Scuola di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica

Tesi di Laurea

TITOLO ITALIANO

ENGLISH TITLE

MATTEO MONICOLINI

Relatore: *Rosario Pugliese*

Anno Accademico 2023-2024

INDICE

ELENCO DELLE FIGURE

"Sarà che prendo troppo spesso Trenitalia ma io non credo nelle coincidenze"
— Pinguini tattici nucleari "Test di ingresso di medicina"

1

INTRODUCTION

The introduction is usually a short chapter that can be read in less than 10 minutes.

The goal of the Introduction is to engage the reader (why you should keep reading). You don't have to discuss anything in detail. Rather, the goal is to tell the reader:

- what is the problem dealt with and why the problem is a problem;
- what is the particular topic you're going to talk about in the thesis;
- what are your goals in doing so, and what methodology do you follow;
- what are the implications of your work.

The above points will be further expanded in the following chapters, so only a glimpse is essential.

It is customary to conclude the Introduction with a summary of the content of the rest of the thesis. One or two sentences are enough for each chapter.

2

STATE OF THE ART

The State of the Art chapter is where you discuss . . . well, the state of the art. Obvious, isn't it?

You have to explain the main research papers related to your thesis and their shortcomings.

It's up to the writer to cite everything or just the sources that are directly relevant to the research topic, and the choice also depends on the kind of thesis work.

For example, one might refer to [5] as one of the first books about algorithms and programming issues, or [9] for the foundations of Game Theory.

Nevertheless, referring to standards (see [2]) for a more accurate technology description is also important.

It is customary to end this chapter by recapping that, given the current state of the art, there is a gap in the knowledge that must be filled - and this is the goal of the present work.

This chapter and the bibliography are an essential part of the thesis. They show that you did your research starting from solid foundations, and they allow the reader to both replicate your results and continue your work. Hence, the bibliography must be good (relevant works of solid reputation) and correct (allow the reader to find the referenced paper).

One of the best ways to do so is to create your bibliography using a tool, e.g., JabRef[1], which will help you in organising the bibliography. JabRef (or an equivalent tool) will help you in creating a bibliographic database (a so-called `.bib` file). Only the entries effectively cited will be imported into the thesis with the correct citation style.

A more straightforward way to organise the bibliographic entries of the papers, books, or whatever you cite in the thesis is to just write them into a text file named *.bib.

1 https://www.jabref.org

Note that most websites allow you to download the bibliographic entry of a paper (or book, or whatever) directly. For example, IEEEXplore has a button "Cite This" that allows you to download the entry and copy-paste it into your bibliographic tool. Just select 'BibTeX', copy-paste, and you will have the correct entry (well, mostly, always double-check).

3

INTEGRAZIONE DI OPENNEBULA CON FACPL

Come già discusso nel capitolo 2, grazie al modo in cui FACPL[6] è costruito risulta molto facile andare a fornire un'implementazione concreta dei PEP (*Policy Enforcement Point*) e PDP (*Policy Decision Point*) avendo conoscenza di quali sono le esigenze a cui devono rispondere.

Per validare ulteriormente questo punto abbiamo deciso di utilizzare FACPL in una situazione in cui fosse necessario interfacciarsi con un software già esistente così da mostrare le effettive potenzialità e semplicità di implementazione della libreria. Il caso concreto che abbiamo deciso di considerare deriva da un possibile sviluppo futuro già proposto all'interno di [8], ovvero un'integrazione con un "sistema di IaaS open-source sul cloud" come OpenNebula[3], software che è stato già presentato all'interno del capitolo 2.

3.1 IDEA DI BASE

Per iniziare abbiamo considerato due esempi di gestione delle risorse disponibili aderenti alla realtà e una spiegazione abbastanza dettagliata del loro funzionamento, presenti all'interno di [8]. Abbiamo inoltre potuto osservare due primi tentativi di implementazione descritti sempre all'interno di [8]:

- il primo sviluppato basandosi su una completa astrazione, ovvero dei files che simulano un sistema con diverse virtual machine, che è presente anche in [7] fra gli esempi nella cartella "*EXAMPLES*"
- il secondo basato su uno XEN Hypervisor, che però non è presente fra gli esempi forniti assieme alla libreria e di cui non vengono esplicitati i dettagli di implementazione.

Con questa conoscenza l'obiettivo iniziale è stato quello di riuscire ad interagire con un reale sistema su cui è installato un cloud manager per

fare si che questo eseguisse i comandi necessari per eseguire le PEP action da noi richieste e ci esponesse tutte le informazioni necessarie al PDP per valutare le richieste. Nel concreto era quindi necessario pensare a due parti distinte:

- una che svolgesse operazioni sulle singole virtual machine (avviarle, inserire nell'host corretto, fermarne l'esecuzione).
- una che recuperasse le informazioni generali sugli host e sul sistema tutto.

