

AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY

FACULTY OF COMPUTER SCIENCE,
ELECTRONICS AND TELECOMMUNICATIONS



TOOL FOR MONITORING OF DISTRIBUTED APPLICATIONS IN AKKA TOOLKIT

USER DOCUMENTATION

Michał Ciołczyk
Mariusz Wojakowski

TUTOR
Maciej Maławski, Ph.D

KRAKÓW 2016

Table of Contents

[1 Code repositories](#)

[2 Requirements](#)

[2.1 Database driver](#)

[2.2 Akka Remote](#)

[2.3 AspectJ](#)

[3 Instruction manual](#)

[3.1 Step by step tutorial](#)

[3.1.1 plugins.sbt file](#)

[3.1.2 akka_tracing.conf file](#)

[3.1.3 application.conf file](#)

[3.1.4 Mixing TracedActor trait](#)

[3.2 Changing database driver](#)

[3.3 Using visualization tool](#)

[4 Configuration](#)

[4.1 akka_tracing.conf file](#)

[4.1.1 remote key](#)

[database key](#)

[akka key](#)

[4.1.2 packages key](#)

[4.2 application.conf file](#)

[5 Examples](#)

[5.1 Simple scenario](#)

[5.2 One-to-many scenario](#)

[References](#)

1 Code repositories

Source code is available online on GitHub service at the following address: <https://github.com/akka-tracing-tool>. At the time of creating this part of the documentation *Akka-Tracing-Tool* consists of 6 repositories:

- akka-tracing-core - the core part of the Akka Tracing library,
- akka-tracing-docs - the documentation of the Akka Tracing library,
- akka-tracing-sbt - SBT plugin for Akka Tracing Tool,
- akka-tracing-examples - examples of usage of the Akka Tracing library,
- akka-tracing-tutorial - simple project that explains how to use Akka Tracing Tool,
- akka-tracing-visualization - visualization tool for visualizing the traces.

Main functionality, i.e. tracing is provided by the **core part** and the **SBT plugin**. Remaining repositories provide showcase of usage and extensions.

2 Requirements

In order to meet all library requirements, it's necessary to provide few more dependencies into the project. Important goal of this tool was to minimize required configuration and make it user friendly.

2.1 Database driver

One of the most important is driver to database: it was impossible to equip library with all possible ones that would be in use. Instead developer has to provide the driver that meets his requirements best.

Under the hood we're relying on *Slick*, which is database query and access library from *Typesafe*. There is only one restriction: your database engine must be conform with it.

Probably the most useful database engine for project that doesn't involve complex persistence layer is *SQLite*. According to *SQLite*'s website [1], it's "*a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine.*" Drivers to use that database engine are widely available in public repositories and it's very easy to integrate them into project.

2.2 Akka Remote

Next important requirement that enables collecting messages is library Akka Remote. It permits to communicate between different actor systems. When project actually includes this dependency, no additional configuration is needed. Otherwise, a simple configuration is

necessary to introduce. However, in most cases example configuration file available on *akka-tracing-examples* is enough.

2.3 AspectJ

Our library use *runtime-weaving* to instrument actors. Hence to make tracing possible, it's essential to introduce artifacts connected with AspectJ library: *Runtime* and *Weaver*. They are also widely available in public repositories. Also, our plugin includes them into project automatically and provide necessary options for JVM so you don't need to bother it.

3 Instruction manual

This part of documentation will be presented in tutorial form. First, you need to clone the repository <https://github.com/akka-tracing-tool/akka-tracing-tutorial>:

```
$ git clone https://github.com/akka-tracing-tool/akka-tracing-tutorial
```

Downloaded project contains already instrumented, using our tracing tool, Akka application. Actors are passing messages in a way that is shown in Figure 1:

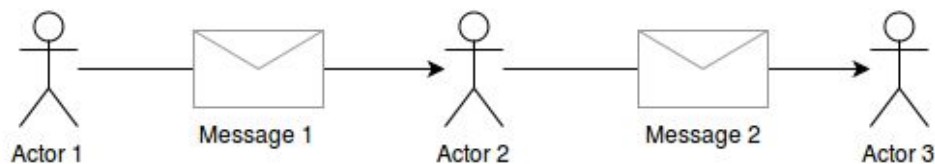


Figure 1. Messages flow in Akka application

Project structure is following:

```
+ project
|   - build.properties (1)
|   - plugins.sbt (2)
+ src/main
|   + resources
|   |   - akka_tracing.conf (3)
|   |   - application.conf (4)
|   + scala/pl/agh/edu/iet/akka_tracing/tutorial (5)
|   |   + actors
|   |   |   - FirstActor.scala
|   |   |   - SecondActor.scala
|   |   |   - ThirdActor.scala
|   |   - Runner.scala
|   - build.sbt (6)
```

