Angry IP Scanner Documentation

สมาชิกกลุ่ม

1. นาย ณภัทร ศรีนามนต์ 62010248
2. นาย ธนพล วงศ์อาษา 62010356
3. นาย ธีรัตม์ บุตรากาศ 62010448
4. นาย ปิยพัทธ์ บุญแถม 62010558
5. นาย ระพีพัฒน์ เพ็งที 62010765
6. นาย วริศ เผ่าทองศุข 62010808

GitHub Repository: https://github.com/angryip/ipscan (version 3.8.2)

ลักษณะของโปรแกรม Angry IP Scanner

Angry IP Scanner เป็นโปรแกรมที่สแกน IP ในรูปแบบต่าง ๆ ได้ โปรแกรมนี้เป็นเครื่องมือที่เหมาะสมสำหรับผู้ดูแลระบบ เครือข่าย ด้วยการใช้งานที่ง่ายและ รวดเร็วในการสแกนหา IP ต่าง ๆ ตามที่ต้องการ เช่น กำหนดโดยช่วงของไอพี (IP Range) จากหมายเลขอะไรถึงหมายเลขอะไร แบบนี้เป็นต้น โดยมันจะสามารถรู้ได้แบบ real time ว่า แต่ละเครื่องที่สแกนออกมากำลังใช้งานอะไรบน network อยู่บ้าง โดยข้อมูลที่จะแสดงออกมามีค่าได้หลากหลายรูปแบบตามที่ผู้ใช้ต้องการ เช่น ping, hostname, open port, website hosted

โปรแกรมนี้ถูกพัฒนาให้ใช้งานได้ง่าย และสามารถรันได้บนระบบปฏิบัติการที่หลากหลายได้แก่ Windows, Mac OS X, และ Linux distro ต่าง ๆ

Architecture ของ Angry IP Scanner

ตัว Software Architecture Styles ที่เห็นชัดได้มากที่สุดของ Angry IP Scanner คือ **Event-Driven Architecture**

ทางกลุ่มได้ค้นพบว่าจาก Document ของผู้ผลิตโปรแกรม Angry IP Scanner ได้ทำการระบุอย่างชัดเจนว่ามีการใช้ Design Pattern หลัก คือ Mediator ซึ่งจะมีการทำงานที่เชื่อมโยงระหว่างการทำงานในส่วน Feeder และ การทำงานในส่วน Fetcher ในการทำการ Scan ข้อมูลของ IP โดยมีลักษณะเด่นเป็นการทำงานแบบ loosely coupled software components กล่าวคือจะมีการสร้าง State การทำงานและตัว State จะส่งผลให้เกิดการทำงานในส่วนอื่นๆต่อไป

ซึ่งจาก Design Pattern หลัก ที่ค้นพบจะเห็นได้ว่าจะมีลักษณะที่เด่นชัดเหมือนกับ Architectural style แบบ Event-Driven Architecture ซึ่งมีลักษณะการทำงาน คือ เมื่อเกิดการทำงานในบางฟังก์ชันที่จำเป็นจะต้องมีการทำงานต่อเนื่องกัน จะทำการส่งค่า event ออกไปและทำให้ฟังก์ชันส่วนอื่นที่ต้องทำงานต่อจะพบ event ที่ถูกส่งมาและจะทำงานเป็นขั้นๆ ไปโดยที่ตัว event จะไม่รู้ว่าฟังก์ชันปลายทางการในการทำงานนั้นจะนำไปใช้ทำอะไรและทำที่ฟังก์ชันใดบ้าง ซึ่งทำให้แต่ละส่วนเปรียบเสมือนมีการทำงานที่ผูกมัดกันแบบหลวม ๆ (loosely coupled software components) ซึ่งมีความสอดคล้อง กับ Design Pattern หลักของโปรแกรมอย่าง Mediator

The Scanning component

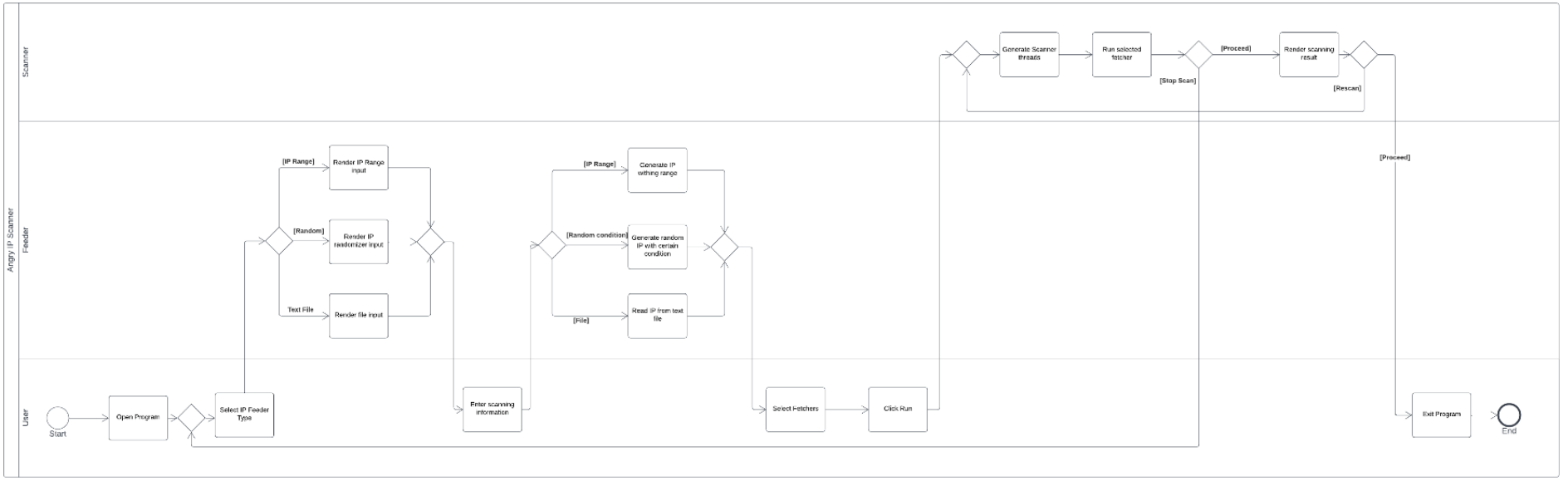
Angry IP Scanner's scanning component is implemented using the Mediator pattern, which routes messages between the user interface, generator of IP addresses (*feeder*), and information retrieving modules (*fetchers*), generating events for other components. This ensures that all components of the program are loosely coupled and therefore reusable and interchangeable.