Il cloud manager che abbiamo deciso di utilizzare è stato OpenNebula per i motivi presentati nel capitolo 2

3.2 OPENNEBULA API

Il primo approccio che avevamo pensato di percorrere era quello di utilizzare il codice Java per invocare dei comandi da shell che fossero in grando di interagire con OpenNebula. Questo approccio sembrava il più immediato anche dato lo studio preliminare di OpenNebula che avevamo svolto che era stato soprattutto attraverso la shell (oltre che la web ui).

Per utilizzare i comandi da shell che permettono di interagire con OpenNebula occorre aver effettuato l'autenticazione come utente OpenNebula[1]. Questa soluzione aveva il principale vantaggio di poter fornire un'interfaccia generica che permettava in un futuro di far interagire con sforzo minimo FACPL anche con comandi di natura completamente diversa, tuttavia presentava anche diverse problematiche come la forte dipendenza dalla versione di OpenNebula installata nel sistema e una grande inefficienza nello svolgimento di alcune operazioni.

La strada su cui ci siamo orientati è stata quindi quella di utilizzare le API di OpenNebula per Java[2] in quanto queste semplificavano l'ottenimento di alcune informazioni. Come è possibile leggere dal sito di OpenNebula stesso, queste API sono a loro volta un wrapper dei metodi XML-RPC[3]. Le API utilizzate sono quelle della versione 5.12.0[4] dato che

1 https://docs.opennebula.io/6.8/management_and_operations/users_groups_management/manage_users.html
2 https://docs.opennebula.io/6.8/integration_and_development/system_interfaces/java.html
3 https://docs.opennebula.io/6.8/integration_and_development/system_interfaces/api.html#api
4 https://downloads.opennebula.io/packages/opennebula-5.12.0/

dopo la major version 5 sono compilate con la versione di Java 11 (al contrario di quato riportato sul sito stesso) e di conseguenza non erano compatibili con la versione di Java 8 con cui è stato sviluppato FACPL. I principali attori che sono stati utilizzati dalle API[5] sono stati:

- *Client*: è la classe principale che gestisce la connesione fra il core di OpenNebula e le chiamate XML-RPC, quasi tutti gli altri oggetti delle API richiedono di passare un oggetto di questo tipo per poter essere istanziati. Questo oggetto può essere istanziato passando uno *username* e una *password* al costruttore, ma presenta anche un costruttore che non richiede parametri e permette di derivare *username password* dall'utente corrente.

- *ClientConfigurationException*: è la classe che rappresenta l'eccezione che viene lanciata se si tenta di istanziare un *Client* con delle impostazioni di autorizzazione sbagliate, in particolare se *username password* sono errate oppure se si tenta di usare il costruttore vuoto lanciando il programma da un utente che non è uno user di OpenNebula.

- *OneResponse*: è la classe che incapsula le risposta XML-RPC di OpenNebula, viene istanziata con un *boolean* e una *String* che rappresentano rispettivamente l'esito della richiesta (positivo o negativo) e un eventuale messaggio che lo descrive. Quasi tutte le azioni eseguibili sui *PoolElement* ritornano un oggetto di questo tipo.

- *Pool*: è la classe che rappresenta un insieme di *PoolElement* e fornisce la possibilità di scorrere gli stessi ed eseguire in modo agevolato alcune operazioni

- *PoolElement*: è la superclasse della maggior parte delle classi che rappresentano gli elementi di OpenNebula, quelli più interessati nel progetto sono stati
  - *VirtualMachine*
  - *Host*
  - *Template*

5 https://docs.opennebula.io/doc/6.4/oca/java/org/opennebula/client/package-summary.html

Listing 1: Metodo make di FileLoggerFactory

```
1 public static Logger make(String fileName, Formatter formatter, Level
      level) {
2    Throwable t = new Throwable();
3    StackTraceElement directCaller = t.getStackTrace()[1];
4    String loggerName = directCaller.getClassName() + "-" + fileName;
5
6    Logger logger = Logger.getLogger(loggerName);
7
8    if (!isFileHandlerAttached(logger, fileName)) {
9       try {
10         FileHandler fileHandler = createFileHandler(fileName,
               formatter, level);
11         logger.addHandler(fileHandler);
12         logger.setUseParentHandlers(false);
13      } catch (IOException e) {
14         throw new RuntimeException("Failed to initialize logger
               handler.", e);
15      }
16   }
17
18   return logger;
19 }
```

3.3 LOGGING

Prima ancora di pensare ai comandi da eseguire sulle virtual machine si è reso necessario pensare ad una modalità di logging. Dato l'ambito di applicazione si è reso fondamentale pensare ad una scrittura di logs su file di modo da rendere gli stessi facilmente accessibili in futuro, anche a distanza di tempo.

