

## *Practical session* An Arachne primer

15 novembre 2004

This is a step by step introduction to Arachne. Question 1 shows how to write simple aspects. The aspects considered in this exercise remain simple because they manipulate a single join point. Through question 1, you will learn how to weave aspects with Arachne and the basics of the aspect syntax. Questions 2 and 3 built on question 1 and introduces more complicated pointcuts. Questions 3 and 4 will prompt yourself to write aspects triggered upon a specific sequence of function invocations or upon a global variable access. Question 5 is a real case study : it considers a poorly written network server. This server is poorly written because it contains a potential buffer overflow. This buffer overflow can be exploited by malicious user to gain privileged (root) access to the machine. Question 5 will prompt you to secure the server on the fly by weaving on the fly an appropriate aspect. Before doing anything, you will have to set up your working environment.

### **Setting up your working environment**

In order to be able to follow this tutorial, you need to have Arachne installed on your machine along with the source code mentioned in this document. The section below describes various ways of installing Arachne and the following explains how to fetch the source code used here.

#### **Installing Arachne**

This is a practical session aiming at teaching how to use Arachne. Hence you need to have Arachne installed on your machine. The Arachne home page provides most of the tools you will need<sup>1</sup>. You can either download a bootable cd image or you can download the Arachne packages as a set of binaries compiled for Linux 2.6. If you belong to the Obasco research team, you may also want to compile Arachne from its source.

If you choose to download the bootable cd image, save the image on your hard drive and burn it on a cd. This cd now contains a modified version of the Knoppix Linux distribution<sup>2</sup>. Leave the cd in your cdrom drive and reboot your machine<sup>3</sup>. At the `boot :` prompt, type `linux26` and press enter. Knoppix will then start and launch the graphical desktop environment. You are now ready to do some serious Arachne development.

If you choose to download the binary version of Arachne, save the file `pubcurrent.tar.bz2` somewhere on your hard drive. Arachne is meant to run on Intel IA-32 machines under the Linux operating system. Under Linux, in a `bash` shell, go the directory (`cd`) where you saved Arachne binary archive (that is go the directory containing `pubcurrent.tar.bz2`). To uncompress the archive please type `bunzip2 ./pubcurrent.tar.bz2` && `tar xvf ./pubcurrent.tar` and press the enter key. Then, request super user privilege by typing the command `su` (note that you will be prompted for the root password). You can now complete the Arachne installation by typing `mv ./Arachne /usr/local/Arachne` and pressing enter. You have successfully installed Arachne on your machine.

If you choose to install Arachne from its source code, you can either fetch the source archive from the Obsaco intranet (<http://x-info.emn.fr/arachne>) or from the Obasco CVS (the module name is `ArachneSrc`, a description of how to access the CVS is available on the Obasco wiki<sup>4</sup>). Unzip the source archive in a directory. The top level directory in the source tree contains a file named `makefile.options`. Edit this file and change

---

<sup>1</sup>Go to <http://www.emn.fr/x-info/arachne>

<sup>2</sup>see <http://www.knoppix.net>

<sup>3</sup>Make sure that the BIOS allows you to boot from the cdrom drive.

<sup>4</sup><http://servinfo.info.emn.fr/wikis/loac/CVS>

the value of the variable `TOP_LEVEL_COMPILATION_DIRECTORY` so that it holds the path to the directory containing the Arachne source code on your computer. In a bash shell, go to the top level source directory (that is `cd` to the directory where you have unzipped the source archive) and type `make`. This will compile the different source. Then, request super user privilege by typing the command `su` (note that you will be prompted for the root password). You can now complete the Arachne installation by typing `make install` and pressing enter. You are now ready to do some serious Arachne development.

**Getting the "Arachne Primer" source code** In a standard Arachne installation, the Arachne Primer source code is located under `/usr/local/Arachne/doc/Primer`. We will create a directory on your user account to hold these files so that you can freely modify them. In a bash shell, please type `mkdir ~/Primer` and press enter. This will create a subdirectory named `Primer` under the root of your user account. Then type `cp -r /usr/local/Arachne/doc/Primer/* ~/Primer`. This command will copy all the files from the `/usr/local/Arachne/doc/Primer` to your `Primer` account directory.

**End of session** These of you using the Arachne bootable cds might encounter a problem at the end of the session. The bootable cds provides a user account that is in fact located in a ram drive. Hence rebooting or shutting the machine down erase their work. It is however possible to save their work on a floppy at the end of the session. To do so, please follow the following steps : first insert a floppy in the floppy drive. Then in a bash shell, type `cp -r ~/Primer /mnt/fl` and press the tabulation key<sup>5</sup>. The LED on the floppy drive should blink and the operating system should complete your command by writing `cp -r ~/Primer /mnt/floppy`. Press enter. All your files have now been copied on the floppy.