Below the most important files and directories are briefly described:

1. `build.properties` - contains information about SBT version.
2. `plugins.sbt` - convention says that here are put information connected with plugins. This project only include our plugin for tracing and dependencies, hence here you can see one method for adding SBT plugin (using *`addSbtPlugin`*) and its dependencies.
3. `akka_tracing.conf` - contains configuration for an actor system responsible for collecting traces, database driver class with settings for it and package names that should be instrumented.
4. `application.conf` - settings that enable remote capabilities in project being traced.
5. source code - all actor classes and runner are in proper directories according to packages. Additionally, every actor being traced has to mix trait `TracedActor`.
6. `build.sbt` - contains another important settings: library dependencies and enabling *`Akka Tracing`* plugin.

3.1 Step by step tutorial

3.1.1 plugins.sbt file

Before instrumenting with our plugin usual content of this file is small and contains one line:

```
logLevel := Level.Warn
```

We need to use method *`addSbtPlugin`* to explicitly include plugin in proper version into the project. Below we need to provide URL path where our plugin is available. It's done by adding new resolver to Bintray's repository. Also it's necessary to add dependencies to database driver. It's used by collector to persist information. In our example we're going to use SQLite database, hence you should provide SQLite JDBC dependency. Content of *`plugins.sbt`* after these additions is following:

```
logLevel := Level.Warn

addSbtPlugin("pl.edu.agh.iet" % "akka-tracing-sbt" % "0.0.2.1")

resolvers += Resolver.url("Akka Tracing",
url("https://dl.bintray.com/salceson/maven/"))(Resolver.ivyStylePatterns)

libraryDependencies += Seq(
  "org.xerial" % "sqlite-jdbc" % "3.8.11.1"
)
```

3.1.2 akka_tracing.conf file

This file contains main configuration settings for plugin and it's expected to be in *resources* directory. Of course, the name of the file is customizable: the only thing you need to change is the value of the key `aspectsConfigurationFile` in the plugin. Below the content of `akka_tracing.conf` is shown:

```
akka_tracing {
  remote {
    database {
      driver = "slick.driver.SQLiteDriver$"
      db {
        driver = "org.sqlite.JDBC"
        url = "jdbc:sqlite:akka-tracing-tutorial.sqlite"
      }
    }
    akka {
      loglevel = "WARNING"
      actor {
        provider = "akka.remote.RemoteActorRefProvider"
      }
      remote {
        enabled-transport = ["akka.remote.netty.tcp"]
        netty.tcp {
          hostname = "127.0.0.1"
          port = 0
        }
      }
    }
  }
}
packages = [
  "pl.edu.agh.iet.akka_tracing.tutorial.actors"
]
```

3.1.3 application.conf file

If your project doesn't use Akka Remote package, this file probably doesn't exist. It enables remoting capabilities without any further changes in source code - only by providing this configuration file:

```
akka {
  actor {
    provider = "akka.remote.RemoteActorRefProvider"
  }
  remote {
    enabled-transport = ["akka.remote.netty.tcp"]
    netty.tcp {
      hostname = "127.0.0.1"
      port = 0
    }
  }
}
```

You also have to ensure that Akka Remote libraries and driver to database are available in the main project. It's done by providing dependencies:

```
val AkkaVersion = "2.3.9"

libraryDependencies += Seq(
  "com.typesafe.akka" %% "akka-actor" % AkkaVersion,
  "com.typesafe.akka" %% "akka-remote" % AkkaVersion,
  "org.xerial" % "sqlite-jdbc" % "3.8.11.1",
  ...
)
```

3.1.4 Mixing TracedActor trait

Every actor being traced has to mix TracedActor trait - it is necessary to provide messages correlation functionality. Initial implementation for one actor looks like this:

```
class SecondActor(val actorRef: ActorRef) extends Actor {
  override def receive: Receive = {
    case a =>
      actorRef ! a
  }
}
```

