Conceptual Architecture Description

Structural View

The Component Behaviour Evaluation (ComponentBee) tool consists of four main packages (Figure 1):

1. The ***test execution package*** provides components capable of producing information from the dynamic behaviour of components and finally writing a test report of the observed behaviours.
2. The ***behaviour presentation package*** provides various kinds of presentation components capable of presenting dynamic behaviour of components.
3. The ***test model package*** provides components enabling software integrators to define various kinds of test models supporting R&A testing of OS components.
4. The ***test creation package*** provides tool components to help software integrators to create test models for the R&A testing.

In order to increase the level of modularity, the components of the packages communicate with each other via predefined interfaces. Each package is divided into core and implementation parts. The core part provides interfaces for the components of the package whereas the implementation part provides the reference implementations for the core interfaces. In addition, predefined extension points enable developers to extend the ComponentBee tool with new Eclipse plug-ins.

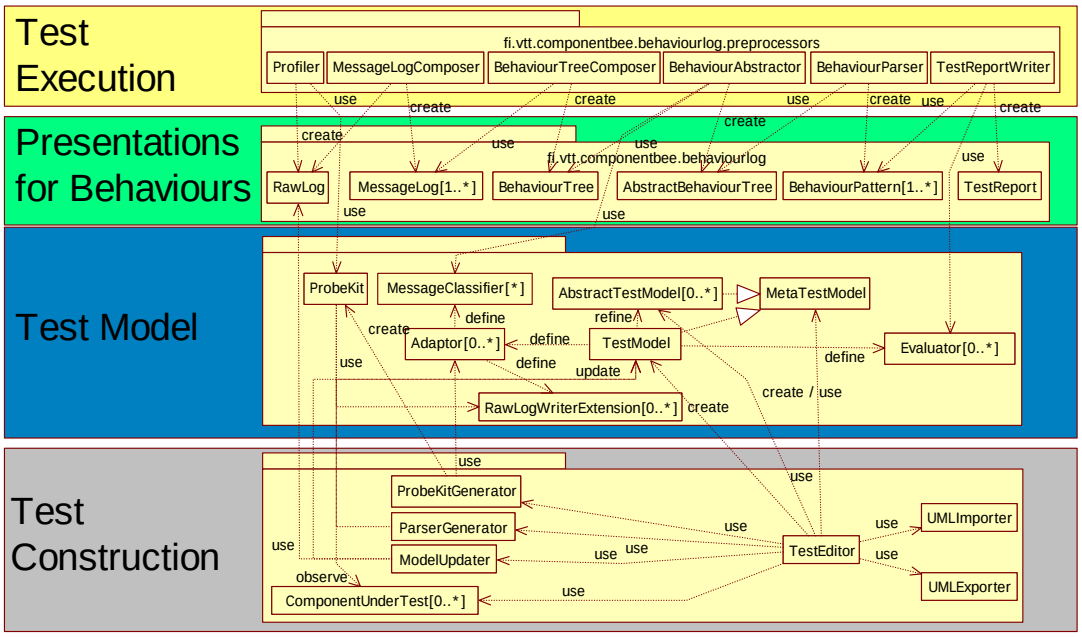

Figure 1. The main packages of ComponentBee.

The requirement specification determines that the ComponentBee must provide access and extension points for downstream plug-ins. They are described in the table 1. The ComponentBee provides two kinds of extension points: Eclipse and dynamic extension points. The Plug-in Development Environment (PDE) provides tools to create, develop, test, debug, build and deploy Eclipse plug-ins for the Eclipse extension points. The dynamic extension points provide a mechanism to extend the ComponentBee at run-time with the plug-ins being developed in the workspace of Eclipse.

Table 1. The extension and access points of the ComponentBee.