All'interno del progetto è quindi fornita una classe `FileLoggerFactory` che utilizza il design pattern *Static Factory Methods* [1] e permette di creare dei file di log, di modo da evitare la necessità di interagire coi file in ogni classe che intende eseguire il log di informazioni oltre che gestire in modo consono gli handler dei files evitando duplicati. Questa classe permette di creare dei `Logger` con un'implementazione di `Level` e `Formatter` di default oppure di specificare questi parametri in input. Nonostante la presenza di questa classe, tutte le classi all'interno del progetto hanno i logger passati tramite dependency injection e forniscono un'implementazione di default che esegue logging nello stsandard

output (solitamente la console). L'unica classe che fa eccezione in questo è proprio `ContextStub_Default` che è il primo punto di ingresso della dipendenza.
L'idea è che nel caso in cui si preferisca eseguire logging in modo diverso si possa definire un logger diverso al posto dell'implementazione proposta. Per farlo occorre semplicemente modificare una riga all'interno del costruttore di `ContextStub_Default`:

Listing 2: Costruttore ContextStub_Default

```
1 public static ContextStub_Default getInstance() {
2    if (instance == null) {
3       try {
4          Configuration config = new
               Configurations().properties(CONFIG_FILE);
5          hyper1HostId = config.getString("hyper1.host.id");
6          hyper2HostId = config.getString("hyper2.host.id");
7          ContextStub_Default.oneClient = new Client();
8          Logger logger =
               FileLoggerFactory.make("logs/virtualMachines.log");
               inizializeStub(oneClient, logger);
9          instance = new ContextStub_Default();
10      } catch (ClientConfigurationException e) {
11         throw new RuntimeException("Failed to initialize Client: " +
               e.getMessage(), e);
12      } catch (ConfigurationException e) {
13         throw new RuntimeException(
14               "Errors in the config gile: " + e.getMessage(), e);
15      } catch (Exception e) {
16         throw new RuntimeException(
17               "Unexpected error during ContextStub_Default
                   initialization: " + e.getMessage(), e);
18      }
19   }
20   return instance;
21 }
```

3.4 GESTIONE DELLE VIRTUAL MACHINE GENERICHE

Per gestire le virtual machine l'approccio iniziale è stato quello di fornire un'interfaccia che permettesse in un futuro di poter interagire con le stesse anche in modo.
E' stata creata una classe wrapper chiamata `VMDescriptor` che contiene

le informazioni sulle virtual machine utili per il nostro progetto, questa classe serve per creare una prima astrazione dalle API utilizzate, in effetti questa stessa classe permetterebbe, ad esempio, di fornire un'implementazione come quella precedentemente pensata basata sui comandi da shell. L'interfaccia `VirtualMachineService` presenta due metodi che servono per lavorare all'effettivo con i `VMDescriptor`.

Listing 3: VirtualMachineService

```
1 public interface VirtualMachineService {
2    List<VMDescriptor> getVirtualMachinesInfo();
3    List<VMDescriptor> getRunningVirtualMachineInfo();
4 }
```

La scelta di avere due metodi separati per restituire tutte le macchine virtuali oppure solo quelle attualmente in esecuzione deriva dal fatto che in effetti spesso queste due liste vorranno essere utilizzate in modo diverso. Di solito le VirtualMachine presenti nel sistema sono molte più di quelle effettivamente in esecuzione e quindi non inserire il secondo metodo avrebbe portato una buona parte delle classi che volevano utilizzare un'implementazione di questa interfaccia a dover sempre inserire un controllo sullo stato delle virtual machine. Nel codice da me scritto viene utilizzata la sua implementazione concreta `OpenNebulaVMService` che, interfacciandosi con le API di OpenNebula già descritte, riesce ad ottenere tutte le informazioni richieste sulle VM e a popolare le liste.
Una volta ottenuta una lista di `VMDescriptor` è possibile filtrare le virtual machine sia a partire dal nome che a partire da altre loro caratteristiche come l'*ID* o il *template* utilizzato per crearle come spiegato nel capitolo 2. È stata implementata un'ulteriormente classe di comodo che mette a disposizione la possibilità di eseguire alcuni tipi di filtraggio sulle virtual machine in automatico, questa classe fornisce inoltre un meccaniscmo di logging ogni volta che viene eseguita un'operazione di filtraggio, come si può vedere nel codice di esempio:

```
1 private List<VMDescriptor> getAndLogVMs() {
2    List<VMDescriptor> vmDescriptors =
         vmService.getRunningVirtualMachineInfo();
3    logger.info("The VMs running are: " + vmDescriptors.toString());
4    return vmDescriptors;
5 }
```

```
1 public List<VMDescriptor> getRunningVMsByHostTemplate(String host,
     String templateId) {
```

```
2    return getAndLogVMs()
3          .stream()
4          .filter(x -> x.getHostId().equals(host) &&
               x.getTemplateId().equals(templateId))
5          .collect(Collectors.toList());
6 }
```

Listing 4: Esempio di metodo di filtraggio e metodo di logging

Non è necessario usare questa classe, però risulta comoda per fare sì che le classi che si occupano di eseguire un comando su una o più virtual machine non abbiano anche il compito di eseguire il filtraggio e fare logging.

