アクターシステムを対象とした リバースデバッギングフレームワーク Actoverse への AOP の導入

柴内一宏・渡部卓雄 (東京工業大学)

概要

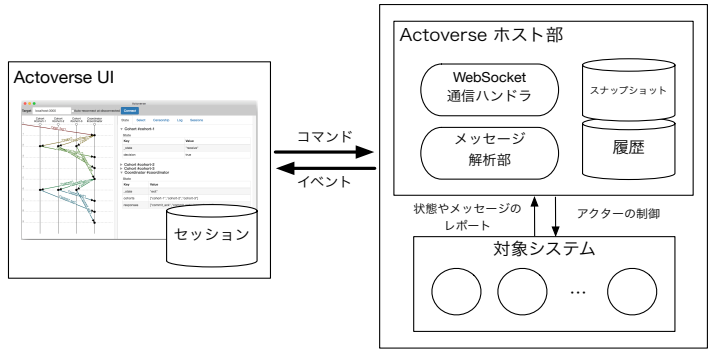
・Actoverseは，アクターモデルにもとづく各種言語および実行系を対象としたデバッギングフレームワークである
・しかしながら，デバッグ対象となるアプリケーションのソースコードに変更を行う必要があった
・AspectJの導入によってこの問題を解決した

Actoverse https://github.com/45deg/Actoverse

アクターシステムのためのデバッギングフレームワーク
[Shibanai&Watanabe AGERE2017]

- リバースデバッギング
- レコード-リプレイデバッギング
- メッセージ到達順序の制御
- メッセージシーケンスの可視化

対象システムとして Akka Actor (Scala) が使用できる.



AspectJ の導入

Java VM 向けのアスペクト指向フレームワーク AspectJ を Akka Actor を対象としたデバッガフレームワークに導入した．実行時ウィービングにより対象のプログラムに対しデバッグ機構が導入される.

タイミング	AspectJ ポイントカットアノテーション	処理概要
アクターシステムの開始時	`@After("execution(* akka.actor.ActorSystemImpl.start(..)) && !cflow(execution(void akka.actoverse.DebuggingSystem.introduce(*))")`	デバッグシステムの導入
アクターの生成時	`@After("execution(* akka.actor.Actor.preStart(..))")`	デバッグシステムへの登録
メッセージ送信時	`@Around("execution(* akka.actor.ScalaActorRef+.$bang(..))")`	追跡情報の付加 デバッガへのレポート
メッセージ受信時	`@Around("execution(* akka.actor.Actor.aroundReceive(..))")`	追跡情報の取得 内容に基づいたメッセージ制御 デバッガへのレポート

導入結果

以前は，デバッガ利用の際にはソースコードの書き換えを行う必要があった．AspectJの導入によって，実際のコードに手を加えずとも，外部の実行オプション指定(sbt) によってデバッグが可能となった.

以前のデバッグ用コード

```
class Pong extends Actor with DebuggingSupporter {
  val receive: Receive = {
    case "ping" => sender !+ "pong"
  }
}
```

AspectJ 適用後 (オリジナルのコードと同等)

```
class Pong extends Actor {
  val receive: Receive = {
    case "ping" => sender ! "pong"
  }
}
```

実験

メッセージを互いに送信し合うサンプルコードについて実行時間の計測を行った．AOPを導入した場合30-50%のオーバーヘッドが生じた．これは，cflow ポイントカットによる影響が大きいと考えられる.