Firstly, you need to enable plugin in your project. It will include *Akka Tracing Core* dependency without manually adding it through *libraryDependencies* setting. It's done by this line in *build.sbt* file:

```
lazy val root = (project in file(".")).enablePlugins(AkkaTracingPlugin)
```

That's all changes you have to provide into your build configuration files! This plugin automatically include any remaining dependencies. Then, if you're using some kind of IDE, you should easily import TracedActor from `pl.edu.agh.iet.akka_tracing` package. Next step is to mix that into actor class as following:

```
import pl.edu.agh.iet.akka_tracing.TracedActor

...

class SecondActor(val actorRef: ActorRef) extends Actor with TracedActor {
  override def receive: Receive = {
    case a =>
      actorRef ! a
  }
}
```

Now you can run your application in the same way as previous: all necessary initializations depends on compilation phase and will be done automatically. Let's run application using SBT command:

```
$ sbt run
```

You can see that after database initialization a new file is created in the project root directory: `akka-tracing-tutorial.sqlite`. It contains data for SQLite database. We can check collected data using *sqlliteman* program:

```
$ sqlliteman akka-tracing-tutorial.sqlite
```

In Figure 1. you can see a screenshot from that program. Database contains two tables - *messages* and *relation* - filled with information collected from your Akka application.

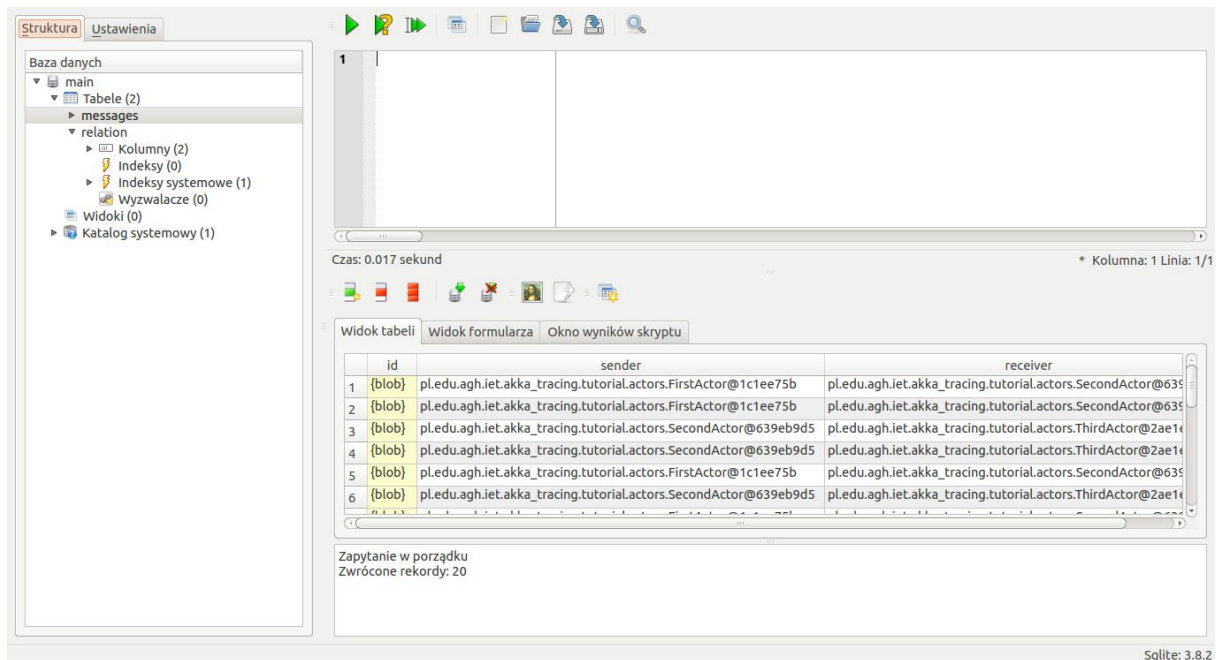


Figure 1. Screenshot from *sqlliteman*

3.2 Changing database driver

Changing database driver is easy. You need to provide different dependencies into your configuration and make sure that plugin can access that database. We show this on example of PostgreSQL database.

