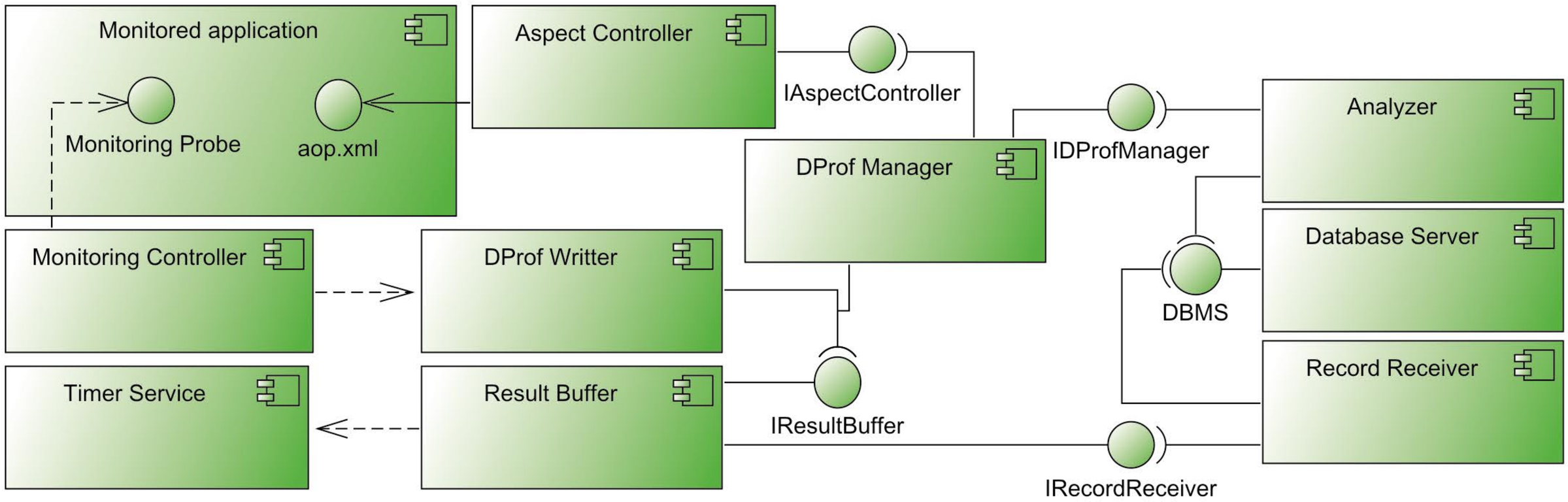
Evaluation of Alternative Instrumentation Frameworks

Dušan Okanović
oki@uns.ac.rs

Milan Vidaković
minja@uns.ac.rs

We have developed ***Dprof*** tool for adaptive monitoring of JEE applications, that monitors specified call trees. Monitoring is turned off in branches if performance parameteres are as expected, and turned on if they are not. By monitoring only "problematic" branches, overall monitoring overhead is reduced.
The default implementation uses AspectJ, but we look to reduce overhead even more by using other instrumentation tools.



How does **Dprof** work?

- **DProf** uses **AspectJ** for code instrumentation with monitoring probes
- *DProfWriter* sends data from probes into the *ResultBuffer*
- *ResultBuffer* holds the data and sends it in bulks to the *RecordReceiver* periodically or on demand
- *RecordReceiver* receives records and stores them into the database
- *Analyzer* analyzes call trees reconstructed from this data, creates a new set of monitoring parameters, and sends these parameters to the *DProfManager*
- *DProfManager* component controls the work of the *ResultBuffer* and the *AspectController*
- *AspectController* configures AspectJ framework

AspectJ Limitations

- AspectJ works on method level
- Support for runtime changes of aspects
- Overhead is higher in some cases
- Possible alternatives
  - low-level tools - very flexible, hard to develop and debug
  - high-level tools - easy to learn, high overhead

DiSL framework

- Syntax similar to AspectJ's makes DiSL easy to learn
- Slightly lower overhead then AspectJ
- Joinpoints can "see" into methods
- Support for runtime changes of aspects (through FRANC framework)

(DiSL and FRANC are still in development)

```
1   public aspect ExecutionTimeMonitoringAspect {
2     // ...
3     pointcut monitoredMethod() :
            execution (@OperationExecutionMonitoringProbe * *(..));
4
5     around() : monitoredMethod() {
6       // ...
7       double startTime = System.nanoTime();
8       try {
9         Object retVal = proceed();
10      } catch (Exception e) {
11        throw e;
12      } finally {
13      double endTime = System.nanoTime();
14      long executionTime = endTime - startTime;
15      DProfExecutionRecord dProfExec =
            new DProfExecutionRecord(..., executionTime);
16      MonitoringController.getInstance().newMonitoringRecord(dProfExec);
17      //...
18      return retVal;
19      }
20    }
21  }
```

AspectJ aspect

Replacing AspectJ aspect with DiSL aspect requires:

- replacing around advice with a before/after pair
- using synthetic local variables for inter-advice communication

```
1   public class ExecutionTimeMonitoring {
2     // ...
3     @SyntheticLocal public static long startTime;
4
5     @Before(marker = BodyMarker.class, guard = DProfAnnotatedGuard.class)
6     static void onMethodEntry() {
7       startTime = System.nanoTime();
8     }
9
10    @After(marker = BodyMarker.class, guard = DProfAnnotatedGuard.class) {
11    static void onMethodExit(MethodStaticContext msc) {
12      double endTime = System.nanoTime();
13      DProfExecutionRecord dProfExec =
            new DProfExecutionRecord(..., endTime - startTime, ...);
14      MonitoringController.getInstance().newMonitoringRecord(dProfExec);
15    }
16  }
```

DiSL aspect