Eclipse Extension Point	**Description**
ProbekitGenerator	An extension point that enables developers to extend ComponentBee with a plugin that generates a probekit for the test model.
BehaviourParserGenerator	An extension point that enables developers to extend ComponentBee with a plugin that generates a behaviour parser for the test model.
RawLogEvaluator	An extension point that enables developers to extend ComponentBee with plugins that process the raw log and produce new presentations for it.
UMLImporter	An extension point that enables developers to extend ComponentBee with plugins that are capable of importing UML models to test models.
UMLExporter	An extension point that enables developers to extend ComponentBee with plugins that are capable of exporting UML models from test models.
Dynamic Extension Point	**Description**
Evaluator element	An extension point that provides means to utilise new plugins in behaviour evaluation.
RawLogWriterExtension element	An extension point that enables software integrators to use various kinds of plugins in raw log information recording.
MessageClassifier element	An extension point that enables software integrators to use various kinds of plugins classifying messages to different classes.
BehaviourParser element	An extension point that enables software integrators to use a behaviour parser in raw log information processing.
Access Point	**Description**
TestModel	An access point that enables UML import and export plugins to access information provided in the test model.
RawLog	An access point that enables evaluator plugin to access collected log information.
MessageLog	An accesss point that enables evaluator plugin to navigate in the message log.
BehaviourTree	An accesss point that enables evaluator plugin to navigate between synchronous and asynchronous messages delivered between components in different threads.
AbstractBehaviourTree	An accesss point that enables evaluator plugin to navigate between synchronous and asynchronous messages delivered between components in different threads. An abstract behaviour tree contains only messages that are defined in the test model.
BehaviourPattern	An access point that enables evaluator plugins to navigate in behaviour pattern tree and to add evaluation data to the extracted behaviour patterns .
TestReport	An access point that provides access to the behaviour patterns recorded to the test report.

Behavioural View

Figure 1 illustrates the communications between the components of main packages.

The *TestEditor* enables software integrators to create new *TestModels* that can refine more abstract test models (Figure 2). The *UMLImporter* is capable of importing UML diagrams to test models and *UMLExporter* is capable of exporting UML models from test models.

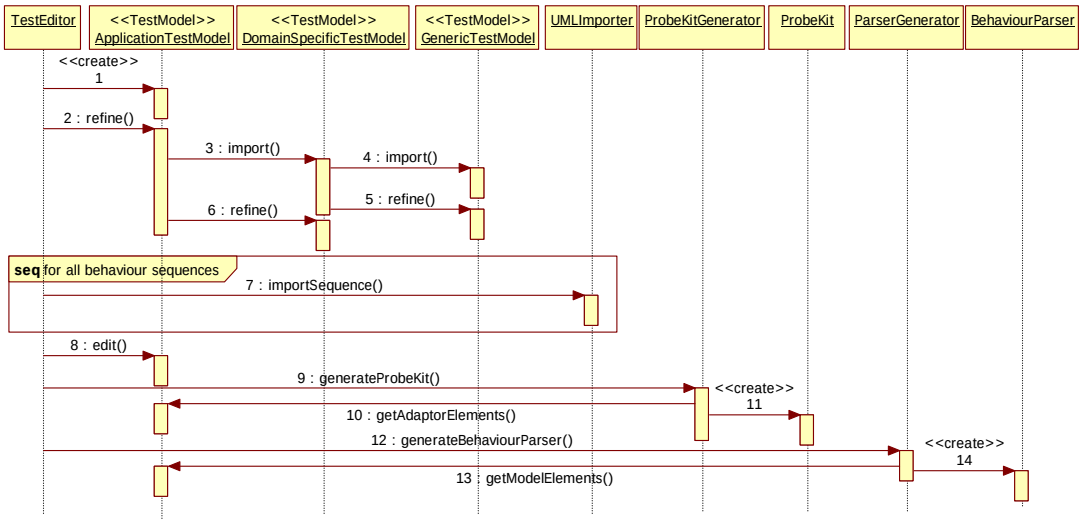

Figure 2. The test creation sequence.

The *ProbeKitGenerator* is capable of generating ProbeKit for the test model. The ProbeKit defines probes that will record raw log information about the input attributes and return values of methods and the states of the components to XML-based behaviour logs. The *ParserGenerator* takes the *test model* as an input and generates a BNF grammar to define production rules for the behaviours defined in the test model and then finally calls the JavaCC compiler-compiler (*https://javacc.dev.java. net/*) to generate a behaviour parser for the grammar.

Profiler attaches probes to the components under test (Figure 3). The probes call *RawLogWriterExtensions* to write information to the raw log file.

