustments that we made were in most cases handled transparently from the irrelevant parts of the translator.

5.1.2 Consistency

The Swift Fox translator was under constant development throughout the duration of this project. Therefore, keeping the team up-to-date with the changes during the whole implementation phase was of paramount importance for various reasons:

- maintaining the *logical consistency* and *semantical correctness* of the translator
- since more than one person might have been working concurrently on implementing the various features of the compiler, we had to ensure that all developers were aware of the latest development status for not implementing the same parts, or even worse, the same parts in a different manner

- tracking the progress of the implemented features, the individuals contributions, and most importantly keeping the the roles of the team separated and clear

In order to achieve this, we relied upon our web-based project management and bug-tracking tools, namely Subversion [7] and Trac [37].

We adopted a distributed development model with a centralized source code repository hosted here at Columbia University (courtesy of the Network Security Laboratory (NSL) [42]). In particular, we merged the code repository with the management system in order to keep track of the development process and deal with the changes in the code in a *differential* manner through the web. Hence, all the members were informed about the progress on the implementation and the latest changes by visiting the appropriate URL on the project web site (*i.e.,* `http://nslvm2.cs.columbia.edu/trac/timeline`). Moreover, by exploiting the reporting and "ticketing" capabilities of Trac, the team could easily observe the status of the various milestones, bugs, and features. Finally, on the long run our group also held weekly meetings (see Section 4.1). Among others, we were also discussing the latest features that have been added to the translator, the current status on the implementation, and most importantly various deviations from the prescribed implementation plan. For the needs of Swift Fox project, it turned out that our approach was more than sufficient.

Chapter 6

Translator Architecture

6.1 Translator Architecture (*Marcin*)

6.1.1 Architectural Block Diagram of the Translator

The SwiftFox compiler architecture consists of four phases, namely lexical analysis, syntax analysis, intermediate code generator, and target nesC code generator. Figure 6.1 depicts all the phases of the SwiftFox compiler.

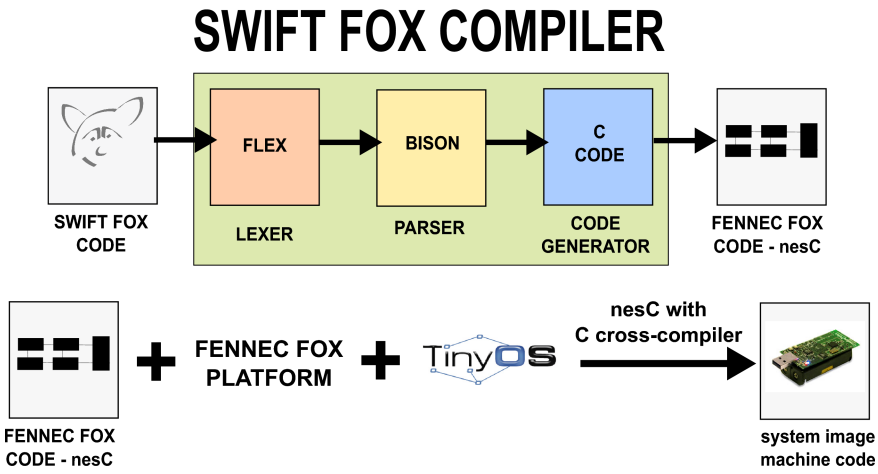

Figure 6.1: Swift Fox compiler architecture.

6.1.2 Modules and Interfaces

First, the lexical analyzer receives a stream of characters from two files:*i.* a library file, and *ii.* a program code file. The library file specifies what libraries are accessible to the program code, and where they are located. The program code file contains code that implements configurations, events, and policies. The lexical analyzer was implemented using flex [33] and saved in `sf.l` file. Besides tokenizing the input text, the lexical analyzer also implements debug code used to verify correctness of the lexer functionality, and also keeps track of the text that is tokenized to be able to return helpful error messages.

Second, the syntax analyzer receives the stream of tokens from the lexical analyzer. All terminals and non-terminals are defined in the syntax analyzer which was implemented using bison [32] and saved in `sf.y` file. As the syntax analyzer is parsing the input, it uses synthesized attributes to create annotated parse tree. The generated parse tree fills out all necessary

information for the further semantic check and code generation processes. These information are related to types, values, and links to the libraries that provide the requested functionality. The parser also tracks how much memory is necessary to be reserved for the code generated by the compiler. The memory management done by the parser is critical to insure that the final machine code will required minimum of ROM space, particularly because it is to be deployed on small embedded devices.

Third, the parse tree generated by the syntax analyzer is traversed twice for semantic checking and for code generation. The semantic check was not fully implemented as we expected. During the code generation the parse tree is traversed and code is generated and stored in files which are part of the Fennec Fox platform. The Swift Fox code generation creates 10 files which are part of the Fennec Fox and contain code written in nesC. The code generation functionality was saved in `code_gen.c` and `code_gen.h` files.

6.1.3 Members Impact on the Architecture

1. **Lexer**: Marcin and Vasileios
2. **Parser**: Marcin
3. **Tree Traversal**: Yiwei (adapted to the parse tree by Marcin)
4. **Semantic Check**: Vasileios (after it was attempted by Linh)
5. **Code Generation**: Marcin
6. **Test Suite**: Vasileios (after it was attempted by Linh)

Chapter 7

Development Environment

7.1 Development Environment (*Yiwei*)

7.1.1 Platforms

The team used various development environments to code different sections of the compiler. Members used whichever platform they were most comfortable and familiar with. Our code was run and tested on Windows, Mac, Linux, and Unix. The full environment consists of Tiny OS [11] and Fennec Fox installations, among other things as software, and the sensor motes as the most important hardware. However, for an experimenter to try out our compiler the aforementioned software and hardware are not needed; the compiler will produce the output nesC code independently. It just that the experimenter will not be able to see the generated code running. That was also the case for our team since we do not have the capability to equip a full system for each team member. Still we are able to develop the code in individual machines throughout the project.

7.1.2 Programming Tools

In order to develop the Swift Fox language, Flex and Bison were used to create the lexer and parser. Any other code in the lexer and parser were written in C. Due to the original Fennec Fox that Swift Fox is intended to be built on top of, our target language is nesC [14]. Since nesC is a close variant of C, we chose to use C code to do code generation into nesC, as well as error checking. Code for defining the tree nodes, traversal, code generation and error generation have their individual `.h` and `.c` files and are linked together using the corresponding makefiles. The Makefile compiles all necessary `.c` files and creates a Swift Fox program. In addition, simple shell scripts were created to compile, run and clean the `.o` files at the end of tests. The majority of the functions required the standard C library as well as the standard I/O library. For code editing, individual members used Eclipse, Notepad++, or Vi.

7.1.3 Management Tools

To synchronize code in a distributed developing environment, the team used Subversion (SVN) as a repository for code (and also documentation). In combination with SVN, Trac was used to keep track of deadlines, documentation, and tickets. Other alternatives such as DropBox, Google Docs were considered but since there is a strong likelihood of future code development of Swift Fox, code needed to be public and a website needed to be created for

both the posting of code and extensive documentation for future developers or users. In the end, a web server running Trac was set up

Trac is a free open source web-development tool to assist with project management. It allows maintaining all aspects of the project in one place, like:

- wiki. Common definition, role assignment, meeting minutes, tools used, and so forth
- timelines. Keeping track of who did what and when, ticketing assignment and progress watching of bite-sized tasks
- code plus document repository. Using track we are able to look at the code resided in SVN repository right from browser window, which is very convenient.

Nevertheless, there were some synchronization problems with SVN. While Trac reduced some synchronization issues, email alerts as to which member was currently editing which part of code were still frequent. Also, certain files and folder needed to be copied and edited locally in order to prevent corruption of the repository. By the end of the project, most of these issues were eliminated.

Chapter 8

Appendix

.1 Source Code Listing

The complete listing of the Swift Fox source code is publicly available at `http://nslvm2.cs.columbia.edu/trac/browser` along with its revision history and identification information regarding its authors.