## Exercise 1 :Simple aspects and Fibonacci numbers

### Question 1 (Tracing the base program execution)

Our purpose here is to teach you how to compile and weave aspects with Arachne.

Go to the directory `Question1` (that is type `cd ~/Primer/Question1` and press enter). You can see the different files hold in this directory with the command `ls`<sup>6</sup>.

`fib.c` is a regular C program. Compile `fib.c` and runs it.

`aspect.c` is an aspect. Compile `aspect.c` into a DLL named `aspect.so`.

Weave the compiled aspect (i.e. the dll `aspect.so`) into `fib.exe`.

Do you think that outputting "Hello world" again and again slows down the base program? Deweave the aspect `aspect.so` in a running `fib.exe` to speed up the execution.

What is the difference between `./fib.exe` and `run ./fib.exe`?

Modify `aspect.c` so that it displays "fib() has been called" instead of "hello world". Which part of the aspect have you modified?

Modify `aspect.c` so that it displays the number of times the function `fib` defined in `fib.c` has been called by the base program `fib.exe`.

---

<sup>5</sup>the Knoppix automounter makes the usage of the tabulation mandatory

<sup>6</sup>if you are uncomfortable with `ls` type `man ls` and press enter. You may also want to type `man man` and press enter.

### Question 2 (From a recursive to an iterative implementation)

The goal of this question is to let you write your first aspect with Arachne. Go to the directory `Question2` (that is type `cd ~/Primer/Question2` and press enter).

Compare `fib.c` and `fibIter.c`. Which version computes the Fibonacci numbers the fastest? Prove your hypothesis using the `time` command (see `man time`). Modify the two programs (`fib.c` and `fibIter.c`) to perform accurate timings.

Write an aspect than once woven into `fib.c` will modify give to the `fib` function behavior, the behavior defined in the `fibIter.c` file. In other words, once woven, `fib.exe` should behave as if it was using the naive recursive implementation but as it was built with an iterative implementation.

### Question 3 (Even more optimization)

The goal of this question is to let you play with the information that is given to the aspect upon the execution of the action. Go to the directory `Question3` (that is type `cd ~/Primer/Question3` and press enter).

If you analyze carefully the behavior of `fib.exe` in the previous question, you will discover that it is not needed to use the iterative version when the argument given to the `fib` function is even. Rewrite your aspect so that it uses the `fib` function provided `fib.c` when the argument given to the `fib` function is even and use an iterative implementation when the argument is odd.

**Hint :** in C, the expression below will be true if the integer (i.e. number) `n` is odd and false otherwise :  
`((n%2)==0)`

## Exercise 2 : cflow and bank accounts

### Question 4 (Working for the IRS)

The goal of this question is to let you write an aspect whose action is triggered when a specific function calls another one in the base program. Go to the directory `Question4` (that is type `cd ~/Primer/Question4` and press enter). This directory contains a small banking application. A `Makefile` is provided to save your typing (see `man make`).

Compile and run the application.

The laws have changed. Now the local government has chosen to apply a tax on every bank transfers made between two accounts. Compile, weave and deweave the `Tax.c` aspect.

Correct the error contained in the `Tax.c` aspect.

### Question 5 (Working for the bank)

The goal of this question is to let you write an aspect whose action is executed when the base program reads a global variable. Go to the directory `Question5` (that is type `cd ~/Primer/Question5` and press enter).

The new tax established by the government has decreased the bank profits. In turn, they want to increase the overdraft interest rate. Rewrite `aspect.c` so that once woven this aspect increases the overdraft interest rate by 10%.

## Exercise 3 : a real case study

### Question 6 (Working for the CIA)

The goal of this question is to show you that Arachne is really useful. Go to the directory `Question6` (that is type `cd ~/Primer/Question6` and press enter).

This directory contains a small echo server and a client. Compiles them together by `make all`. In a first shell, runs the server by typing `make run`. In a second shell, run the client by typing `./client.exe`; this will prompt you for some data to be sent to the server. Type `test` and press enter. The client will report that the server returned `test`. In the shell of the server, an address has been displayed. Make sure that the displayed address corresponds with the address contained in the file `magic.h`. Press the `ctrl` key with the `c` key in the shell of the server to stop it. Then type `make clean` and type again `make all`. Restart the server by typing `make run`. Now, in another shell run a client by typing `client.exe`. When prompted for data to be sent to the server, type, `hack`. The client will then behave as a remote shell running on the sever machine.

