endava

—

OpenL Tablets Introduction

OR HOW TO EMPOWER THE BUSINESSPEOPLE FOR CREATING AND MAINTAINING THE BUSINESS RULES USED IN JAVA-BASED APPLICATIONS

11 JUN 2020

CIPRIAN MIHALACHE

Intro

Every application uses some business rules. But where should we keep them?

- In the code? For each change, a new code delivery (release / hotfix) is necessary, which is not cheap…
- In Database, with an UI?
    - If it is something simple, it is not very useful… For sure some numbers can be changed easily, but changing a criteria will not work
    - If it is very flexible, there will be a huge investment (months / years) to build such complex Business Rules Engine.
- In some files? Then a DSL will emerge, that will become more and more sophisticated… There will be an important learning effort for every new team member, that is a waste, because, in the end, it is our DSL. Do we really want to reinvent the wheel?

What if we need to deal with limited business rules? (for instance on 8th of March, there will be different rules for women than on the rest of the year)

Fortunately, there are some good open-source Business Rules Engines available and one of them is OpenL Tablets. In this presentation we will learn how to write some OpenL Tablets rules, how do we test them, how do we use them in our java applications and what are some utility applications that we have available when dealing with these type of rules.

Agenda

1. WHY?
2. OPENL TABLETS
3. BUSINESS RULES ENGINE
4. BUSINESS RULES MANAGEMENT SYSTEM
   - WEBSTUDIO
   - RULES REPOSITORY
   - WEB SERVICES
5. ACCESSING RULES REPOSITORY FROM EXTERNAL JAVA APPLICATIONS
6. FINAL THOUGHTS

If you ever talk to a great programmer, you'll find they know their tools like an artist knows their paintbrushes.

Bill Gates

Agenda

1. WHY?
2. OPENL TABLETS
3. BUSINESS RULES ENGINE
4. BUSINESS RULES MANAGEMENT SYSTEM
   - WEBSTUDIO
   - RULES REPOSITORY
   - WEB SERVICES
5. ACCESSING RULES REPOSITORY FROM EXTERNAL JAVA APPLICATIONS
6. FINAL THOUGHTS

Given 2 employees in an insurance company…

Businessperson

Software developer

After deep analytics on the existing data, and intensive predictive computations, we reached to the conclusion that we can no longer afford to insure some vehicles


So, our application should have some conditions in place before it offers a premium quote?

Yes, exactly

What are these conditions?

Vehicles should not be older than 25 years and their market value should not be less than 5000$

Vehicles should not be older than 25 years and their market value should not be less than 5000$

```
Premium computePremium(args) {
    if(isPremiumAllowed(args)) {
        return doComputePremium(args);
    }
    return null;
}
boolean isPremiumAllowed(args) {
     return vehicleAge <= 25 &&
                    vehiclePrice >= 5000;
}
```


Sure, it will be implemented in the next release

Some time after the release…

Considering the current market state, we want to relax a bit the premium allowed conditions, so we should accept vehicles with market value at least 4500$

Some time after the release…

Considering the current market state, we want to relax a bit the premium allowed conditions, so we should accept vehicles with market value at least 4500$

```
Premium computePremium(args) {
    if(isPremiumAllowed(args)) {
        return doComputePremium(args);
    }
    return null;
}
boolean isPremiumAllowed(args) {
    return vehicleAge <= 25 &&
            vehiclePrice >= 50004500;
}
```


Sure, it will be implemented in the next release

NEXT RELEASE???
Our business rules will change several times until the next release

Sure, it will be implemented in the next release

But we cannot wait until the next release… We need to modify the existing behavior of our app as soon as possible…

We could deliver a hotfix which will be put in production in several days

SEVERAL DAYS???

We could deliver a hotfix which will be put in production in several days

Is it possible to implement a mechanism to allow us to modify these rules while the application is running, without the need for a hotfix?

Is it possible to implement a mechanism to allow us to modify these rules while the application is running, without the need for a hotfix?

Max vehicle age: 25
Min vehicle price: 4500

Yes, we can. The rules will be added in the database and there will be a place in the UI where they can be modified. The new values will be used by the application immediately.


This will be implemented in the next release

Some time after the release…

The new UI functionality is great, but too simple. We now want to accept SUVs with value at least 5000$, convertibles with value at least 4500$ and all other vehicles that have the market value at least 4000$

The new UI functionality is great, but too simple. We now want to accept SUVs with value at least 5000$, convertibles with value at least 4500$ and all other vehicles that have the market value at least 4000$

Now he wants to consider the body type of the car??? We never discussed about that. How will the UI look like, after this change request?

The new UI functionality is great, but too simple. We now want to accept SUVs with value at least 5000$, convertibles with value at least 4500$ and all other vehicles that have the market value at least 4000$

```
Max vehicle age:      25
Min vehicle price:    5000   for type:  SUV
Min vehicle price:    4500   for type:  Convertible
Min vehicle price:    4000   for type:  <other>
```

The new UI functionality is great, but too simple. We now want to accept SUVs with value at least 5000$, convertibles with value at least 4500$ and all other vehicles that have the market value at least 4000$

But now we have 3 rules with the car price. Maybe this will change in the future…
We should offer a mechanism to multiply the rule with the price…
Or even better, we should have one line for each body type…
It is a little verbose, but it does the job.
Great…

The new UI functionality is great, but too simple. We now want to accept SUVs with value at least 5000$, convertibles with value at least 4500$ and all other vehicles that have the market value at least 4000$

But, what if, in the future, some other criteria will have to be implemented?

I must admit, I did not foresee that the vehicle body type will be used in the rule that allows the premium to be offered. But it is not a difficult change, we can deliver it in the next release

You know that we cannot wait until the next release… I was expecting now that we can define the premium allowed conditions on the fly, without the need of a special software delivery

Yes, but we did not expect that the car body type will be included in the rule…
By the way, do you foresee other conditions that will be included in the rule?

These days we are running some simulations that involve the young drivers and powerful cars. We will define soon a rule on that area…

These days we are running some simulations that involve the young drivers and powerful cars. We will define soon a rule on that area…

WHAT???

But, honestly, it could be anything:
Details of car, like age, type, price, engine power or fuel of the car, combined or not with details of the owner like age, address or something else, combined or not with details of the driver like age, criminal record, time since he has the driving license or other details we may consider relevant in the future

But, honestly, it could be anything:
Details of car, like age, type, price, engine power or fuel of the car, combined or not with details of the owner like age, address or something else, combined or not with details of the driver like age, criminal record, time since he has the driving license or other details we may consider relevant in the future

WHAT???
I am not even able to imagine the UI that allows such flexibility

We need at least 6 months to implement a such generic mechanism

WHAT???
6 MONTHS???

We need at least 6 months to implement a such generic mechanism

Where are the old Excel tools? 20 years ago we were able to deliver in no time a new version of the Excel tool by mail to the call center employees…

Where are the old Excel tools? 20 years ago we were able to deliver in no time a new version of the Excel tool by mail to the call center employees…

If there would be some mechanism to pass our data objects to some “function” written and maintained by the businesspeople, and it is their responsibility to return whether the premium is allowed or not, based on whatever rules they consider to be appropriate for the moment…
But maybe there is…

After several days…


I worked on the Excel files since… forever…

If you agree to write the business rules in Excel files, I can reduce my estimation from 6 months to 6 hours…

Let me understand… There is a way to define business rules in Excel files, based on the actual objects used in the application? And, we can define in a simple way the desired logic, using the properties of those objects?

Yes, indeed…
And it is amazing how powerful are the things you can write in Excel files, from simple rules to real programs. You could even call back methods in the app, that could do… anything…
You can define even the timings for the rules…
And they offer also an application that helps you write, maintain and deploy the rules. I can’t wait to tell you more…

Wow, this sounds great…
Finally, I will be able to define the necessary business rules in time, without waiting for the developers to update the code or implement fancy UIs that I can never use for my needs…

Yes, indeed…
And it is amazing how powerful are the things you can write in Excel files, from simple rules to real programs. You could even call back methods in the app, that could do… anything…
You can define even the timings for the rules…
And they offer also an application that helps you write, maintain and deploy the rules. I can't wait to tell you more…

It sounds promising,
let’s give it a try

It sounds promising,
let’s give it a try

Hurray…
No more ultra-sophisticated
“future-proof” rules engine that
we need to build…


Could you please tell me more about this framework?

Sure, its name is **OpenL Tablets** and I can briefly present some of its features in the next minutes…

Agenda

1. WHY?
2. OPENL TABLETS
3. BUSINESS RULES ENGINE
4. BUSINESS RULES MANAGEMENT SYSTEM
   - WEBSTUDIO
   - RULES REPOSITORY
   - WEB SERVICES
5. ACCESSING RULES REPOSITORY FROM EXTERNAL JAVA APPLICATIONS
6. FINAL THOUGHTS

OpenL Tablets (http://openl-tablets.org)

- Targets the gap between business requirements (rules and policies) and software implementation
- Designed to be straightforward and intuitive for businesspeople
- It is an open source
  - business rules engine (BRE)
  - business rules management system (BRMS)

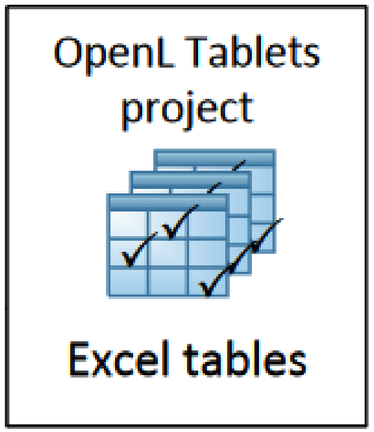
  based on tables presented in Excel documents

- In a very simplified overview, OpenL Tablets can be considered as a table processor that extracts tables from Excel documents and makes them accessible from software applications

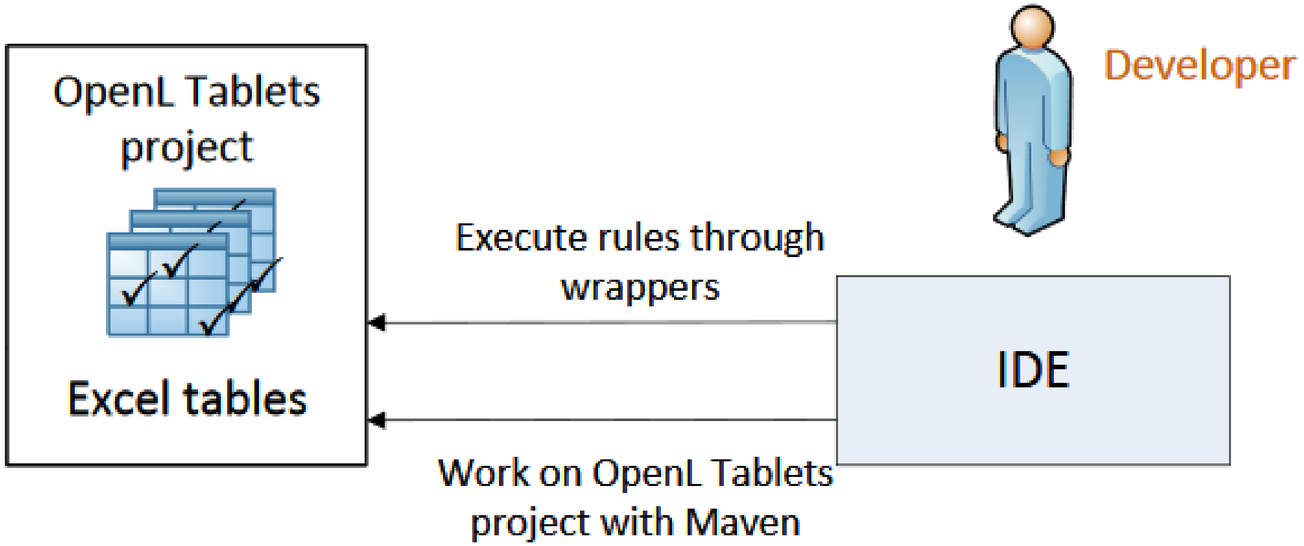
⇒ It reduces costly enterprise software development errors