First, you need to change dependencies. It has to be done in 2 files: `build.sbt` and `project/plugins.sbt`. In `build.sbt` you need to change this line:

```
libraryDependencies += Seq(  
  ...  
  "org.xerial" % "sqlite-jdbc" % "3.8.11.1",  
  ...  
)
```


Into this:

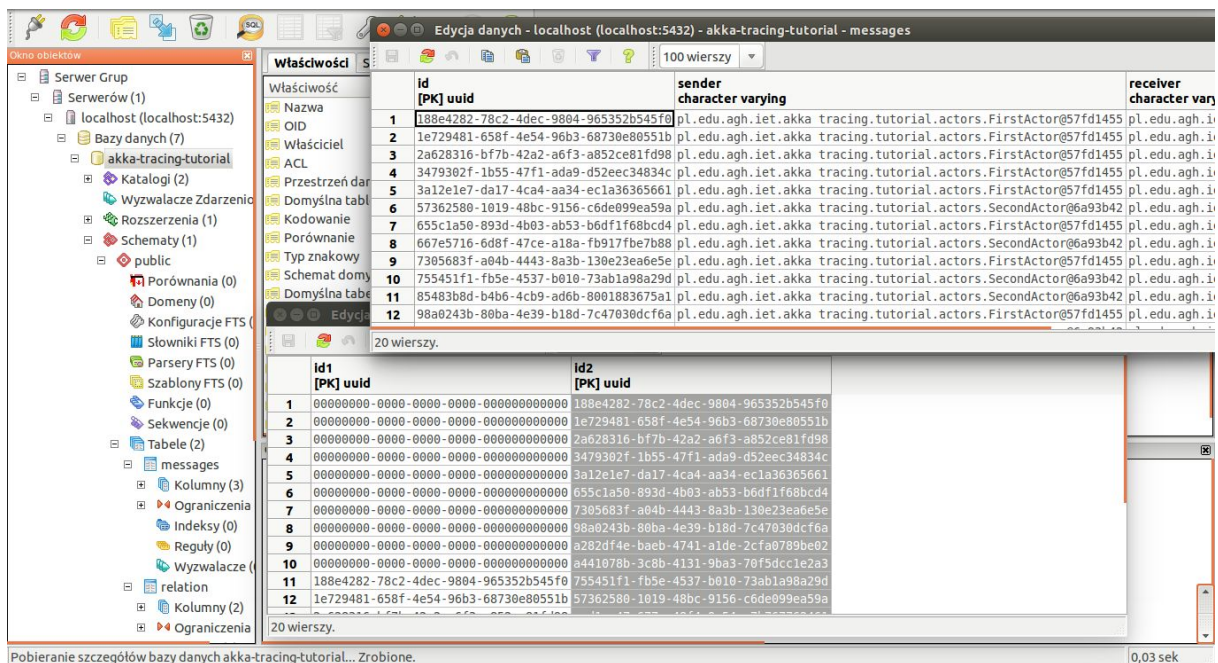
```
libraryDependencies ++= Seq(  
  ...  
  "org.postgresql" % "postgresql" % "9.4-1201-jdbc41",  
  ...  
)
```

The same change has to be done in project/plugins.sbt.

Last change you need to introduce is in akka_tracing.conf. It's necessary to provide correct drivers and parameters for connection. Below is presented one possible configuration:

```
akka_tracing {  
  remote {  
    database {  
      driver = "slick.driver.PostgresDriver$"  
      db {  
        driver = "org.postgresql.Driver"  
        url = "jdbc:postgresql://localhost:5432/akka-tracing-tutorial"  
        user = "postgres"  
        password = "postgres"  
      }  
    }  
  }  
  ...  
}
```

Now it's ready! You can run application and check results. In Figure 2. we're showing output from *pgadmin3*.