In fact, `client.c` exploits a bug known as a buffer overflow to take control of the server. Locate the bug and write an aspect (`aspect.c`) that will allow the server to be properly secured on the fly.

## Exercise 4 : THE exam

### Question 7 (Exam)

Go to the directory `Question7` (that is type `cd ~/Primer/Question7` and press enter).

Write an aspect (that you will store in a file named `aspect.c`) that will displays "call to compute" every time the base program calls the function `compute`.

Complete `aspect.c` so that an appropriate warning is displayed before the function `compute` overflows the integer `global`.

Explains why the aspect reports many warnings and not only one.

## *Practical session* An Arachne primer

### — SOLUTIONS —

15 novembre 2004

#### Solution 1 (Tracing the base program execution)

##### Question 1 (Tracing the base program execution)

Our purpose here is to teach you how to compile and weave aspects with Arachne.

Go to the directory `Question1` (that is type `cd ~/Primer/Question1` and press enter). You can see the different files hold in this directory with the command `ls`<sup>6</sup>.

`fib.c` is a regular C program. Compile `fib.c` and runs it.

`aspect.c` is an aspect. Compile `aspect.c` into a DLL named `aspect.so`.

Weave the compiled aspect (i.e. the dll `aspect.so`) into `fib.exe`.

Do you think that outputting "Hello world" again and again slows down the base program ? Deweave the aspect `aspect.so` in a running `fib.exe` to speed up the execution.

What is the difference between `./fib.exe` and `run ./fib.exe` ?

Modify `aspect.c` so that it displays "fib() has been called" instead of "hello world". Which part of the aspect have you modified ?

Modify `aspect.c` so that it displays the number of times the function `fib` defined in `fib.c` has been called by the base program `fib.exe`.

To compile `fib.c` type :

```
gcc -o fib.exe fibo.c
```

To run `fib.exe` type :

```
./fib.exe
```

To compile `aspect.c` type :

```
source /usr/local/Arachne/site/sourceMe.bash  
acc aspect.so aspect.c
```

To weave the aspect, first run the base program in a first shell by typing :

```
source /usr/local/Arachne/site/sourceMe.bash  
run ./fib.exe
```

And weave the aspect using a second shell I by typing :

```
source /usr/local/Arachne/site/sourceMe.bash  
weave $(pidof fib.exe) WEAVE ./aspect.so
```

Press a key in the first shell. You will see "hello world" printed at every call to `fib`.

To deweave the aspect, type in the shell where you have already used the `weave` command :

```
weave $(pidof fib.exe) DEWEAVE ./aspect.so
```

The "hello world" will stop being printed and the computation will run much faster.

When one runs `fib.exe` by typing `./fib.exe`, the Arachne kernel is not loaded in the base program (i.e. `fib.exe`). When one runs `fib.exe` by typing `run ./fib.exe`, the Arachne kernel is loaded by the base program. The latter is necessary to let the Arachne kernel properly handle weaving requests inside `fib.exe`<sup>1</sup>.

To display `fib()` instead of "hello world", it is necessary to modify the action part of the aspect. `aspect.c` should look as follows :

```
1 #include <stdio.h>
2 HelloWorld [:
3     int fib(int n) [: {
4         printf("fib()\n");
5         return continue_fib(n);
6     } :]
7 :]
```

To count the number of time the base program called the `fib` function, it is necessary to add a state to the aspect. The simplest solution is to use a global variable as follows :

```
1 #include <stdio.h>
2 /*below is a global variable*/
3 int count = 0;
4 HelloWorld [:
5     int fib(int n) [: {
6         count = count+1;
7         printf("fib() invoked %i times \n",count);
8         return continue_fib(n);
9     } :]
10 :]
```

## Solution 2 (From a recursive to an iterative implementation)

### Question 2 (From a recursive to an iterative implementation)

The goal of this question is to let you write your first aspect with Arachne. Go to the directory `Question2` (that is type `cd ~/Primer/Question2` and press enter).

Compare `fib.c` and `fibIter.c`. Which version computes the Fibonacci numbers the fastest? Prove your hypothesis using the `time` command (see `man time`). Modify the two programs (`fib.c` and `fibIter.c`) to perform accurate timings.

Write an aspect than once woven into `fib.c` will modify give to the `fib` function behavior, the behavior defined in the `fibIter.c` file. In other words, once woven, `fib.exe` should behave as if it was using the naive recursive implementation but as it was built with an iterative implementation.