.2 Swift Fox Lex Definition

```
%{
/* Swift Fox Compiler v0.3
 * Authors: Marcin Szczodrak and Vasileios P. Kemerlis
 * Date: May 9, 2010
 */

#include <stdio.h>
#ifdef __DEBUG__          /* link with the testing suite */
#include <unistd.h>
#include "common.h"
#else                     /* link with YACC/Bison */
#include <fcntl.h>
#include <ctype.h>
#include <stdlib.h>
#include "y.tab.h"
#include "sf.h"

#define YY_NO_INPUT
#endif

%}

%option nounput

delim             [ \t\v\f\r]
whitespace        {delim}+
letter            [A-Za-z]
digit             [0-9]
number            [1-9]{digit}*
newline           \n.*
identifier        ({letter}|_)({letter}|{digit}|_)*
numtype           (C|F|msec|sec|min|hr)
constant          {number}{numtype}?
path              (http:\/\/|\.|\/|\$\(FENNEC_FOX_LIB\))({letter
    }|{digit}|\.|\/|_|-|~)*
comment           #.*

%%
```

```
{newline}       {
#ifndef __DEBUG__
                (void)memset(linebuf, 0, BUF_SZ);
                strncpy(linebuf, yytext + 1, BUF_SZ - 1)
                    ; lineno++;
                tokenpos = 0; yyless(1);
#else
                yyless(1);
#endif
                return LF;
                    }
{whitespace}    {
#ifndef __DEBUG__
                tokenpos += yyleng;
#endif
                                                    }
{comment}       {
#ifndef __DEBUG__
                tokenpos += yyleng;
#endif
                                                    }
any             {
#ifndef __DEBUG__
                tokenpos += yyleng;
                yylval.symp = symlook(yytext);
#endif
                return ANY;                         }
configuration   {
#ifndef __DEBUG__
                tokenpos += yyleng;
#endif
                return CONFIGURATION;               }
,               {
#ifndef __DEBUG__
                tokenpos += yyleng;
#endif
                return COMMA;                       }
nothing         {
#ifndef __DEBUG__
                tokenpos += yyleng;
                yylval.symp = symlook(yytext);
#endif
                return NOTHING;                     }
event-condition {
#ifndef __DEBUG__
                tokenpos += yyleng;
#endif
                return EVENT_CONDITION;             }
```

```
from            {
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
                        return FROM;                    }
goto            {
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
                        return GOTO;                    }
start           {
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
                        return START;                   }
use             {
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
                        return  USE;                    }
application     {
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
                        return APPLICATION;             }
network         {
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
                        return NETWORK;                 }
qoi         {
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
                        return QOI;                 }
mac         {
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
                        return MAC;                 }
radio           {
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
                        return RADIO;                   }
address         {
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
```

```
                        return ADDRESS;                 }
source          {
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
                        return SOURCE;                  }
once            {
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
                        return ONCE;                    }
when            {
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
                        return WHEN;                    }
and             {
#ifndef __DEBUG__
                        tokenpos += yyleng;
                        yylval.ival = AND;
#endif
                        return AND;                     }
or              {
#ifndef __DEBUG__
                        tokenpos += yyleng;
                        yylval.ival = OR;
#endif
                        return OR;                      }
not             {
#ifndef __DEBUG__
                        tokenpos += yyleng;
                        yylval.ival = NOT;
#endif
                        return NOT;                     }
"<"             {
#ifndef __DEBUG__
                        tokenpos += yyleng;
                        yylval.ival = LT;
#endif
                        return RELOP;                   }
">"             {
#ifndef __DEBUG__
                        tokenpos += yyleng;
                        yylval.ival = GT;
#endif
                        return RELOP;                   }
"<="            {
#ifndef __DEBUG__
                        tokenpos += yyleng;
```

```
                        yylval.ival = LE;
#endif
                        return RELOP;                   }
">="            {
#ifndef __DEBUG__
                        tokenpos += yyleng;
                        yylval.ival = GE;
#endif
                        return RELOP;                   }
"<>"            {
#ifndef __DEBUG__
                        tokenpos += yyleng;
                        yylval.ival = NE;
#endif
                        return RELOP;                   }
"="             {
#ifndef __DEBUG__
                        tokenpos += yyleng;
                        yylval.ival = EQ;
#endif
                        return RELOP;                   }
\{              {
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
                        return OPEN_BRACE;              }
\}              {
#ifndef __DEBUG__
                        tokenpos += yyleng;
#endif
                        return CLOSE_BRACE;             }
{identifier}    {
#ifndef __DEBUG__
                        tokenpos += yyleng;
                        yylval.symp = symlook(yytext);
#endif
                        return IDENTIFIER;              }
{path}          {
#ifndef __DEBUG__
                        tokenpos += yyleng;
                        yylval.libp = liblook(yytext);
#endif
                        return PATH;                    }
{constant}      {
#ifndef __DEBUG__
                        tokenpos += yyleng;
                        yylval.symp = symlook(yytext);
#endif
                        return CONSTANT;                }
```

```
virtual-network {
                        return VIRTUAL_NETWORK;          }

%%

#ifdef __DEBUG__        /* link with the testing suite */
int
main(int argc, char **argv) {

        int tok;        /* token from the scanner */

        /* try to open the first argument for input */
        if ((argc != 1) && ((yyin = fopen(argv[1], "r")) == NULL
            ))
                /* failed */
                yyin = stdin;   /* read from stdin */

        /* call the scanner repetitively */
        while ((tok = yylex()) != 0)
                /* print the identified token */
                (void)fprintf(stdout, "%d ", tok);
        /* EOL */
        (void)fprintf(stdout, "\n");

        /* cleanup */
        (void)fclose(yyin);

        /* finish */
        return EXIT_SUCCESS;
}
#endif
```