⇒ It dramatically shortens the software development cycle

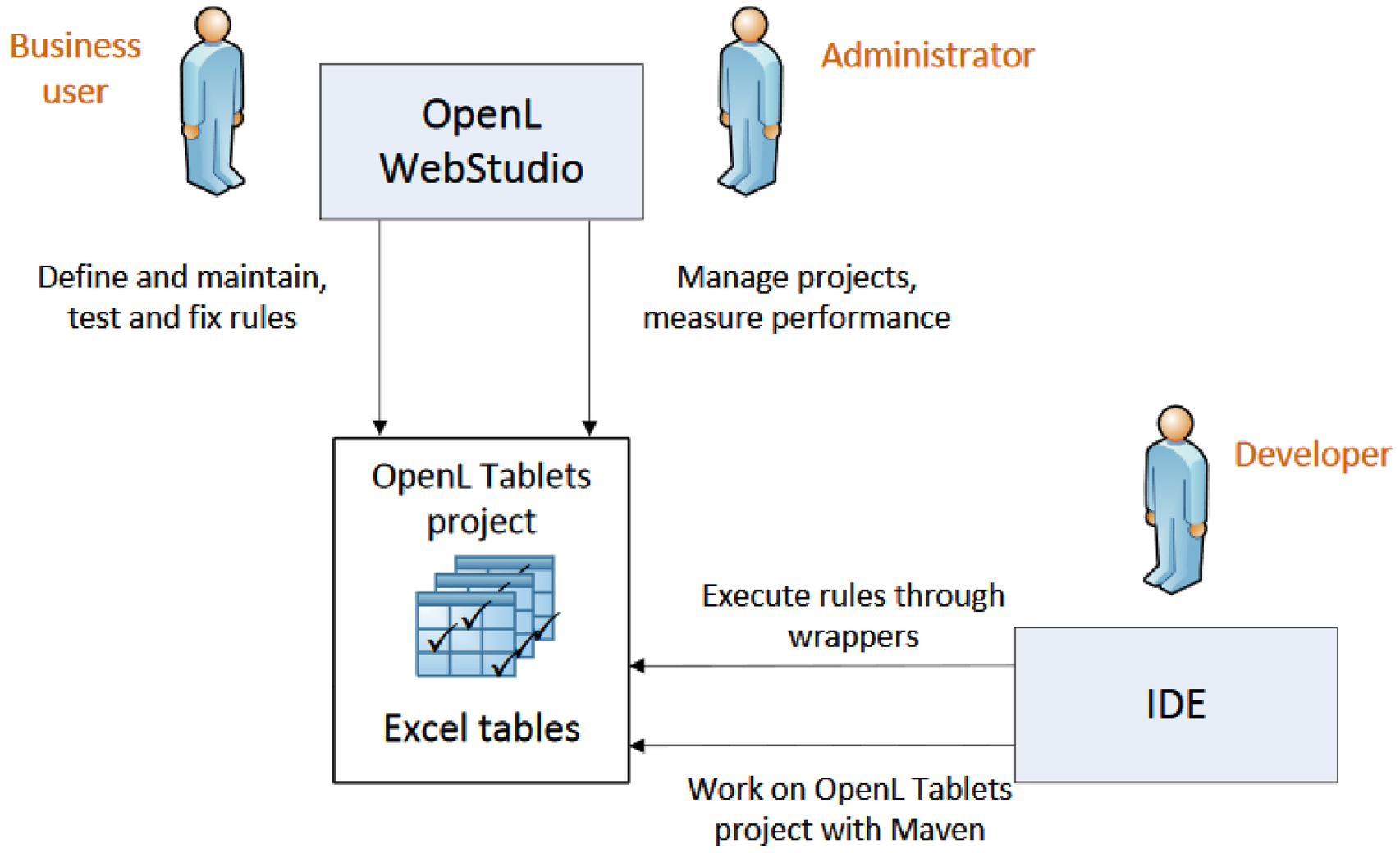
System Overview



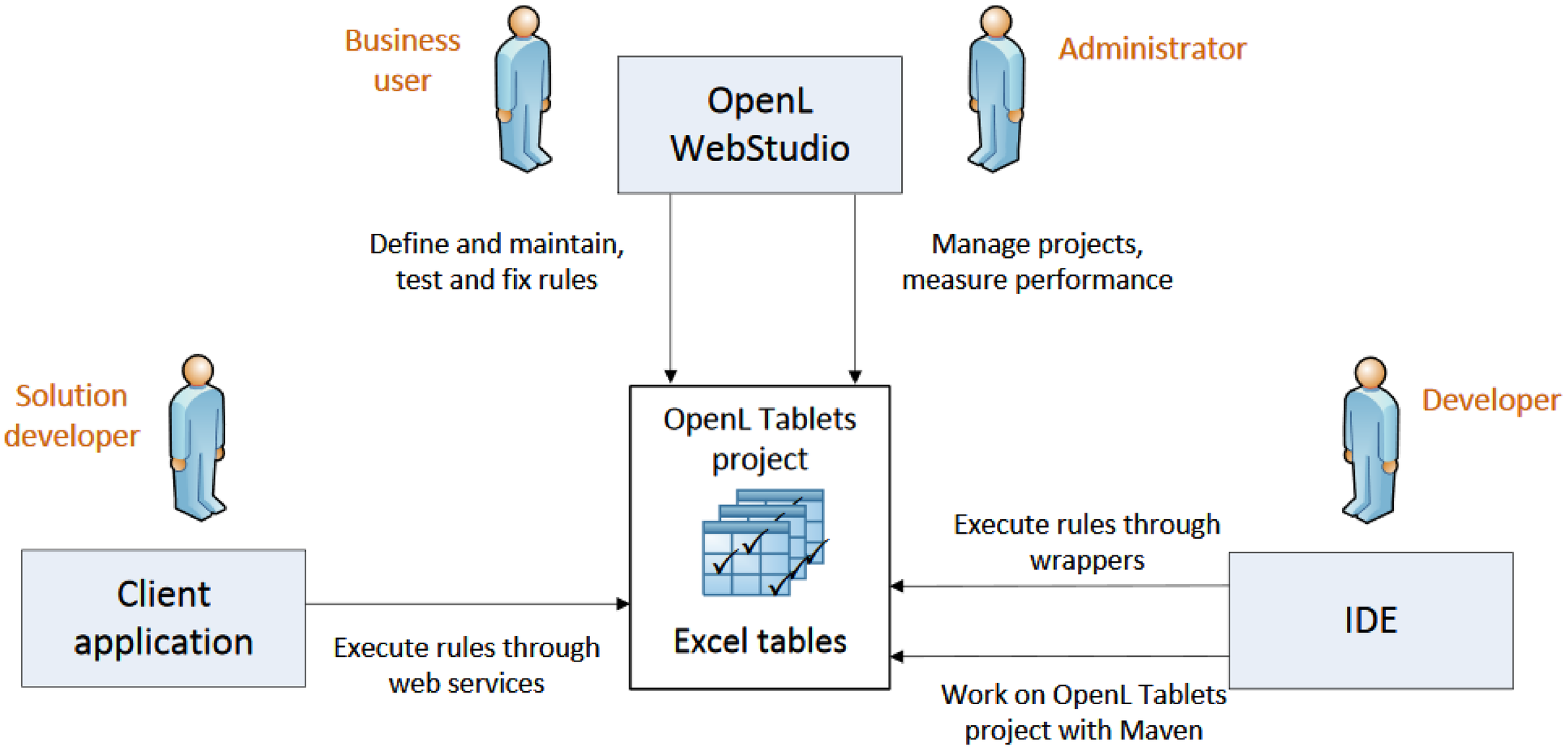
System Overview



System Overview



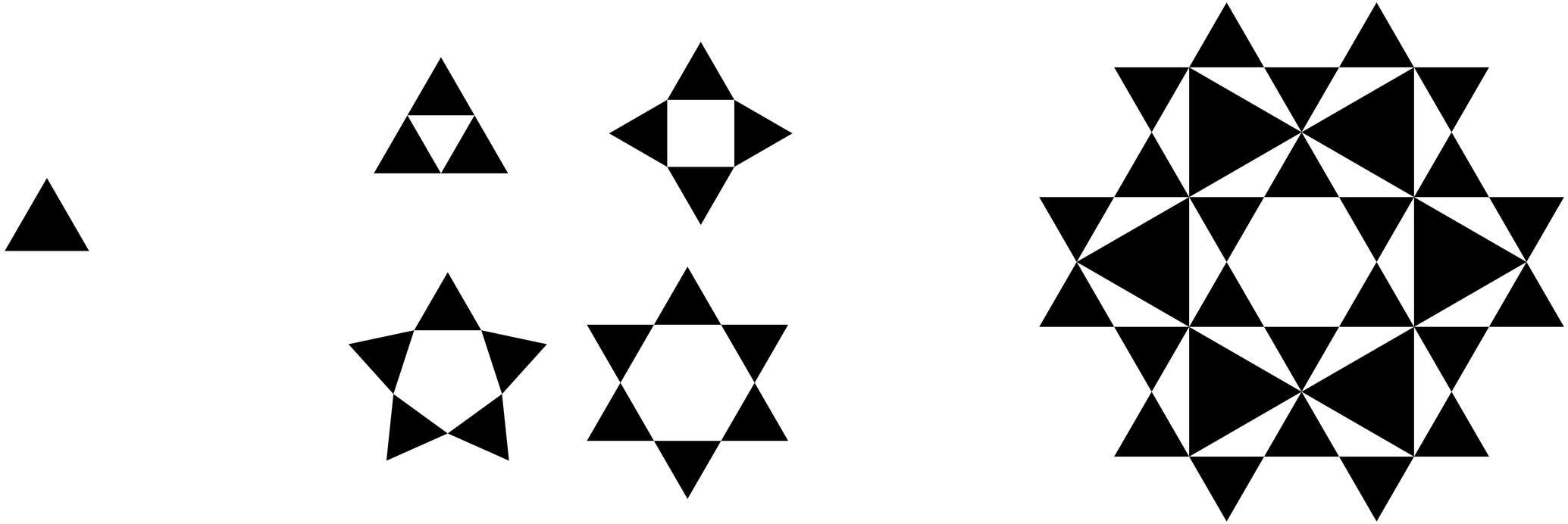
System Overview



Agenda

1. WHY?
2. OPENL TABLETS
3. BUSINESS RULES ENGINE
4. BUSINESS RULES MANAGEMENT SYSTEM
   - WEBSTUDIO
   - RULES REPOSITORY
   - WEB SERVICES
5. ACCESSING RULES REPOSITORY FROM EXTERNAL JAVA APPLICATIONS
6. FINAL THOUGHTS

Simple is powerful, but OpenL Tablets equips the user also with more complex constructs, according to the level of necessary power



Scenario #1

Define an Excel OpenL Rule with the following characteristics:

Name	Return Type	Parameters
isPremiumAllowed	**Boolean**	• **vehicleAge : int** • **vehicleValue : int**

Desired behavior:

- Return FALSE When vehicleAge > 25 OR vehicleValue < 5000
- Return TRUE in all other cases

Other requirements: None

We will not start with the Excel file…

The first thing you ask is not *“What code will I write?”*
The first thing you ask is *“How will I know that I’ve solved the problem?”*

(Tim King)

(http://sd.jtimothyking.com/2006/07/11/twelve-benefits-of-writing-unit-tests-first/)

The test:

```
public class TestScenario1 {

    private static final String EXCEL_FILE = "src/test/resources/Scenario1.xlsx";

    @Test
    void testPremiumAllowedRule() throws Exception {
        RulesEngineFactory<MyRules> rulesFactory = new RulesEngineFactory<>(EXCEL_FILE, MyRules.class);
        MyRules rules = rulesFactory.newEngineInstance();
        assertFalse(rules.isPremiumAllowed(26, 10000));
        assertTrue(rules.isPremiumAllowed(25, 10000));
        assertTrue(rules.isPremiumAllowed(5, 5000));
        assertFalse(rules.isPremiumAllowed(5, 4999));
    }

    private static interface MyRules {
        boolean isPremiumAllowed(int vehicleAge, int currentVehicleMarketValue);
    }

}
```

The Excel Rule:

SimpleRules Boolean isPremiumAllowed(int vehicleAge, int vehicleValue)		
Age	Price	Decision
26+		no
	< 5000	no
		yes

In OpenL Tablets, everything is a table

And the table boundaries are:

- Empty row
- Empty column
- Sheet margins

Comments can be added on table top/down but leave one empty row between comment and table

SimpleRules Boolean isPremiumAllowed(int vehicleAge, int vehicleValue)		
Age	Price	Decision
26+		no
	< 5000	no
		yes

<- And one empty column
if the comment is on table side

First row of a table is the Table Header

It is a best practice to merge all the cells that compose the table header

SimpleRules Boolean isPremiumAllowed(int vehicleAge, int vehicleValue)		
Age	Price	Decision
26+		no
	< 5000	no
		yes

The first word of the table header is a reserved word

Its value specifies the table type. All the other table data and table structure is parsed according to this word

Possible values: Constants, ColumnMatch, Data, Datatype, Environment, Method, Properties, Rules, Run, SimpleLookup, SimpleRules, SmartLookup, SmartRules, Spreadsheet, TablePart, TBasic or Algorithm, Test

SimpleRules Boolean isPremiumAllowed(int vehicleAge, int vehicleValue)		
Age	Price	Decision
26+		no
	< 5000	no
		yes

Rules, SimpleLookup, SimpleRules, SmartLookup, SmartRules are different types of Decision Table

Decision Tables: The rest of the table header is the method signature

```
private static interface MyRules {
    boolean isPremiumAllowed(int vehicleAge, int currentVehicleMarketValue);
}
```

SimpleRules Boolean isPremiumAllowed(int vehicleAge, int vehicleValue)		
Age	Price	Decision
26+		no
	< 5000	no
		yes

SimpleRules: The table rows

The next row after the table header contains the column headers. It is mandatory, but in this case, the value of the headers is not parsed, it is only for the human eyes:

SimpleRules Boolean isPremiumAllowed(int vehicleAge, int vehicleValue)		
Age	Price	Decision
26+		no
	< 5000	no
		yes

The other rows contain the actual rules:

SimpleRules Boolean isPremiumAllowed(int vehicleAge, int vehicleValue)		
Age	Price	Decision
26+		no
	< 5000	no
		yes

SimpleRules: The table columns

There must be exactly n+1 columns, where the n is the number of method parameters. The value specified in the first column is compared to the value of the first input parameter, and so on. The last column is the return column and its value will be the result.

We will use the term condition column for any of the first n columns.

SimpleRules Boolean isPremiumAllowed(int vehicleAge, int vehicleValue)		
Age	Price	Decision
26+		no
	< 5000	no
		yes

Decision Tables: Matching rules

- ✓ An empty value in a condition column means MATCH or TRUE
- ✓ It is performed a logical AND between the Boolean match value of all condition columns (all conditions must be fulfilled)

SimpleRules Boolean isPremiumAllowed(int vehicleAge, int vehicleValue)		
Age	Price	Decision
26+		no
	< 5000	no
		yes

SimpleRules Boolean isPremiumAllowed(int vehicleAge, int vehicleValue)		
Age	Price	Decision
26+	<ANY PRICE>	no
<ANY AGE>	< 5000	no
<ANY AGE>	<ANY PRICE>	yes

Decision Tables: (by default) The first matching rule wins

- ✓ The rules are evaluated top – down
- ✓ The first one that matches is the winner, so the value on its last column will be returned
- ✓ It is possible to return an array. In this case, specify on the last column the array values, separated by comma (,)
  In case of SimpleRules, the header would be: SimpleRules <returnType>[] <name>(<parameters>)

SimpleRules Boolean isPremiumAllowed(int vehicleAge, int vehicleValue)		
Age	Price	Decision
26+		no
	< 5000	no
		yes

- ✓ It is possible to return the values of all matching rules.
  In case of SimpleRules, the header would be:
  → For an array: SimpleRules Collect <returnElementType>[] <name>(<parameters>)
  → For a list: SimpleRules Collect as <returnElementType> List <name>(<parameters>)

SimpleRules: Using IntRange

SimpleRules Boolean isPremiumAllowed(int vehicleAge, int vehicleValue)		
Age	Price	Decision
26+		no
	< 5000	no
		yes

Integer.MIN_VALUE

< 5000 → -∞ … , 4998, 4999

26+ ⇔ >=26 → 26, 27, 28, … +∞

[1; 5] ⇔ [1 .. 5] ⇔ [1 – 5] ⇔ >=1 <=5 ⇔ <=5 >=1 ⇔ 1 .. 5 ⇔ 1 – 5 → 1, 2, 3, 4, 5

(1; 5) ⇔ (1 .. 5) ⇔ (1 – 5) ⇔ >1 <5 ⇔ <5 >1 ⇔ 1 ... 5 → 2, 3, 4

[1; 5) ⇔ [1 .. 5) ⇔ [1 – 5) ⇔ >=1 <5 ⇔ <5 >=1 → 1, 2, 3, 4

(1; 5] ⇔ (1 .. 5] ⇔ (1 – 5] ⇔ >1 <=5 ⇔ <=5 >1 → 2, 3, 4, 5

✓ There are other range data types available, dealing with floating point numbers, characters, dates etc.