The scanning component itself is very abstract – it knows nothing about what information is being collected. Information is gathered with the help of *fetchers* that are selected by the user. Angry IP Scanner contains a number of built-in *fetchers* (e.g. mentioned above), but additional third-party fetchers can be used with the help of plugins. This ensures very good scanning flexibility and extensibility of the program – each user can have very different and non-standard needs, especially if the user is an administrator of a large network.

During scanning, the scanning component is controlling states. In the scanning state it iterates IP addresses provided by *fetcher* and gives control to *fetchers* in order to do the actual scanning. All this would be very slow without doing most of the work in parallel.

reference: https://angryip.org/documentation/

UML Diagram ของสถาปัตยกรรม


Angry IP Scanner
Scanner
Feeder
User
Start
Open Program
Select IP Feeder Type
[IP Range]
Render IP Range input
[Random]
Render IP randomizer input
Text File
Render file input
Enter scanning information
[IP Range]
Generate IP within range
[Random condition]
Generate random IP with certain condition
[File]
Read IP from text file
Select Fetchers
Click Run
Generate Scanner threads
Run selected fetcher
[Stop Scan]
[Proceed]
Render scanning result
[Rescan]
[Proceed]
Exit Program
End

Quality Attribute ของ Angry IP Scanner

1. **Scalability**

เนื่องจากการสแกน IP หนึ่ง ๆ จะมีข้อมูลที่ต้องการดูแตกต่างกัน ข้อมูลที่ต้องการดูแต่ละอย่างจะถูกเช็คด้วย Fetcher แต่ละตัว นอกจากนั้นแล้วการสแกน IP ส่วนใหญ่จะทำการสแกนหลาย ๆ IP พร้อมกันด้วย ผู้พัฒนาจึงเลือกใช้วิธี multithread ในการสแกนนำค่าที่ต้องการออกมาดู

Tactic ที่ใช้

Fetcher แต่ละตัวจะไม่ได้ถูกเรียกทำงานโดยตรง แต่ว่าจะถูกเรียกผ่าน Scanner ที่อยู่ภายใน ScannerDispatcherThread อีกที ซึ่งจะมีหน้าที่ในการรัน Scanner ซึ่งเรียกใช้ Fetcher อีกที (กล่าวคือ ScannerDispatcherThread => Scanner หลายตัว => Fetcher หลายตัว) โดยการสร้าง thread จะสร้างได้สูงสุดถึง 100 thread ในการตั้งค่าเริ่มต้น ซึ่งถ้าต้องการเพิ่มจำนวน thread ก็สามารถแก้ไขที่ source code ให้มีจำนวน thread เพิ่มได้ ทั้งนี้ข้อจำกัดในแง่ของจำนวน thread จะขึ้นกับระบบปฏิบัติการของผู้ใช้ด้วย

Proof of Evidence

> Because the maximum number of threads may be very different, Angry IP Scanner uses no more than 100 threads at a time by default. The user has the possibility to increase this number if their hardware and software allows that, or the opposite. Some latest combinations can even handle 500 scanning threads with no problems, however this number may be close to the situation when threads finish their jobs before the scanner is able to reach the limit by starting new ones. Another limitation may be due to instability of some network adapters or their drivers (especially wireless ones) – they just cannot process so many simultaneous connections or packets, so they start loosing them, rendering scanning results unreliable. The same problem sometimes is created artificially mostly on Windows platforms by rate limiting of connection attempts (unpatched Windows XP SP2 limits to 10 incomplete outbound connection attempts at a time) with the goal of preventing scanning by worms that are unfortunately likely to get into the system.

reference: https://angryip.org/documentation/

2. **Performance**

ปกติแล้วการสแกน IP ต่าง ๆ เป็นการทำงานที่ใช้เวลาค่อนข้างนาน รวมทั้งการสแกนที่ถ้าใช้เวลานาน (ไม่ว่าจะเกิดจาก latency ของ internet หรือว่าเกิดจากความล่าช้าในการรัน process ต่าง ๆ ก็ตาม) จะส่งผลให้ได้ผลลัพธ์ที่ผิดพลาด เช่น latency ของระบบที่สูงเกิน หรือการที่นานเกินไปจนโปรแกรมตีความผิดพลาดว่า host unreachable การแก้ปัญหานี้ก็สามารถแก้ได้โดยการใช้งาน multithread เช่นเดียวกัน