../../src/sf/sf.l

.3 Swift Fox YACC Definition

```
%{
/* Swift Fox Compiler v0.3
 * Authors: Marcin Szczodrak and Vasileios P. Kemerlis
 * Date: May 9, 2010
 */

#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <string.h>
```

```
#include <sys/stat.h>
#include <ctype.h>
#include "sf.h"
#include "traverse.h"

char *relopToLetter(int i);
int editConst(struct symtab *entry);
int negateOperator(int i);
void initialize(void);
void gc(void);
void checkForRemotePath(struct libtab*);

int conf_counter        = 0;
int event_counter       = 1;
int policy_counter      = 0;
int virtual_counter     = 0;
int app_id_counter      = 0;
int net_id_counter      = 0;
int qoi_id_counter      = 0;
int mac_id_counter      = 0;
int radio_id_counter    = 0;
int event_id_counter    = 0;
int address_id_counter  = 0;
FILE *fcode             = NULL;

struct eventnodes *last_evens = NULL;

%}

%union {
        struct symtab           *symp;
        struct libtab           *libp;
        int                     ival;
        char                    *str;
        double                  dval;
        struct confnode         *confp;
        struct confnodes        *confsp;
        struct eventnode        *evep;
        struct eventnodes       *evesp;
        struct policy           *pol;
        struct policies         *pols;
        struct initnode         *initp;
        struct program          *prog;
}

%token CONFIGURATION COMMA EVENT_CONDITION
%token FROM GOTO START USE WHEN
%token APPLICATION NETWORK QOI MAC RADIO ADDRESS
%token SOURCE LF VIRTUAL_NETWORK
```

```
%token LT GT LE GE NE EQ
%token OPEN_BRACE CLOSE_BRACE ONCE

%token <symp>    CONSTANT
%token <symp>    IDENTIFIER
%token <libp>    PATH
%token <ival>    RELOP
%token <ival>    AND
%token <ival>    OR
%token <ival>    NOT
%token <symp>    NOTHING
%token <symp>    ANY
%type <symp>     conf_param
%type <symp>     from_configurations
%type <confp>    configuration
%type <confsp>   configurations
%type <confsp>   defined_configurations
%type <evep>     event_condition
%type <evesp>    defined_events
%type <pol>      policy;
%type <pols>     policies;
%type <initp>    initial_configuration
%type <prog>     program
%type <str>      type
%type <ival>     when_events
%type <ival>     one_event
%type <ival>     virtual_network
%type <ival>     virtual_networks
%type <symp>     configuration_ids

%%

swiftfox: library program
                {
                        /* verbose; to be removed */
                        /* printTable(); */

                        /* root node for the constructed AST */
                        struct program *p = $2;

                        /* traverse the AST for semantic
                            checking */
                        traverse_program($2,
                                         TREE_CHECK_SEMANTIC,
                                         policy_counter);

                        /* traverse the AST for code generation
                            */
                        traverse_program($2,
```

```
                                                TREE_GENERATE_CODE,
                                                policy_counter);
                }
        ;

program: defined_configurations defined_events policies
    virtual_networks initial_configuration
                {
                        /* root node */
                        $$              = calloc(1, sizeof(
                            program));

                        /* link the node appropriately */
                        $1->parent      = NULL;
                        if ($2 != NULL )
                                $2->parent = NULL;
                        if ($3 != NULL )
                                $3->parent = NULL;

                        /* init */
                        $$->defcon      = $1;
                        $$->defeve      = $2;
                        $$->defpol      = $3;
                        $$->init        = $5;
                }
        ;

defined_configurations: configurations configuration
                {
                        /* configurations set */
                        $$              = calloc(1, sizeof(
                            confnodes));

                        /* link the child nodes */
                        if ($1 != NULL)
                                $1->parent = $$;
                        $2->parent      = $$;

                        $$->confs       = $1;
                        $$->conf        = $2;
                }
        ;

configurations: configurations configuration
                {
                        /* configurations set */
                        $$              = calloc(1, sizeof(
                            confnodes));
```

```
                        /* link the child nodes */
153                     if ($1 != NULL)
                                $1->parent = $$;
                        $2->parent      = $$;

                        $$->confs       = $1;
158                     $$->conf        = $2;
                }
        |
                {
                        $$ = NULL;
163             }
        ;

configuration: CONFIGURATION IDENTIFIER OPEN_BRACE conf_param
    conf_param conf_param conf_param conf_param conf_param
    conf_param CLOSE_BRACE newlines
                {
168                     /* configuration node */
                        $$              = calloc(1, sizeof(
                            confnode));

                        /* init */
                        $2->type        = "configuration_id";
173                     $2->value       = conf_counter;
                        $$->id          = $2;
                        $$->counter     = conf_counter;

                        /* link child nodes */
178                     $$->app         = $4;
                        $$->net         = $5;
                        $$->net_addr    = $6;
                        $$->qoi         = $7;
                        $$->mac         = $8;
183                     $$->mac_addr    = $9;
                        $$->radio       = $10;

                        ++conf_counter;
                }
188     ;

conf_param: IDENTIFIER
                {
                        $$ = $1;
193
                }
        |       NOTHING
                {
                        $$ = $1;
```

```
198                 }
        ;


defined_events: defined_events event_condition
203                 {
                            /* event-conditions set */
                            $$              = calloc(1, sizeof(
                                eventnodes));

                            /* link child nodes */
208                         if ($1 != NULL) {
                                    $1->parent = $$;
                                    last_evens = $$;
                            }
                            $2->parent = $$;
213
                            $$->evens       = $1;
                            $$->even        = $2;
                    }
        |
218                 {
                            $$ = NULL;
                    }
        ;

223 event_condition: EVENT_CONDITION IDENTIFIER OPEN_BRACE
    IDENTIFIER RELOP CONSTANT CLOSE_BRACE newlines
                    {
                            /* event-condition node */
                            $2->value       = event_counter;
                            $2->type        = "event_id";
228                         $2->lib         = $4->lib;

                            /* init */
                            $$ = calloc(1, sizeof(eventnode));
                            $$->id          = $2;
233                         $$->counter     = $2->value;

                            /* link child nodes */
                            $$->src         = $4;
                            $$->cst         = $6;
238                         $$->cst->value  = editConst($6);

                            struct evtab *ev= evlook($2->name);
                            ev->num         = event_counter;
                            ev->op          = $5;
243                         ev->value       = $$->cst->value;
```

```
                        event_counter++;
                }
        ;

policies: policies policy
                {
                        /* policies set */
                        $$ = calloc(1, sizeof(policies));

                        /* link child nodes */
                        if ($1 != NULL)
                                $1->parent = $$;
                        $2->parent = $$;

                        $$->pols        = $1;
                        $$->pol         = $2;
                }
        |
                {
                        $$ = NULL;
                }
        ;

policy: FROM from_configurations GOTO IDENTIFIER WHEN
    when_events newlines
                {
                        /* policy node */
                        $$              = calloc(1, sizeof(
                            policy));

                        /* link child nodes */
                        $$->from        = $2;
                        $$->to          = $4;

                        $$->mask_l      = 0;
                        $$->mask_r      = -1;
                        $$->mask_l      = $6;
                        $$->counter     = policy_counter;

                        ++policy_counter;
                }

        |       FROM from_configurations GOTO IDENTIFIER WHEN
            when_events OR when_events newlines
                {
                        /* policy node */
                        $$              = calloc(1, sizeof(
                            policy));
```

```
                        /* link child nodes */
                        $$->from        = $2;
                        $$->to          = $4;
293
                        $$->mask_l      = 0;
                        $$->mask_r      = 0;
                        $$->mask_l      = $6;
                        $$->mask_r      = $8;
298                     $$->counter     = policy_counter;

                        ++policy_counter;
                        ++policy_counter;
                }
303
        ;

from_configurations: IDENTIFIER
                {
308                     $$ = $1;
                }

        |       ANY
                {
313                     $$ = $1;
                }
        ;

when_events: one_event AND when_events
318             {
                        $$ = $1 + $3;
                }
        |       one_event
                {
323                     $$ = $1;
                }
        ;


328 one_event: IDENTIFIER
                {
                        /* iterator */
                        struct evtab *ep = NULL;
                        /* flag */
333                     int found = 0;

                        /* check for undeclared identifiers */
                        for (ep = evtab; ep < &evtab[NEVS]; ep
                            ++)
```

```
                        if (ep->name && !strcmp(ep->name
                            , $1->name)) {
338                             /* found */
                                found = 1;
                                break;
                        }

343                 /* undeclared event identifier */
                    if (!found)
                            yyerror("undeclared event-
                                condition identifier");

                    $$ = 1 << (($1->value) - 1);
348         }
        | NOT IDENTIFIER
            {
                    /* iterator */
                    struct evtab *ep = NULL;
353                 /* flag */
                    int found = 0;

                    /* check for undeclared identifiers */
                    for (ep = evtab; ep < &evtab[NEVS]; ep
                        ++)
358                         if (ep->name && !strcmp(ep->name
                                , $2->name)) {
                                /* found */
                                found = 1;
                                break;
                        }
363
                    /* undeclared event identifier */
                    if (!found)
                            yyerror("undeclared identifier")
                                ;

368                 /* create new event in symtab */
                    int len = strlen($2->name) + strlen("
                        not_") + 1;
                    char *new = calloc(len, 1);
                    (void)snprintf(new, len, "not_%s", $2->
                        name);
                    struct symtab *sp = symlook(new);
373
                    if (!sp->type) {
                            sp->value = event_counter;
                            sp->type = strdup($2->type);
                            sp->lib = $2->lib;
378
```

```
                                        /*
                                         * make a new event entry by
                                             negating the
                                         * operator
                                         */
383                                     struct evtab *ev_old = evlook($2
                                            ->name);
                                        struct evtab *ev_new = evlook(sp
                                            ->name);

                                        ev_new->num = event_counter;
                                        ev_new->op = negateOperator(
                                            ev_old->op);
388                                     ev_new->value = ev_old->value;

                                        event_counter++;
                                }

393                             $$ = 1 << ((sp->value) - 1);
                        }
                ;


398 initial_configuration: START IDENTIFIER newlines
                        {
                                /* start node */
                                $$ = calloc(1, sizeof(initnode));

403                             /* link child nodes */
                                $$->id = $2;
                                $$->init = symlook($2->name)->value;
                        }
                ;
408

    virtual_networks: virtual_networks virtual_network
                        {
                                $$              = $1;
413                     }
                |
                        {
                                $$ = 0;