Decision Tables: Flexible Boolean values

- ✓ OpenL supports the native Excel Boolean values TRUE / FALSE
- ✓ Additional to these, it supports other text values:
  - true, yes, y
  - false, no, n

SimpleRules Boolean isPremiumAllowed(int vehicleAge, int vehicleValue)		
Age	Price	Decision
26+		no
	< 5000	no
		yes

Scenario 1 complete

```
public class TestScenario1 {

    private static final String EXCEL_FILE = "src/test/resources/Scenario1.xlsx";

    @Test
    void testPremiumAllowedRule() throws Exception {
        RulesEngineFactory<MyRules> rulesFactory = new RulesEngineFactory<>(EXCEL_FILE, MyRules.class);
        MyRules rules = rulesFactory.newEngineInstance();
        assertFalse(rules.isPremiumAllowed(26, 10000));
        assertTrue(rules.isPremiumAllowed(25, 10000));
        assertTrue(rules.isPremiumAllowed(5, 5000));
        assertFalse(rules.isPremiumAllowed(5, 4999));
    }

    private static interface MyRules {
        boolean isPremiumAllowed(int vehicleAge, int currentVehicleMarketValue);
    }

}
```

SimpleRules Boolean isPremiumAllowed(int vehicleAge, int vehicleValue)		
Age	Price	Decision
26+		no
	< 5000	no
		yes

Wait a minute…
If the business users are defining the Excel Rules and software developers are maintaining the JUnit tests… the software developers might have to change the tests after each update…

Tests can be defined as Excel tables… But first we check they are correct

```
public class TestScenario1WithAssertionsInExcel {
	private static final String EXCEL_FILE = "src/test/resources/Scenario1.xlsx";
	private static final Logger LOG = LoggerFactory.getLogger(TestScenario1WithAssertionsInExcel.class);

	@Test
	void testPremiumAllowedRule() throws Exception {
		RulesEngineFactory<?> rulesFactory = new RulesEngineFactory<>(EXCEL_FILE);
		IOpenClass openClass = rulesFactory.getCompiledOpenClass().getOpenClass();
		SimpleRulesRuntimeEnv env = new SimpleRulesVM().getRuntimeEnv();
		Object target = openClass.newInstance(env);
		// IOpenMethod testMethod = openClass.getMethod("isPremiumAllowedTest", new IOpenClass[0]);
		for (IOpenMethod testMethod : getTestMethods(openClass)) {
			LOG.info("Executing tests in table: {}", testMethod.getName());
			@SuppressWarnings("unchecked")
			TestUnitsResults results = (TestUnitsResults) testMethod.invoke(target, new Object[0], env);
			if(results.getNumberOfAssertionFailures() > 0) {
				for (ITestUnit testUnit : results.getTestUnits()) {
					for (ComparedResult cRes : testUnit.getComparisonResults()) {
						if(cRes.getStatus() != TestStatus.TR_OK) {
							LOG.info("Expected {} but was {}", cRes.getExpectedValue(), cRes.getActualValue());
						}
					}
				}
				fail("There are test failures in table '" + testMethod.getName() + "'");  }}}

	private List<IOpenMethod> getTestMethods(IOpenClass openClass) {
		List<IOpenMethod> out = new ArrayList<IOpenMethod>();
		for (IOpenMethod method : openClass.getMethods()) {
			String methodName = method.getName().toLowerCase();
			if(methodName.startsWith("test") || methodName.endsWith("test")) { // our convention
				out.add(method);
			}}
		return out; }}
```

In real-world situations it is unlikely to use a similar code. The tests are executed automatically by OpenL at maven build and in WebStudio

Now there will be 2 tables in Excel, one is “production” and the other is test

SimpleRules Boolean isPremiumAllowed(int vehicleAge, int vehicleValue)		
Age	Price	Decision
26+		no
	< 5000	no
		yes

Test isPremiumAllowed isPremiumAllowedTest		
vehicleAge	vehicleValue	_res_
Age	Value	Expected Result
26	10000	FALSE
25	10000	TRUE
5	5000	TRUE
5	4999	FALSE

Another type of table: The “Test” one

SimpleRules Boolean isPremiumAllowed(int vehicleAge, int vehicleValue)		
Age	Price	Decision
26+		no
	< 5000	no
		yes

Test isPremiumAllowed isPremiumAllowedTest		
vehicleAge	vehicleValue	_res_
Age	Value	Expected Result
26	10000	FALSE
25	10000	TRUE
5	5000	TRUE
5	4999	FALSE

The next word in the header is the name of the table we are testing

SimpleRules Boolean isPremiumAllowed(int vehicleAge, int vehicleValue)		
Age	Price	Decision
26+		no
	< 5000	no
		yes

Test isPremiumAllowed isPremiumAllowedTest		
vehicleAge	vehicleValue	_res_
Age	Value	Expected Result
26	10000	FALSE
25	10000	TRUE
5	5000	TRUE
5	4999	FALSE

The last word in the header is the name of the test table

SimpleRules Boolean isPremiumAllowed(int vehicleAge, int vehicleValue)		
Age	Price	Decision
26+		no
	< 5000	no
		yes

Test isPremiumAllowed isPremiumAllowedTest		
vehicleAge	vehicleValue	_res_
Age	Value	Expected Result
26	10000	FALSE
25	10000	TRUE
5	5000	TRUE
5	4999	FALSE

```
IOpenMethod testMethod =
        openClass.getMethod("isPremiumAllowedTest",
                            new IOpenClass[0]);
```

The next row in the table is for the framework (filter similar methods)

SimpleRules Boolean isPremiumAllowed(int vehicleAge, int vehicleValue)		
Age	Price	Decision
26+		no
	< 5000	no
		yes

Keyword

Test isPremiumAllowed isPremiumAllowedTest		
vehicleAge	vehicleValue	_res_
Age	Value	Expected Result
26	10000	FALSE
25	10000	TRUE
5	5000	TRUE
5	4999	FALSE

The next row contains the column headers (only for human eyes)

SimpleRules Boolean isPremiumAllowed(int vehicleAge, int vehicleValue)		
Age	Price	Decision
26+		no
	< 5000	no
		yes

Test isPremiumAllowed isPremiumAllowedTest		
vehicleAge	vehicleValue	_res_
Age	Value	Expected Result
26	10000	FALSE
25	10000	TRUE
5	5000	TRUE
5	4999	FALSE

The remaining rows contain the tests

SimpleRules Boolean isPremiumAllowed(int vehicleAge, int vehicleValue)		
Age	Price	Decision
26+		no
	< 5000	no
		yes

Test isPremiumAllowed isPremiumAllowedTest		
vehicleAge	vehicleValue	_res_
Age	Value	Expected Result
26	10000	FALSE
25	10000	TRUE
5	5000	TRUE
5	4999	FALSE

Scenario #2

Define an Excel OpenL Rule with the following characteristics:

Name	Return Type	Parameters
isPremiumAllowed	**Boolean**	• **vehicleAge : int** • **vehicleValue : int** • **vehicleBody: String**

Desired behavior:

- Return FALSE When vehicleAge > 25
- Return FALSE When (vehicleBody=SUV AND vehicleValue < 5000) OR (vehicleBody=Convertible AND vehicleValue < 4500) OR ((vehicleBody is not SUV or Convertible) AND vehicleValue < 4000)
- Return TRUE in all other cases

Other requirements: None

The rule and test:

SimpleRules Boolean isPremiumAllowed(int vehicleAge, int vehicleValue, String vehicleBody)			
Age	Price	Body Type	Decision
26+			no
	< 5000	SUV	no
	< 4500	Convertible	no
	< 4000		no
			yes

Test isPremiumAllowed isPremiumAllowedTest			
vehicleAge	vehicleValue	vehicleBody	_res_
Age	Value	Body	Expected Result
26	10000	Any	FALSE
25	10000	Any	TRUE
5	5000	SUV	TRUE
5	4999	SUV	FALSE
5	4500	Convertible	TRUE
5	4499	Convertible	FALSE
5	4000	Any	TRUE
5	3999	Any	FALSE

What does “Any” mean?

Scenario #3 = Scenario #2 + restrictions on the car body type

Define an Excel OpenL Rule with the following characteristics:

Name	Return Type	Parameters
isPremiumAllowed	**Boolean**	• **vehicleAge : int** • **vehicleValue : int** • **vehicleBody: String with restricted values**

Desired behavior:

- Return FALSE When vehicleAge > 25
- Return FALSE When (vehicleBody=SUV AND vehicleValue < 5000) OR (vehicleBody=Convertible AND vehicleValue < 4500) OR ((vehicleBody is not SUV or Convertible) AND vehicleValue < 4000)
- Return TRUE in all other cases

Other requirements: The possible body types are: Convertible, Coupe, Hatchback, Jeep, Sedan, SUV, Van, Wagon

Now there will be 2 “production” tables in Excel

SimpleRules Boolean isPremiumAllowed(int vehicleAge, int vehicleValue, CarBodyType vehicleBody)			
Age	Price	Body Type	Decision
26+			no
	< 5000	SUV	no
	< 4500	Convertible	no
	< 4000		no
			yes

Datatype CarBodyType <String>
Convertible
Coupe
Hatchback
Jeep
Sedan
SUV
Van
Wagon

Another type of table: The “Datatype” one

SimpleRules Boolean isPremiumAllowed(int vehicleAge, int vehicleValue, CarBodyType vehicleBody)			
Age	Price	Body Type	Decision
26+			no
	< 5000	SUV	no
	< 4500	Convertible	no
	< 4000		no
			yes

Datatype CarBodyType <String>
Convertible
Coupe
Hatchback
Jeep
Sedan
SUV
Van
Wagon

The next word in the header is the name of the new type

SimpleRules Boolean isPremiumAllowed(int vehicleAge, int vehicleValue, CarBodyType vehicleBody)			
Age	Price	Body Type	Decision
26+			no
	< 5000	SUV	no
	< 4500	Convertible	no
	< 4000		no
			yes

Datatype CarBodyType <String>
Convertible
Coupe
Hatchback
Jeep
Sedan
SUV
Van
Wagon

The last word in the header specifies that caller can use String values

SimpleRules Boolean isPremiumAllowed(int vehicleAge, int vehicleValue, CarBodyType vehicleBody)			
Age	Price	Body Type	Decision
26+			no
	< 5000	SUV	no
	< 4500	Convertible	no
	< 4000		no
			yes

Datatype CarBodyType <String>
Convertible
Coupe
Hatchback
Jeep
Sedan
SUV
Van
Wagon

```
private static interface MyRules {

    boolean isPremiumAllowed(int vehicleAge, int currentVehicleMarketValue,
                             String vehicleBody);

}
```

The remaining rows contain the possible values

SimpleRules Boolean isPremiumAllowed(int vehicleAge, int vehicleValue, CarBodyType vehicleBody)			
Age	Price	Body Type	Decision
26+			no
	< 5000	SUV	no
	< 4500	Convertible	no
	< 4000		no
			yes

Datatype CarBodyType <String>
Convertible
Coupe
Hatchback
Jeep
Sedan
SUV
Van
Wagon

The tests

```
public class TestScenario3 {

	private static final String EXCEL_FILE = "src/test/resources/Scenario3.xlsx";

	@Test
	void testPremiumAllowedRule() throws Exception {
		RulesEngineFactory<MyRules> rulesFactory = new RulesEngineFactory<>(EXCEL_FILE, MyRules.class);
		MyRules rules = rulesFactory.newEngineInstance();
		assertFalse(rules.isPremiumAllowed(26, 10000, "Coupe"));
		assertTrue(rules.isPremiumAllowed(25, 10000, "Coupe"));
		assertTrue(rules.isPremiumAllowed(5, 5000, "SUV"));
		assertFalse(rules.isPremiumAllowed(5, 4999, "SUV"));
		assertTrue(rules.isPremiumAllowed(5, 4500, "Convertible"));
		assertFalse(rules.isPremiumAllowed(5, 4499, "Convertible"));
		assertTrue(rules.isPremiumAllowed(5, 4000, "Sedan"));
		assertFalse(rules.isPremiumAllowed(5, 3999, "Sedan"));
		OutsideOfValidDomainException e = assertThrows(OutsideOfValidDomainException.class,
				() -> rules.isPremiumAllowed(5, 5000, "Any"));
		String expectedExceptionMessage = "Object 'Any' is outside of valid domain 'CarBodyType'. “
				+ "Valid values: [Convertible, Coupe, Hatchback, Jeep, Sedan, SUV, Van, Wagon]",
		assertEquals(expectedExceptionMessage, e.getMessage());
	}

	private static interface MyRules {
		boolean isPremiumAllowed(int vehicleAge, int currentVehicleMarketValue, String vehicleBody);
	}

}
```

Test isPremiumAllowed isPremiumAllowedTest			
vehicleAge	vehicleValue	vehicleBody	_res_
Age	Value	Body	Expected Result
26	10000	Coupe	FALSE
25	10000	Coupe	TRUE
5	5000	SUV	TRUE
5	4999	SUV	FALSE
5	4500	Convertible	TRUE
5	4499	Convertible	FALSE
5	4000	Sedan	TRUE
5	3999	Sedan	FALSE

Unfortunately, we can test the behavior only for proper datatype values

Sharing data type definitions between Java programs and Excel rules

There are 2 approaches:

1. Define data types in Excel. OpenL Tablets provide mechanisms to generate the Java classes based on the Excel definition. These generated classes will be used in Java programs.

It is possible to define complex data types in Excel:

Datatype ZipCode	
String	zip1
String	zip2

Datatype Address	
String	street
String	city
ZipCode	zip

2. Define Java classes as usual and “import” them in Excel, by using another table type:

Environment	
import	intro.openl.rules.scenario4.model

Of course, it is possible to combine those 2 approaches.

Scenario #4 = Scenario #3 + The car body type is java Enumeration

Define an Excel OpenL Rule with the following characteristics:

Name	Return Type	Parameters
isPremiumAllowed	**Boolean**	• **vehicleAge : int** • **vehicleValue : int** • **vehicleBody: intro.openl.rules.scenario4.model.CarBodyType**

Desired behavior:

- Return FALSE When vehicleAge > 25
- Return FALSE When (vehicleBody=SUV AND vehicleValue < 5000) OR (vehicleBody=Convertible AND vehicleValue < 4500) OR ((vehicleBody is not SUV or Convertible) AND vehicleValue < 4000)
- Return TRUE in all other cases

Other requirements: None

The Excel rules, Java Enumeration and Interface used in client applications

Environment	
import	intro.openl.rules.scenario4.model

SimpleRules Boolean isPremiumAllowed(int vehicleAge, int vehicleValue, CarBodyType vehicleBody)			
Age	Price	Body Type	Decision
26+			no
	< 5000	SUV	no
	< 4500	Convertible	no
	< 4000		no
			yes

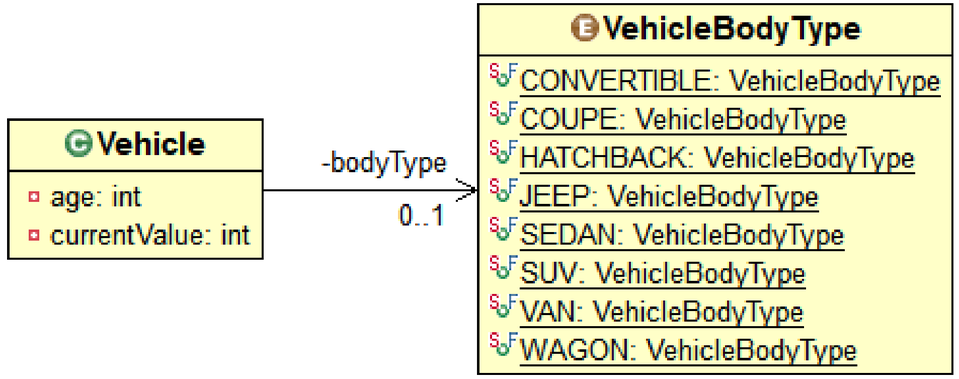
Test isPremiumAllowed isPremiumAllowedTest			
vehicleAge	vehicleValue	vehicleBody	_res_
Age	Value	Body	Expected Result
26	10000	Coupe	FALSE
25	10000	Coupe	TRUE
5	5000	SUV	TRUE
5	4999	SUV	FALSE
5	4500	Convertible	TRUE
5	4499	Convertible	FALSE
5	4000	Sedan	TRUE
5	3999	Sedan	FALSE

```
package intro.openl.rules.scenario4.model;

public enum CarBodyType {
        CONVERTIBLE, COUPE, HATCHBACK, JEEP, SEDAN, SUV,
VAN, WAGON;
}
```

```
private static interface MyRules {
        boolean isPremiumAllowed(int vehicleAge,
                                 int currentVehicleMarketValue,
                                 CarBodyType vehicleBody);
}
```

But if the Excel rules can work with our java classes, can we send a Vehicle object having the age, price and body as properties?