3.5 GESTIONE DELLE VIRTUAL MACHINE DI OPENNEBULA

Da qui in avanti saranno presentate tutte le classi che interagiscono effettivamente con degli oggetti che rappresentano delle `VirtualMachine` come indicato nelle API di OpenNebula. La prima classe implementata è `OpenNebulaActionContext`, questa classe può essere istanziata usando un `Client` (e opzionalmente anche un `Logger`) e fornisce semplicemente uno stato da utilizzare per eseguire i comandi che richiedono informazioni sulle `VirtualMachine` attualmente in esecuzione, che in questo caso concreto saranno tutti tranne la creazione di una nuova virtual machine. La classe per eseguire i comandi ha la seguente base:

Listing 5: Classe astratta per i comandi

```
1 public abstract class OpenNebulaActionBase implements IPepAction{
2    protected final OpenNebulaActionContext ONActionContext;
3
4    public OpenNebulaActionBase(OpenNebulaActionContext ONActionContext) {
5       this.ONActionContext = ONActionContext;
6    }
7
8    public abstract void eval(List<Object> args);
9
10   protected void logResponse(OneResponse response) {
11      if (response.isError()) {
12         ONActionContext.getLogger().severe(response.getErrorMessage());
13      } else {
14         ONActionContext.getLogger().info(response.getMessage());
15      }
16   }
17 }
```

Per questa classe è stato valutato l'utilizzo di un `Themplate method`, tuttavia risultava abbastanza scomodo da applicare dato che alcune delle classi concrete potrebbero dover seguire un workflow diverso fra loro (es. sospensione di una virtual machine e creazione di una virtual machine) per il modo in cui le API di OpenNebula sono scritte. Inoltre si suppone la possibilità di scrivere comandi futuri che agiscano su più di una virtual machine in un solo comando, questa logica era già stata testata ma non si è rivelata necessaria nel nostro caso concreto e di conseguenza è stata successivamente rimossa ma il modo in cui è scritta la classe astratta lascia spazio ad una facile implementazione concreta in questo senso.

La scelta del nome `eval` è obbligata dal modo in cui FACPL accederà alla classe per eseguire i comandi. L'approccio scelto è quello di costruire l'oggetto di tipo `VirtualMachine` corrispondente all'ID ottenibile dal `VMdescriptor`. Questo passaggio può sembrare controintuitivo perchè partendo da un oggetto `VirtualMachine` si ottengono le sue informazioni per poi andare a ricreare un oggetto sostanzialmente identico per eseguire le operazioni, tuttavia questi passaggi hanno diversi lati positivi:

- Rendono il codice indipendente dalla modalità con cui si ottengono le informazioni sulle virtual machine
- Rendono il codice più aperto a modifiche e future implementazioni
- Rendono il codice molto più semplice da testare
- Isolano l'ottenimento delle informazioni ad una classe che estende `OpenNebulaVMService`

Le classi concrete per eseguire i comandi sulle virtual machine a partire da questa classe hanno tutte chiaramente una forma simile sebbene con alcuni accorgimenti.

Listing 6: Classe per avviare una `VirtualMachine`

```
1 public class CreateVM extends OpenNebulaActionBase {
2   public CreateVM(OpenNebulaActionContext ONActionContext) {
3     super(ONActionContext);
4   }
5
6   public void eval(List<Object> args) {
7     ONActionContext.getLogger().info("Starting VM: " + "[" + args.get(2) + ", " +
          args.get(1) + "]");
8     Template template = new Template((int) args.get(2), ONActionContext.getClient());
9     OneResponse instantiateResponse = template.instantiate((String) args.get(1));
```

```
10      logResponse(instantiateResponse);
11      if (!instantiateResponse.isError()){
12        VirtualMachine vm =
13            new VirtualMachine(instantiateResponse.getIntMessage(),
                  ONActionContext.getClient());
14        logResponse(vm.deploy((int) args.get(0)));
15      }
16    }
17  }
```

Listing 7: Classe per freezzare(sospendere) una `VirtualMachine`

```
1  public class FreezeVM extends OpenNebulaActionBase {
2    public FreezeVM(OpenNebulaActionContext ONActionContext) {
3      super(ONActionContext);
4    }
5
6    public void eval(List<Object> args) {
7      ONActionContext.getLogger().info("Suspending (Freezing) 1 VM of [host, template]:
           " + "[" + args.get(0) + " " + args.get(2) + "]");
8      List<VMDescriptor> suspendList =
9          ONActionContext.getVMsInfo()
10               .getRunningVMsByHostTemplate((String)args.get(0), (String)args.get(2));
11     if (suspendList.isEmpty()) {
12       ONActionContext.getLogger().severe("No VM found");
13         return;
14      }
15     logResponse(
16         new VirtualMachine(Integer.parseInt(suspendList.get(0).getVmId()),
               ONActionContext.getClient())
17         .suspend());
18   }
19 }
```