`fibIter.c` computes Fibonacci numbers faster than `fib.c`. To perform accurate timings, `fib.c` should be rewritten as follow :

```
1 #include <stdio.h>
2 int fib(int n) {
3     if(n<=1) {
4         return 1;
5     }
6     return fib(n-1)+fib(n-2);
7 }
```

---

<sup>1</sup>Actually, there are versions of Arachne that allows the Arachne kernel to be injected on the fly in the base program (i.e. `fib.exe` but this feature is still unstable.

```

8 int main(int argc, char** argv) {
9     printf("%d\n", fib(42));
10    return 0;
11 }

```

It should be compiled using :

```
gcc -o fib.exe fib.c
```

Timing of fib.exe shall be performed as follow :

```
time ./fib.exe
```

Likewise, to perform accurate timings, fiIterb.c should be rewritten as follow :

```

1 #include <stdio.h>
2 int fib(int n) {
3     int i;
4     int last=1;
5     int previous=1;
6     int result;
7     if( n <= 1 ) {
8         return 1;
9     }
10    for(i=2; i<=n;i++ ) {
11        result = last + previous;
12        previous = last;
13        last = result;
14    }
15    return result;
16 }
17 int main(int argc, char** argv) {
18    printf("%d\n", fib(42));
19    return 0;
20 }

```

It should be compiled using :

```
gcc -o fiboIter.exe fiboIter.c
```

Timing of fib.exe shall be performed as follow :

```
time ./fiboIter.exe
```

The timing reported by time should show that the iterative implementation (i.e. fibIter.c) runs faster than the recursive one (i.e. fib.c).

The aspect below, once woven in fib.exe will makes it behave as if it was using an iterative implementation :

```

1 ReplaceFib [:
2     int fib(int n) [: {
3         int i;
4         int last=1;
5         int previous=1;
6         int result;
7         if( n <= 1 ) {
8             return 1;
9         }
10        for(i=2; i<=n;i++ ) {
11            result = last + previous;
12            previous = last;
13            last = result;
14        }
15        return result;
16    } :]
17 :]

```

Assuming that you store the aspect source code presented below in a file named aspect.c, this aspect can be compiled through :

```
source /usr/local/Arachne/site/sourceMe.bash
```

```
acc aspect.so aspect.c
```

To weave the compiled aspect, first run fib.exe in a first shell by typing :

```
source /usr/local/Arachne/site/sourceMe.bash
```

```
run ./fib.exe
```

And weave the aspect using a second shell by typing :

```
source /usr/local/Arachne/site/sourceMe.bash
weave $(pidof fib.exe) WEAVE ./aspect.so
```

Press a key in the first shell. You will note that the computation is carried out slightly faster.

### Solution 3 (Even more optimization)

#### Question 3 (Even more optimization)

The goal of this question is to let you play with the information that is given to the aspect upon the execution of the action. Go to the directory `Question3` (that is type `cd ~/Primer/Question3` and press enter).

If you analyze carefully the behavior of `fib.exe` in the previous question, you will discover that it is not needed to use the iterative version when the argument given to the `fib` function is even. Rewrite your aspect so that it uses the `fib` function provided `fib.c` when the argument given to the `fib` function is even and use an iterative implementation when the argument is odd.

**Hint :** in C, the expression below will be true if the integer (i.e. number) `n` is odd and false otherwise :  
`((n%2)==0)`

The point is to understand that an aspect receives the argument of the function it replaces in the base program. Hence, the aspect source code of the previous code should be rewritten as follow :

```
1 ReplaceFib [:
2     int fib(int n) [: {
3         int i;
4         int last=1;
5         int previous=1;
6         int result;
7         if(n <= 1 ) {
8             return 1;
9         }
10        if((n%2)==0) {
11            return continue_fib(n);
12        }
13        for(i=2; i<=n;i++ ) {
14            result = last + previous;
15            previous = last;
16            last = result;
17        }
18        return result;
19    } :]
20 :]
```

### Solution 4 (Working for the IRS)

#### Question 4 (Working for the IRS)

The goal of this question is to let you write an aspect whose action is triggered when a specific function calls another one in the base program. Go to the directory `Question4` (that is type `cd ~/Primer/Question4` and press enter). This directory contains a small banking application. A `Makefile` is provided to save your typing (see `man make`).

Compile and run the application.

The laws have changed. Now the local government has chosen to apply a tax on every bank transfers made between two accounts. Compile, weave and deweave the `Tax.c` aspect.

Correct the error contained in the `Tax.c` aspect.