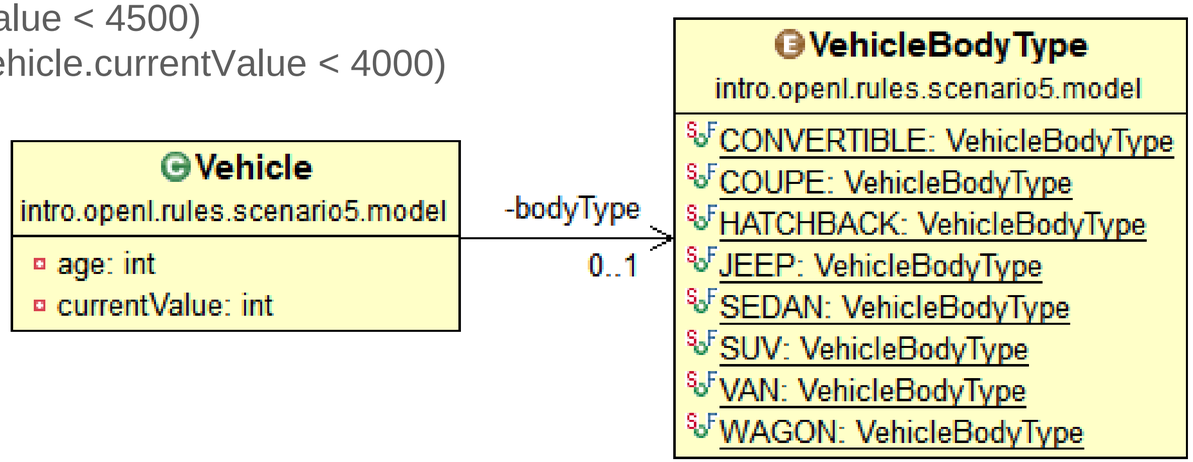
Scenario #5 = Scenario #4, but a single input parameter, a domain POJO

Define an Excel OpenL Rule with the following characteristics:

Name	Return Type	Parameters
isPremiumAllowed	Boolean	• vehicle: intro.openl.rules.scenario5.model.Vehicle

Desired behavior:

- Return FALSE When vehicle.age > 25
- Return FALSE When (vehicle.bodyType=SUV AND vehicle.currentValue < 5000)
  OR (vehicle.bodyType=Convertible AND vehicle.currentValue < 4500)
  OR ((vehicle.bodyType is not SUV or Convertible) AND vehicle.currentValue < 4000)
- Return TRUE in all other cases



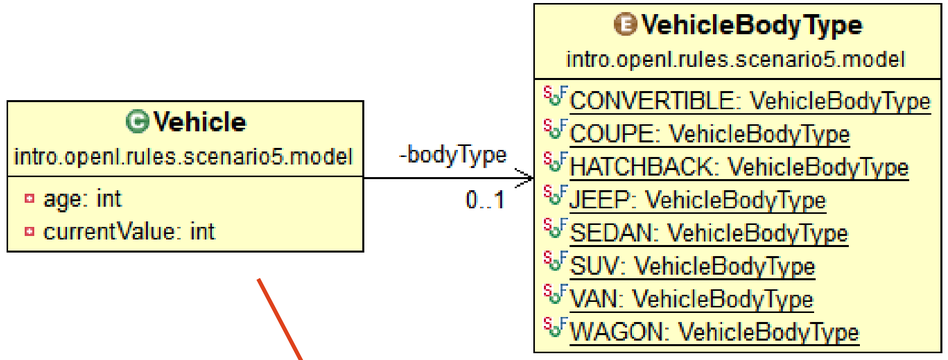
Other requirements: None

The Excel rules and Java Interface used in client applications

Environment	
import	intro.openl.rules.scenario5.model

SmartRules Boolean isPremiumAllowed(Vehicle vehicle)			
Age	Value	Body Type	Decision
26+			no
	< 5000	SUV	no
	< 4500	Convertible	no
	< 4000		no
			yes

Test isPremiumAllowed isPremiumAllowedTest			
vehicle.age	vehicle.currentValue	vehicle.bodyType	_res_
Age	Value	Body	Expected Result
26	10000	Coupe	FALSE
25	10000	Coupe	TRUE
5	5000	SUV	TRUE
5	4999	SUV	FALSE
5	4500	Convertible	TRUE
5	4499	Convertible	FALSE
5	4000	Sedan	TRUE
5	3999	Sedan	FALSE




```
private static interface MyRules {
        boolean isPremiumAllowed(Vehicle vehicle);
}
```

Another type of table: The “SmartRules” one

Environment	
import	intro.openl.rules.scenario5.model

SmartRules Boolean isPremiumAllowed(Vehicle vehicle)			
Age	Value	Body Type	Decision
26+			no
	< 5000	SUV	no
	< 4500	Convertible	no
	< 4000		no
			yes

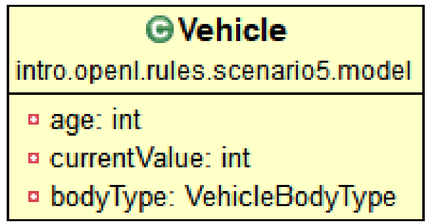
Test isPremiumAllowed isPremiumAllowedTest			
vehicle.age	vehicle.currentValue	vehicle.bodyType	_res_
Age	Value	Body	Expected Result
26	10000	Coupe	FALSE
25	10000	Coupe	TRUE
5	5000	SUV	TRUE
5	4999	SUV	FALSE
5	4500	Convertible	TRUE
5	4499	Convertible	FALSE
5	4000	Sedan	TRUE
5	3999	Sedan	FALSE

Unlike SimpleRules table, now the column headers matter

Environment	
import	intro.openl.rules.scenario5.model

SmartRules Boolean isPremiumAllowed(Vehicle vehicle)			
Age	Value	Body Type	Decision
26+			no
	< 5000	SUV	no
	< 4500	Convertible	no
	< 4000		no
			yes

Test isPremiumAllowed isPremiumAllowedTest			
vehicle.age	vehicle.currentValue	vehicle.bodyType	_res_
Age	Value	Body	Expected Result
26	10000	Coupe	FALSE
25	10000	Coupe	TRUE
5	5000	SUV	TRUE
5	4999	SUV	FALSE
5	4500	Convertible	TRUE
5	4499	Convertible	FALSE
5	4000	Sedan	TRUE
5	3999	Sedan	FALSE



This type of table tries to parse the name of the column headers and match these name with the attributes of input/output objects.

Thus, the column headers are both for human eyes and for OpenL Tablets parsing engine:

- *Flexibility*: The header value does not need to match exactly the name of the attribute. There can be words in the header, not necessary all the words from the attribute, and not only the words from the attribute. For instance “Vehicle Body” can be placed instead of “Body Type” and the table remains valid.

- *Restriction*: But we cannot put something totally different. For instance we cannot have “Price” instead of “Value”, for instance, as we had on the SimpleRules table examples.

With this table type, custom POJOs can be returned as result

The test table is also slightly changed

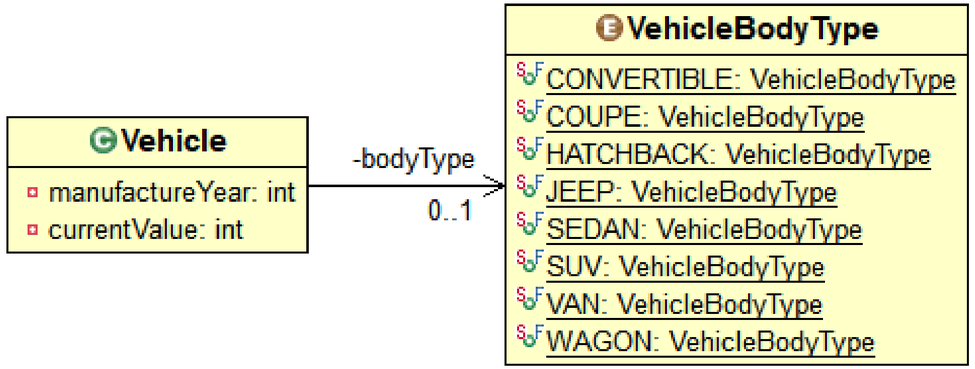
Environment	
import	intro.openl.rules.scenario5.model

SmartRules Boolean isPremiumAllowed(Vehicle vehicle)			
Age	Value	Body Type	Decision
26+			no
	< 5000	SUV	no
	< 4500	Convertible	no
	< 4000		no
			yes

Test isPremiumAllowed isPremiumAllowedTest			
vehicle.age	vehicle.currentValue	vehicle.bodyType	_res_
Age	Value	Body	Expected Result
26	10000	Coupe	FALSE
25	10000	Coupe	TRUE
5	5000	SUV	TRUE
5	4999	SUV	FALSE
5	4500	Convertible	TRUE
5	4499	Convertible	FALSE
5	4000	Sedan	TRUE
5	3999	Sedan	FALSE

Because the input columns are now properties of an input object

What if there is not all the necessary data in the input object?
For instance, is it possible to compute the age of the vehicle in Excel, in case we have only the manufacture year in the Vehicle class?



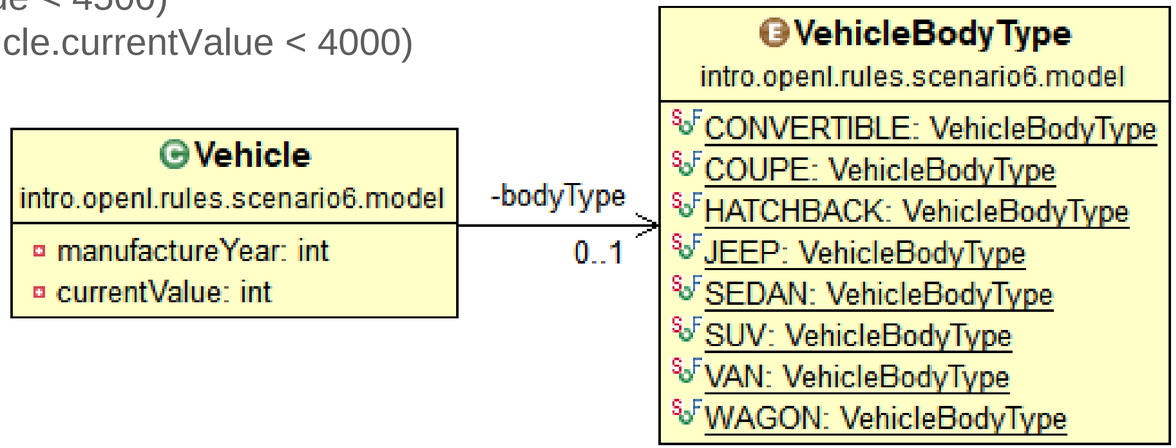
Scenario #6 = Scenario #5, but vehicle has manufactureYear instead of age

Define an Excel OpenL Rule with the following characteristics:

Name	Return Type	Parameters
isPremiumAllowed	**Boolean**	• **vehicle: intro.openl.rules.scenario6.model.Vehicle**

Desired behavior:

- Return FALSE When vehicle.age > 25
- Return FALSE When (vehicle.bodyType=SUV AND vehicle.currentValue < 5000)
  OR (vehicle.bodyType=Convertible AND vehicle.currentValue < 4500)
  OR ((vehicle.bodyType is not SUV or Convertible) AND vehicle.currentValue < 4000)
- Return TRUE in all other cases



Other requirements: None

The “production” Excel tables:

Environment	
import	intro.openl.rules.scenario6.model

Rules Boolean isPremiumAllowed(Vehicle vehicle)			
C1	C2	C3	RET
currentYear() - vehicle.manufactureYear > vaMin	vehicle.currentValue < cvvMin	vehicle.bodyType	allowed
int vaMin	int cvvMin	VehicleBodyType	boolean allowed
Age	Value	Body Type	Decision
25			no
	5000	SUV	no
	4500	Convertible	no
	4000		no
			yes

Method int **currentYear**()
return year(new Date());

Another type of table: The “Method” one

Method int **currentYear**()
return year(new Date());

- This type of table allows java code in Excel file
- Thus, it is possible to achieve maximum flexibility

Another type of table: The “Method” one

Method int **currentYear**()
return year(new Date());

- This type of table allows java code in Excel file
- Thus, it is possible to achieve maximum flexibility
- On the table header, after the “Method” keyword, it is the method signature
- The method can have parameters of any type visible to OpenL Tablets

Another type of table: The “Method” one

Method int **currentYear**()
return year(new Date());

- This type of table allows java code in Excel file
- Thus, it is possible to achieve maximum flexibility
- On the table header, after the “Method” keyword, it is the method signature
- The method can have parameters of any type visible to OpenL Tablets
- The second row and the following ones, compose the method body
- In the method body, it is possible to call methods:
  - On the method parameters
  - Public static methods on any class visible to OpenL Tablets

Another type of table: The “Method” one

Method int **currentYear**()
return year(new Date());

- This type of table allows java code in Excel file
- Thus, it is possible to achieve maximum flexibility
- On the table header, after the “Method” keyword, it is the method signature
- The method can have parameters of any type visible to OpenL Tablets
- The second row and the following ones, compose the method body
- In the method body, it is possible to call methods:
  - On the method input parameters
  - Public static methods on any class visible to OpenL Tablets
  - Utility methods exposed by OpenL Tablets framework

OpenL Tablets utility methods for working with java.util.Date

Method int **currentYear**()
return year(new Date());

Any of these can be used in any other table type, not only in Method

- toDate(String str)
- toDate(String str, String dateFormat)
- toString(Date dt)
- toString(Date dt, String dateFormat)
- dateDif(startDate, endDate, unit)
- second(Date dt)
- minute(Date dt)
- hour(Date dt)
- hourOfDay(Date dt)
- dayOfMonth(Date dt)
- dayOfWeek(Date dt)
- dayOfYear(Date dt)
- weekOfMonth(Date dt)
- weekOfYear(Date dt)
- month(Date dt)
- quarter(Date dt)
- year(Date dt)
- lastDayOfMonth(Date dt)
- firstDateOfQuarter(int absQuarter)
- lastDateOfQuarter(int absQuarter)
- …

There are many other utility methods available that deal with: Math, Arrays, Strings, etc.

Another type decision table: The “Rules” one

Environment	
import	intro.openl.rules.scenario6.model

Rules Boolean isPremiumAllowed(Vehicle vehicle)			
C1	C2	C3	RET
currentYear() - vehicle.manufactureYear > vaMin	vehicle.currentValue < cvvMin	vehicle.bodyType	allowed
int vaMin	int cvvMin	VehicleBodyType	boolean allowed
Age	Value	Body Type	Decision
25			no
	5000	SUV	no
	4500	Convertible	no
	4000		no
			yes

Method int **currentYear**()
return year(new Date());

There are 3 new rows, in order to instruct the OpenL Tablets engine

Environment	
import	intro.openl.rules.scenario6.model

Rules Boolean isPremiumAllowed(Vehicle vehicle)			
C1	C2	C3	RET
currentYear() - vehicle.manufactureYear > vaMin	vehicle.currentValue < cvvMin	vehicle.bodyType	allowed
int vaMin	int cvvMin	VehicleBodyType	boolean allowed
Age	Value	Body Type	Decision
25			no
	5000	SUV	no
	4500	Convertible	no
	4000		no
			yes

Method int **currentYear**()
return year(new Date());

1st row specifies the column type

Environment	
import	intro.openl.rules.scenario6.model

Rules Boolean isPremiumAllowed(Vehicle vehicle)			
C1	C2	C3	RET
currentYear() - vehicle.manufactureYear > vaMin	vehicle.currentValue < cvvMin	vehicle.bodyType	allowed
int vaMin	int cvvMin	VehicleBodyType	boolean allowed
Age	Value	Body Type	Decision
25			no
	5000	SUV	no
	4500	Convertible	no
	4000		no
			yes

Method int **currentYear**()
return year(new Date());

Possible values:

- Cn = Condition Column
- MCn = Merged rows Condition Column
- HCn = Horizontal Condition Column
- An = Action Column
- RETn = Return Column

2nd row: expression statements for condition / action / return value

Environment	
import	intro.openl.rules.scenario6.model

Rules Boolean isPremiumAllowed(Vehicle vehicle)			
C1	C2	C3	RET
currentYear() - vehicle.manufactureYear > vaMin	vehicle.currentValue < cvvMin	vehicle.bodyType	allowed
int vaMin	int cvvMin	VehicleBodyType	boolean allowed
Age	Value	Body Type	Decision
25			no
	5000	SUV	no
	4500	Convertible	no
	4000		no
			yes

Method int **currentYear**()
return year(new Date());

3rd row contains type and name of parameters in the cells below

Environment	
import	intro.openl.rules.scenario6.model

Rules Boolean isPremiumAllowed(Vehicle vehicle)			
C1	C2	C3	RET
currentYear() - vehicle.manufactureYear > vaMin	vehicle.currentValue < cvvMin	vehicle.bodyType	allowed
int vaMin	int cvvMin	VehicleBodyType	boolean allowed
Age	Value	Body Type	Decision
25			no
	5000	SUV	no
	4500	Convertible	no
	4000		no
			yes

Method int **currentYear**()
return year(new Date());

4th row contains column descriptions (it is only for the human eyes)

Environment	
import	intro.openl.rules.scenario6.model

Rules Boolean isPremiumAllowed(Vehicle vehicle)			
C1	C2	C3	RET
currentYear() - vehicle.manufactureYear > vaMin	vehicle.currentValue < cvvMin	vehicle.bodyType	allowed
int vaMin	int cvvMin	VehicleBodyType	boolean allowed
Age	Value	Body Type	Decision
25			no
	5000	SUV	no
	4500	Convertible	no
	4000		no
			yes

Method int **currentYear**()
return year(new Date());

5th+ rows contain concrete parameter values

Environment	
import	intro.openl.rules.scenario6.model

Rules Boolean isPremiumAllowed(Vehicle vehicle)			
C1	C2	C3	RET
currentYear() - vehicle.manufactureYear > vaMin	vehicle.currentValue < cvvMin	vehicle.bodyType	allowed
int vaMin	int cvvMin	VehicleBodyType	boolean allowed
Age	Value	Body Type	Decision
25			no
	5000	SUV	no
	4500	Convertible	no
	4000		no
			yes

Method int **currentYear**()
return year(new Date());

Only values, not matching rules, as we had at SimpleRules and SmartRules tables.