La classe `CreateVM`, utilizza un oggetto di tipo `Template`, che, come già anticipato all'interno del capitolo 2 permette di definire le caratteristiche per la creazione di una specifica virtual machine. Nel nostro caso concreto l'oggetto template risulta utilissimo per fare sì che la definizione delle caratteristiche delle virtual machine da utilizzare sia fatta attraverso la UI di OpenNebula (o comunque con dei file appositi validati da OpenNebula tramite UI o comando da shell). Nel caso di studio[6] vengono considerati due tipi di virtual machine istanziabili, quindi nei nostri test abbiamo

6 [8]

spesso considerato due template che fossero in grado di riprodurre i comportamenti richiesti, tuttavia grazie al `Template` si apre la strada all'utilizzo di virtual machine dalle caratteristiche più disparate senza neanche bisogno di cambiare il codice Java. Infatti grazie al modo in cui è scritto il codice basta definire in OpenNebula un nuovo template e utilizzare il suo ID all'interno di policy e richieste FACPL per poter creare (o distruggere, frezzare ecc.) delle virtual machine di quel tipo.

La classe `suspendVM` dall'altra parte raffigura la struttura che hanno tutti i comandi che agiscono sulle virtual machine attualmente in esecuzione, viene eseguita una ricerca per `Host` e `Template` fra tutte le virtual machine in esecuzione e dopodichè viene creato un oggetto di tipo `VirtualMachine` su cui eseguire in effettivo il comando, il tutto con appropriato logging a contorno. La necessità di filtrare per `Host` e `Template` è definita dalle politiche che stiamo applicando nel caso concreto.

3.6 INTERAZIONE CON IL CONTEXTSTUB E LE PEPACTIONS

La gestione della classe `ContextStub_Default` è definita dall'implementazione in Java di FACPL, la classe è stata quindi soltanto modificata affinchè potesse recuperare le informazioni necessarie ad eseguire le valutazioni sullo stato del sistema, in particolare sono stati aggiunte le seguenti variabili di sistema:

Listing 8: Context di OpenNebula

```
1  @Override
2  public Object getContextValues(AttributeName attribute) {
3     ...
4     if (attribute.getCategory().equals("system") &&
          attribute.getIDAttribute().equals("vm-name")) {
5        return UUID.randomUUID().toString();
6     }
7
8     if (attribute.getCategory().equals("system") &&
          attribute.getIDAttribute().equals("hyper1.vm-names")) {
9        Set runningHyper1VMs = new Set();
10       vmsInfo.getRunningVMsByHost(hyper1HostId).forEach(vm ->
             runningHyper1VMs.addValue(vm.getVmName()));
11       return runningHyper1VMs;
12    }
13    if (attribute.getCategory().equals("system") &&
          attribute.getIDAttribute().equals("hyper2.vm-names")) {
```

```
14      Set runningHyper2VMs = new Set();
15      vmsInfo.getRunningVMsByHost(hyper2HostId).forEach(vm ->
            runningHyper2VMs.addValue(vm.getVmName()));
16      return runningHyper2VMs;
17    }
18    if (attribute.getCategory().equals("system") &&
          attribute.getIDAttribute().equals("hyper1.vm1-counter")) {
19      return vmsInfo.countRunningVMsByHost(hyper1HostId).doubleValue();
20    }
21    if (attribute.getCategory().equals("system") &&
          attribute.getIDAttribute().equals("hyper2.vm1-counter")) {
22      return vmsInfo.countRunningVMsByHost(hyper2HostId).doubleValue();
23    }
24    if (attribute.getCategory().equals("system")
25          && attribute.getIDAttribute().equals("hyper1.availableResources")) {
26      return hostInfo.getAvailableCpu(hyper1HostId);
27    }
28    if (attribute.getCategory().equals("system")
29          && attribute.getIDAttribute().equals("hyper2.availableResources")) {
30      return hostInfo.getAvailableCpu(hyper2HostId);
31    }
32    return null;
33  }
```

Guardando questo snippet notiamo alcune parti da evidenziare:

- `UUID.randomUUID().toString()` è un metodo che permette di ottenere un identificativo unico che verrà usato come nome per le virtual machine. In OpenNebula le virtual machine non sono istanziabili a partire da un ID, dato che questo viene assegnato dal software alla creazione, tuttavia è possibile assegnare un nome ad una virtual machine e quindi quello che abbiamo deciso di fare è usare il nome come identificativo all'interno del nostro progetto. Questa logica permette di assegnare degli identificativi più significativi in un futuro se fosse necessario e permette di disaccoppiare la nostra logica dalla logica con cui OpenNebula associa gli ID.
- `Set` è una classe libreria di FACPL per Java che permette di creare un insieme di oggetti, l'utilizzo di un oggetto di questo tipo è obbligatorio per usare i metodi di confronto presenti nella libreria, usare un oggetto della classe `Set` incluso nelle Collections causerà un errore nell'analisi delle Policy

- `hostInfo` è un oggetto della classe `HostInfo`, questa classe non è stata presentata ma è semplicemente una classe che permette, come si può notare dal codice, di ottenere informazioni riguardo CPU e quantità di memoria disponibili.
- Come si può notare ci sono diverse parti di codice in cui si fa riferimento a `hyper1HostId` e `hyper2HostId`, questi due parametri sono letti direttamente dal file `config.properties` locato nella directory del progetto grazie all'utilizzo di alcuni metodi forniti all'interno della libreria `Apache Commons`[10], come si può vedere guardando il codice del costruttore (listing 2).