Tactic ที่ใช้

การทำ multithread เพื่อสแกนค่า IP นอกจากจะตอบโจทย์ด้าน Quality Attribute ของ Scalability แล้ว ยังสามารถทำให้การสแกน IP ต่าง ๆ เป็นไปได้ด้วยความรวดเร็วอีกด้วย รวมถึงการเลือกใช้ built-in function ต่าง ๆ ซึ่งอาจจะดูไม่ใช่วิธีการใช้ที่ถูกต้องตามจุดประสงค์การใช้งานหลัก (hack) แต่ว่าสามารถทำให้รันได้เร็วขึ้นก็มีการเลือกใช้และ document ไว้เช่นเดียวกัน

Proof of Evidence

The easiest and most reliable way to make code run in parallel is the usage of threads, because in this case the operating system is dealing with all the complexity of task switching and scheduling, making programming a lot easier. The OS can even run several threads really in parallel if the machine has several CPUs, which is another great advantage over manual parallelizing. The programmer must only take care of proper synchronization.

But let's assume that the machine has only one CPU. Then, as opposed to microprocessor systems, threads cannot just magically increase the performance, especially in the case, when each thread needs 100% of processor time, which would result in performance degradation due to too frequent context switching compared to sequential program.

```
@Override public boolean hasNext() {
        // equals() is faster than greaterThan()
        return !currentIP.equals(endIP);
}
```

reference: https://angryip.org/documentation/ และ https://github.com/angryip/ipscan/blob/master/src/net/azib/ipscan/feeders/RangeFeeder.java (บรรทัดที่ 78-81)

3. **Maintainability**

Angry IP Scanner เป็นโปรแกรม open source ซึ่งถึงแม้จะดูมีความเป็น monolithic มากก็ตาม ผู้พัฒนาก็ได้เล็งเห็นถึงปัญหาตรงนี้ โดยการทำการ document โค้ดที่ดีและการเขียนโค้ดตาม best practice เพื่อให้สามารถอ่านได้ง่าย นอกจากนี้ผู้ที่ต้องการนำโปรแกรมไปพัฒนาเพิ่มเติมต่อก็สามารถนำโค้ดมาเพิ่มเติมและดัดแปลงได้โดยง่ายผ่าน interface ต่าง ๆ ที่ผู้พัฒนาได้ทำไว้ให้ ไม่ว่าจะเป็น Exporter ที่ใช้ในการแปลงผลการสแกนเป็นไฟล์ หรือ Feeder ซึ่งเป็นการป้อน IP ในรูปแบบต่าง ๆ

Tactic ที่ใช้

ออกแบบส่วนต่าง ๆ ของโปรแกรมให้แยกส่วนการทำงานอย่างชัดเจน และการเลือกใช้ class abstraction ในรูปแบบต่าง ๆ เช่น Feeder, Exporter, Fetcher ซึ่งเป็น interface ซึ่งมี concrete class ต่าง ๆ มา implement เพื่อเรียกใช้งานจริงอีกทีหนึ่ง หรือการเลือกใช้ Dependency Injection ที่ทำให้สามารถเปลี่ยนแปลงส่วนต่าง ๆ ของโปรแกรมที่ต้องการ class หรือ interface ชนิดใดชนิดหนึ่งได้ง่าย

Proof of Evidence

Angry IP Scanner is an open-source program. At first sight it may seem that open-source software can be monolithic – users will be able to extend its functionality anyway, by editing the source code. However, in most cases this is not true. Very often, in order to make a significant change in the code, developer must spend quite a lot of time reading and debugging the code to understand how it works and where to make the exact modification. And it is widely known that reading of code is often more difficult than writing, especially if the original author has not put any effort to make the code extensible.

The Linux kernel has gone this path in the past: it started with 100% monolithic code, however, as it grew and attracted more and more developers, a more modular approach was taken. Now, Linux kernel has modules, which can be either integrated into the base kernel binary, or can be loaded separately on demand. This and some other improvements resulted in much quicker development that can be easily noticed by the increased pace of kernel releases.

Thus, if a modular extensible system is in place, any individual is able to add additional functionality to the software with much less effort, because extensibility points are likely to be well documented and have simple interface. On the other hand, that allows to reduce the bloat of the original application, making the code simpler and possibly the application itself faster, because some “optional” plugins are not loaded at all if they are not used.

A plugin is usually an external software component that can be loaded dynamically in order to add or extend functionality of the base program. The internal design of Angry IP Scanner aims to be as modular as possible in order to be able to introduce even more either internal extensions or external plugins later.

reference: https://angryip.org/documentation/

จุดอ่อนของสถาปัตยกรรมและวิธีแก้ปัญหา

Portability

Windows รุ่นเก่าที่ใช้สถาปัตยกรรม NT จะใช้งานสแต็ก TCP/IP ได้ดีกว่า Windows รุ่นใหม่ ๆ เนื่องจากมีการจำกัดด้านการเชื่อมต่อขาออกต่อนาที และการยกเลิกการใช้ raw socket ที่ใช้ในการสแกนที่ซับซ้อน ในขณะที่ Open Source OS นั้นมีความเหมาะสมแก่การใช้ในงานทาง networking มากกว่าเมื่อคำนึงถึงความปลอดภัย, ความสามารถ และราคา ตัวอย่างเช่น Linux ที่สามารถรองรับ feature ของ Angry IP Scanner ได้มากกว่าและยัง scan ได้คุณภาพที่ดีกว่าและเร็วกว่า