In this table type, the matching rule is specified in the 2nd row after the table header

The test table is almost identical with the one in the scenario5

But now we need to specify the year when the vehicle was manufactured, instead of the age:

Test isPremiumAllowed isPremiumAllowedTest			
vehicle.manufactureYear	vehicle.currentValue	vehicle.bodyType	_res_
Year of creation	Value	Body	Expected Result
1994	10000	Coupe	FALSE
1995	10000	Coupe	TRUE
2015	5000	SUV	TRUE
2015	4999	SUV	FALSE
2015	4500	Convertible	TRUE
2015	4499	Convertible	FALSE
2015	4000	Sedan	TRUE
2015	3999	Sedan	FALSE

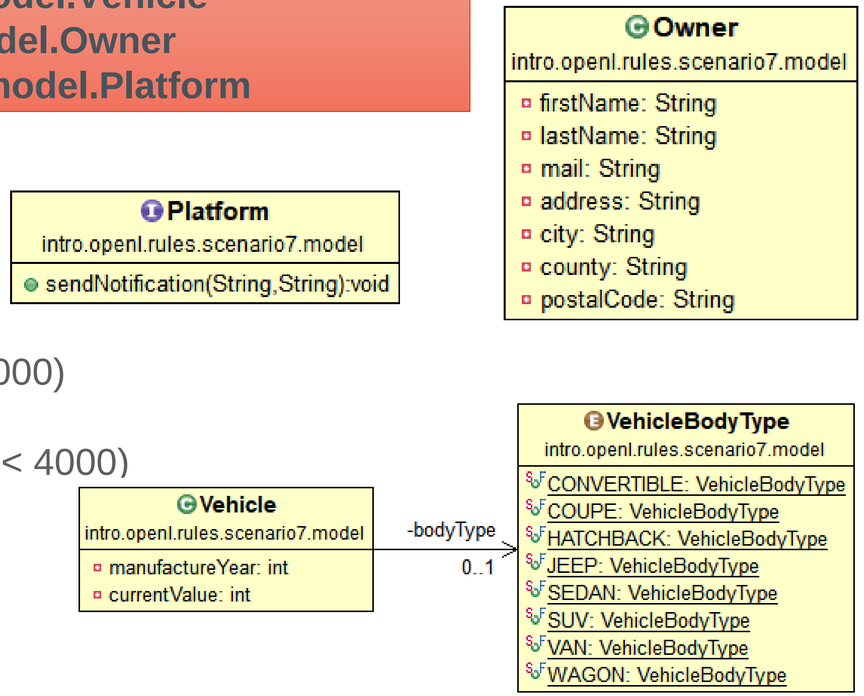
Note: We are now in year 2020. Of course we need to adjust some of the manufacture years after the New Year celebration.

What is an action column, in a “Rules” table?

Scenario #7 = Scenario #6 + notifications (call backs)

Define an Excel OpenL Rule with the following characteristics:

Name	Return Type	Parameters
isPremiumAllowed	**Boolean**	• **vehicle: intro.openl.rules.scenario7.model.Vehicle** • **owner: intro.openl.rules.scenario7.model.Owner** • **platform: intro.openl.rules.scenario7.model.Platform**



Desired behavior:

- Return FALSE When vehicle.age > 25
- Return FALSE When (vehicle.bodyType=SUV AND vehicle.currentValue < 5000)
  OR (vehicle.bodyType=Convertible AND vehicle.currentValue < 4500)
  OR ((vehicle.bodyType is not SUV or Convertible) AND vehicle.currentValue < 4000)
- Return TRUE in all other cases

Other requirements: There is a special survey for rich vehicle owners in Surrey county only. So, the notifications must be sent with code 3575 for cars with value between 50.000 and 100.000, and with the code 3576 for cars with value greater than 100.000. In both cases the message is a CSV containing the last name and the mail of the owner

The “production” Excel tables:

Environment	
import	intro.openl.rules.scenario7.model

Rules Boolean isPremiumAllowed(Vehicle vehicle, Owner o, Platform p)								
C1	C2	C3	C4	C5	C6	RET	A1	
currentYear() - vehicle.manufactureYear > vaMin	vehicle.currentValue < cvvMin	vehicle.bodyType	vehicle.currentValue >= cvvnMin	vehicle.currentValue <= cvvnMax	o.county	allowed	p.sendNotification(c, m)	
int vaMin	int cvvMin	VehicleBodyType	int cvvnMin	int cvvnMax	String	boolean allowed	String c	String m
Age	Value	Body Type	Min range value	Max range value	Owner County	Decision	Code	Message
			50000	100000	Surrey		3575	= o.lastName + "," + o.mail
			100001		Surrey		3576	= o.lastName + "," + o.mail
25						no		
	5000	SUV				no		
	4500	Convertible				no		
	4000					no		
						yes		

Method int **currentYear**()
return year(new Date());

There are 5 new columns

Environment	
import	intro.openl.rules.scenario7.model

Rules Boolean isPremiumAllowed(Vehicle vehicle, Owner o, Platform p)								
C1	C2	C3	C4	C5	C6	RET	A1	
currentYear() - vehicle.manufactureYear > vaMin	vehicle.currentValue < cvvMin	vehicle.bodyType	vehicle.currentValue >= cvvnMin	vehicle.currentValue <= cvvnMax	o.county	allowed	p.sendNotification(c, m)	
int vaMin	int cvvMin	VehicleBodyType	int cvvnMin	int cvvnMax	String	boolean allowed	String c	String m
Age	Value	Body Type	Min range value	Max range value	Owner County	Decision	Code	Message
			50000	100000	Surrey		3575	= o.lastName + "," + o.mail
			100001		Surrey		3576	= o.lastName + "," + o.mail
25						no		
	5000	SUV				no		
	4500	Convertible				no		
	4000					no		
						yes		

Method int **currentYear**()
return year(new Date());

3 new condition columns (necessary for action)

2 action columns (because there are 2 arguments on method call)

And 2 new lines

Environment	
import	intro.openl.rules.scenario7.model

Rules Boolean isPremiumAllowed(Vehicle vehicle, Owner o, Platform p)								
C1	C2	C3	C4	C5	C6	RET	A1	
currentYear() - vehicle.manufactureYear > vaMin	vehicle.currentValue < cvvMin	vehicle.bodyType	vehicle.currentValue >= cvvnMin	vehicle.currentValue <= cvvnMax	o.county	allowed	p.sendNotification(c, m)	
int vaMin	int cvvMin	VehicleBodyType	int cvvnMin	int cvvnMax	String	boolean allowed	String c	String m
Age	Value	Body Type	Min range value	Max range value	Owner County	Decision	Code	Message
			50000	100000	Surrey		3575	= o.lastName + "," + o.mail
			100001		Surrey		3576	= o.lastName + "," + o.mail
25						no		
	5000	SUV				no		
	4500	Convertible				no		
	4000					no		
						yes		

Method int **currentYear**()
return year(new Date());

In this case it is important to leave the return column empty for the new rows, otherwise they will be part of the decision

An important detail: If a table contains action columns, the engine executes actions for all rules with true conditions. If a table has a return column, the engine stops processing rules after the first executed rule with true conditions and non-empty result found

It is possible to execute code in a cell

Environment	
import	intro.openl.rules.scenario7.model

Rules Boolean isPremiumAllowed(Vehicle vehicle, Owner o, Platform p)								
C1	C2	C3	C4	C5	C6	RET	A1	
currentYear() - vehicle.manufactureYear > vaMin	vehicle.currentValue < cvvMin	vehicle.bodyType	vehicle.currentValue >= cvvnMin	vehicle.currentValue <= cvvnMax	o.county	allowed	p.sendNotification(c, m)	
int vaMin	int cvvMin	VehicleBodyType	int cvvnMin	int cvvnMax	String	boolean allowed	String c	String m
Age	Value	Body Type	Min range value	Max range value	er County	Decision	Code	Message
			50000	10			3575	= o.lastName + "," + o.mail
			100001				3576	= o.lastName + "," + o.mail
25								
	5000	SUV						
	4500	Convertible						
	4000					no		
						yes		

Method int **currentYear**()
return year(new Date());

Just use: '= in front of the code (otherwise it will be interpreted as String)

This is interesting, because it makes possible to replace hardcoded values with method calls

I do not want to be querulous, but I see two distinct tables merged into one. Wouldn’t be cleaner if we separate them?

Make it work, make it right, make it fast.

Kent Beck

Scenario #8 = Scenario #7, but 2 tables: decision and notifications

Define an Excel OpenL Rule with the following characteristics:

Name	Return Type	Parameters
isPremiumAllowed	**Boolean**	• **vehicle: intro.openl.rules.scenario8.model.Vehicle**
sendNotifications	**void**	• **vehicle: intro.openl.rules.scenario8.model.Vehicle** • **owner: intro.openl.rules.scenario8.model.Owner** • **platform: intro.openl.rules.scenario8.model.Platform**

Desired behavior:

- Return FALSE When vehicle.age > 25
- Return FALSE When (vehicle.bodyType=SUV AND vehicle.currentValue < 5000)
  OR (vehicle.bodyType=Convertible AND vehicle.currentValue < 4500)
  OR ((vehicle.bodyType is not SUV or Convertible) AND vehicle.currentValue < 4000)
- Return TRUE in all other cases

Other requirements: There is a special survey for rich vehicle owners in Surrey county only. So, the notifications must be sent with code 3575 for cars with value between 50.000 and 100.000, and with the code 3576 for cars with value greater than 100.000. In both cases the message is a CSV containing the last name and the mail of the owner

The “production” Excel tables:

Environment	
import	intro.openl.rules.scenario8.model

Rules Boolean isPremiumAllowed(Vehicle vehicle)			
C1	C2	C3	RET
currentYear() - vehicle.manufactureYear > vaMin	vehicle.currentValue < cvvMin	vehicle.bodyType	allowed
int vaMin	int cvvMin	VehicleBodyType	boolean allowed
Age	Value	Body Type	Decision
25			no
	5000	SUV	no
	4500	Convertible	no
	4000		no
			yes

Rules void sendNotifications(Vehicle vehicle, Owner o, Platform p)				
C1	C2	C3	A1	
vehicle.currentValue >= cvvnMin	vehicle.currentValue <= cvvnMax	o.county	p.sendNotification(c, m)	
int cvvnMin	int cvvnMax	String	String c	String m
Min range value	Max range value	Owner County	Code	Message
50000	100000	Surrey	3575	= o.lastName + "," + o.mail
100001		Surrey	3576	= o.lastName + "," + o.mail

Method int **currentYear**()
return year(new Date());

There is an obvious and serious drawback with this approach

The Java client has one method for each Rules table, so it could call them independently:

```
private static interface MyRules {
    boolean isPremiumAllowed(Vehicle vehicle);
    void sendNotifications(Vehicle vehicle, Owner owner, Platform platform);
}
```

It still does not look ok…
What if a client forgets to call the notifications?
Why should the client know about notifications,
in the first place?
Isn’t this a misplaced responsibility?
What we really need is to call a mini-algorithm,
and it will be the responsibility of that algorithm
to call whatever parts it needs…

Scenario #9 = Scenario #7 for clients, but contains an algorithm inside

Define an Excel OpenL Rule with the following characteristics:

Name	Return Type	Parameters
isPremiumAllowed	**Boolean**	• **vehicle: intro.openl.rules.scenario7.model.Vehicle** • **owner: intro.openl.rules.scenario7.model.Owner** • **platform: intro.openl.rules.scenario7.model.Platform**

Desired behavior:

- Return FALSE When vehicle.age > 25
- Return FALSE When (vehicle.bodyType=SUV AND vehicle.currentValue < 5000)
  OR (vehicle.bodyType=Convertible AND vehicle.currentValue < 4500)
  OR ((vehicle.bodyType is not SUV or Convertible) AND vehicle.currentValue < 4000)
- Return TRUE in all other cases

Other requirements: There is a special survey for rich vehicle owners in Surrey county only. So, the notifications must be sent with code 3575 for cars with value between 50.000 and 100.000, and with the code 3576 for cars with value greater than 100.000. In both cases the message is a CSV containing the last name and the mail of the owner

The “production” Excel tables:

Environment	
import	intro.openl.rules.scenario7.model

Spreadsheet Boolean isPremiumAllowed(Vehicle vehicle, Owner owner, Platform platform)	
Step	Formula
x	= sendNotifications(vehicle, owner, platform)
vehicleAge	= currentYear() - vehicle.manufactureYear
RETURN	= isPremiumAllowedRule($vehicleAge, vehicle.currentValue, vehicle.bodyType)

SimpleRules Boolean isPremiumAllowedRule(int vAge, int vValue, VehicleBodyType vBodyType)			
Age	Value	Body Type	Decision
26+			no
	< 5000	SUV	no
	< 4500	Convertible	no
	< 4000		no
			yes

Rules void sendNotifications(Vehicle vehicle, Owner o, Platform p)				
C1	C2	C3	A1	
vehicle.currentValue >= cvvnMin	vehicle.currentValue <= cvvnMax	o.county	p.sendNotification(c, m)	
int cvvnMin	int cvvnMax	String	String c	String m
Min range value	Max range value	Owner County	Code	Message
50000	100000	Surrey	3575	= o.lastName + "," + o.mail
100001		Surrey	3576	= o.lastName + "," + o.mail

Method int **currentYear**()
return year(new Date());

Another type of table: The “Spreadsheet” one

Environment	
import	intro.openl.rules.scenario7.model

Spreadsheet Boolean isPremiumAllowed(Vehicle vehicle, Owner owner, Platform platform)	
Step	Formula
x	= sendNotifications(vehicle, owner, platform)
vehicleAge	= currentYear() - vehicle.manufactureYear
RETURN	= isPremiumAllowedRule($vehicleAge, vehicle.currentValue, vehicle.bodyType)