Per quanto riguarda le `PEPAction`, anche in questo caso basandosi sulla struttura fornita dalla libreria FACPL non ci sono molte scelte implementative che si possono fare e il risultato è il seguente:

Listing 9: Classe PEPAction adattata

```
1  public class PEPAction{
2     public static HashMap<String, IPepAction> getPepActions() {
3        ContextStub_Default.getInstance();
4        HashMap<String, IPepAction> pepAction = new HashMap<String,
             IPepAction>();
5        pepAction.put("release",
6              new ReleaseVM(ContextStub_Default.getONContext()));
7        pepAction.put("create",
8              new CreateVM(ContextStub_Default.getONContext()));
9        pepAction.put("freeze",
10             new FreezeVM(ContextStub_Default.getONContext()));
11
12       return pepAction;
13    }
14 }
```

3.7 ACCESSO DALL'ESTERNO DEL PROGETTO

Per il modo in cui il progetto è scritto ci sono diversi punti di entrata da cui una persona esterna può iniziare sviluppare codice che sfrutti le classi sopra discusse, alcuni dei quali sono anche stati già esposti durante la presentazione delle classi stesse. Il progetto è distribuito con due package contenenti le due implementazioni concrete delle tecniche di gestione presentate in [8] oltre che con il codice FACPL che le ha generate, di conseguenza aprendo il progetto con Eclipse si può facilmente cominciare a generare nuove richieste e trasformarle in codice Java tramite la UI di

Eclipse[7]. Inoltre è anche fornita la cartella `/opennebula_context_actions` che contiene i file java delle classi `PEPAction` e `ContextStub_Default`

Nonstante questo si è pensato di fornire delle classi che permettessero, dato un file FACPL, di: validarlo, generare il codice Java corrispondente, compilarlo e, nel caso lo si voglia, anche di eseguire direttamente il `main` di `MainFACPL`. Il motivo principale per cui le classi che saranno discusse in questo paragrafo sono state ideate è che all'interno della documentazione di FACPL non è mai esplicitato un workflow da seguire per poter eseguire a runtime la decisione delle policy e/o la generazione di una nuova richiesta, di conseguenza si è reso utile idearne uno.

Listing 10: Classe FACPLHandlingTemplate

```
1  public abstract class FACPLHandlingTemplate {
2
3    private final String CONFIG_FILE = "config.properties";
4     protected Logger logger;
5     protected CodeExecutorInterface executor;
6     protected String javaFilesDir;
7     protected ClassSetup setupper;
8
9     public FACPLHandlingTemplate(String logFilePath, String javaFilesDir) throws
          IOException {
10       this.logger = FileLoggerFactory.make(logFilePath);
11       this.javaFilesDir = javaFilesDir;
12    }
13
14    public FACPLHandlingTemplate(Logger logger, String javaFilesDir) throws
          IOException {
15       this.logger = logger;
16       this.javaFilesDir = javaFilesDir;
17    }
18
19    public final void execute(String[] args) throws Exception {
20       try {
21          List<String> fileLocations = Arrays.asList(args[0]);
22          initializeConcreteSetupperExecutor(fileLocations);
23          setup();
24          compile();
25          postProcess();
26       } catch (Exception e) {
27          logger.severe("An error occurred: " + e.getMessage());
28          throw e;
29       }
30    }
31
32    protected abstract void initializeConcreteSetupperExecutor(List<String>
          fileLocations) throws Exception;
33
34    protected void setup() throws Exception {
```

7 https://facpl.readthedocs.io/en/latest/txt/facpl.html

```
35      setupper.setup(new
            Configurations().properties(CONFIG_FILE).getString("context.file.location"),
            javaFilesDir);
36    }
37
38    protected void compile() throws Exception {
39       boolean success = executor.compileJavaFiles();
40       if (success) {
41          logger.info("Compilation successful.");
42       } else {
43          logger.severe("Compilation failed.");
44          throw new RuntimeException("Compilation failed");
45       }
46    }
47
48    protected abstract void postProcess() throws Exception;
49 }
```

La classe base che è stata ideata è `FACPLHandlingTemplate`, come si può intuire dal nome e dalla struttura della classe, è una rappresentazione del pattern *Template Method*[4]. Questa classe permette di istanziare degli oggetti a partire da una locazione dove verranno inseriti e successivamente compilati, i file Java. Lanciando il comando `execute` infatti quello che succede effetivamente è:

1. Vengono inizializzati gli oggetti che servono per eseguire il setup della cartella che dovrà contenere i file Java, compilarli ed eseguirli.
2. I file Java prodotti a partire dai file FACPL vengono messi tutti nella cartella di destinazione finale, che è determinata dal parametro con cui viene eseguito il metodo `execute`
3. I file aggiuntivi (solitamente `PEPAction` e `ContextStub_Default`) vengono messi nella cartella finale
4. I file vengono compilati
5. I file compilati possono essere eseguiti o, più in generale, utilizzati per qualunque scopo si voglia definire.

