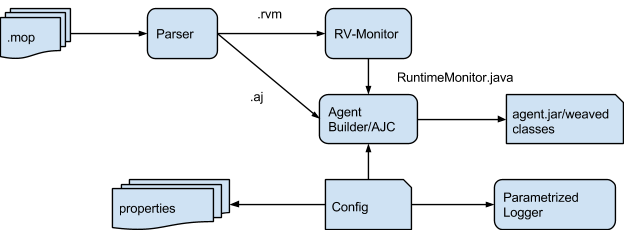
JavaMOP Improvement

**Overall architecture**



Here we present the overall architecture of JavaMOP. JavaMOP takes as input a number of .mop files. Those files will first go through the parser. The parser will parse the .mop files and generate corresponding .aj and .rvm files. The .rvm files will be passed to RVMonitor and thus generate RuntimeMonitor.java files. The agent builder will take .aj and .rvm files, as well as a configuration file together to generate agent.jar. The configuration file will include information about what properties to be included in the agent and how error messages will be logged.

**Goal**

Have a **clean, reliable, easy to use, well tested and documented, and easy to deploy and maintain** JavaMOP.

**Clean:** We want to have a modular system. Every module should be self-contained and only communicate with other modules through well defined APIs, with very suggestive names. We also want to remove duplicate code with RVMonitor especially in the parser (but hopefully everywhere). If support from RV-Monitor is needed, for example to export and well-defined and well-named API, then the RV company people will need to get involved to help with that.

**Reliable:** We want to have an installer for JavaMOP generated automatically with each release, and we want JavaMOP to be able to build agents which run on all platforms out-of-the-box. The generated agents should not crash when used in real world, large applications. Weaving the generated .aj files statically should also be reliable and well documented, especially the part about what needs to be added to the classpath in order for ajc to weave it properly.

**Easy to use:** We want to simplify the process to build an Agent. Users should be able to configure what properties to be built and how to log error message in an agent using a simple configuration file. Similarly for the statically weaved .aj file if the users choose so.

**Well tested and documented:** JavaMOP should also have unit tests for complex methods. The usage of agents should be well documented and provided as a user manual. JavaMOP should also be tried on large real world applications and fix possible problems.

**Easy to deploy and maintain:** This should be a natural consequence of the above, but we should not save any effort that will make JavaMOP easy to deploy and maintain. Maven scripts will allow to generate the desired releases, both with RV-Monitor/AspectJ incorporated in one working system or with RV-Monitor/AspectJ as a dependence, both with the Agent incorporating all the rt.jar files needed or allowing users to include them explicitly, and so on.

**Current status and tasks**

**Pending pull requests (Philip or Qingzhou/Owolabi)**

Currently we have a few pending requests, most of them are from Cody (#42 - #46). Those PRs include major refactoring, such as use Maven instead of Ant, remove some duplicate classes in parser and so on. So this will be the **very first step** before we can proceed with any other further changes.

Action: merge all the pending pull requests, close corresponding issues.

**JavaMOP Installer (Yi with help from Philip)**

Currently JavaMOP installer needs some more work and testing. We decided to put several required jars into generated agent.jar, so this would simplify the installer: we don’t need to put jars in classpath.

Action: Have a well tested installer. Change the code to build agent.jar, so all the necessary jars will be included in agent.jar. Possibly provide different modes if the size of agent.jar would be a concern. Ideally, we should work closely with RV.com and use their generic installer (possibly generalize it if needed to allow us to do what we need in JavaMOP).

**Parser (He with help from Philip/Traian)**

The main functionality of the parser is to separate .mop files into .rvm files and .aj files. Currently it has the full parser from the old JavaMOP, so it overlaps a lot with RVMonitor. The idea is to **remove such duplication**. One way to solve that would be to modify the RV-Monitor parser to export its AST data-structures as a .jar and to leave several boxes for JavaMOP to fill in, but there may be other possible solutions.

Action: Write a design doc to propose a solution and discuss with other people before writing code. Make pull request after we all agree with the proposed solution.

**Agent Builder with configuration (Owolabi and Qingzhou, logger part coordinate with Philip/Traian)**

Currently the agent is built with a plain text configuration file, which specifies on the package level which properties to be included in the agent.jar. It also specifies in each package, which level of specifications (warning/severe/…) should be included.

We want to think about a better way to make such configuration. For example, user can specify in a finer granularity (file level) what properties to be included/excluded, whether to build the agent locally or from remote repository, etc. Moreover, the configuration should also specify how to log error messages: to stdout, to stderr, into a file, into memory (special data-structures, then spit out at the end or regulrly), into a DB, etc. Currently we are using RVMLogger, so this part will probably go into RVMonitor. But it needs to be designed first and foremost

Action: Write a proposal and design with other people before making changes.

Agent also has some issues on Windows platform with bootclasspath (PR #62). Moreover, building and using an agent would require more sophisticated documents.

Action: Resolve the issue and write documents, being able to automatically generate user manual like RVMonitor.

**Merging properties (Owolabi, Qingzhou, Yi)**

According to Owolabi, some of the properties diverged when RVMonitor copied all the properies and Choonghwan made changes to the old properties on google doc. We need to resolve all conflicts and merge them together.

Action: Review conflicts and merge.

**Pending Issues and Test Cases (Owolabi, Qingzhou, Yi)**

Currently JavaMOP has 22 pending issues. Some of them are bugs, for example, for those properties we currently ignored in omit.txt. We need to understand those issues and resolve them. Write regression unit tests for bugs and complex methods.

Action: Resolve all the pending issues, either fix them or close them with a reason why not going to fix. Write unit tests to cover most bugs and complex methods.

**Revised Architecture proposed by (Ali, He, Owolabi and Yi)**