                        }
418             ;

    virtual_network: VIRTUAL_NETWORK IDENTIFIER OPEN_BRACE
        configuration_ids CLOSE_BRACE newlines
                        {
                                $$              = 0;
```

```
423                             $2->type        = "virtual_id";
                                $2->value       = virtual_counter;


                                ++virtual_counter;
428                     }
                ;

configuration_ids: configuration_ids IDENTIFIER
                        {
433                             $$ = $1;
                        }

                |
                        {
438                             $$ = NULL;
                        }
                ;

/* Library section */
443

library: newlines definitions
                ;

448 definitions: definitions definition
                |
                ;

definition: USE type IDENTIFIER PATH
453                     {
                                /* iterator */
                                char *p = NULL;

                                /* lookup */
458                             struct symtab *sp = NULL;

                                /* check for library re-declarations */
                                if ((sp = symlook($3->name)) != NULL &&
                                                        sp->type != NULL
                                                                )
463                                     /* failed */
                                        yyerror("redeclaration of
                                            library");

                                /* fix the child properties */
                                $3->type = $2;
468                             $3->lib = $4;
```

```
                /* differentiate based on the definition
                   type */
                if (!strcmp($3->type, "application")) {
                        /* application */
473                     $4->type = TYPE_APPLICATION;
                        $4->used = 0;
                        $4->id = 0;
                }
                if (!strcmp($3->type, "network")) {
478                     /* network */
                        $4->type = TYPE_NETWORK;
                        $4->used = 0;
                        $4->id = 0;
                }
483             if (!strcmp($3->type, "qoi")) {
                        /* qoi */
                        $4->type = TYPE_QOI;
                        $4->used = 0;
                        $4->id = 0;
488             }
                if (!strcmp($3->type, "mac")) {
                        /* mac */
                        $4->type = TYPE_MAC;
                        $4->used = 0;
493                     $4->id = 0;
                }
                if (!strcmp($3->type, "radio")) {
                        /* radio */
                        $4->type = TYPE_RADIO;
498                     $4->used = 0;
                        $4->id = 0;
                }
                if (!strcmp($3->type, "address")) {
                        /* address */
503                     $4->type = TYPE_ADDRESS;
                        $4->used = 0;
                        $4->id = 0;
                }
                if (!strcmp($3->type, "source")) {
508                     /* source */
                        $4->type = TYPE_EVENT;
                        event_id_counter++;
                        $4->used = 0;
                        $4->id = event_id_counter;
513             }

                /* extract the name from the path */
                if ((p = rindex($4->path, '/')) == NULL)
                        $4->name = $4->path;
```

```
518                         else
                                    $4->name = ++p;

                            $4->used = 0;

523                         /* save the name as it was defined */
                            $4->def = $3->name;

                    }
                    newlines
528         ;

    type: APPLICATION { $$ = "application"; }
            | NETWORK { $$ = "network"; }
            | QOI { $$ = "qoi"; }
533         | MAC { $$ = "mac"; }
            | RADIO { $$ = "radio"; }
            | ADDRESS { $$ = "address"; }
            | SOURCE { $$ = "source"; }
            ;
538
    newlines: newlines newline
            |
            ;

543 newline: LF
            ;

    %%

548 char program_file[PATH_SZ];
    char library_file[PATH_SZ];
    int done = 0;

    extern FILE *yyin;
553 extern char *yytext;
    extern int yyleng;

    /* main entry point */
    int
558 start_parser(int argc, char *argv[]) {

            /* check the file extention */
            if (rindex(argv[1], '.') != NULL && !strcmp(rindex(argv
                [1], '.'), ".sfp")) {
                    argv[1][strlen(argv[1])-4] = '\0';
563         }

            /* init */
```

```
        (void)memset(program_file, 0, PATH_SZ);
        (void)memset(library_file, 0, PATH_SZ);
568     (void)snprintf(library_file, PATH_SZ, "%s.sfl", argv[1])
            ;
        (void)snprintf(program_file, PATH_SZ, "%s.sfp", argv[1])
            ;

        /* process libraries */
        lineno = 1;
573     tokenpos = 0;

        /* cleanup */
        (void)atexit(gc);

578     /* open the specific library file */
        yyin = fopen(library_file, "r");

        /* try fennec fox standart libray located at (
            $FENNEC_FOX_LIB)/STD_FENNEC_FOX_LIB */
        if (!yyin) {
583             (void)snprintf(library_file, PATH_SZ, "%s/%s",
                    getenv("FENNEC_FOX_LIB"), STD_FENNEC_FOX_LIB)
                    ;
                yyin = fopen(library_file, "r");
        }

        if (!yyin) {
588             /* failed */
                (void)fprintf(stderr,
                        "%s.sfl: no such file or directory and
                            no standard library\n",
                        argv[1]);
                exit(1);
593     }

        /* main loop */
        do {
                initialize();
598             yyparse();

        } while(!feof(yyin));

        /* byeZzz */
603     return 0;
}

/* error reporting */
yyerror(char *errmsg) {
608
```

```
        /* error in program */
        if (done)
                (void)fprintf(stderr, "\nsfc in program: %s\n",
                    program_file);
        else
                (void)fprintf(stderr, "\nsfc in library: %s\n",
                    library_file);

        /* line */
        (void)fprintf(stderr, "%s at %d at line:\n", errmsg,
            lineno);
        (void)fprintf(stderr, "%s\n", linebuf);
        (void)fprintf(stderr, "%*s\n", 1 + tokenpos - yyleng, "^
            ");

        /* terminate */
        exit(1);
}

/* restart the parser with the program file */
int
yywrap(void) {
        /* finish; done with library and program */
        if (done)
                return 1;
        else {
                /* re-init */
                lineno          = 1;
                tokenpos        = 0;

                /* open the program file */
                yyin            = fopen(program_file, "r");
                if (!yyin) {
                        /* failed */
                        (void)fprintf(stderr,
                                "%s: no such file or directory\n
                                    ",
                                program_file);
                        exit(1);
                }

                /* done with the library */
                done = 1;
        }

        /* default */
        return 0;
}
```

```
/* symbol lookup */
struct symtab *
symlook(char *s) {

        /* iterator */
        struct symtab *sp = NULL;

        /* loop */
        for(sp = symtab; sp < &symtab[NSYMS]; sp++) {
                /* is it already here? */
                if (sp->name && !strcmp(sp->name, s))
                        return sp;

                /* is it free */
                if(!sp->name) {
                        sp->name = strdup(s);
                        return sp;
                }
                /* otherwise continue to next */
        }
        yyerror("symtab is full");
}

/* library lookup */
struct libtab*
liblook(char *l) {

        /* iterator */
        struct libtab *lp = NULL;

        /* loop */
        for(lp = libtab; lp < &libtab[NLIBS]; lp++) {
                /* is it already here? */
                if (lp->path && !strcmp(lp->path, l))
                        yyerror("library already exists");

                /* is it free */
                if(!lp->path) {
                        lp->path = strdup(l);
                        checkForRemotePath(lp);
                        lp->used = 0;
                        return lp;
                }

                /* otherwise continue to next */
        }
        yyerror("libtab is full");
}
```

```
/* event-condition lookup */
struct evtab*
evlook(char *name) {

        /* iterator */
        struct evtab *ev = NULL;

        /* loop */
        for(ev = evtab; ev < &evtab[NEVS]; ev++) {
                /* found */
                if (ev->name && !strcmp(ev->name, name))
                        return ev;

                /* insert */
                if (!ev->name) {
                        ev->name = strdup(name);
                        return ev;
                }
        }

        /* failed */
        yyerror("evtab is full");
}

/* initialize the keywords set */
void
initialize(void) {

        /* keywords set */
        char *keywords[] = {"configuration", "start", "use", "
            application",
                        "network", "source", "event-condition",
                            "from", "goto",
                        "qoi", "mac", "radio", "virtual-network"
                            ,   // new keywords
                        "when", "nothing", "any", "once", "event
                            "};

        /* size of the keywords set */
        int k_num               = sizeof(keywords)/sizeof(char*)
            ;

        /* iterate */
        int i                   = 0;
        struct symtab *sp       = NULL;

        /* init */
        for(i = 0, sp = symtab; i < k_num; i++, sp++) {
                sp->name = keywords[i];
```

```
                sp->value = 0;
                sp->type = "keyword";
        }
}

/* get the negation of an operator */
int
negateOperator(int i) {
        switch(i) {
                case LT:
                        return GE;
                case GT:
                        return LE;
                case LE:
                        return GT;
                case GE:
                        return LT;
                case NE:
                        return EQ;
                case EQ:
                        return NE;
                default:
                        yyerror("unknown RELOP operator");
        }
}

/* parse constants */
int
editConst(struct symtab *entry ) {

        /* get the constant */
        char *sp = entry->name;

        /* extract the integer part */
        int v = atoi(sp);
        for (; isdigit(*sp); sp++);

        /* set the type accordingly */

        /* sec */
        if (!strcmp(sp, "sec")) {
                entry->type = strdup("timer");
                v *= SEC_CONV;
                return v;
        }

        /* min */
        if (!strcmp(sp, "min")) {
                entry->type = strdup("timer");
```

```
                v *= MIN_CONV;
                return v;
        }

        /* hr */
        if (!strcmp(sp, "hr")) {
                entry->type = strdup("timer");
                v = HR_CONV;
                return v;
        }

        /* Celsius */
        if (!strcmp(sp, "C")) {
                entry->type = strdup("temperature");
                return v;
        }

        /* Fahrenheit */
        if (!strcmp(sp, "F")) {
                entry->type = strdup("temperature");
                v = (v - 32) * 5 / 9 ;
                return v;
        }

        /* default */
        entry->type = strdup("number");

        return v;
}

printTable() {

       struct symtab *sp;
        printf("\n");
        for(sp = symtab; sp < &symtab[NSYMS]; sp++) {
                /* is it already here? */
                if (sp->name ) {
                                printf("%s   %s   %d\n", sp->
                                    name, sp->type, sp->value);
                } else {
                        printf("\n\n");
                        break;
                }
        }

        struct evtab *ev;
        for(ev = evtab; ev < &evtab[NEVS]; ev++) {
                if (ev->name) {
```

```
                        printf("%s      %d      %d      %d\n", ev->
                            name, ev->num, ev->op, ev->value);
843                 } else {
                        printf("\n\n");
                        return;
                    }
            }
848 }

    /* garbage collection */
    void
    gc(void) {
853
            /* cleanup */
            (void)close(yyin);
    }

858 /* check if the path is remote */
    void
    checkForRemotePath(struct libtab *lp) {

            /* check if it is http:// */
863         if (!strncmp(lp->path, "http://",7)) {

                    /* make temp dir for all downloaded libraries */
                    mkdir(TEMP_DIR, S_IRWXU);

868                 char *p = NULL;
                    /* extract the name from the path */
                    if ((p = rindex(lp->path, '/')) != NULL) {
                            p++;
                    }
873
                    /* make dir for this library */
                    char *new = malloc(strlen(p)+strlen(TEMP_DIR)+2)
                        ;
                    sprintf(new, "%s/%s", TEMP_DIR, p);
                    mkdir(new, S_IRWXU);
878
                    /* prepare system command */
                    char *command = malloc(strlen(lp->path)+strlen(
                        new)+40);
                    sprintf(command, "wget -q -nd -r --no-parent -A.
                        nc -P %s %s", new, lp->path);

883                 if( system(command) ) {
                            fprintf(stderr, "Swift Fox Error: Could
                                not download files from %s\n", lp->
                                path);
```

```
                        exit(1);
                    }
                    free(lp->path);
888                 lp->path = strdup(new);
                    free(command);
                    free(new);
            }
}
```

