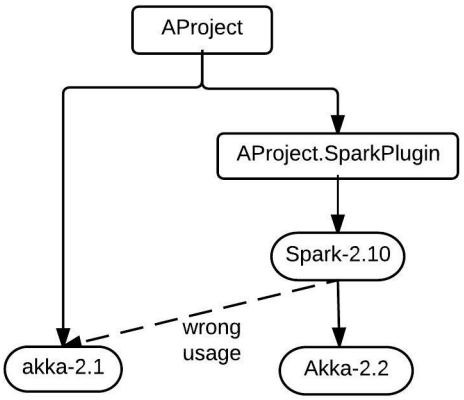
January 18, 2016

1. PROBLEM

To obtain necessary functionality, developers often use third-party libraries. Due to the complexity of current software systems, the dependency issue arises when the system depend on different and incompatible versions of the same library. This issue, called 'dependency hell' [7] may break other dependencies or push the problem to another set of libraries. We use an example to illustrate how the 'dependency hell' causes a build error, how it is localized, how it is fixed in the next session.

2. EXAMPLE



What is the 'dependency hell'?

We illustrate the building error caused by 'dependency hell' in a Maven project shown in Figure 1. As shown in part A, the parent project uses akka 2.1.1 to initialize `ActorRef` (underline) by creating a new `Props` object. The developer Alice is asked to implement a new sub-module named as SparkPlugin using Spark framework. Without knowing

(A) Code snippet in parent project using akka 2.1.1

```
/* AProject - pom.xml */
<dependency>
  <groupId>com.typesafe.akka</groupId>
  <artifactId>akka-actor_2.1</artifactId>
  <version>2.1.1</version>
<dependency>

/* AkkaSystemExecutor.java */
ActorSystem system = ActorSystem.create("MySystem");
ActorRef myActor = system.actorOf(new Props(MyUntypedActor.class),"myactor");
```

(B) Code snippet in a sub-module using akka 2.2.3 transitively

```
/* AProject.sparkPlugin - pom.xml */
<parent>
  <artifactId>AProject</artifactId>
  <groupId>..</groupId>...
</parent>

<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-core_2.10</artifactId>
  <version>1.0.0</version>
</dependency>
/* SparkUtility.java */
SparkConf sparkConf = new SparkConf();
JavaSparkContext sc = new JavaSparkContext(sparkConf);...
```

(C) Building error when integrating SparkPlugin with parent projects

```
java.lang.NoSuchMethodException: akka.remote.RemoteActorRefProvider.<init>
(java.lang.String, akka.actor.ActorSystem$Settings, akka.event.EventStream,
akka.actor.Scheduler, akka.actor.DynamicAccess)
 at java.lang.Class.getConstructor0(Class.java:3082)...
```

(D) code snippet in spark-core_2.10

```
/* spark-core_2.10, version 1.0.0 pom.xml */
 <properties> ...
    <java.version>1.6</java.version>
    <scala.version>2.10.4</scala.version>
    <akka.group>org.spark-project.akka</akka.group>
    <akka.version>2.2.3-shaded-protob</akka.version>
</properties>
...
<dependency>
  <groupId>$akka.group</groupId>
  <artifactId>akka-remote_$scala.binary.version</artifactId>
</dependency>
//some simplification by tranfering the raw scala code to Java code
ActorRef myActor = system.actorOf(Props.create(MyUntypedActor.class),"myactor");
```

Figure 1: Example from shifu.ml building error

that Spark is dependent on akka-2.2.3, she implements the entire sub-module with Spark-1.0.0, writes the tests, and fully tests the single submodule before integrating with the parent project. However, she encounters the building error shown as part (C) (The complete building error is shown at [1]).

How to localize 'dependency hell'?

Starting from the `NoSuchMethodException`, she investigates the `akka.remote.RemoteActorRefProvider.<init>` in akka-2.2.3 as declared in Spark's pom.xml and finds the static `Props.create` method shown in part (D). Alice get confused on the `NoSuchMethodException` and she has to use step-by-step debugging. She finally notices that the maven build system mistakenly uses akka-2.1.1 inherited from parent project, rather than the akka-2.2.3 that is required for Spark library.

How to fix 'dependency hell'?

Alice has several options to fix this bug as below:

1. change akka version in parent class to 2.2+, which is not feasible as the parent project heavily uses akka and it requires tremendous effort for upgrading.

2. exclude all transitive dependencies inherited from parent project with the feature provided by Maven 3.2.2+ [2], aiming to resolve dependency hell [4]. This approach is not feasible as well because SparkPlugin itself relies on the parent projects and Alice has to re-import all dependencies that the submodule uses.

3. include both akka versions and enforce Spark to use akka-2.2.3. This approach is similar to the well-known side-by-side mechanism used in NIX package manager for Linux-based system [5]. This might fix the build error, but Alice is concerned that compiling both versions might increase the building time of the entire system.

4. exclude akka-2.1.1 from the dependency declaration of Spark-1.0.0. Alice decides to use this solution as it won't have ripple effect on the rest of the system.

Using this example, we illustrate that it is not easy to localize dependency hell without knowing the entire dependency graph and it can be error-prone when trying to fix the error.

3. APPROACH

We propose an approach to check compatibility on the interfaces that use different versions for the same libraries, which has the possibility to introduce 'dependency hell'. Instead of strict backward compatibility checker for the entire class [6], we regard it as safe if the used API is compatible for the other version in the current usage context. We make this trade-off because developers often embrace changes, making multiple versions incompatible to each other [3]. To save build time and space, we include multiple versions of the files from the same library only if they are incompatible with each other, instead of maintaining multiple versions for the same library at the same time. To save time for analysis, we only consider the API calls when it has the possibility to generate 'dependency hell', since there is no need to check the dependency issue if the libraries are consistent used in the system with a single version.

4. REFERENCES

[1] Build error. In *https://travis-ci.org/lisahua/shifu/builds/29112815*, last visited 1/15/2016.

[2] Maven release note. In *https://maven.apache.org/docs/3.2.1/release-notes.html*, last visited 1/15/2016.

[3] C. Bogart, C. Kästner, and J. Herbsleb. When it breaks, it breaks: How ecosystem developers reason about the stability of dependencies. In *Proceedings of the ASE Workshop on Software Support for Collaborative and Global Software Engineering (SCGSE) 2015*, 2015.

[4] T. O. Brien. Maven-3.2.1 provides a new cure for dependency hell. In *http://discursive.com/2014/03/17/maven-3-2-1-provides-a-new-cure-for-dependency-hell/*, last visited 1/15/2016.

[5] P. Prins, J. Suresh, and E. Dolstra. Nix fixes dependency hell on all linux distributions. In *http://archive09.linux.com/feature/155922*, last visited 1/15/2016.

[6] Y. Welsch and A. Poetzsch-Heffter. Verifying backwards compatibility of object-oriented libraries using boogie. In *Proceedings of the 14th Workshop on Formal Techniques for Java-like Programs, FTfJP 2012, Beijing, China, June 12, 2012*, pages 35–41, 2012.

[7] Wikipedia. Dependency hell. In *https://en.wikipedia.org/wiki/Dependency_hell*, last visited 1/15/2016.