# OpenL Tablets. Concurrent Edition

## Preface

The goal of the Concurrent Edition (CE) is to utilize resources of modern multi-core CPUs that became de-facto the only way of improving brute-force performance in modern software development. By using Concurrent Edition developers can take advantage of new commercially available CPUs to decrease time required to run long transactions using OpenL Tablets that can be run in parallel. The good examples of such transactions could be the rating of insurance policies with many risk items, or multiple coverages that can be processed in parallel.

### Limitations of Parallel Execution

1. In Java parallel execution is based on Java threads and on concurrent framework provided by JDK library classes. Due to OpenL specifics, it’s calculations demand only CPU and memory resources, and almost never I/O operations. It means that when executing OpenL tasks in parallel it makes no sense to create more threads than number of available CPU cores. In addition to this, in cases, when OpenL application runs on a server, a special consideration should be made to other applications running on the server, including the fact that OpenL itself may be running inside of multi-threaded container.
2. Using CE framework is not recommended when CPU is already under high load. The reason for this is that the total throughput for parallel execution is usually lower than the sum of the individual sequential executions. In the future, we plan to be able to automatically determine high-load state of the host and automatically adjust CE settings. Currently, use CE with caution in a high-load environment. At the same time CE can drastically speed up execution of long requests, and by doing this reduce possible congestion.
3. Using CE framework (and parallel execution in general) impose some level of overhead on the host system. Because of this overhead, CE framework demonstrates significant improvement in performance when a) the length of sequential execution paths is more than a certain threshold and b) the length of each execution path is close to the length of other paths (symmetry). CE framework provides means to configure these values to tune up execution

### Using Concurrent Framework in OpenL Runtime

Currently, there are two cases where OpenL CE framework can automatically parallel OpenL execution: a) OpenL function call with array arguments and b) calculation of OpenL Spreadsheet

Let’s consider both cases in more detail:

1. OpenL function call with array argument is a known OpenL syntax shortcut. For example we calculate a single coverage premium using method with a signature

SpreadsheetResult calcCoverage(Vehicle v, Coverage c)

To calculate premiums for each of the coverages we can use the following construction:

SpreadsheetResult[] premiums = calcCoverage(v, v.coverages);

OpenL will automatically call calcCoverage for each coverage in the array and return an array of results back. It is easy to see how this syntax feature is a very good candidate for a parallel execution.

To enable parallel execution of method calcCoverage CE framework must be configured using the following XML element:

<bean class="org.openl.util.ce.conf.ComponentMTBean">

<property name ="componentId" value="calcCoverage"/>

<!-- run time in nano-seconds -->

<property name ="componentLength" value="500000"/>

<property name ="callComponentUsingMT" value="true"/>

</bean>

Or Java API call:

ServiceMTConfiguration config = ServiceMT.*getService*().getConfig();

config.setCallComponentUsingMT("calcCoverage2", 500000, **true**);

1. Spreadsheet calculation presents a less straightforward case for parallel execution. It is based on analyzing spreadsheet *dependency graph*  and finding parallel execution paths. This case does not possess an intrinsic symmetry of the first case, as the result there will be less suitable candidates for parallel execution.

The configuration of a spreadsheet component for parallel execution is done similar to the configuration of the previous case, the only difference is the name of the property executeComponentUsingMT.

<bean class="org.openl.util.ce.conf.ComponentMTBean">

<property name ="componentId" value="calcPremium2"/>

<property name ="executeComponentUsingMT" value="true"/>

</bean>

Or Java API call:

ServiceMTConfiguration config = ServiceMT.*getService*().getConfig();

config.setExecuteComponentUsingMT("calcPremium2", 0, **true**);

## Configuration

To use CE framework in Java application all you need is to add **org.openl.rules.ce*${version}*.jar** to the classpath and configure CE framework settings

## Classpath Configuration

In java application make sure that **org.openl.rules.ce*${version}*.jar** is located*before*the jar file **org.openl.rules*${version}*.jar**

If you use maven dependencies, make sure that module **org.openl.rules.ce** is located *before* the modul**e org.openl.rules**

## CE Framework Settings

CE Framework can be configured using either or XML file or Java API.

#### Using XML File

XML file **rules-ce-conf.xml**  is loaded from java classpath.

Here is the sample rules-ce-conf.xml file:

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.1.xsd"  default-autowire="no">    <bean id="rules-ce-conf" class="org.openl.util.ce.conf.ServiceMTConfiguration">    <!-- optional property totalParallelLevel. Default = Runtime.getRuntime().availableProcessors() .  This property limits the total number of active threads in CE framework  -->  <property name="totalParallelLevel" value="8"/>  <!-- optional property maxPerRequestParallelLevel. Default = totalParallelLevel.  This property limits the total number of threads allocated to a single CE request to avoid having a single request using too much of system CPU resources -->  <property name="maxPerRequestParallelLevel" value="4"/>  <!-- optional property minSequenceLengthNs. Default = 100000 (100 microseconds) -->  <property name="minSequenceLengthNs" value="200000"/>  <property name="componentMTBeans">  <list>  <bean class="org.openl.util.ce.conf.ComponentMTBean">  <property name ="componentId" value="calcCoverage2"/>  <property name ="callComponentUsingMT" value="true"/>  </bean>  <bean class="org.openl.util.ce.conf.ComponentMTBean">  <property name ="componentId" value="calcCoverageX"/>  <property name ="executeComponentUsingMT" value="true"/>  </bean>  </list>  </property>  </bean>  </beans> |

#### Using Java API calls

// Access current service configuration

IServiceMT serviceMT = ServiceMT.getService();

ServiceMTConfiguration conf = serviceMT.getConfig();

…..

// use CE framework when calling OpenL method named calcCoverage2

ServiceMTConfiguration config = ServiceMT.*getService*().getConfig();

config.setCallComponentUsingMT("calcCoverage2", 500000, **true**);

// use CE framework when executing OpenL method named calcPremium

ServiceMTConfiguration config = ServiceMT.*getService*().getConfig();

config.setExecuteComponentUsingMT("calcPremium", 0, **true**);