../../src/sf/sf.y

.4 Swift Fox Lexer Testing Suite

```
#!/bin/bash
# Swift Fox Compiler v0.1
# Authors: Vasileios P. Kemerlis
# Date: April 20, 2010

# pool with lexer input
POOL_DIR="lex_pool"
# Swift Fox extensions
SF_EXT="*.sfp"
# regression test suite
REG_TEXT="./regtest_lex"
# temporary output
TMP_OUT="/tmp/real_out.sfp"

#verbose
echo "Lexer regression testing"

# test counter
count=0;

# get Swift Fox programs
for PROG in `ls $POOL_DIR/$SF_EXT`; do
        # expected output
        EXP_OUT=$PROG.out

        # call the regression test suite
        $REG_TEXT $PROG > $TMP_OUT

        # verbose
        count=$((++count));
        echo -n "     [$count]"
        echo -n `head -n1 $PROG`;
        echo -n ": ";
```

```
        # check for expected behavior
        diff $EXP_OUT $TMP_OUT >> /dev/null
        if (( $? == 0 )); then
                echo "Success";
        else
                 echo "Failed ($PROG)"
        fi
done
```

../../src/regression/test_lex

.5 Swift Fox Symbol Table and Parse Tree Nodes

```
/* Swift Fox Compiler v0.3
 * Authors: Marcin Szczodrak and Vasileios P. Kemerlis
 * Date: May 27, 2010
 */

#ifndef SF_H
#define SF_H

#define TREE_TRAVERSE           1
#define TREE_CHECK_SEMANTIC     2
#define TREE_GENERATE_CODE      3

#define NSYMS                   256
#define NLIBS                   256
#define NEVS                    256

#define TYPE_APPLICATION        1
#define TYPE_NETWORK            2
#define TYPE_EVENT              3
#define TYPE_QOI                4
#define TYPE_MAC                5
#define TYPE_RADIO              6
#define TYPE_ADDRESS            7

#define PATH_SZ                 256
#define ARGC_MAX                2

#define SEC_CONV                1024
#define MIN_CONV                60 * SEC_CONV
#define HR_CONV                 60 * MIN_CONV

#define BUF_SZ                  1024

#define TEMP_DIR                "/tmp/swift_fox"
```

```
#define STD_FENNEC_FOX_LIB      "support/sfc/fennec.sfl"

int lineno;
char linebuf[BUF_SZ];
int tokenpos;

/* Symbol Table */
struct symtab {
        char            *name;
        int             value;
        char            *type;
        struct libtab   *lib;
} symtab[NSYMS];

struct symtab *symlook(char *);

struct libtab {
        char    *path;
        char    *def;
        char    *name;
        int     used;
        int     type;
        int     id;
} libtab[NLIBS];

struct libtab *liblook(char *);

struct evtab {
        char    *name;
        int     num;
        int     op;
        int     value;
} evtab[NEVS];

struct evtab *evlook(char *);

struct confnode {
        struct confnodes        *parent;
        struct symtab           *id;
        struct symtab           *app;
        struct symtab           *net;
        struct symtab           *net_addr;
        struct symtab           *qoi;
        struct symtab           *mac;
        struct symtab           *mac_addr;
        struct symtab           *radio;
        int                     counter;
} confnode;
```

```
struct confnodes {
        struct confnodes        *parent;
        struct confnodes        *confs;
        struct confnode         *conf;
} confnodes;

struct eventnode {
        struct eventnodes       *parent;
        struct symtab           *id;
        struct symtab           *src;
        int                     op;
        int                     counter;
        struct symtab           *cst;
} eventnode;

struct eventnodes {
        struct eventnodes       *parent;
        struct eventnodes       *evens;
        struct eventnode        *even;
} eventnodes;

struct policy {
        struct policies         *parent;
        struct symtab           *from;
        struct symtab           *to;
        int                     mask_l;
        int                     mask_r;
        int                     counter;
} policy;

struct policies {
        struct policies         *parent;
        struct policies         *pols;
        struct policy           *pol;
} policies;

struct initnode {
        struct symtab   *id;
        int             init;
} initnode;

struct program {
        struct confnodes        *defcon;
        struct eventnodes       *defeve;
        struct policies         *defpol;
        struct initnode         *init;
} program;
```

```
133 #endif
```

../../src/sf/sf.h

.6 Swift Fox Code Generation Functions

```
/* Swift Fox Compiler v0.2
 * Authors: Marcin Szczodrak
 * Date: May 9, 2010
 */

#ifndef __CODE_GEN_H__
#define __CODE_GEN_H__


#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <ctype.h>
#include "sf.h"
#include "traverse.h"

extern char *appC;
extern char *appM;

extern char *netC;
extern char *netM;

extern char *macC;
extern char *macM;

extern char *qoiC;
extern char *qoiM;

extern char *radioC;
extern char *radioM;

extern char *cachesM;

extern char *app_netC;
extern char *app_netM;

extern char *net_macC;
extern char *net_macM;

extern char *mac_radioC;
```

```
42 extern char *mac_radioM;

   extern char *addrC;
   extern char *addrM;

47 extern int app_id_count;
   extern int net_id_count;
   extern int qoi_id_count;
   extern int mac_id_count;
   extern int radio_id_count;
52 extern int addr_id_count;

   extern int missing_policy_mac;
   extern int missing_policy_address;
   extern int missing_policy_radio;
57
   extern char *policy_configuration_mac;
   extern char *policy_configuration_address;
   extern char *policy_configuration_radio;

62 void setFiles();
   void setFennecExtra();

   void initialCodeGeneration(int policy_counter);
   void find_policy_mac_and_radio();
67 void finishCodeGeneration();

   void generateConfiguration(struct confnode* c);

   void generateFrontCaches(int event_counter, int conf_counter,
       int policy_counter);
72 void generateEvent();
   void generateEndCaches();

   void generatePolicy(struct policy* p);
   void generateInitial(struct initnode *i);
77
   void generateApplicationC();
   void generateNetworkC();
   void generateQoIC();
   void generateMacC();
82 void generateRadioC();
   void generateEventC();
   void generateAddressingC();

   void generateApplicationNetworkC();
87 void generateNetworkMacC();
   void generateMacRadioC();
```

```
void generateApplicationP();
void generateNetworkP();
void generateMacP();
void generateQoIP();
void generateRadioP();
void generateEventP();
void generateAddressingP();

void generateApplicationNetworkP();
void generateNetworkMacP();
void generateMacRadioP();

char *relopToLetter(int i);
#endif
```