id [PK] uuid	sender character varying	receiver character varying
188e4282-78c2-4dec-9804-965352b545f0	pl.edu.agh.iet.akka_tracing.tutorial.actors.FirstActor@57fd1455	pl.edu.agh.iet.akka_tracing.tutorial.actors.FirstActor@57fd1455
1e729481-658f-4e54-96b3-68730e80551b	pl.edu.agh.iet.akka_tracing.tutorial.actors.FirstActor@57fd1455	pl.edu.agh.iet.akka_tracing.tutorial.actors.FirstActor@57fd1455
2a628316-bf7b-42a2-a6f3-a852ce81fd98	pl.edu.agh.iet.akka_tracing.tutorial.actors.FirstActor@57fd1455	pl.edu.agh.iet.akka_tracing.tutorial.actors.FirstActor@57fd1455
3479302f-1b55-47f1-ada9-d52ecc34834c	pl.edu.agh.iet.akka_tracing.tutorial.actors.FirstActor@57fd1455	pl.edu.agh.iet.akka_tracing.tutorial.actors.FirstActor@57fd1455
3a12e1e7-da17-4ca4-aa34-ec1a36365661	pl.edu.agh.iet.akka_tracing.tutorial.actors.FirstActor@57fd1455	pl.edu.agh.iet.akka_tracing.tutorial.actors.FirstActor@57fd1455
57362580-1019-48bc-9156-c6de099ea59a	pl.edu.agh.iet.akka_tracing.tutorial.actors.SecondActor@6a93b42	pl.edu.agh.iet.akka_tracing.tutorial.actors.SecondActor@6a93b42
655c1a50-893d-4b03-ab53-b6df1f68bcd4	pl.edu.agh.iet.akka_tracing.tutorial.actors.FirstActor@57fd1455	pl.edu.agh.iet.akka_tracing.tutorial.actors.FirstActor@57fd1455
667e5716-6d8f-47ce-a18a-fb917fb7b88	pl.edu.agh.iet.akka_tracing.tutorial.actors.SecondActor@6a93b42	pl.edu.agh.iet.akka_tracing.tutorial.actors.SecondActor@6a93b42
7395683f-a04b-4443-8a3b-130e23ea6e5e	pl.edu.agh.iet.akka_tracing.tutorial.actors.FirstActor@57fd1455	pl.edu.agh.iet.akka_tracing.tutorial.actors.FirstActor@57fd1455
755451f1-fb5e-4537-b010-73ab1a98a29d	pl.edu.agh.iet.akka_tracing.tutorial.actors.SecondActor@6a93b42	pl.edu.agh.iet.akka_tracing.tutorial.actors.SecondActor@6a93b42
85483b8d-b4b6-4cb9-ad6b-8001883675a1	pl.edu.agh.iet.akka_tracing.tutorial.actors.SecondActor@6a93b42	pl.edu.agh.iet.akka_tracing.tutorial.actors.SecondActor@6a93b42
98a0243b-80ba-4e39-b18d-7c47030dcf6a	pl.edu.agh.iet.akka_tracing.tutorial.actors.FirstActor@57fd1455	pl.edu.agh.iet.akka_tracing.tutorial.actors.FirstActor@57fd1455

Figure 2. Screenshot from *pgadmin3*

3.3 Using visualization tool

When you have all information in database, you can process the data and e.g. visualize them. On *Github* repository is available simple tool for visualizing traces collected using Akka Tracing Tool. In order to use it, please clone the repository:

<https://github.com/akka-tracing-tool/akka-tracing-visualization> by running the command:

```
$ git clone https://github.com/akka-tracing-tool/akka-tracing-visualization
```

Downloaded project contains Play application which can visualize traces in web browser. You need only to provide configuration file. Example is available in the repository under the name `database.conf.example`:

```
//Postgres configuration
driver = "slick.driver.PostgresDriver$"
db {
  connectionPool = disabled
  driver = "org.postgresql.Driver"
  url = "jdbc:postgresql://localhost:5432/<DB_NAME>"
  user = ""
  password = ""
  numThreads = 10
}

//SQLite configuration
driver = "slick.driver.SQLiteDriver$"
db {
  driver = "org.sqlite.JDBC"
  url = "jdbc:sqlite:<<FILE>>"
}
```

Here you can see example of configuration for 2 drivers. Below is shown minimal configuration for PostgreSQL:

```
driver = "slick.driver.PostgresDriver$"
db {
  driver = "org.postgresql.Driver"
  url = "jdbc:postgresql://localhost:5432/akka-tracing-tutorial"
  user = "postgres"
  password = "postgres"
}
```

When all configuration changes are done you can run Play application:

```
$ sbt run
```

Under the URL `localhost:9000` you can see visualization of collected traces, similar to shown in Figure 3.

Akka Tracing Visualization Tool

This is a simple tool which enables to see the messages passed between actors.