Questi passaggi all'apparenza molto semplici in realtà nella loro implementazione concreta necessitano di diversi passaggi intermedi

Le implementazioni concrete di questa classe astratta che vengono fornite sono `ApplyPolicy` e `RequestExecution`, che servono rispettivamente per applicare una policy al sistema e eseguire la valutazione di una richiesta, ed entrambe utilizzano come `setupper` la classe `OpenNebulaFACPLClassSetup` che presenta il seguente metodo `setup`:

Listing 11: Metodo di setup per OpenNebula

```
1  @Override
2  public void setup(String additionalFilesFolder, String outputFolder) {
3     logger.info("Starting setup...");
4
5     try {
6        classGenerator.generateClasses("tmp/FACPLFiles");
7        logger.info("Class generation completed successfully.");
8     } catch (Exception e) {
9        logger.severe("Failed to generate classes: " + e.getMessage());
10       e.printStackTrace();
11       return;
12    }
13
14    try {
15       FolderContentHandler folderManager = new
             FolderContentHandler(logger);
16       folderManager.processFolderContents("tmp/FACPLFiles/",
             outputFolder, new MoveStrategy());
17       folderManager.processFolderContents(additionalFilesFolder,
             outputFolder, new CopyStrategy());
18       logger.info("Folder contents handled successfully.");
19    } catch (IOException e) {
20       logger.severe("Failed to move folder contents: " + e.getMessage());
21       e.printStackTrace();
22    }
23 }
```

In questo caso si evidenziano due punti necessari di ulteriore spiegazioni:

- `classGenerator` è un oggetto creato a partire dalla classe `OpenNebulaFACPLClassGenerator` che sfrutta la libreria di FACPL e il generatore fornito come esempio dalla stessa per generare tutte le classi Java necessarie.
- `FolderContentHandler` è una classe che grazie ad uno stragegy[4] applicato ad un metodo, permette di definire diversi modi per gestire i contenuti di due cartelle. In questo caso vengono utilizzate due implementazioni dello strategy `MoveStrategy` e `CopyStrategy` per muovere e copiare i file dalla prima cartella nella seconda.

Il modo in cui questa parte di codice è scritta lascia molta possibilità in un futuro di implementare classi concrete che gestiscono i file FACPL in modi completamente diversi. Il package `entryPoint` infatti è pensato per essere utile anche in progetti di altra natura rispetto a quello in esame,

fornendo però un metodo semplice per eseguire il logging su un file e uno scheletro delle operazioni da eseguire.

3.8 UTILIZZO DI MAVEN

3.9 HOW DID I DISCOVER A NOVEL TYPE OF HEATED WATER

Explain in detail what are the steps to heat the water in a novel way.

3.10 HOW MY HEATED WATER DIFFERS FROM THE PREVIOUS ONES

Describe why and how your findings are different from the past versions. Here you might want to add code (see for example Listing 12), or tables (see Table 1).
Note that figures, listings, tables, and so on, should never be placed 'manually'. Let LaTeX decide where to put them - you'll avoid headaches (and bad layouts). Furthermore, each of them must be referred to at least once in the body of the thesis.

Tabella 1: Example table

Country	Country code	ISO codes
Canada	1	CA / CAN
Italy	39	IT / ITA
Spain	34	ES / ESP
United States	1	US / USA

```
1  import java.awt.Rectangle;
2
3  public class ObjectVarsAsParameters
4  {
5      public static void main(String[] args)
6      {        go();
7      }
8
9      public static void go()
10     {
11         Rectangle r1 = new Rectangle(0,0,5,5);
12         System.out.println("In method go. r1 " + r1 + "\n");
13         // could have been
14         //System.out.prinltn("r1" + r1.toString());
15         r1.setSize(10, 15);
16         System.out.println("In method go. r1 " + r1 + "\n");
17         alterPointee(r1);
18         System.out.println("In method go. r1 " + r1 + "\n");
19
20         alterPointer(r1);
21         System.out.println("In method go. r1 " + r1 + "\n");
22     }
23
24     public static void alterPointee(Rectangle r)
25     {
26         System.out.println("In method alterPointee. r " + r + "\n");
27         r.setSize(20, 30);
28         System.out.println("In method alterPointee. r " + r + "\n");
29     }
30
31     public static void alterPointer(Rectangle r)
32     {
33         System.out.println("In method alterPointer. r " + r + "\n");
34         r = new Rectangle(5, 10, 30, 35);
35         System.out.println("In method alterPointer. r " + r + "\n");
36     }
37 }
```