../../src/sf/code_gen.h

```
/* Swift Fox Compiler v0.3
 * Authors: Marcin Szczodrak
 * Date: May 27, 2010
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "FennecParameters.h"
#include "sf.h"
#include "code_gen.h"
#include "y.tab.h"

char *appC = NULL;
char *netC = NULL;
char *qoiC = NULL;
char *macC = NULL;
char *radioC = NULL;
char *eventC = NULL;
char *app_netC = NULL;
char *net_macC = NULL;
char *mac_radioC = NULL;
char *addrC = NULL;

char *appM = NULL;
char *netM = NULL;
char *qoiM = NULL;
char *macM = NULL;
char *radioM = NULL;
char *eventM = NULL;
char *app_netM = NULL;
char *net_macM = NULL;
```

```
char *mac_radioM = NULL;
char *addrM = NULL;

int app_id_count = 1;
int net_id_count = 1;
int qoi_id_count = 1;
int mac_id_count = 1;
int radio_id_count = 1;
int addr_id_count = 1;

char *cachesM = NULL;

char *fex = NULL;

char *policy_configuration_mac = NULL;
char *policy_configuration_address = NULL;
char *policy_configuration_radio = NULL;

/* flags to see if there are any missing policy configuration
   mac or radio */
int missing_policy_mac = 1;
int missing_policy_address = 1;
int missing_policy_radio = 1;

void setFiles() {

  char *fennec_fox_lib = getenv("FENNEC_FOX_LIB");

  if (fennec_fox_lib == NULL) {
    fprintf(stderr, "\n\nFENNEC_FOX_LIB is not set!\n");
    exit(1);
  }

  char *p_appC = "Application/ApplicationC.nc";
  char *p_appM = "Application/ApplicationP.nc";
  appC = malloc(strlen(p_appC)+strlen(fennec_fox_lib)+2);
  sprintf(appC, "%s/%s", fennec_fox_lib, p_appC);
  appM = malloc(strlen(p_appM)+strlen(fennec_fox_lib)+2);
  sprintf(appM, "%s/%s", fennec_fox_lib, p_appM);

  char *p_netC = "Network/NetworkC.nc";
  char *p_netM = "Network/NetworkP.nc";
  netC = malloc(strlen(p_netC)+strlen(fennec_fox_lib)+2);
  sprintf(netC, "%s/%s", fennec_fox_lib, p_netC);
  netM = malloc(strlen(p_netM)+strlen(fennec_fox_lib)+2);
  sprintf(netM, "%s/%s", fennec_fox_lib, p_netM);

  char *p_qoiC = "QoI/QoIC.nc";
  char *p_qoiM = "QoI/QoIP.nc";
```

```
    qoiC = malloc(strlen(p_qoiC)+strlen(fennec_fox_lib)+2);
82  sprintf(qoiC, "%s/%s", fennec_fox_lib, p_qoiC);
    qoiM = malloc(strlen(p_qoiM)+strlen(fennec_fox_lib)+2);
    sprintf(qoiM, "%s/%s", fennec_fox_lib, p_qoiM);

    char *p_macC = "Mac/MacC.nc";
87  char *p_macM = "Mac/MacP.nc";
    macC = malloc(strlen(p_macC)+strlen(fennec_fox_lib)+2);
    sprintf(macC, "%s/%s", fennec_fox_lib, p_macC);
    macM = malloc(strlen(p_macM)+strlen(fennec_fox_lib)+2);
    sprintf(macM, "%s/%s", fennec_fox_lib, p_macM);
92
    char *p_radioC = "Radio/RadioC.nc";
    char *p_radioM = "Radio/RadioP.nc";
    radioC = malloc(strlen(p_radioC)+strlen(fennec_fox_lib)+2);
    sprintf(radioC, "%s/%s", fennec_fox_lib, p_radioC);
97  radioM = malloc(strlen(p_radioM)+strlen(fennec_fox_lib)+2);
    sprintf(radioM, "%s/%s", fennec_fox_lib, p_radioM);

    char *p_addrC = "Addressing/AddressingC.nc";
    char *p_addrM = "Addressing/AddressingP.nc";
102 addrC = malloc(strlen(p_addrC)+strlen(fennec_fox_lib)+2);
    sprintf(addrC, "%s/%s", fennec_fox_lib, p_addrC);
    addrM = malloc(strlen(p_addrM)+strlen(fennec_fox_lib)+2);
    sprintf(addrM, "%s/%s", fennec_fox_lib, p_addrM);

107 char *p_eventC = "Events/EventsC.nc";
    char *p_eventM = "Events/EventsP.nc";
    eventC = malloc(strlen(p_eventC)+strlen(fennec_fox_lib)+2);
    sprintf(eventC, "%s/%s", fennec_fox_lib, p_eventC);
    eventM = malloc(strlen(p_eventM)+strlen(fennec_fox_lib)+2);
112 sprintf(eventM, "%s/%s", fennec_fox_lib, p_eventM);

    char *p_cachesM = "Fennec/CachesP.nc";
    cachesM = malloc(strlen(p_cachesM)+strlen(fennec_fox_lib)+2);
    sprintf(cachesM, "%s/%s", fennec_fox_lib, p_cachesM);
117
    char *p_fex = "support/sfc/fennec.extra";
    fex = malloc(strlen(p_fex)+strlen(fennec_fox_lib)+2);
    sprintf(fex, "%s/%s", fennec_fox_lib, p_fex);

122 char *p_app_netC = "Fennec/ApplicationNetworkC.nc";
    char *p_app_netM = "Fennec/ApplicationNetworkP.nc";
    app_netC = malloc(strlen(p_app_netC)+strlen(fennec_fox_lib)+2)
        ;
    sprintf(app_netC, "%s/%s", fennec_fox_lib, p_app_netC);
    app_netM = malloc(strlen(p_app_netM)+strlen(fennec_fox_lib)+2)
        ;
127 sprintf(app_netM, "%s/%s", fennec_fox_lib, p_app_netM);
```

```
    char *p_net_macC = "Fennec/NetworkMacC.nc";
    char *p_net_macM = "Fennec/NetworkMacP.nc";
    net_macC = malloc(strlen(p_net_macC)+strlen(fennec_fox_lib)+2)
        ;
132 sprintf(net_macC, "%s/%s", fennec_fox_lib, p_net_macC);
    net_macM = malloc(strlen(p_net_macM)+strlen(fennec_fox_lib)+2)
        ;
    sprintf(net_macM, "%s/%s", fennec_fox_lib, p_net_macM);

    char *p_mac_radioC = "Fennec/MacRadioC.nc";
137 char *p_mac_radioM = "Fennec/MacRadioP.nc";
    mac_radioC = malloc(strlen(p_mac_radioC)+strlen(fennec_fox_lib
        )+2);
    sprintf(mac_radioC, "%s/%s", fennec_fox_lib, p_mac_radioC);
    mac_radioM = malloc(strlen(p_mac_radioM)+strlen(fennec_fox_lib
        )+2);
    sprintf(mac_radioM, "%s/%s", fennec_fox_lib, p_mac_radioM);
142
}

void setFennecExtra() {

147 FILE *fp_fe = fopen(fex, "w");

    if (fp_fe == NULL) {
      fprintf(stderr, "You do not have a permission to write into
          file: %s\n", fex);
      exit(1);
152 }

    fprintf(fp_fe, "CFLAGS+=-I$(FENNEC_FOX_LIB)/interfaces\n");
    fprintf(fp_fe, "CFLAGS+=-I$(FENNEC_FOX_LIB)/libs\n");
    fprintf(fp_fe, "CFLAGS+=-I$(FENNEC_FOX_LIB)/chips\n");
157 fprintf(fp_fe, "CFLAGS+=-I$(FENNEC_FOX_LIB)/system\n");
    fprintf(fp_fe, "CFLAGS+=-I$(FENNEC_FOX_LIB)/Fennec\n");
    fprintf(fp_fe, "CFLAGS+=-I$(FENNEC_FOX_LIB)/Policy\n");
    fprintf(fp_fe, "CFLAGS+=-I$(FENNEC_FOX_LIB)/PowerMgmt\n");
    fprintf(fp_fe, "CFLAGS+=-I$(FENNEC_FOX_LIB)/Network\n");
162 fprintf(fp_fe, "CFLAGS+=-I$(FENNEC_FOX_LIB)/QoI\n");
    fprintf(fp_fe, "CFLAGS+=-I$(FENNEC_FOX_LIB)/Events\n");
    fprintf(fp_fe, "CFLAGS+=-I$(FENNEC_FOX_LIB)/Application\n");
    fprintf(fp_fe, "CFLAGS+=-I$(FENNEC_FOX_LIB)/Addressing\n");
    fprintf(fp_fe, "CFLAGS+=-I$(FENNEC_FOX_LIB)/Application/Dummy\
        n");
167 fprintf(fp_fe, "CFLAGS+=-I$(FENNEC_FOX_LIB)/Mac\n");
    fprintf(fp_fe, "CFLAGS+=-I$(FENNEC_FOX_LIB)/Radio\n");
    fprintf(fp_fe, "CFLAGS+=-I$(FENNEC_FOX_LIB)/Sensors\n");
    fprintf(fp_fe, "CFLAGS+=-I$(FENNEC_FOX_LIB)/Serial\n");
```

```
    fprintf(fp_fe, "CFLAGS+=-I$(FENNEC_FOX_LIB)/tinyos-2.x\n");
172 fprintf(fp_fe, "include $(FENNEC_FOX_LIB)/Sensors/sensors.
        extra\n");
    fprintf(fp_fe, "include $(FENNEC_FOX_LIB)/PowerMgmt/power.
        extra\n");
    fprintf(fp_fe, "include $(FENNEC_FOX_LIB)/libs/libs.extra\n");
    fprintf(fp_fe, "include $(FENNEC_FOX_LIB)/chips/chips.extra\n"
        );

177 fprintf(fp_fe, "CFLAGS+=-I$(FENNEC_FOX_LIB)/Mac/%s\n",
        policy_configuration_mac);
    fprintf(fp_fe, "CFLAGS+=-I$(FENNEC_FOX_LIB)/Radio/%s\n",
        policy_configuration_radio);
    fprintf(fp_fe, "CFLAGS+=-I$(FENNEC_FOX_LIB)/Addressing/%s\n",
        policy_configuration_address);
    /* just in case add tinyos radio for TOSSIM */
    fprintf(fp_fe, "CFLAGS+=-I$(FENNEC_FOX_LIB)/Radio/%s\n", "
        tinyos");
182
    struct libtab *lp;
    for(lp = libtab; lp < &libtab[NSYMS]; lp++) {
      if (lp->path && lp->used == 1) {
        fprintf(fp_fe, "CFLAGS+=-I%s\n", lp->path);
187   }
    }

    fclose(fp_fe);
}
192
void initialCodeGeneration(int policy_counter) {

    int conf_counter = 0;
    int event_counter = 0;
197 struct symtab *sp;

    for(sp = symtab; sp < &symtab[NSYMS]; sp++) {
      if (sp->name) {
        if (!strcmp(sp->type, "event_id")) {
202       event_counter++;