Tactic ที่ใช้ในการแก้ปัญหา

มีการเพิ่มกลไกการตรวจสอบหาจำนวน thread สูงสุดและ timeout เมื่อเริ่มต้น Angry IP Scanner เป็นครั้งแรก จากนั้นโปรแกรมจะเปิดพอร์ตในเครื่องและพยายามสแกนเชิงรุก หรือจะทำการขอให้ผู้ใช้หาโฮสต์และพอร์ตที่เปิดอยู่ในเพื่อทำการทดสอบ หลังจากนั้นโปรแกรมจะทำการตั้งค่าต่าง ๆ ให้มีความเหมาะสมมากที่สุดกับอุปกรณ์นั้น ๆ อย่างไรก็ตามมันจะทำให้การสแกนช้าลงมากในเครื่อง Windows รุ่นใหม่ ๆ แต่ผลการสแกนจะแม่นยำกว่า

reference: https://angryip.org/documentation/

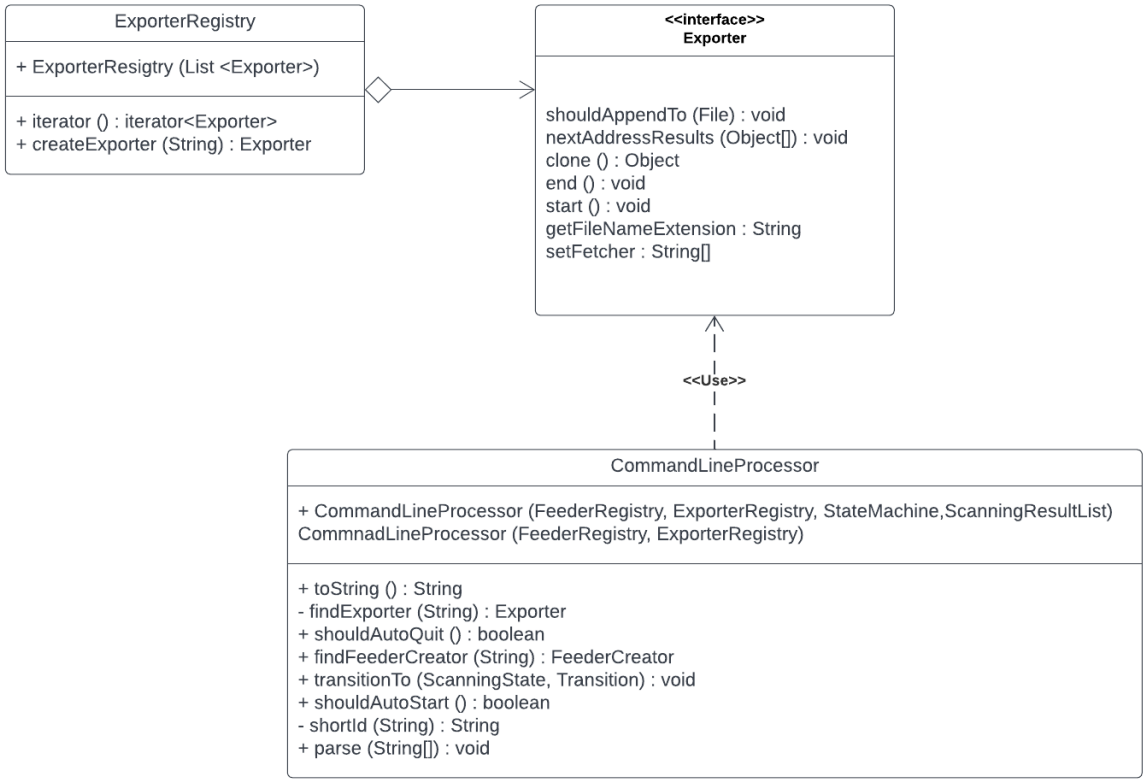
Design Pattern ของ Angry IP Scanner

1. Prototype

การใช้ Prototype pattern จะทำให้ class ที่มีการเรียกใช้ Exporter ใช้งานได้ง่ายขึ้นโดยการเรียกใช้งานฟังก์ชัน createExporter ในไฟล์ ExporterRegistry.java ที่ทำการบันทึก instance ของ Exporter ทั้งหมดที่เป็นไปได้ เพียงส่งชื่อไฟล์ไปเป็น parameter สำหรับฟังก์ชัน ก็จะได้ Exporter สำหรับชนิดไฟล์ที่ต้องการเช่น หากใส่ชื่อไฟล์เป็น scan.csv ฟังก์ชันนี้ก็จะให้ CSVExporter ออกไปใช้งาน

ข้อดีของการใช้ Pattern นี้คือโค้ดที่มีการเรียกใช้ Exporter จะไม่มีความ dependent กับ concrete class ที่ทำการ implement Exporter interface เลย ตัวอย่างการใช้งานจะพบได้ในคลาส CommandLineProcessor.java

UML Class Diagram



- https://github.com/angryip/ipscan/blob/master/src/net/azib/ipscan/exporters/Exporter.java (บรรทัดที่ 28): Exporter interface มีการ extends Cloneable

```
public interface Exporter extends Cloneable, Plugin {
```