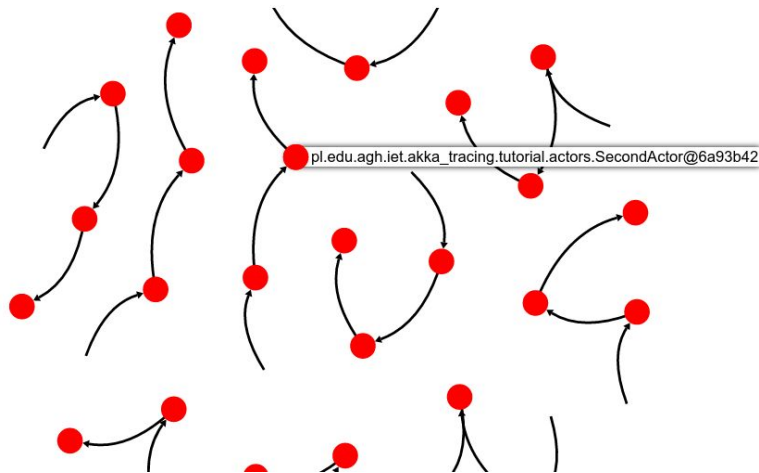


Figure 3. Screenshot from web browser showing *Akka Tracing Visualization Tool*

4 Configuration

Configuration needed by Akka Tracing Tool is divided into 2 files:

- akka_tracing.conf
- application.conf

Below we describe each of the configuration files.

4.1 akka_tracing.conf file

The akka_tracing.conf file contains information about 3 things:

- settings for database
- packages with actors being instrumented
- Akka Remote configuration of collector

In order to make configuration readable and looking similar to Typesafe's format, we've adopted HOCON format. HOCON stands for Human-Optimized Config Object Notation and is widely used in Akka configuration. Whole configuration in akka_tracing.conf is available as a value under the key akka_tracing. This object consists of 2 parts: remote and packages keys.

```
akka_tracing {
  remote { ... }
  packages = [ ... ]
}
```

4.1.1 remote key

Settings under the remote key contains information for database connection - under the database key and Akka Remote configuration - under akka key.

database key

The database key consists of: driver and db keys. First key, driver, has string value which specifies Slick's driver for chosen database. Value for it should be picked from classes available in Slick library in `slick.driver` package. Some of them are listed below:

- `H2Driver$`
- `MySQLDriver$`
- `PostgresDriver$`
- `SQLiteDriver$`

Slick provides drivers for most widely used database systems. Most of them are included in the Slick's core package.

Three commercial database systems (Microsoft SQL Server, Oracle and IBM DB2) have their Slick drivers in the separate dependency called `slick-extensions`. This part of the Slick library is closed-sourced and it is not free for use in the production environment. If you want to use it, the following code:

```
libraryDependencies += "com.typesafe.slick" %% "slick-extensions" % "3.1.0"
resolvers += "Typesafe Releases" at "http://repo.typesafe.com/typesafe/maven-releases/"
```

should be added to both `build.sbt` and `plugins.sbt` files in your project. For more information about `slick-extensions` please see the official Slick documentation [2].

Settings under db key depend on choice of database system. Basically, you need to provide values for JDBC driver to your database - under the driver key and url to database - under url key. Sometimes it's necessary to introduce other settings, e.g. user, password, maxConnections etc.

```
akka_tracing {
  remote {
    database {
      driver = "slick.driver.SQLiteDriver$"
      db {
        driver = "org.sqlite.JDBC"
        url = "jdbc:sqlite:simple-scenario.sqlite"
      }
    }
    ...
  }
  ...
}
```

akka key

Under akka key you provide Akka Remote configuration. All settings under this key are directly fed into collector's actor system to enable remoting functionalities. Format of the value for this key is created by Akka Remote developers - we've adopted it to ease developers usage of our tool. For more information please see Akka documentation about Remoting [3].

```
akka_tracing {
  remote {
    ...
    akka {
      loglevel = "WARNING"
      actor {
        provider = "akka.remote.RemoteActorRefProvider"
      }
      remote {
        enabled-transport = ["akka.remote.netty.tcp"]
        netty.tcp {
          hostname = "127.0.0.1"
          port = 0
        }
      }
    }
  }
  ...
}
```

4.1.2 packages key

Key packages contains array with package names as strings. It's necessary to produce resource with information about weaving: in which packages instrument actors and collect data. AspectJ reads configuration from that file and based on that injects advice code into actors. You have to specify at least one package name.

```
akka_tracing {
  ...
  packages = [
    "pl.edu.agh.iet.akka_tracing.examples.actors"
  ]
}
```