SimpleRules Boolean isPremiumAllowedRule(int vAge, int vValue, VehicleBodyType vBodyType)			
Age	Value	Body Type	Decision
26+			no
	< 5000	SUV	no
	< 4500	Convertible	no
	< 4000		no
			yes

Rules void sendNotifications(Vehicle vehicle, Owner o, Platform p)				
C1	C2	C3	A1	
vehicle.currentValue >= cvvnMin	vehicle.currentValue <= cvvnMax	o.county	p.sendNotification(c, m)	
int cvvnMin	int cvvnMax	String	String c	String m
Min range value	Max range value	Owner County	Code	Message
50000	100000	Surrey	3575	= o.lastName + "," + o.mail
100001		Surrey	3576	= o.lastName + "," + o.mail

Method int **currentYear**()
return year(new Date());

One step at a time…

Environment	
import	intro.openl.rules.scenario7.model

Spreadsheet Boolean isPremiumAllowed(Vehicle vehicle, Owner owner, Platform platform)	
Step	Formula
x	= sendNotifications(vehicle, owner, platform)
vehicleAge	= currentYear() - vehicle.manufactureYear
RETURN	= isPremiumAllowedRule($vehicleAge, vehicle.currentValue, vehicle.bodyType)

SimpleRules Boolean isPremiumAllowedRule(int vAge, int vValue, VehicleBodyType vBodyType)			
Age	Value	Body Type	Decision
26+			no
	< 5000	SUV	no
	< 4500	Convertible	no
	< 4000		no
			yes

Rules void sendNotifications(Vehicle vehicle, Owner o, Platform p)				
C1	C2	C3	A1	
vehicle.currentValue >= cvvnMin	vehicle.currentValue <= cvvnMax	o.county	p.sendNotification(c, m)	
int cvvnMin	int cvvnMax	String	String c	String m
Min range value	Max range value	Owner County	Code	Message
50000	100000	Surrey	3575	= o.lastName + "," + o.mail
100001		Surrey	3576	= o.lastName + "," + o.mail

Method int **currentYear**()
return year(new Date());

Each “Step” has 2 parts:

- Left side: variable name
- Right side: variable value

The left side is mandatory to be defined, in case we want to execute the right side, even if the result of execution is **void**

2 flavors of Spreadsheet

Environment	
import	intro.openl.rules.scenario7.model

Spreadsheet Boolean isPremiumAllowed(Vehicle vehicle, Owner owner, Platform platform)	
Step	Formula
x	= sendNotifications(vehicle, owner, platform)
vehicleAge	= currentYear() - vehicle.manufactureYear
RETURN	= isPremiumAllowedRule($vehicleAge, vehicle.currentValue, vehicle.bodyType)

SimpleRules Boolean isPremiumAllowedRule(int vAge, int vValue, VehicleBodyType vBodyType)			
Age	Value	Body Type	Decision
26+			no
	< 5000	SUV	no
	< 4500	Convertible	no
	< 4000		no
			yes

Rules void sendNotifications(Vehicle vehicle, Owner o, Platform p)				
C1	C2	C3	A1	
vehicle.currentValue >= cvvnMin	vehicle.currentValue <= cvvnMax	o.county	p.sendNotification(c, m)	
int cvvnMin	int cvvnMax	String	String c	String m
Min range value	Max range value	Owner County	Code	Message
50000	100000	Surrey	3575	= o.lastName + "," + o.mail
100001		Surrey	3576	= o.lastName + "," + o.mail

Method int **currentYear**()
return year(new Date());

1) If the return type is SpreadsheetResult, all the variables from the left side will be made available to the clients, thus, there is no need for RETURN on the last row

2) If the return type is something else, the last row should have RETURN on the left side and value (of a compatible type with the Spreadsheet return type) in the right side

This table facilitates the break of complex scenarios into simple ones

Environment	
import	intro.openl.rules.scenario7.model

Spreadsheet Boolean isPremiumAllowed(Vehicle vehicle, Owner owner, Platform platform)	
Step	Formula
x	= sendNotifications(vehicle, owner, platform)
vehicleAge	= currentYear() - vehicle.manufactureYear
RETURN	= isPremiumAllowedRule($vehicleAge, vehicle.currentValue, vehicle.bodyType)

SimpleRules Boolean isPremiumAllowedRule(int vAge, int vValue, VehicleBodyType vBodyType)			
Age	Value	Body Type	Decision
26+			no
	< 5000	SUV	no
	< 4500	Convertible	no
	< 4000		no
			yes

Rules void sendNotifications(Vehicle vehicle, Owner o, Platform p)				
C1	C2	C3	A1	
vehicle.currentValue >= cvvnMin	vehicle.currentValue <= cvvnMax	o.county	p.sendNotification(c, m)	
int cvvnMin	int cvvnMax	String	String c	String m
Min range value	Max range value	Owner County	Code	Message
50000	100000	Surrey	3575	= o.lastName + "," + o.mail
100001		Surrey	3576	= o.lastName + "," + o.mail

Method int **currentYear**()
return year(new Date());

In our example we computed the age of the vehicle in a dedicated step

⇒ It does not need to be computed in decision table

⇒ Decision table can be simpler

If you are looking for more advanced algorithm functionality (like loops for instance), TBasic table type is more appropriate

The client code is unchanged

```
public class TestScenario7 {
    private static final String EXCEL_FILE_PATH = "src/test/resources/";

    @ParameterizedTest
    @ValueSource(strings = { "Scenario7.xlsx", "Scenario9.xlsx" })
    void testPremiumAllowedRule(String excelFileName) throws Exception {
        RulesEngineFactory<MyRules> rulesFactory = new RulesEngineFactory<>(EXCEL_FILE_PATH + excelFileName,
                MyRules.class);
        MyRules rules = rulesFactory.newEngineInstance();
        TestPlatform p = new TestPlatform();
        verifyDecisions(rules, p);
        verifyNotifications(rules, p);
    }

    private void verifyDecisions(MyRules rules, TestPlatform p) {
        Owner oSurrey = createOwner("John", "Max", "john@x.com", "141 Blackborough Rd", "Reigate", "Surrey", "RH2 7DA");
        assertFalse(rules.isPremiumAllowed(new Vehicle(1994, 10000, VehicleBodyType.COUPE), oSurrey, p));
```

I wonder how do we switch rules?
If we want to change a rule at midnight, in a certain day, is it possible? Or, maybe there are some special days where we want to apply different rules…

Scenario #10: Two versions of the same rule

Define an Excel OpenL Rule with the following characteristics:

Name	Return Type	Parameters
getBaseDiscount	**double**	

Desired behavior:

- Return 0 always

Other requirements: There should be another version of the same rule that, on 11 June returns 0.1 (10%), because that day is special for our company and we want to celebrate it by offering that discount to all our customers.

The “production” Excel tables:

6/11/2020 11:59:59 PM

B	C	D	E

SimpleRules double getBaseDiscount()
Discount
0%

SimpleRules double getBaseDiscount()		
properties	startRequestDate	11-Jun-2020
	endRequestDate	6/11/20 23:59
Discount		
10%		

Test getBaseDiscount getBaseDiscountTest	
context.requestDate	_res_
Request Date	Result
6/10/2020	0%
6/11/2020	10%
6/12/2020	0%

There are now 2 rules with the same signature

6/11/2020 11:59:59 PM

B	C	D	E

SimpleRules double getBaseDiscount()
Discount
0%

SimpleRules double getBaseDiscount()		
properties	startRequestDate	11-Jun-2020
	endRequestDate	6/11/20 23:59
Discount		
10%		

Test getBaseDiscount getBaseDiscountTest	
context.requestDate	_res_
Request Date	Result
6/10/2020	0%
6/11/2020	10%
6/12/2020	0%

But one of them has some properties defined

6/11/2020 11:59:59 PM

B	C	D	E

SimpleRules double getBaseDiscount()
Discount
0%

SimpleRules double getBaseDiscount()		
properties	startRequestDate	11-Jun-2020
	endRequestDate	6/11/20 23:59
Discount		
10%		

Test getBaseDiscount getBaseDiscountTest	
context.requestDate	_res_
Request Date	Result
6/10/2020	0%
6/11/2020	10%
6/12/2020	0%

Some possible properties:

- Business Properties:
  - effectiveDate
  - expirationDate
  - startRequestDate
  - endRequestDate
  - lob
  - usregion
  - country
  - currency
  - lang
  - state
  - caProvinces
  - caRegions
  - region
  - origin
  - nature
- Table Versioning
  - version
  - active
- Info Properties
  - category
  - description
  - tags
  - createdBy
  - createdOn
  - modifiedBy
  - modifiedOn
- Dev Properties
  - id
  - buildPhase
  - validateDT
  - failOnMiss
  - scope
  - datatypePackage
  - recalculate
  - cacheable
  - precision
  - autotype
  - parallel

Now, the context must be specified in the tests and java clients

6/11/2020 11:59:59 PM

B	C	D	E

SimpleRules double getBaseDiscount()
Discount
0%

SimpleRules double getBaseDiscount()		
properties	startRequestDate	11-Jun-2020
	endRequestDate	6/11/20 23:59
Discount		
10%		

Test getBaseDiscount getBaseDiscountTest	
context.requestDate	_res_
Request Date	Result
6/10/2020	0%
6/11/2020	10%
6/12/2020	0%

```
public class TestScenario10 {
	private static final String EXCEL_FILE = "src/test/resources/Scenario10.xlsx";
	private SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss");

	@Test
	void testBaseDiscount() throws Exception {
		RulesEngineFactory<MyRules> rulesFactory = new RulesEngineFactory<>(EXCEL_FILE, MyRules.class);
		MyRules rules = rulesFactory.newEngineInstance();

		IRuntimeEnv env = ((IEngineWrapper) rules).getRuntimeEnv();
		IRulesRuntimeContext context = RulesRuntimeContextFactory.buildRulesRuntimeContext();
		env.setContext(context);

		context.setRequestDate(formatter.parse("2020-06-10T23:59:59"));
		assertEquals(0, rules.getBaseDiscount(), 0);

		context.setRequestDate(formatter.parse("2020-06-11T00:00:00"));
		assertEquals(0.1, rules.getBaseDiscount(), 0);

		context.setRequestDate(formatter.parse("2020-06-11T23:59:59"));
		assertEquals(0.1, rules.getBaseDiscount(), 0);

		context.setRequestDate(formatter.parse("2020-06-12T00:00:00"));
		assertEquals(0, rules.getBaseDiscount(), 0);
	}

	private static interface MyRules {
		double getBaseDiscount();
	}
}
```

Any Excel format can be used, including date + time if necessary…

6/11/2020 11:59:59 PM

B	C	D	E

SimpleRules double getBaseDiscount()
Discount
0%

SimpleRules double getBaseDiscount()		
properties	startRequestDate	11-Jun-2020
	endRequestDate	6/11/20 23:59
Discount		
10%		

Test getBaseDiscount getBaseDiscountTest	
context.requestDate	_res_
Request Date	Result
6/10/2020	0%
6/11/2020	10%
6/12/2020	0%