- https://github.com/angryip/ipscan/blob/master/src/net/azib/ipscan/exporters/ExporterRegistry.java (บรรทัดที่ 43-58): มีการใช้ method clone() สำหรับการส่ง Exporter ออกไปใช้งานผ่าน method createExporter(String)

```
public Exporter createExporter(String fileName) throws ExporterException {
        int extensionPos = fileName.lastIndexOf('.') + 1;
        String extension = fileName.substring(extensionPos);

        Exporter prototype = exporters.get(extension);
        if (prototype == null) {
                throw new ExporterException("exporter.unknown");
        }
        try {
                return (Exporter) prototype.clone();
        }
        catch (CloneNotSupportedException e) {
                // this is not possible
                throw new RuntimeException(e);
        }
}
```

- https://github.com/angryip/ipscan/blob/master/src/net/azib/ipscan/config/CommandLineProcessor.java (บรรทัดที่ 29, 151): การใช้งาน method createExporter(String) ผ่าน CommandLineProcessor

```
public class CommandLineProcessor implements CommandProcessor, StateTransitionListener {
        private final FeederRegistry feederRegistry;
        private final ExporterRegistry exporters;
        private StateMachine stateMachine;
        private ScanningResultList scanningResults;
```

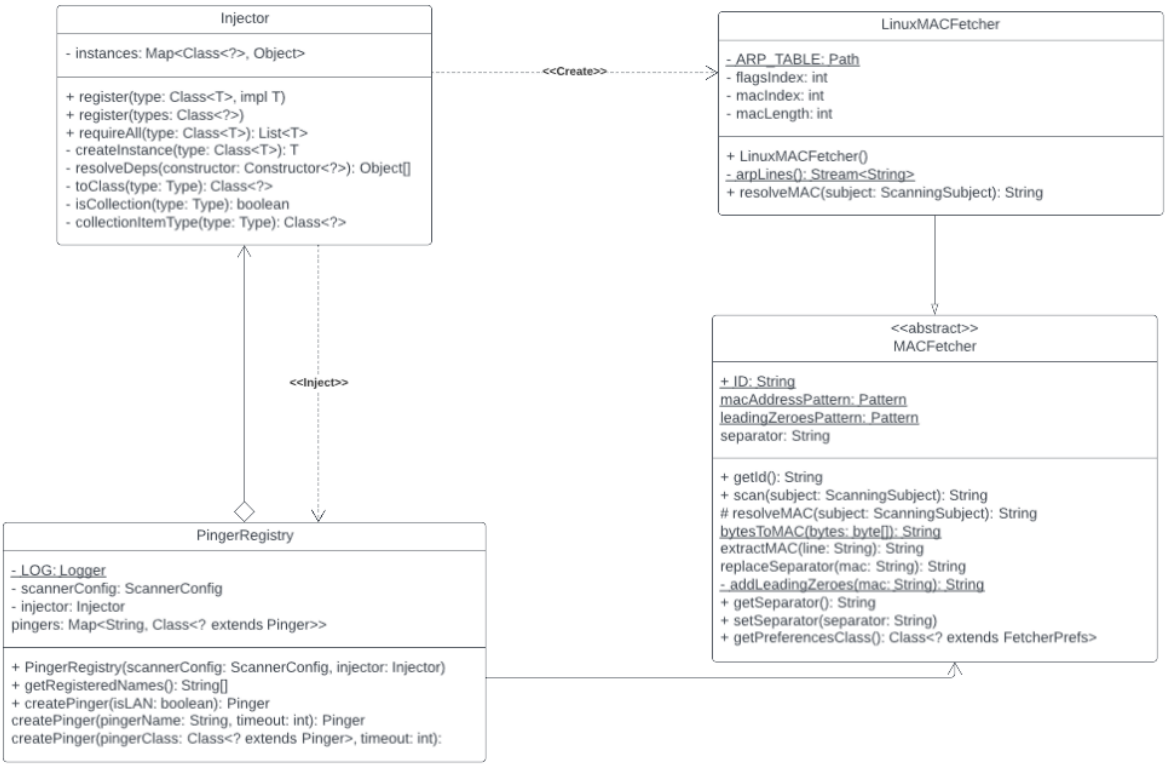
```
 private Exporter findExporter(String outputFilename) {
         return exporters.createExporter(outputFilename);
 }
```

2. Dependency Injection

เนื่องจากชิ้นส่วนต่าง ๆ ของตัว IP Scanner นี้มีมากมาย และการอนุญาตให้ developer สามารถพัฒนาส่วนเสริม (Plug-ins) เพิ่มเติมได้อีก การใช้ Dependency Injection จึงเป็น design pattern ที่ถูกเลือกใช้เพื่อให้สามารถทำการเพิ่ม plugin ต่าง ๆ และการเพิ่มเติม sub-class ใหม่และนำไปใช้แทน class ตัวเก่าก็สามารถทำได้โดยง่าย เพียงแค่เปลี่ยน class ที่จะทำการ inject เข้าไปสู่ระบบเท่านั้น

การใช้ pattern นี้ยังให้ความสะดวกในการ implementation ได้ด้วย เนื่องจากโค้ดในส่วนฟังก์ชัน require มีลักษณะคล้ายกับเป็น singleton pattern ซึ่งจะสามารถการันตีได้ว่าทั้งซอฟต์แวร์มีการใช้งาน instance ของ class ต่าง ๆ ตัวเดียวกัน ทำให้ลดการใช้ทรัพยากรได้อีกด้วย ตัวอย่างการใช้งานจะพบได้ในไฟล์ PingerRegistry.java ที่มีการขอใช้งาน MACFetcher (class ที่ extends MACFetcher ทั้งหมดได้แก่ LinuxMACFetcher,WinMACFetcher,UnixMACFetcher ในที่นี้ขอยกตัวอย่างการ inject ด้วย class LinuxMACFetcher)