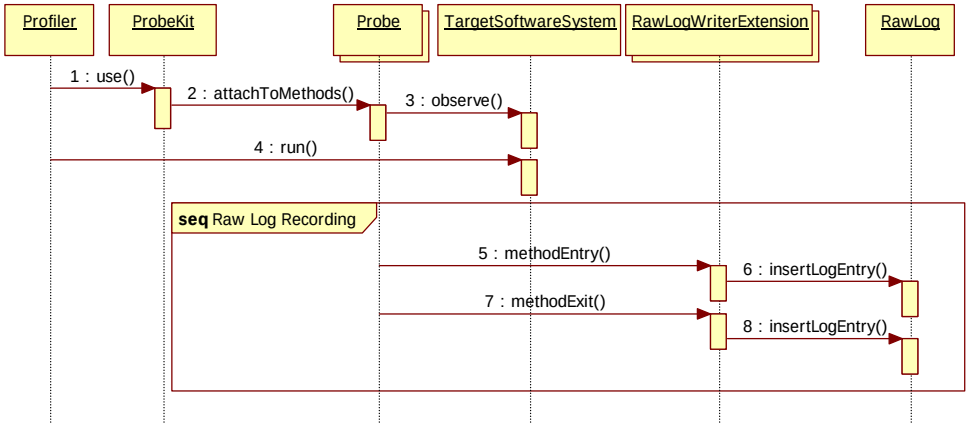

Figure 3. The raw log recording sequence.

The pre-processors will read the raw log file and create evaluation data in the following steps (Figure 4):

1. ***MessageLogComposer*** creates message logs of the raw log information, defining a sequence of messages that are delivered between different components in various threads during the test.

2. ***BehaviourTreeComposer*** creates an overall presentation (a behaviour tree) for the message sequences delivered in different threads.

3. ***BehaviourAbstractor*** calls message classifiers to classify the messages of the behaviour tree first and then creates an abstract behaviour tree of the composed behaviour tree containing only the messages defined in the test model. The model-based message classifier gives names for the messages that are defined in the test model. In addition, the message classifiers defined in test model can use the recorded log information, do data and message sequence-based message classifications, and finally give more specific classifications for the messages.

4. ***BehaviourParser*** extracts behavior patterns and composes a symbol tree of the abstract behaviour tree. The nodes of the tree define extracted behaviour patterns and messages related to these.

5. ***TestReportWriter*** calls evaluators that are defined in the test model to attach evaluation data to the behaviour patterns first and then writes the behaviour patterns to the test report.

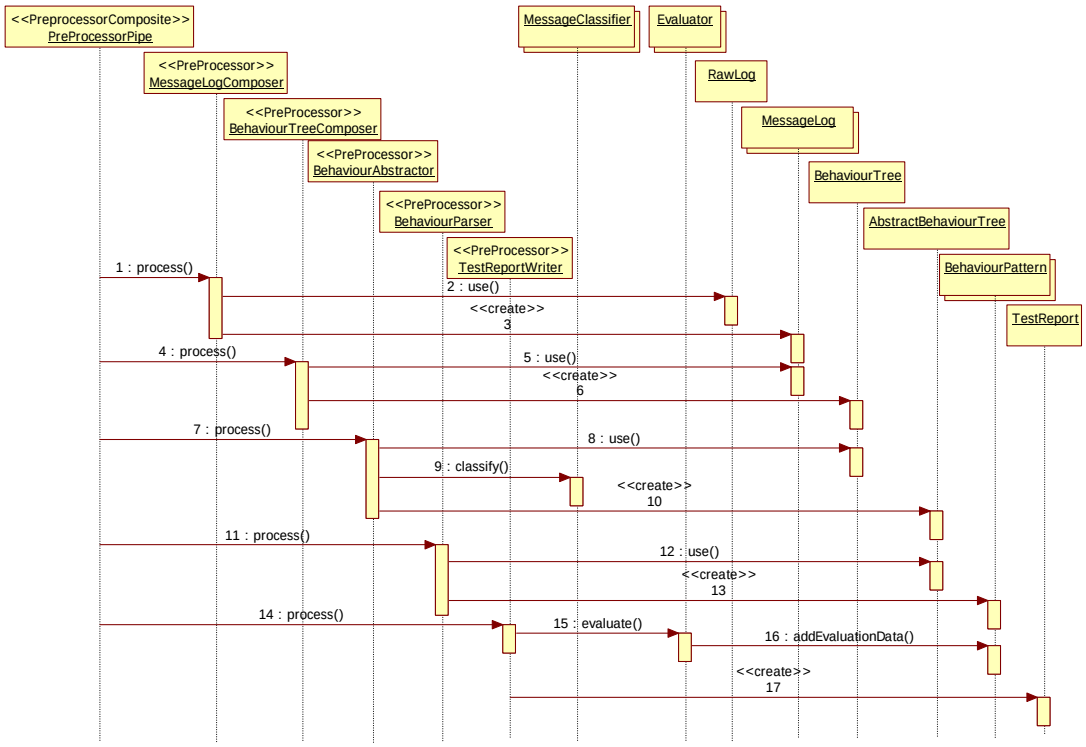

Figure 4. The evaluation sequence of raw log information.

