Beagle

Implementation Report

Annika Berger, Joshua Gleitze, Roman Langrehr,
Christoph Michelbach, Ansgar Spiegler, Michael Vogt

14th of February 2016

at the Department of Informatics
Institute for Program Structures and Data Organization (IPD)

Reviewer:	Jun.-Prof. Dr.-Ing. Anne Koziolek
Advisor:	M.Sc. Axel Busch
Second advisor:	M.Sc. Michael Langhammer

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

1 Design Changes

2 Implemented Tasks

Each task should be implemented in the week it belongs to and tested in the week after that.

2.1 Changes to the Implementation Plan

We decided to split up the PCM task (#050) into two:

- In #050A, the loading part should be implemented (read the PCM repository and store the information on the blackboard). This should be done in the first week.
- In #050B, the storing part should be implemented (convert the evaluable expressions from the blackboard to stochastic expressions and store them on the blackboard). This should be done in the last week.

2.2 Mandatory Tasks

All mandatory tasks have been implemented.

Week 1 (2016-01-11 – 2016-01-17)

Task:	SEFF Classes
Nr:	#040
Description/Classes:	Implement `SEFFLoop`, `SEFFBranch`, `ResourceDemandingInternalAction` `ExternalCallParameter`.
Depends on:	–
Implemented by:	Roman Langrehr
Tested by:	Annika Berger

Task:	PCM Repository Loader
Nr:	#050A
Description/Classes:	Create a class that provides all information of the PCM which is relevant for Beagle and a factory that uses this information to create the `SEFFLoops`, `SEFFBranches`, `ResourceDemandingInternalActions` and `ExternalCallParameters`.
Depends on:	#040 (SEFF classes)
Implemented by:	Ansgar Spiegler
Tested by:	Christoph Michelbach

Task:	Blackboard
Nr:	#060
Description/Classes:	Implement `Blackboard`.
Depends on:	– (Not depending on #050 as the `Blackboard` only needs the class stubs for `SEFFLoop`, `SEFFBranch`, `ResourceDemandingInternalAction`, `ExternalCallParameter` and the `ParameterisationDependentMeasurementResult` subclasses and not their functionality).
Implemented by:	Michael Vogt
Tested by:	Ansgar Spiegler

Task:	Blackboard Views
Nr:	#070
Description/Classes:	Implement `Read-Only Measurement Controller Blackboard View`, `Measurement Controller Blackboard View`, `Read-Only Measurement Result Analyser Blackboard View`, `Measurement Result Analyser Blackboard View`, `Read-Only Proposed Expression Analyser Blackboard View`, `Proposed Expression Analyser Blackboard View`.
Depends on:	– (Does not depend on #060 as the blackboard views only need the blackboard's method stubs.)
Implemented by:	Micheal Vogt
Tested by:	Ansgar Spiegler

Task:	Integrate Prototype for Context Menus
Nr:	#080
Description/Classes:	Integrate the prototype for Context Menus into Beagle.
Depends on:	#010
Implemented by:	Roman Langrehr
Tested by:	Michael Vogt

Task:	Integrate Prototype for Eclipse Extension Points
Nr:	#090
Description/Classes:	Integrate the prototype for Eclipse Extension Points into Beagle.
Depends on:	#020
Implemented by:	Roman Langrehr
Tested by:	Michael Vogt

Task:	Prototype for GUI
Nr:	#100
Description/Classes:	Integrate the prototype for Context Menus into Beagle.
Depends on:	#030
Implemented by:	Christoph Michelbach
Tested by:	Micheal Vogt

Week 2 (2016-01-18 – 2016-01-24)

Task:	Implement Evaluable Expressions
Nr:	#110
Description/Classes:	Implement all subclasses of the interface `EvaluableExpression`. (Package “Evaluable Expressions”)
Depends on:	–
Implemented by:	Annika Berger
Tested by:	Joshua Gleitze

Task:	Implement Measurement Results
Nr:	#120
Description/Classes:	Implement `ParameterisationDependentMeasurementResult` and all of its subclasses. (Package “Measurement”)
Depends on:	–
Implemented by:	Roman Langrehr
Tested by:	Annika Berger

Task:	Implement Measurement Order
Nr:	#125
Description/Classes:	Implement `MeasurementOrder` and `CodeSection`.
Depends on:	#040 (SEFF Classes)
Implemented by:	Roman Langrehr
Tested by:	Annika Berger

Task:	Implement Blackboard Controllers
Nr:	#130
Description/Classes:	Implement `BeagleController` and `MeasurementController`.
Depends on:	#060 (`Blackboard`) and #070 (Blackboard Views)
Implemented by:	Christoph Michelbach
Tested by:	Roman Langrehr

Task:	Implement Final Judge
Nr:	#137
Description/Classes:	Implement `Final Judge`.
Depends on:	#060 (`Blackboard`) and #120 (Measurement Results) (Not depending on #160 (Fitness Function), because only method stubs are needed here.)
Implemented by:	Christoph Michelbach
Tested by:	Joshua Gleitze