UML Class Diagram



- https://github.com/angryip/ipscan/blob/master/src/net/azib/ipscan/di/Injector.java: Injector implementation

```
public class Injector {
	private final Map<Class<?>, Object> instances = new LinkedHashMap<>();
	{ register(Injector.class, this); }

	public <T> void register(Class<T> type, T impl) {
		instances.put(type, impl);
	}

	public <T> T require(Class<T> type) {
		// unfortunately, HashMap.computeIfAbsent() doesn't put values properly in a recursive scenario
		T value = (T) instances.get(type);
		if (value == null) instances.put(type, value = createInstance(type));
		return value;
	}

	public void register(Class<?> ... types) {
		stream(types).forEach(this::require);
	}

	public <T> List<T> requireAll(Class<T> type) {
		return instances.entrySet().stream().filter(e -> type.isAssignableFrom(e.getKey())).map(e -> (T) e.getValue()).collect(toList());
	}
```

- https://github.com/angryip/ipscan/blob/master/src/net/azib/ipscan/core/net/PingerRegistry.java (บรรทัดที่ 33, 68, 81, 87): การใช้ Injector เพื่อส่ง concrete class ที่เป็นชนิด MACFetcher ให้กับการสร้าง ARPPinger instance และการ register Pinger เข้าไปใน Injector

```
public class PingerRegistry {
	private static final Logger LOG = LoggerFactory.getLogger();

	private ScannerConfig scannerConfig;
	private Injector injector;

	/** All available Pinger implementations */
	Map<String, Class<? extends Pinger>> pingers;

	@SuppressWarnings("unchecked")
	public PingerRegistry(ScannerConfig scannerConfig, Injector injector) throws ClassNotFoundException {
		this.scannerConfig = scannerConfig;
		this.injector = injector;
```

```java
public Pinger createPinger(boolean isLAN) throws FetcherException {
        Class<? extends Pinger> pingerClass = pingers.get(scannerConfig.selectedPinger);
        if (pingerClass == null) {
                Map.Entry<String, Class<? extends Pinger>> first = pingers.entrySet().iterator().next();
                scannerConfig.selectedPinger = first.getKey();
                pingerClass = first.getValue();
        }
        Pinger mainPinger = createPinger(pingerClass, scannerConfig.pingTimeout);
        if (isLAN) return new ARPPinger(injector.require(MACFetcher.class), mainPinger);
        return mainPinger;
}
```

```java
Pinger createPinger(Class<? extends Pinger> pingerClass, int timeout) throws FetcherException {
        try {
                return injector.require(pingerClass);
        }
        catch (InjectException ie) {
                try {
                        Constructor<? extends Pinger> constructor = pingerClass.getConstructor(int.class);
                        Pinger pinger = constructor.newInstance(timeout);
                        injector.register((Class<Pinger>) pingerClass, pinger);
                        return pinger;
                }
                catch (Exception e) {
                        Throwable t = e instanceof InvocationTargetException ? e.getCause() : e;
                        String message = "Unable to create pinger: " + pingerClass.getSimpleName();
                        LOG.log(SEVERE, message, t);
                        if (t instanceof RuntimeException) throw (RuntimeException) t;
                        throw new FetcherException("pingerCreateFailure");
                }
        }
}
```

- https://github.com/angryip/ipscan/blob/master/src/net/azib/ipscan/fetchers/LinuxMACFetcher.java (บรรทัดที่ 11):

LinuxMACFetcher มีการ extends จาก MACFetcher จึงสามารถนำมาใช้ในการ inject เมื่อต้องการ instance ของ MACFetcher

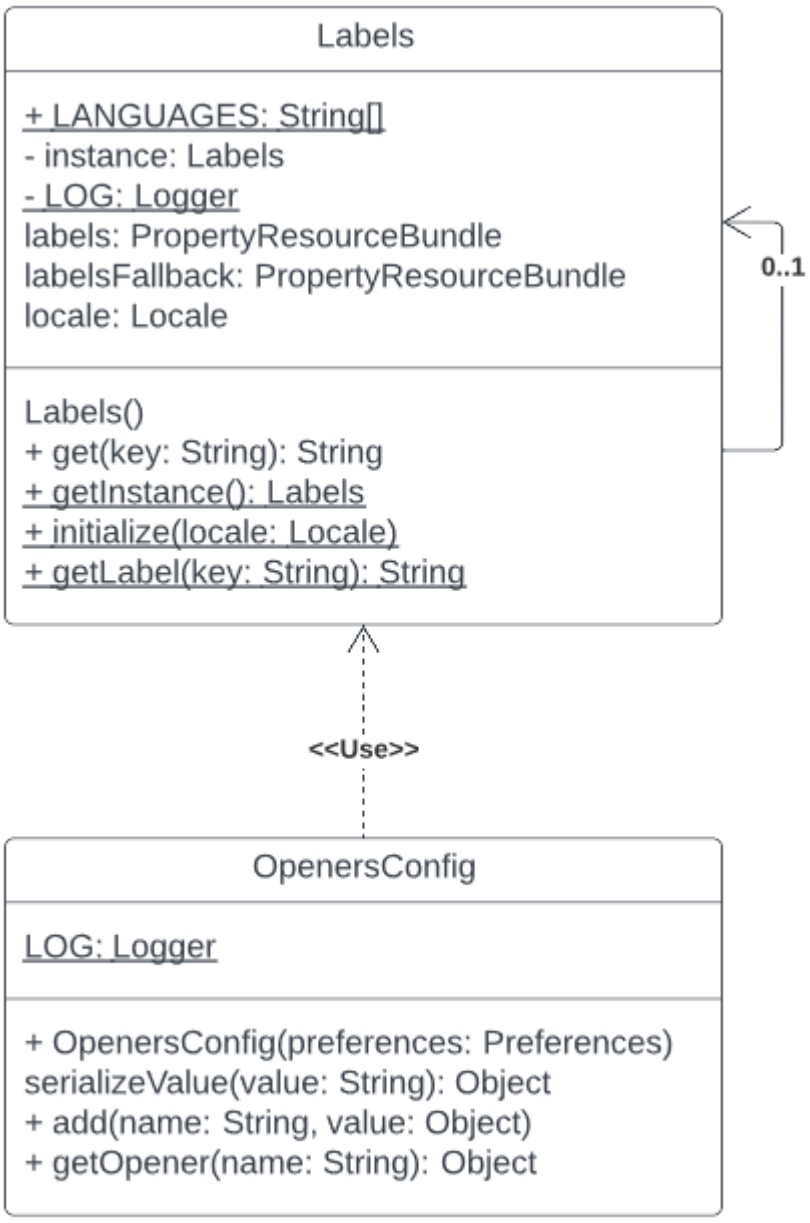
```java
public class LinuxMACFetcher extends MACFetcher {
```