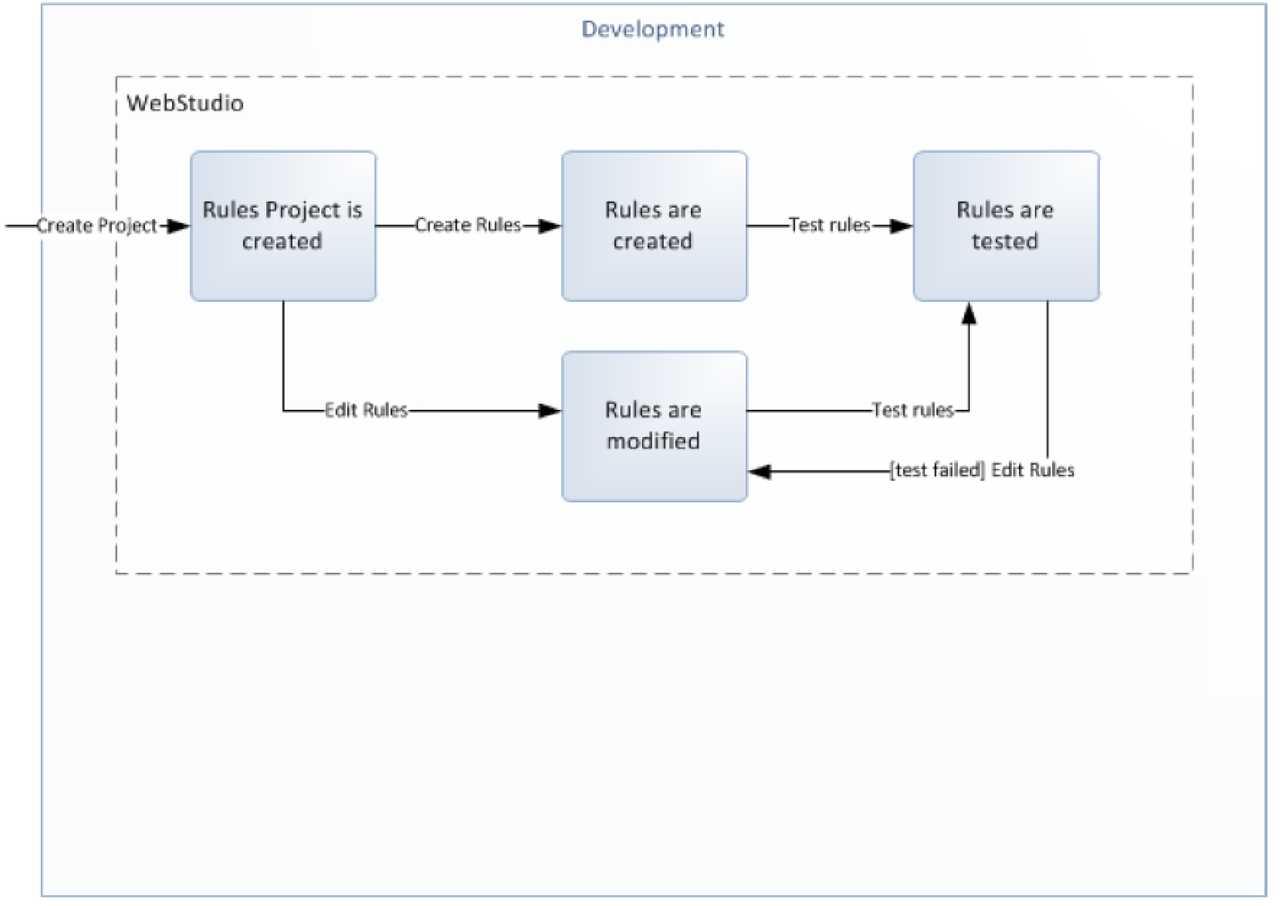
it can also be 0.1

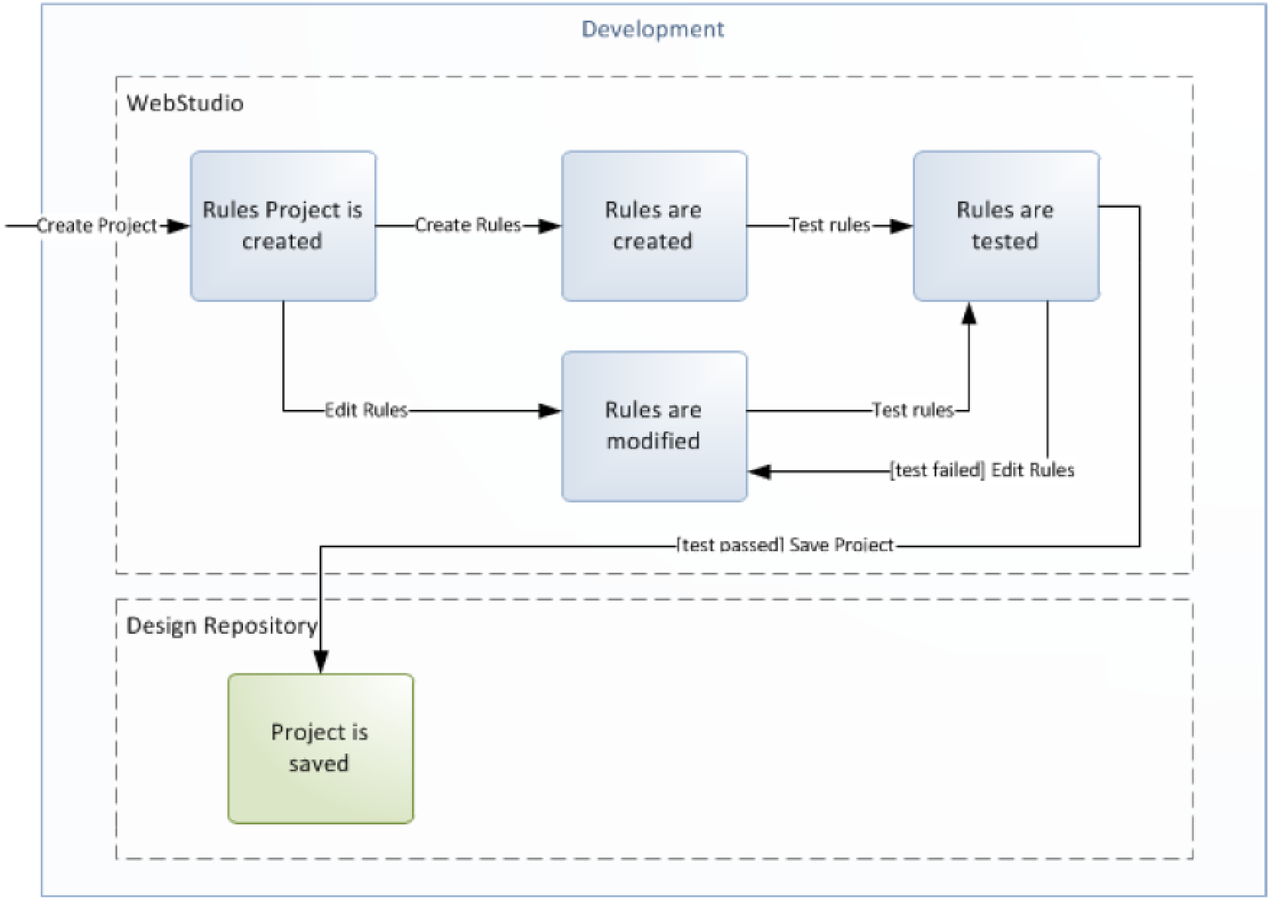
There is much more…

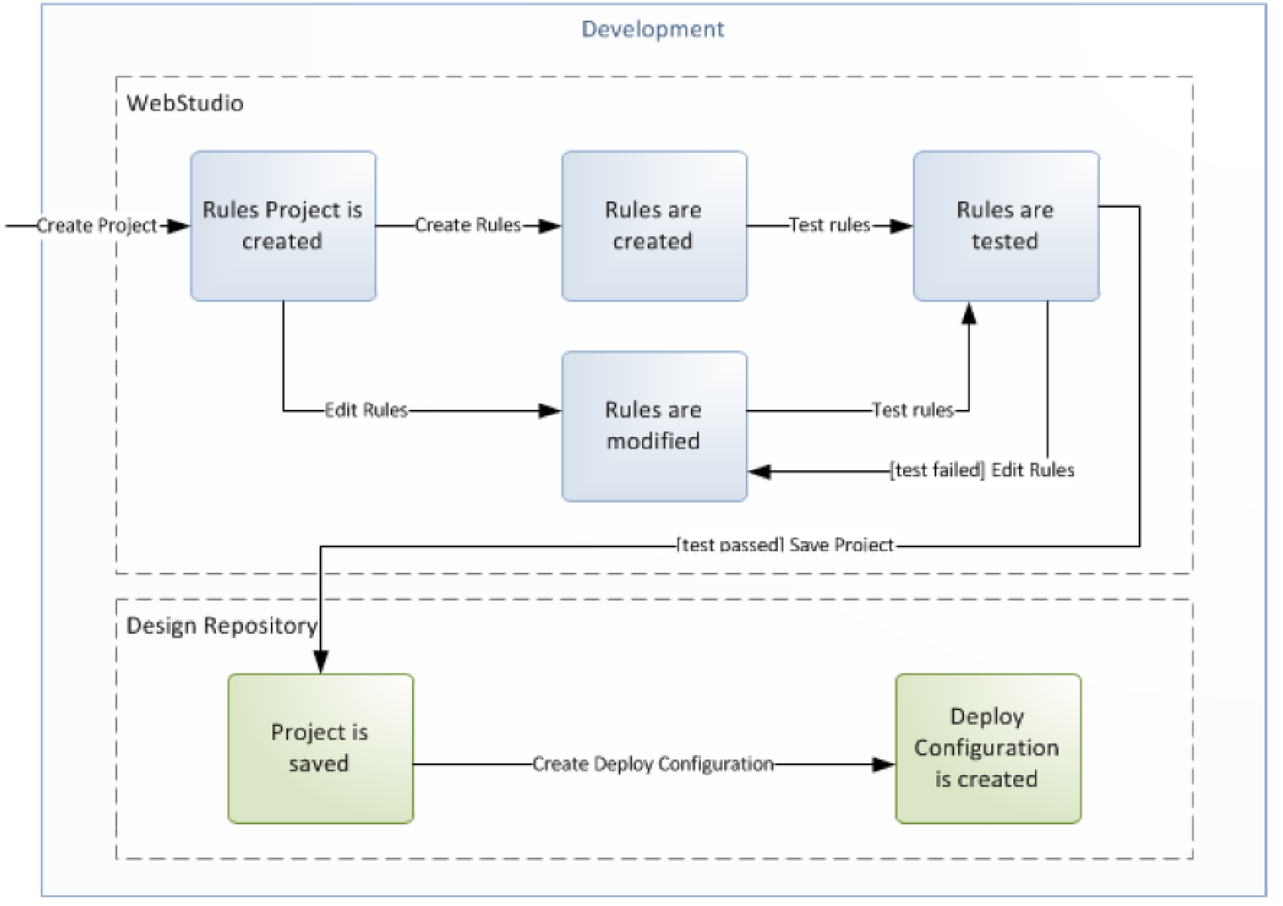
- Usually we work with projects, that are composed of multiple Excel files (called modules)
- There is an OpenL Tablets Business Expression (BEX) language
- We barely touched:
  - ✓ some of the possible table types: Constants, ColumnMatch, Data, Datatype, Environment, Method, Properties, Rules, Run, SimpleLookup, SimpleRules, SmartLookup, SmartRules, Spreadsheet, TablePart, TBasic or Algorithm, Test
  - ✓ Working with functions
  - ✓ Properties, and their scope:
    - ➢ One table
    - ➢ All tables in a specific category
    - ➢ Module
    - ➢ Project
- Start with http://openl-tablets.org/documentation/user-guides (Reference Guide) and http://openl-tablets.org/documentation/videos

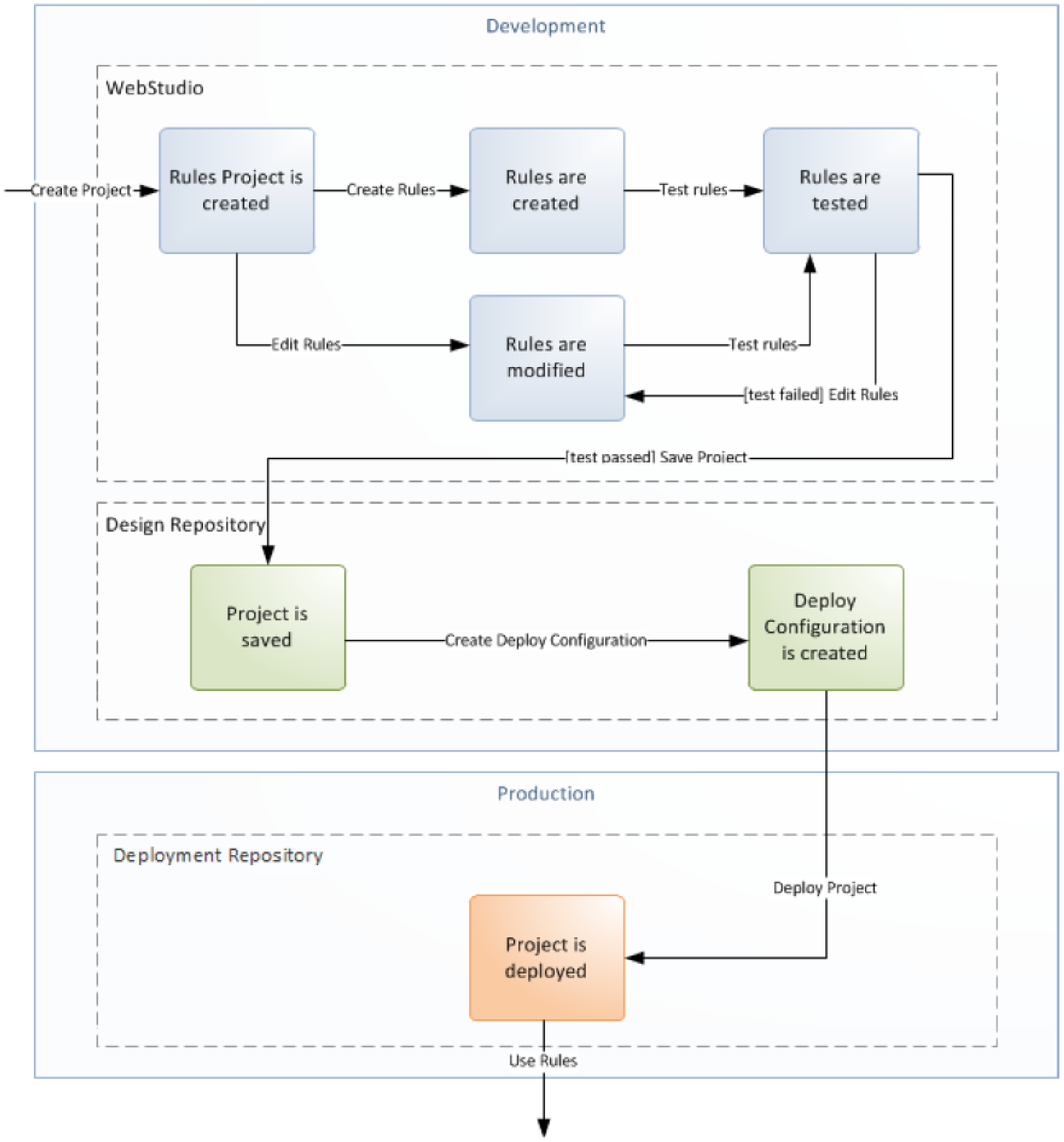
Agenda

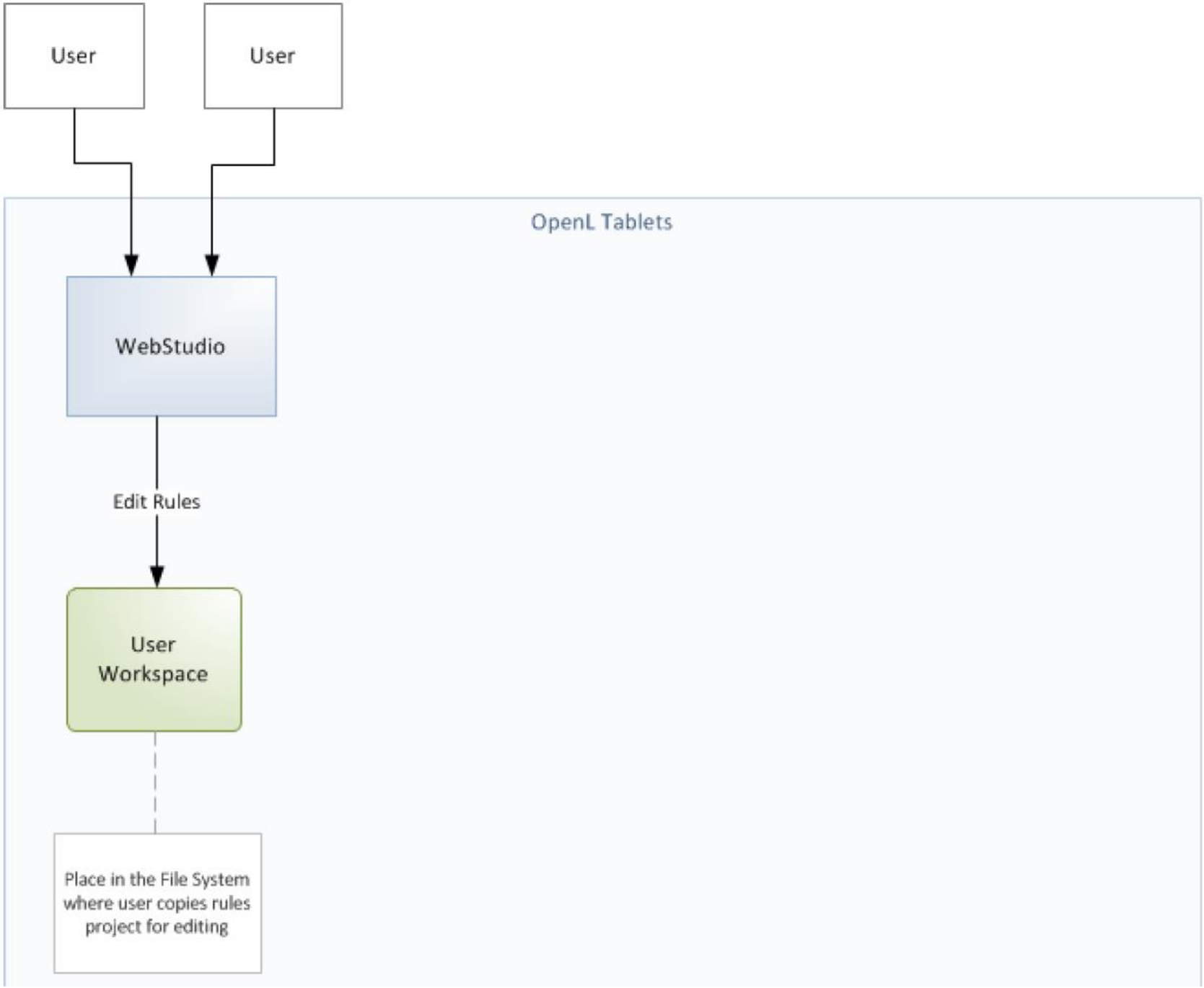
1. WHY?
2. OPENL TABLETS
3. BUSINESS RULES ENGINE
4. BUSINESS RULES MANAGEMENT SYSTEM
   - WEBSTUDIO
   - RULES REPOSITORY
   - WEB SERVICES
5. ACCESSING RULES REPOSITORY FROM EXTERNAL JAVA APPLICATIONS
6. FINAL THOUGHTS