Week 3 (2016-01-25 – 2016-01-31)

Task:	Kieker Measurement Tool
Nr:	#140
Description/Classes:	Build a `MeasurementTool` executing Kieker.
Depends on:	#120 (Measurement Results), #060 (`Blackboard`), #070 (Blackboard Views), #040 (SEFF classes) and #125 (Measurement Order).
Implemented by:	Joshua Gleitze
Tested by:	Annika Berger

Task:	Averaging Measurement Result Analyser
Nr:	#150
Description/Classes:	Build a `MeasurementResultAnalyser` which takes a `ConstantExpression` as final `EvaluableExpression` for each `MeasurableSeffElement` with the average of all available measurement results.
Depends on:	#120 (Measurement Results), #060 (`Blackboard`), #070 (Blackboard Views), #040 (SEFF classes) and #133 (tool helper classes).
Implemented by:	Ansgar Spiegler
Tested by:	Annika Berger

Task:	Implement Evaluable Expression Fitness Function
Nr:	#160
Description/Classes:	Implement a `EvaluableExpessionFitnessFunction`.
Depends on:	#120 (Measurement Results) and #040 (SEFF classes)
Implemented by:	Christoph Michelbach
Tested by:	Annika Berger

Week 4 (2016-02-08 – 2016-02-14)

Task:	PCM Repository Storer
Nr:	#050B
Description/Classes:	Create a class that writes back all information from the blackboard to the PCM.
Depends on:	#040 (SEFF classes) #110 (Evaluable Expressions)
Implemented by:	Ansgar Spiegler
Tested by:	Annika Berger

Tasks Not Covered by the Implementation Plan

Task:	Facade
Nr:	–
Description/Classes:	We realised that the communication between the GUI and Beagle Core is complex because in addition to what we planned, the GUI has to pass information about the project to analyse, e.g. to be able to access the source code files. This is the reason for `BeagleController` being now more than a controller. It is a facade with some additional classes. The `UserConfiguration` has been renamed to `BeagleConfiguration` because it is involved in the facade, too.
Depends on:	–
Implemented by:	Joshua Gleitze, Chirstoph Michelbach, Roman Langrehr
Tested by:	Annika Berger

2.3 Optional Tasks

Week 1 (2016-01-11 – 2016-01-17)

Task:	Gradle
Nr:	–
Description/Classes:	Our project can be build automatically using gradle.
Depends on:	–
Implemented by:	Joshua Gleitze

Week 3 (2016-01-25 – 2016-01-31)

Task:	Fail API
Nr:	-
Description/Classes:	On many parts of our project, errors which can't be handled where they emerge can occur. E.g. a source file from the project under test can't be read or doesn't compile. Therefore, we created the "Fail API": Everywhere in our project, these exceptions are reported to the "Fail API" and handled there. At the moment, the Fail API only throws an exception but when we want these exceptions to be reported to the user so the user can choose in a dialogue whether to abort or to retry, this has to be changed only in the Fail API and not on multiple places in the code.
Depends on:	–
Implemented by:	Joshua Gleitze
Tested by:	Michael Vogt

2.3.1 Preparation for Other Optional Tasks

We did a lot of preparation to implement the following optional tasks:

- Parametrisation of the results.
- Using genetic programming for the analysis.

The classes for the parameterisation were created (but are empty) and are used by all classes which don't depend on the implementation of them.

For the genetic programming approach, we already created the fitness function. The tools which combine results to the next generation can be added easily as new `ProposedExpressionAnalysers`.

3 Delays

Most of the tasks were scheduled to be done during the first and seconds week which lead to some small delays. Some of the tasks of the first week where finished in the second week and some of the tasks in the second week were finished in the third week. The tasks for the third week were therefore started at the end of the third week and we used the empty week four for them, too.

Getting these delays was intentional because starting many tasks at the beginning allowed us to recognise which tasks can be implemented fast and which tasks are complicated and therefore need more time early in the implementation phase. These tasks were:

- The PCM connection and the context menus because nobody in our team had experience with the EMF framework and a lot of problem arose and caused delays.
- The Kieker measurement tool because we had to instrument the code on our own.

4 Unit Tests

Everything that can be meaningfully tested with JUnit Tests is tested with JUnit tests. The only parts not tested with JUnit tests are:

- The GUI classes because it is not possible to test GUIs meaningfully with JUnit. Instead, we created a document describing what should be done to test the GUI manually.
- The extension points are not tested with JUnit tests because an automated test would rather test the Eclipse extension point mechanism and not whether we used it correctly. Instead, we created some Eclipse plugins with stubs for `MeasurementTool`, `MeasurementResultAnalyser`, and `ProposedExpressionAnalyser` which use the extension point of Beagle Core. Additionally, we created a plugin having BeagleCore as a dependency which adds a button to Eclipse allowing the user to see all tools loaded by Beagle Core (using Beagle Core's classes for that). For manual testing of the extension points, a document describing the process and the relevant parts to be tested has been added.

Detailed information on each JUnit test can be found in the Javadoc of the particular test.