3. Singleton

การแปล User Interface เป็นภาษาต่าง ๆ ถูกรองรับด้วย Class Labels ที่มีหน้าที่ในการอ่านไฟล์ messages_XX.properties เพื่อเลือกภาษาที่นำมาใช้แสดงตามที่ผู้ใช้งานต้องการ

จะเห็นว่างานในการอ่านภาษาจากไฟล์ขึ้นมาและใช้กับทุกส่วนของ User Interface เป็นงานที่ทำครั้งเดียวก็เพียงพอ (ทุกส่วนของ User Interface ใช้ภาษาเดียวกัน) ดังนั้นการเลือกใช้ Singleton Pattern จึงสามารถตอบโจทย์นี้ได้ทั้งในแง่ของความสะดวกในการใช้งานและการประหยัดทรัพยากรทั้งการอ่านไฟล์และการ initialize instance ของ class นี้ขึ้นมา

UML Class Diagram



- https://github.com/angryip/ipscan/blob/master/src/net/azib/ipscan/config/Labels.java (บรรทัดที่ 34-46, 53-83): private constructor ของ labels และการ initialize ค่าให้กับ instance

```java
public final class Labels {
	public static final String[] LANGUAGES = { "system", "en", "ru", "de", "hu", "lt", "es", "fr", "it", "ku", "tr", "gr", "pt_BR", "zh_CN", "zh_TW"};
	private static final Logger LOG = Logger.getLogger(Labels.class.getName());
	private static Labels instance;

	PropertyResourceBundle labels, labelsFallback;
	Locale locale;

	static {
		// this is needed for Visual Editor to display
		// labels at design time
		initialize(Locale.getDefault());
	}

	Labels() {
		// private constructor
	}

	public static Labels getInstance() {
		return instance;
	}
```

```java
public static void initialize(Locale locale) {
	if (instance != null && locale.equals(instance.locale)) {
		// do not reload locale, because it was already initialized in the static block
		return;
	}
	// create a new instance
	instance = new Labels();

	instance.locale = locale;
	try (InputStream labelsStream = Labels.class.getClassLoader().getResourceAsStream("messages.properties")) {
		if (labelsStream == null) {
			throw new MissingResourceException("Labels not found!", Labels.class.getName(), "messages");
		}
		instance.labelsFallback = new PropertyResourceBundle(new InputStreamReader(labelsStream, "UTF-8"));
	}
	catch (IOException e) {
		throw new MissingResourceException(e.toString(), Labels.class.getName(), "messages");
	}

	try (InputStream labelsStream = Labels.class.getClassLoader().getResourceAsStream("messages_" + locale.toString() + ".properties")) {
		instance.labels = new PropertyResourceBundle(new InputStreamReader(labelsStream, "UTF-8"));
	}
	catch (Exception e) {
		try (InputStream labelsStream = Labels.class.getClassLoader().getResourceAsStream("messages_" + locale.getLanguage() + ".properties")) {
			instance.labels = new PropertyResourceBundle(new InputStreamReader(labelsStream, "UTF-8"));
		}
		catch (Exception e2) {
			instance.labels = instance.labelsFallback;
		}
	}
}
```

- https://github.com/angryip/ipscan/blob/master/src/net/azib/ipscan/config/OpenersConfig.java (บรรทัดที่ 25): การเรียกใช้งาน Labels โดยใช้ผ่าน static method getInstance()