Deployment View

Figure 5 illustrates how the ComponentBee tool is deployed.

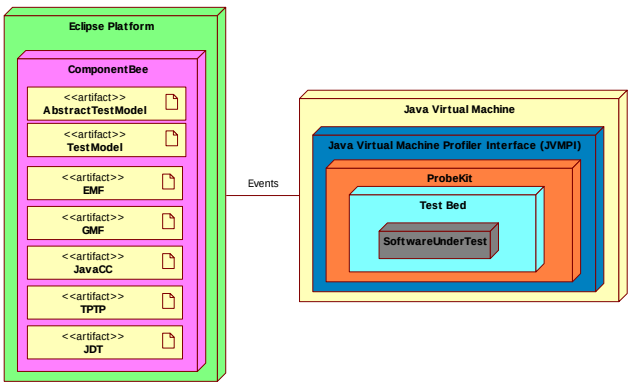

Figure 5. The ComponentBee deployment.

The ComponentBee is an extension to the Eclipse IDE that is typically running in a desktop computer. The dynamic testing of software components is performed in a test bed that runs in a Java Virtual Machine. The raw log information of dynamic behaviour of components is collected by utilising Java Virtual Machine Profiler Interface (JVMPI) and ProbeKits.

Development View

The ComponentBee tool is by no means on isolated island – it is dependent on various tools developed by other open source communities. Figure 6 shows the technologies utilised in ComponentBee.

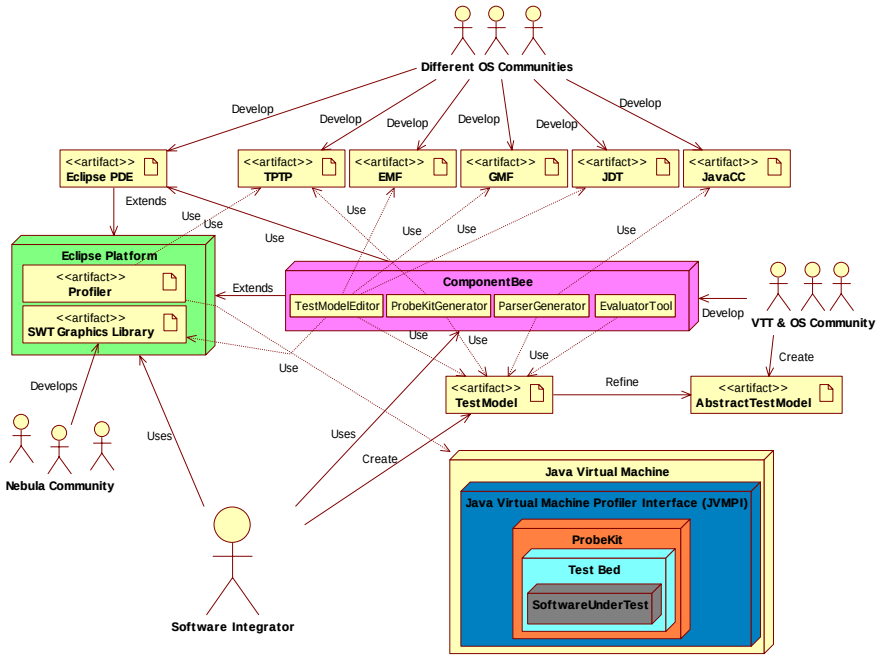

Figure 6. The development view of ComponentBee.

Test models are EMF models that are created with the test model editor that is in turn constructed by utilising GMF. The behaviour parsers are generated with JavaCC, the dynamic behaviour is logged by

utilising the ProbeKit of TPTP. The components, methods, data structures, and comments of the components under test are obtained by utilising the methods of JDT.

Eclipse Platform and Eclipse Plug-in Development Environment (PDE) are developed by respective communities under the official Eclipse project. The development of Standard Widgets Toolkit is managed by the Eclipse platform community, but new widgets originate from the Nebula project which is a source of supplemental SWT widgets and an “incubator” for SWT. The TPTP, EMF, GMF, JDT, and JavaCC components are developed in various OS projects. The further development of ComponentBee relates to these projects. VTT and OS community can extend ComponentBee with new features or then provide more abstract test models supporting Reliability and Availability (R&A) testing of various kinds of application-domains and component-based software systems. Software integrators can later refine these abstract test models and use them in R&A testing of actual component-based software implementations.