4.2 application.conf file

This configuration file contains informations for application's actor system. If you haven't used in your project Akka Remote, you need to provide different ActorProvider and some other settings because of remoting capabilities. Structure is presented below and it's the same as akka_tracing.remote.akka key as in the file akka_tracing.conf (see Section 4.1.1 - "akka key").

```
akka {  
  actor {  
    provider = "akka.remote.RemoteActorRefProvider"  
  }  
  remote {  
    enabled-transport = ["akka.remote.netty.tcp"]  
    netty.tcp {  
      hostname = "127.0.0.1"  
      port = 0  
    }  
  }  
}
```

5 Examples

As earlier mentioned, in Akka Tracing Tool organization on GitHub, there is available one repository with examples. At the time of creating this documentation, that repository contains 2 examples. Below are briefly described scenarios used in that project.

5.1 Simple scenario

This example provides simple scenario where actors passing messages in chain: from the Actor 1, through Actor 2 to Actor 3. Messages flow is shown in Figure 4.

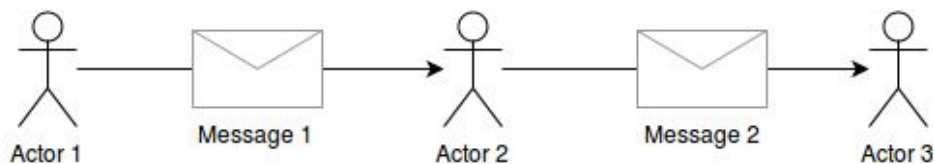


Figure 4. Simple scenario messages flow

It's very common scenario in real world application.

5.2 One-to-many scenario

In this scenario we've tried to simulate situation when one actor send something to more than one receiver. In that case, one actor passes the same message to bunch of different actors. You can see messages flow in Figure 5.

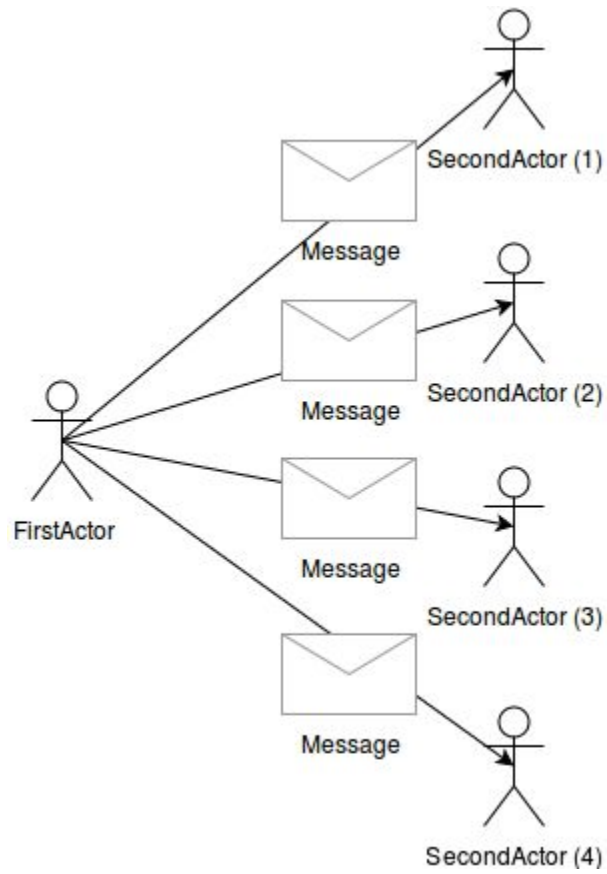


Figure 5. One-to-many scenario messages flow

This situation is similar to master-worker pattern with redundancy, which is common in large scale computation.

References

- [1] SQLite project website. Available at: <https://www.sqlite.org>
[Online; accessed 01.01.2016]
- [2] Slick documentation, “Extensions” section. Available at:
<http://slick.typesafe.com/doc/3.1.1/extensions.html> [Online; accessed 03.01.2016]
- [3] Akka documentation, “Remoting” section. Available at:
<http://doc.akka.io/docs/akka/snapshot/scala/remoting.html>
[Online; accessed 31.12.2015]