        }
        if (!strcmp(sp->type, "configuration_id")) {
          conf_counter++;
        }
207   }
    }

    find_policy_mac_and_radio();
    setFiles();
```

```
212   generateFrontCaches(event_counter, conf_counter,
          policy_counter);
}

void find_policy_mac_and_radio() {

217   struct libtab *lp;

      for(lp = libtab; lp < &libtab[NSYMS]; lp++) {
        if (lp->path && lp->type == TYPE_MAC && !strcmp(lp->def,
            POLICY_CONFIGURATION_MAC)) {
          policy_configuration_mac = lp->name;
222       break;
        }
      }

      if (policy_configuration_mac == NULL) {
227     fprintf(stderr, "Cannot find policy Mac defined as: %s\n",
            POLICY_CONFIGURATION_MAC);
        exit(1);
      }

      for(lp = libtab; lp < &libtab[NSYMS]; lp++) {
232     if (lp->path && lp->type == TYPE_RADIO && !strcmp(lp->def,
            POLICY_CONFIGURATION_RADIO)) {
          policy_configuration_radio = lp->name;
          break;
        }
      }
237
      if (policy_configuration_radio == NULL) {
        fprintf(stderr, "Cannot find policy Radio defined as: %s\n",
            POLICY_CONFIGURATION_RADIO);
        exit(1);
      }
242
      for(lp = libtab; lp < &libtab[NSYMS]; lp++) {
        if (lp->path && lp->type == TYPE_ADDRESS && !strcmp(lp->def,
            POLICY_CONFIGURATION_ADDRESS)) {
          policy_configuration_address = lp->name;
          break;
247     }
      }

      if (policy_configuration_address == NULL) {
        fprintf(stderr, "Cannot find policy addr defined as: %s\n",
            POLICY_CONFIGURATION_ADDRESS);
252     exit(1);
      }
```

```
}


void finishCodeGeneration() {
  generateEvent();
  generateEndCaches();

  generateApplicationC();
  generateNetworkC();
  generateQoIC();
  generateMacC();
  generateRadioC();
  generateEventC();
  generateApplicationNetworkC();
  generateNetworkMacC();
  generateMacRadioC();
  generateAddressingC();

  generateApplicationP();
  generateNetworkP();
  generateQoIP();
  generateMacP();
  generateRadioP();
  generateEventP();
  generateApplicationNetworkP();
  generateNetworkMacP();
  generateMacRadioP();
  generateAddressingP();

  setFennecExtra();
}

/* Generate Events */

void generateEventC() {

  struct symtab *sp;
  FILE *fp_eventC = fopen(eventC, "w");

  if (fp_eventC == NULL) {
    fprintf(stderr, "You do not have a permission to write into
        file: %s\n", eventC);
    exit(1);
  }

  fprintf(fp_eventC, "/* Swift Fox generated code for Fennec Fox
      Event configuration */\n");
  fprintf(fp_eventC, "\n#include <Fennec.h>\n\n");
  fprintf(fp_eventC, "configuration EventsC {\n");
```

```
    fprintf(fp_eventC, "  provides interface Mgmt;\n");
302 fprintf(fp_eventC, "}\n\n");
    fprintf(fp_eventC, "implementation {\n\n");
    fprintf(fp_eventC, "  components EventsP;\n");
    fprintf(fp_eventC, "  Mgmt = EventsP;\n\n");
    fprintf(fp_eventC, "  components CachesC;\n");
307 fprintf(fp_eventC, "  EventsP.EventCache -> CachesC;\n");
    fprintf(fp_eventC, "  /* Defined and linked event handlers */\
        n");

    for(sp = symtab; sp < &symtab[NSYMS]; sp++) {
      if (sp->name && !strcmp(sp->type, "event_id")) {
312     fprintf(fp_eventC, "\n  components new %sEventC() as %
            sEvent%dC;\n", sp->lib->name, sp->lib->name, sp->value)
            ;
        fprintf(fp_eventC, "  EventsP.%sEvent%d -> %sEvent%dC;\n",
             sp->lib->name, sp->value, sp->lib->name, sp->value);
      }
    }

317 fprintf(fp_eventC, "\n}\n");
    fclose(fp_eventC);
  }

  void generateEventP() {
322
    struct symtab *sp;
    FILE *fp_eventM = fopen(eventM, "w");

    if (fp_eventM == NULL) {
327   fprintf(stderr, "You do not have a permission to write into
          file: %s\n", eventM);
      exit(1);
    }

    fprintf(fp_eventM, "/* Swift Fox generated code for Fennec Fox
        Event module */\n");
332 fprintf(fp_eventM, "\n#include <Fennec.h>\n\n");
    fprintf(fp_eventM, "module EventsP {\n\n");
    fprintf(fp_eventM, "  provides interface Mgmt;\n");
    fprintf(fp_eventM, "  uses interface EventCache;\n");

337 for(sp = symtab; sp < &symtab[NSYMS]; sp++) {
      if (sp->name && !strcmp(sp->type, "event_id")) {
        fprintf(fp_eventM, "  uses interface Event as %sEvent%d;\n
            ", sp->lib->name, sp->value);
      }
    }
342
```

```
    fprintf(fp_eventM, "\n}\n\n");
    fprintf(fp_eventM, "implementation {\n\n");
    fprintf(fp_eventM, "  void turnEvents(bool flag);\n");
    fprintf(fp_eventM, "  void setEvent(uint8_t ev_num, bool flag)
        ;\n\n");
347 fprintf(fp_eventM, "  command error_t Mgmt.start() {\n");
    fprintf(fp_eventM, "    turnEvents(ON);\n");
    fprintf(fp_eventM, "    dbg(\"Events\", \"Events started\\n\")
        ;\n");
    fprintf(fp_eventM, "    signal Mgmt.startDone(SUCCESS);\n");
    fprintf(fp_eventM, "    return SUCCESS;\n");
352 fprintf(fp_eventM, "  }\n\n");
    fprintf(fp_eventM, "  command error_t Mgmt.stop() {\n");
    fprintf(fp_eventM, "    turnEvents(OFF);\n");
    fprintf(fp_eventM, "    dbg(\"Events\", \"Events stopped\\n\")
        ;\n");
    fprintf(fp_eventM, "    signal Mgmt.stopDone(SUCCESS);\n");
357 fprintf(fp_eventM, "    return SUCCESS;\n");
    fprintf(fp_eventM, "  }\n\n");
    fprintf(fp_eventM, "  void turnEvents(bool flag) {\n");
    fprintf(fp_eventM, "    uint8_t i;\n");
    fprintf(fp_eventM, "    for(i = 1 ; i < MAX_NUM_EVENTS; i++ )
        {\n");
362 fprintf(fp_eventM, "      if ( call EventCache.eventStatus(i))
         {\n");
    fprintf(fp_eventM, "        setEvent(i,flag);\n");
    fprintf(fp_eventM, "      }\n");
    fprintf(fp_eventM, "    }\n");
    fprintf(fp_eventM, "  }\n\n");
367 fprintf(fp_eventM, "  void setEvent(uint8_t ev_num, bool flag)
         {\n\n");
    fprintf(fp_eventM, "    switch(ev_num) {\n\n");
    fprintf(fp_eventM, "      case 0:\n");
    fprintf(fp_eventM, "        break;\n\n");

372 for(sp = symtab; sp < &symtab[NSYMS]; sp++) {
      if (sp->name && !strcmp(sp->type, "event_id")) {
        fprintf(fp_eventM, "      case %d:\n", sp->value);
        fprintf(fp_eventM, "        flag ? call %sEvent%d.start(
            call EventCache.getEntry(%d)) : call %sEvent%d.stop();\
            n", sp->lib->name, sp->value, sp->value, sp->lib->name,
             sp->value);
        fprintf(fp_eventM, "        break;\n\n");
377   }
    }

    fprintf(fp_eventM, "      default:\n");
    fprintf(fp_eventM, "        dbg(\"Events\", \"Events: there is
         no event with number %s\\n\", ev_num);\n", "%d");
```

```
382   fprintf(fp_eventM, "    }\n");
      fprintf(fp_eventM, "  }\n\n");

      for(sp = symtab; sp < &symtab[NSYMS]; sp++) {
        if (sp->name && !strcmp(sp->type, "event_id")) {
387       fprintf(fp_eventM, "  event void %sEvent%d.occured(bool oc
              ) {\n", sp->lib->name, sp->value);
          fprintf(fp_eventM, "    oc ? call EventCache.setBit(%d) :
              call EventCache.clearBit(%d);\n", sp->value, sp->value)
              ;
          fprintf(fp_eventM, "  }\n\n");
        }
      }
392
      fprintf(fp_eventM, "\n}\n");
      fclose(fp_eventM);
    }

397 char *relopToLetter(int i) {
      switch(i) {
        case LT:
          return "LT";
        case GT:
402       return "GT";
        case LE:
          return "LE";
        case GE:
          return "GE";
407     case NE:
          return "NE";
        case EQ:
          return "EQ";
        default:
412       fprintf(stderr, "Unknown RELOP operator\n");
          exit(1);
        }
    }
```