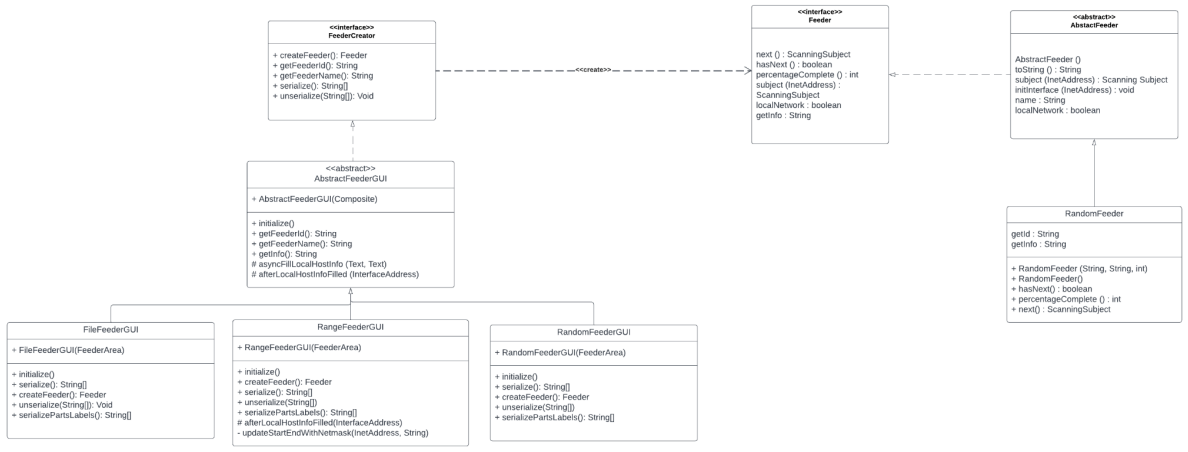
```java
public class OpenersConfig extends NamedListConfig {

	static final Logger LOG = LoggerFactory.getLogger();

	public OpenersConfig(Preferences preferences) {
		super(preferences, "openers");

		if (size() == 0) {
			Labels labels = Labels.getInstance();
```

4. Factory Method

feeder นั้นจะถูกสร้างขึ้นในการ scan แต่ละครั้งเพื่อป้อน IP ให้กับ scanner เมื่อมีการสร้างแล้วไม่ควรที่จะเปลี่ยนแปลงการตั้งค่าภายใน feeder ที่สร้างได้ โดยในการใช้ factory method ด้วยให้การสร้าง feeder แต่ละครั้งจะผ่าน createFeeder ใน FeederCreator ที่เป็น interface ของ AbstractFeederGUI

การใช้ Factory Method ในที่นี้เพื่อให้ทาง client สามารถเลือกว่าจะใช้เป็น feeder ระหว่าง IP range, random, text file ได้ และสามารถมีการใช้ feeder ซ้ำได้แทนที่จะสร้าง feeder ตัวใหม่ ถ้าทำการ rescan

UML Class Diagram



- https://github.com/angryip/ipscan/blob/master/src/net/azib/ipscan/feeders/Feeder.java (บรรทัดที่ 27) - interface Feeder

```
public interface Feeder extends Plugin {
```

- https://github.com/angryip/ipscan/blob/master/src/net/azib/ipscan/feeders/AbstractFeeder.java (บรรทัดที่ 24) - AbstractFeeder ทำการ implement interface Feeder

```
public abstract class AbstractFeeder implements Feeder {
```

- https://github.com/angryip/ipscan/blob/master/src/net/azib/ipscan/feeders/RandomFeeder.java (บรรทัดที่ 20) - RandomFeeder มีการ extend AbstractFeeder

```
public class RandomFeeder extends AbstractFeeder {
```

- https://github.com/angryip/ipscan/blob/master/src/net/azib/ipscan/feeders/FeederCreator.java (บรรทัดที่ 15-20) - interface FeederCreator มี method ชื่อ createFeeder() ซึ่งมี return type เป็น Feeder ออกมา

```
public interface FeederCreator {
        /**
         * Initializes a Feeder instance using the parameters, provided by the GUI.
         * @return initialized feeder instance
         */
        Feeder createFeeder();
```

- https://github.com/angryip/ipscan/blob/master/src/net/azib/ipscan/gui/feeders/AbstractFeederGUI.java (บรรทัดที่ 27, 54-59) - AbstractFeederGUI ทำการ implements interface FeederCreator และมีการเรียกใช้ createFeeder() แต่ยังไม่มีการ implement ตัว createFeeder()

```
public abstract class AbstractFeederGUI extends Composite implements FeederCreator {
```

```
/**
 * @return the feeder's name and the information about its current settings
 */
public String getInfo() {
        return getFeederName() + ": " + createFeeder().getInfo();
}
```

- https://github.com/angryip/ipscan/blob/master/src/net/azib/ipscan/gui/feeders/FileFeederGUI.java (บรรทัดที่ 27, 61-64) - FileFeederGUI มีการ extends AbstractFeederGUI และมีการ implement createFeeder()

```
public class FileFeederGUI extends AbstractFeederGUI {
```

```
public Feeder createFeeder() {
        feeder = new FileFeeder(fileNameText.getText());
        return feeder;
}
```

- https://github.com/angryip/ipscan/blob/master/src/net/azib/ipscan/gui/feeders/RandomFeederGUI.java (บรรทัดที่ 23, 99-102) - RandomFeederGUI มีการ extends AbstractFeederGUI และมีการ implement createFeeder()

```
public class RandomFeederGUI extends AbstractFeederGUI {
```

```
public Feeder createFeeder() {
        feeder = new RandomFeeder(ipPrototypeText.getText(), ipMaskCombo.getText(), countSpinner.getSelection());
        return feeder;
}
```

- https://github.com/angryip/ipscan/blob/master/src/net/azib/ipscan/gui/feeders/FileFeederGUI.java (บรรทัดที่ 32, 61-64) - RangeFeederGUI มีการ extends AbstractFeederGUI และมีการ implement createFeeder()

```
public class RangeFeederGUI extends AbstractFeederGUI {
```

```
public Feeder createFeeder() {
        return feeder = new RangeFeeder(startIPText.getText(), endIPText.getText());
}
```