To compile the application, type :

```
make compile
and press the enter key.
```

To run the application, type :

```
make run
and press the enter key.
```

To compile the `Tax.c` aspect, type :

```
make aspect
and press the enter key.
```

Once you have run the application in a first shell, go to a second shell and type :

```
make weave
and press the enter key.To deweave the aspect, type :
make deweave
and press the enter key.
```

`Tax.c` is incorrect. It taxes all withdrawal. It should be rewritten as follow :

```
1 #include <stdio.h>
2 #include "BankAccount.h"
3 #define TAX 0.206
4 Tax [:
5     int transfer(float amount,BankAccount * from,BankAccount *to) [:
6         int withdraw(float amount,BankAccount * account) [: {
7             int status ;
8             status =continue_withdraw(amount,account);
9             if(status == SUCCESS) {
10                printf("#####\n");
11                printf("Message from IRS: Application of Tax on bank transfers\n");
12                printf("#####\n");
13                status =withdraw(amount*TAX,account);
14            }
15            return status;
16        } :]
17     :]
18 :]
```

### Solution 5 (Working for the bank)

#### Question 5 (Working for the bank)

The goal of this question is to let you write an aspect whose action is executed when the base program reads a global variable. Go to the directory `Question5` (that is type `cd ~/Primer/Question5` and press enter).

The new tax established by the government has decreased the bank profits. In turn, they want to increase the overdraft interest rate. Rewrite `aspect.c` so that once woven this aspect increases the overdraft interest rate by 10%.

Write `aspect.c` as follows :

```
1 #include <stdio.h>
2 require OVERDRAFT_INTEREST_RATE as int *;
3 overdraftInterest [:
4     float OVERDRAFT_INTEREST_RATE [: {
5         printf("#####\n");
6         printf("in aspect\n");
7         return (*OVERDRAFT_INTEREST_RATE)*1,1;
8     } :]
9 :]
```

### Solution 6 (Working for the CIA)

### Question 6 (Working for the CIA)

The goal of this question is to show you that Arachne is really useful. Go to the directory `Question6` (that is type `cd ~/Primer/Question6` and press enter).

This directory contains a small echo server and a client. Compiles them together by `make all`. In a first shell, runs the server by typing `make run`. In a second shell, run the client by typing `./client.exe`; this will prompt you for some data to be sent to the server. Type `test` and press enter. The client will report that the server returned `test`. In the shell of the server, an address has been displayed. Make sure that the displayed address corresponds with the address contained in the file `magic.h`. Press the `ctrl` key with the `c` key in the shell of the server to stop it. Then type `make clean` and type again `make all`. Restart the server by typing `make run`. Now, in another shell run a client by typing `client.exe`. When prompted for data to be sent to the server, type, `hack`. The client will then behave as a remote shell running on the sever machine.

In fact, `client.c` exploits a bug known as a buffer overflow to take control of the server. Locate the bug and write an aspect (`aspect.c`) that will allow the server to be properly secured on the fly.

The buffer overflow concerns the local variable `in` in the function `handleRequest` from the file `server.c`. It is easy to write an aspect that replaces this function by a correct one :

```
1 #include <stdio.h>
2 #include <sys/types.h>
3 #include <unistd.h>
4 int handleRequestWithoutBufferOverflow() {
5     char in[256], *p=in, c;
6     do { read(0,p,1); p++ ;} while( ((*p-1) != '\n') && (p<(in+254)) );
7     *(p-1)=\0;
8     if(p==(in+254)) do { read(0,&c,1) ; } while (( c!=\0) && (c!= '\n'));
9     if(printf("%s\n",in)<=0) return 0;
10    if(fflush(stdout)!=0 ) return 0;
11    return 1;
12 }
13 secureAspect [ :
14     int handleRequest() [ : {
15         return handleRequestWithoutBufferOverflow();
16     } :]
17 :]
```

### Solution 7 (Exam)

#### Question 7 (Exam)

Go to the directory `Question7` (that is type `cd ~/Primer/Question7` and press enter).

Write an aspect (that you will store in a file named `aspect.c`) that will displays "call to compute" every time the base program calls the function `compute`.

Complete `aspect.c` so that an appropriate warning is displayed before the function `compute` overflows the integer `global`.

Explains why the aspect reports many warnings and not only one.

To trace all calls to `compute`, just write `aspect.c` as follow :

```
1 #include <stdio.h>
2 require global as int*;
3 integerOverflow [ :
4     void compute() [ : {
5         printf("call to compute\n");
6         continue_compute();
7     }
8     :]
9 :]
```

To detect an overflow just write `aspect.c` as follow :

```
1 #include <stdio.h>
2 require global as int*;
3 integerOverflow [:
4     void compute() [:
5         int global [: {
6             if((*global) +200000)<= 0)
7                 printf("Warning overflow detected\n");
8             return *global;
9         } :]
10     :]
11 :]
```