../../src/sf/code_gen.c

Bibliography

[1] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, & Tools*. Addison-Wesley, 2nd edition, 2007.

[2] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38:393–422, 2002.

[3] Jude Allred, Ahmad Bilal Hasan, Saroch Panichsakul, William Pisano, Peter Gray, Jyh Huang, Richard Han, Dale Lawrence, and Kamran Mohseni. SensorFlock: An airbone wireless sensor network of micro-air vehicles. In *Proceedings of the 5th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 117–129, Sydney, Australia, November 2007.

[4] Shah Bhatti, James Carlson, Hui Dai, Jing Deng, Jeff Rose, Anmol Sheth, Brian Shucker, Charles Gruenwald, Adam Torgerson, and Richard Han. MANTIS OS: An embedded multithreaded operating system for wireless micro sensor platforms. *ACM/Kluwer Mobile Networks & Applications (MONET)*, 10:563–579, 2005.

[5] Qing Cao, Tarek Abdelzaher, John Stankovic, and Tian He. The LiteOS operating system: Towards unix-like abstractions for wireless sensor networks. In *International Conference on Information Processing in Sensor Networks (IPSN)*, pages 233–244, St. Louis, MO, USA, April 2008.

[6] Luca Cardelli. Typeful programming. *Formal Description of Programming Concepts*, pages 431–507, 1991.

[7] CollabNet. Apache subversion. Online, May 2010.

[8] David Culler, Deborah Estrin, and Mani Srivastava. Guest editors' introduction: Overview of sensor networks. *Computer*, 37:41–49, 2004.

[9] Johan den Haan. 7 recommendations for domain specific language design based on domain-driven design. Online, May 2009.

[10] Adam Dunkels. Full TCP/IP for 8-bit architectures. In *Proceedings of the 1st ACM/USENIX International Conference on Mobile Systems, Applications and Services (MobiSys)*, pages 85–98, San Francisco, CA, USA, May 2003.

[11] Adam Dunkels, Bjrn Grnvall, and Thiemo Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the 1st IEEE Workshop on Embedded Networked Sensors (Emnets-I)*, pages 455–462, Tampa, FL, USA, November 2004.

[12] Mathilde Durvy, Julien Abeillé, Patrick Wetterwald, Colin O'Flynn, Blake Leverett, Eric Gnoske, Michael Vidales, Geoff Mulligan, Nicolas Tsiftes, Niclas Finne, and Adam Dunkels. Making sensor networks IPv6 ready. In *Proceedings of the 6th ACM Conference on Networked Embedded Sensor Systems (SenSys)*, 2008. poster session.

[13] Martin Fowler. Introduction to domain specific languages. Online, October 2006.

[14] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesC language: A holistic approach to network embedded systems. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 1–11, San Diego, CA, USA, June 2003.

[15] Omprakash Gnawali, Ben Greenstein, Ki-Young Jang, August Joki, Jeongyeup Paek, Marcos Vieira, Deborah Estrin, Ramesh Govindan, and Eddie Kohler. The TENET architecture for tiered sensor networks. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 153–156, Boulder, Colorado, USA, October 2006.

[16] Naomi Hamilton. The A-Z of programming languages: AWK. Online, May 2008.

[17] Chih-Chieh Han, Ram Kumar, Roy Shea, Eddie Kohler, and Mani Srivastava. A dynamic operating system for sensor nodes. In *Proceedings of the 3rd ACM/USENIX International Conference on Mobile*

Systems, Applications and Services (MobiSys), pages 163–176, Seattle, WA, USA, June 2005.

[18] Carl Hartung, Richard Han, Carl Seielstad, and Saxon Holbrook. FireWxNet: A multi-tiered portable wireless system for monitoring weather conditions in wildland fire environments. In *Proceedings of the 4th ACM/USENIX International Conference on Mobile Systems, Applications and Services (MobiSys)*, pages 28–41, Uppsala, Sweden, June 2006.

[19] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. *ACM SIGPLAN Notices*, 35:93–104, 2000.

[20] Jing Huang. Introduction to semantics of programming languages. Online, September 1994.

[21] John Hughes. Why functional programming matters. *The Computer Journal*, 32:98–107, 1989.

[22] Information and Quantum Systems Labs – HP. Central nervous system for the earth (CeNSE). Online, February 2010.

[23] Stephen C. Johnson. Yacc: Yet another compiler-compiler. Technical Report 32, July 1975. Computing Science Technical Report, Bell Laboratories.

[24] Ali El Kateeb. Hardware reconfiguration capabilities for third-generation sensor nodes. *Computer*, 42:95–97, 2009.

[25] Younghun Kim, Thomas Schmid, Zainul M Charbiwala, Jonathan Friedman, and Mani B Srivastava. NAWMS: Nonintrusive autonomous water monitoring system. In *Proceedings of the 6th ACM Conference on Networked Embedded Sensor Systems (SenSys)*, pages 309–322, Raleigh, NC, USA, November 2008.

[26] Michael E. Lesk and Eric Schmidt. Lex: A lexical analyzer generator. *UNIX Research System: Programmer's Manual (10th edition)*, 2:375–387, 1990.

[27] Philip Levis and David Culler. Mate: A tiny virtual machine for sensor networks. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, San Jose, CA, USA, October 2002.

[28] Philip Levis, Neil Patel, David Culler, and Scott Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proceedings of the 1st USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)*, pages 15–28, San Francisco, CA, USA, March 2004.

[29] Jorge Lobo, Randeep Bhatia, and Shamim Naqvi. A policy description language. In *Proceedings 16th National Conference on Artificial Intelligence (AAAI) and 11th Conference on Innovative Applications of Artificial Intelligence (IAAI)*, pages 291–298, Orlando, FL, USA, July 1999.

[30] Konrad Lorincz, Bor rong Chen, Jason Waterman, Geoff Werner-Allen, and Matt Welsh. Resource aware programming in the Pixie OS. In *Proceedings of the 6th ACM Conference on Networked Embedded Sensor Systems (SenSys)*, pages 211–224, Raleigh, NC, USA, November 2008.

[31] New York City Department of Housing Preservation and Development. Residential tenants: Heat and water. Online, March 2010.

[32] Bison-GNU parser generator. Gnu. Online, March 2010.

[33] Vern Paxson, Jef Poskanzer, and Kevin Gong. Flex–fast lexical analyzer generator. Online, March 2010.

[34] Mark Pilgrim. *Dive Into Python*. Apress, 2004.

[35] Curt Schurgers, Gautam Kulkarni, and Mani B. Srivastava. Distributed on-demand address assignment in wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 13:1056–1065, 2002.

[36] Gyula Simon, Miklos Maroti, Akos Ledeczi, Gyorgy Balogh, Bronislav Kusy, Andras Nadas, Gabor Pap, Janos Sallai, and Ken Frampton. Sensor network-based countersniper system. In *Proceedings of the 5th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 1–12, Baltimore, MD, USA, November 2004.

[37] Edgewall Software. The trac project. Online, May 2010.

[38] Diomidis Spinellis and V. Guruprasad. Lightweight languages as software engineering tools. In *Proceedings of the Conference on Domain-Specific Languages (DSL)*, pages 67–76, Santa Barbara, CA, USA, January 1997.

[39] Marcin Szczodrak, Vasileios P. Kemerlis, Xuan Linh Vu, and Yiwei Gu. Swift fox tutorial. Technical report, March 2010. Computer Science Department, Columbia University.

[40] Marcin Szczodrak, Vasileios P. Kemerlis, Xuan Linh Vu, and Yiwei Gu. Swift fox whitepaper. Technical report, February 2010. Computer Science Department, Columbia University.

[41] Robert Szewczyk, Eric Osterweil, Joseph Polastre, Michael Hamilton, Alan Mainwaring, and Deborah Estrin. Habitat monitoring with sensor networks. *Communications of the ACM*, 47:34–40, 2004.

[42] Columbia University. Network security laboratory (nsl). Online, May 2010.

[43] Brett Warneke, Matt Last, Brian Liebowitz, and Kristofer S.J. Pister. Smart dust: Communicating with a cubic-millimeter computer. *IEEE Computer*, 34:44–51, 2001.