Listing 12: Java example

```
1 import java.awt.Rectangle;
2
3 public class ObjectVarsAsParameters
4 { public static void main(String[] args)
5    { go();
6    }
7
8    public static void go()
9    { Rectangle r1 = new Rectangle(0,0,5,5);
10       System.out.println("In method go. r1 " + r1 + "\n");
11       // could have been
12       //System.out.prinltn("r1" + r1.toString());
13       r1.setSize(10, 15);
14       System.out.println("In method go. r1 " + r1 + "\n");
15       alterPointee(r1);
16       System.out.println("In method go. r1 " + r1 + "\n");
17
18       alterPointer(r1);
19       System.out.println("In method go. r1 " + r1 + "\n");
20    }
21
22    public static void alterPointee(Rectangle r)
23    { System.out.println("In method alterPointee. r " + r + "\n");
24       r.setSize(20, 30);
25       System.out.println("In method alterPointee. r " + r + "\n");
26    }
27
28    public static void alterPointer(Rectangle r)
29    { System.out.println("In method alterPointer. r " + r + "\n");
30       r = new Rectangle(5, 10, 30, 35);
31       System.out.println("In method alterPointer. r " + r + "\n");
32    }
33 }
```

4

NUMERICAL RESULTS

This is where you show that the novel 'thing' you described in Chapter 3 is, indeed, much better than the existing versions of the same.
You will probably use figures (try to use a high-resolution version), graphs, tables, and so on. An example is shown in Figure 1.

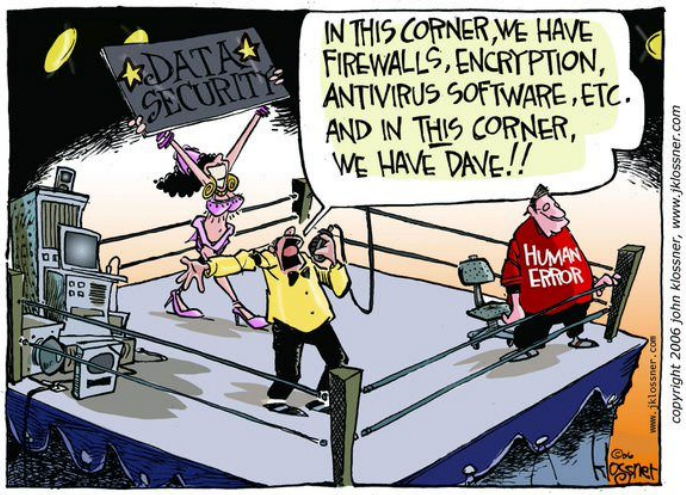

Figura 1: Network Security - the sad truth

Note that, likewise tables and listings, you shall not worry about where the figures are placed. Moreover, you should not add the file extension (LaTeX will pick the 'best' one for you) or the figure path.

5

CONCLUSIONS AND FUTURE WORK

They say that the conclusions are the shortened version of the introduction, and while the Introduction uses future verbs (we will), the conclusions use the past verbs (we did). It is basically true.
In the conclusions, you might also mention the shortcomings of the present work and outline what are the likely, necessary, extension of it. E.g., we did analyse the performance of this network assuming that all the users are pedestrians, but it would be interesting to include in the study also the ones using bicycles or skateboards.
Finally, you are strongly encouraged to carefully spell check your text, also using automatic tools (like, e.g., Grammarly[1] for English language).

1 https://www.grammarly.com/

BIBLIOGRAFIA

[1] Joshua Bloch. *Effective Java*. Addison-Wesley, 3rd edition, 2018.

[2] Ross Callon. The Twelve Networking Truths. RFC 1925, April 1996.

[3] OpenNebula Community. Opennebula. https://opennebula.io/.

[4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1994. Professional eBook edition.

[5] Donald E. Knuth. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*. Addison-Wesley, Reading, Mass., 3rd ed. edition, 1997.

[6] Andrea Margheri. Facpl documentation. https://facpl.readthedocs.io/en/latest/index.html, 2024.

[7] Andrea Margheri. Facpl: Flexible and adaptive control programming language. https://github.com/andreamargheri/FACPL, 2024.

[8] Andrea Margheri, Massimiliano Masi, Rosario Pugliese, and Francesco Tiezzi. Developing and enforcing policies for access control, resource usage, and adaptation. In Emilio Tuosto and Chun Ouyang, editors, *Web Services and Formal Methods*, pages 85–105, Cham, 2014. Springer International Publishing.

[9] John Nash. Non-cooperative games. *Annals of Mathematics*, 54(2):286–295, 1951.

[10] The Apache Software Foundation. Apache commons, 2024.