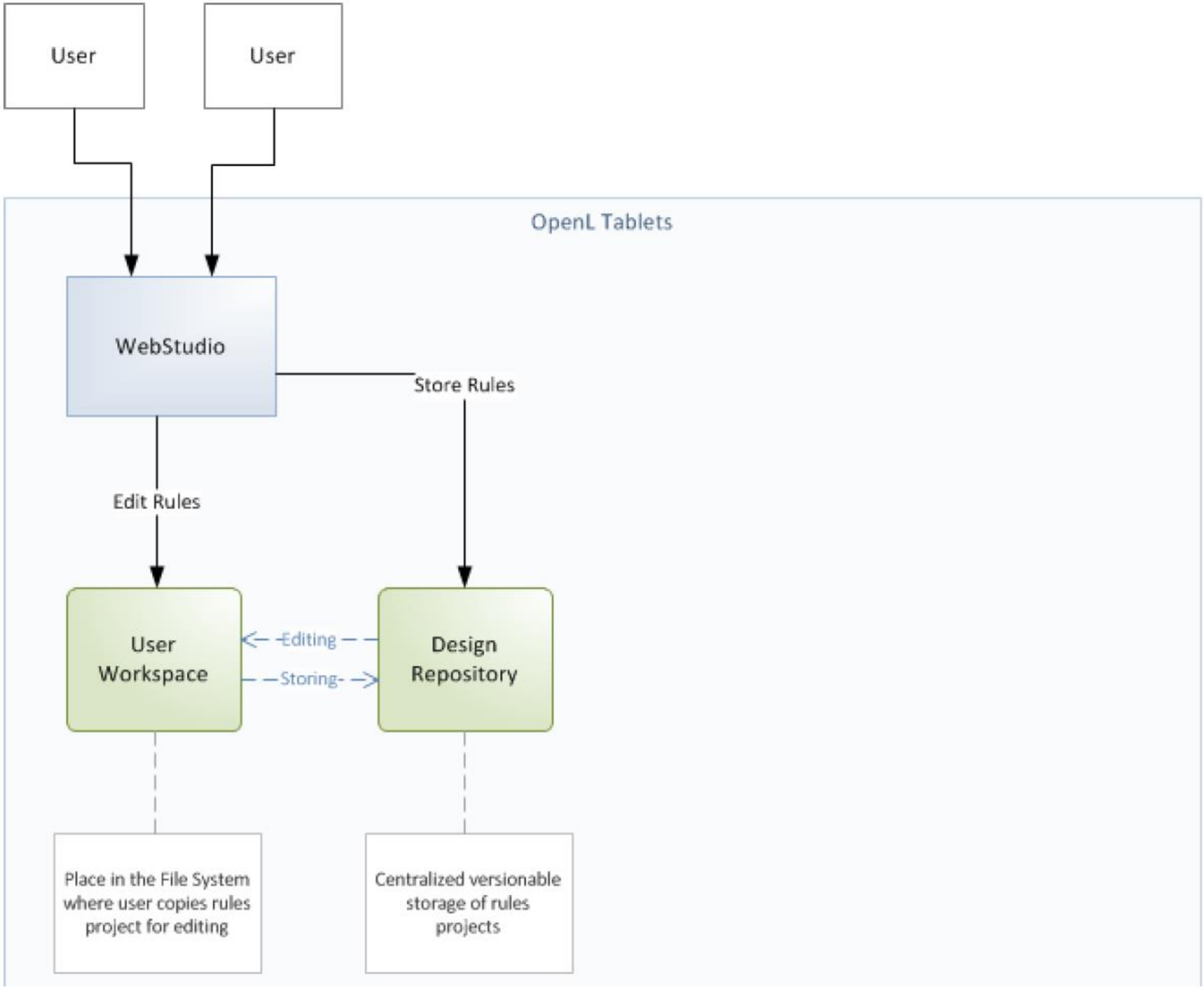

Development
WebStudio
Create Project
Rules Project is created
Create Rules
Rules are created
Test rules
Rules are tested
Edit Rules
Rules are modified
Test rules
[test failed] Edit Rules

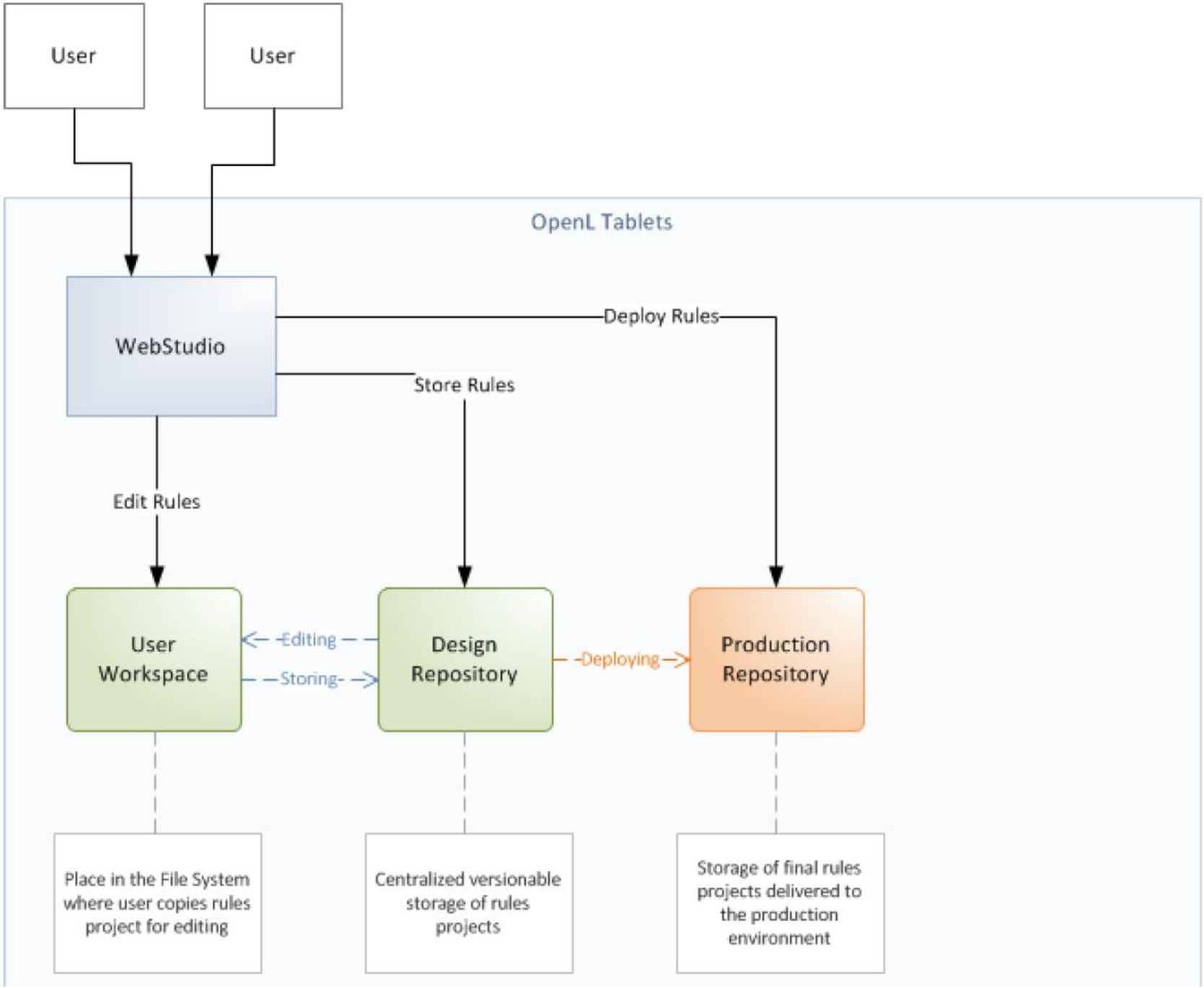

Development
WebStudio
Create Project
Rules Project is created
Create Rules
Rules are created
Test rules
Rules are tested
Edit Rules
Rules are modified
Test rules
[test failed] Edit Rules
[test passed] Save Project
Design Repository
Project is saved

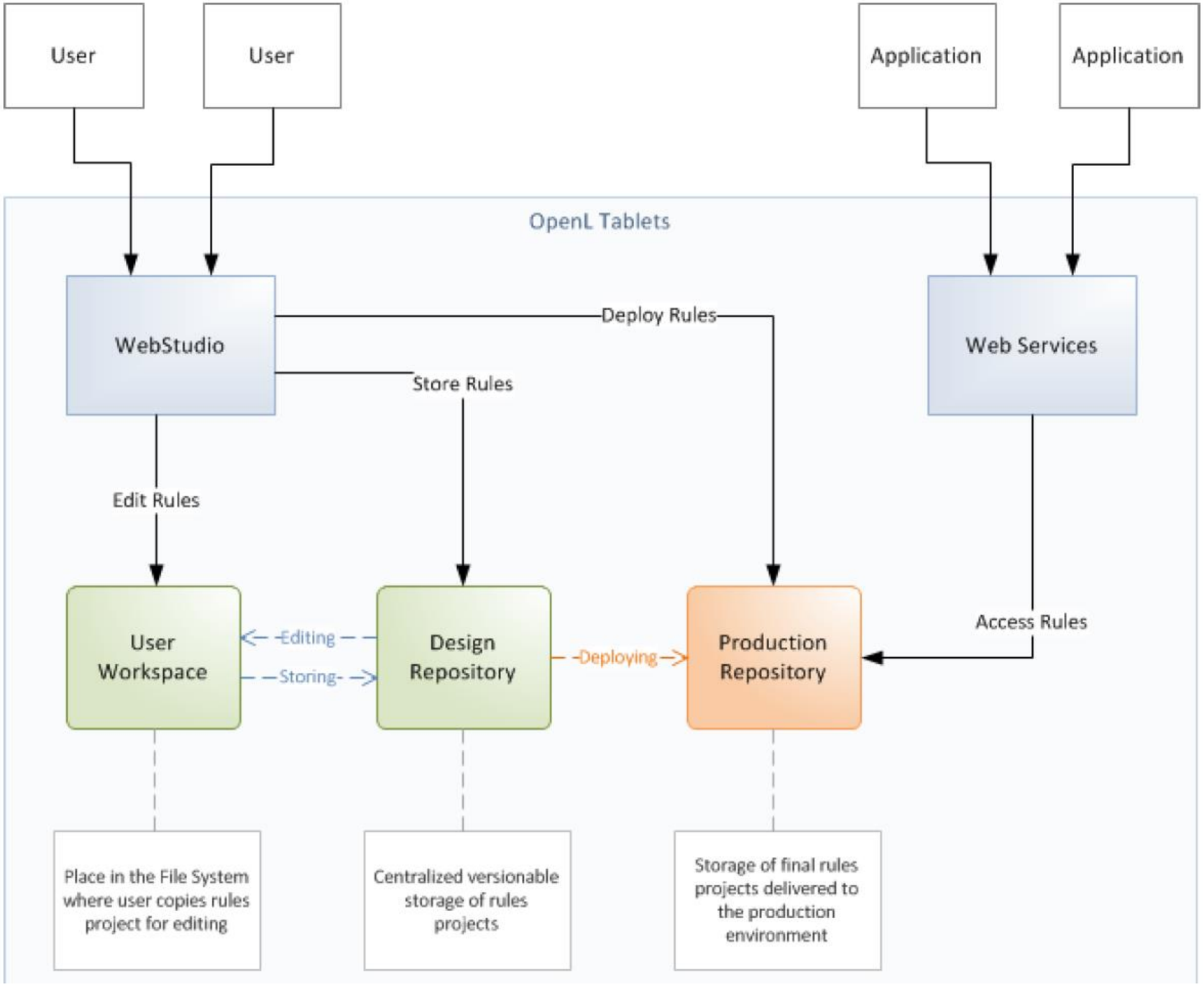

Development
WebStudio
Create Project
Rules Project is created
Create Rules
Rules are created
Test rules
Rules are tested
Edit Rules
Rules are modified
Test rules
[test failed] Edit Rules
[test passed] Save Project
Design Repository
Project is saved
Create Deploy Configuration
Deploy Configuration is created


Development
WebStudio
Create Project
Rules Project is created
Create Rules
Rules are created
Test rules
Rules are tested
Edit Rules
Rules are modified
Test rules
[test failed] Edit Rules
[test passed] Save Project
Design Repository
Project is saved
Create Deploy Configuration
Deploy Configuration is created
Production
Deployment Repository
Deploy Project
Project is deployed
Use Rules


User
User
OpenL Tablets
WebStudio
Edit Rules
User Workspace
Place in the File System where user copies rules project for editing


User
User
OpenL Tablets
WebStudio
Store Rules
Edit Rules
User Workspace
Editing
Storing
Design Repository
Place in the File System where user copies rules project for editing
Centralized versionable storage of rules projects


User
User
OpenL Tablets
WebStudio
Deploy Rules
Store Rules
Edit Rules
User Workspace
Editing
Storing
Design Repository
Deploying
Production Repository
Place in the File System where user copies rules project for editing
Centralized versionable storage of rules projects
Storage of final rules projects delivered to the production environment


User
User
Application
Application
OpenL Tablets
WebStudio
Deploy Rules
Web Services
Store Rules
Edit Rules
User Workspace
Editing
Storing
Design Repository
Deploying
Production Repository
Access Rules
Place in the File System where user copies rules project for editing
Centralized versionable storage of rules projects
Storage of final rules projects delivered to the production environment

Some starting resources:

- http://openl-tablets.org/documentation/getting-started: in this page are links to online demo deployments of OpenL Tablets WebStudio and WebServices, and how to download a zip file containing Tomcat and the war files of these 2 applications.
- http://openl-tablets.org/documentation/user-guides: in this page are the user guides. There are dedicated guides for different parts of the OpenL Tablets journey.
- http://openl-tablets.org/documentation/videos: some videos, not very recent, but very good for the first steps
- http://openl-tablets.org/documentation/tutorials: The zip files and descriptions of several projects, very useful to play with

Agenda

1. WHY?
2. OPENL TABLETS
3. BUSINESS RULES ENGINE
4. BUSINESS RULES MANAGEMENT SYSTEM
   - WEBSTUDIO
   - RULES REPOSITORY
   - WEB SERVICES
5. ACCESSING RULES REPOSITORY FROM EXTERNAL JAVA APPLICATIONS
6. FINAL THOUGHTS

Code demo

Agenda

1. WHY?
2. OPENL TABLETS
3. BUSINESS RULES ENGINE
4. BUSINESS RULES MANAGEMENT SYSTEM
   - WEBSTUDIO
   - RULES REPOSITORY
   - WEB SERVICES
5. ACCESSING RULES REPOSITORY FROM EXTERNAL JAVA APPLICATIONS
6. FINAL THOUGHTS

Things to watch out:

- OpenL Tablets is open source, but it has LGPL license, which is not the most permissive one…
- The documentation is extensive, but most of it makes sense only after the reader is already familiar with the framework and tools.
- Some non-standard actions (for instance deployment of rules from a java class, not from WebStudio) are not documented at all, and they can be achieved only by performing reverse engineering on the existing OpenL Tablets code
- The video tutorials are quite old (they are all from February 2014)
- There is a good support for maven build (openl-maven-plugin), but there is no support for gradle build.
- There is (almost) no documentation on how to perform deployment of the OpenL Tablets project zip file in a database, through openl-maven-plugin. Even with reverse engineering on the code, in order to find the proper setup, I have my doubts that it works on every build (because the *deploy* method of *RulesDeployerService* is called with overridable=false, which means the deployment will only work once for a database)

I am grateful for the pictures used in this presentation to…

- https://freesvg.org/man-using-computer
- https://freesvg.org/sad-computer-user
- https://freesvg.org/man-ready-for-business
- https://freesvg.org/publicdomainq-business-man-inviting
- https://freesvg.org/publicdomainq-business-man-phone-call
- https://freesvg.org/angry-business-man
- https://freesvg.org/strong-man-vector-image
- https://freesvg.org/publicdomainq-business-man-fist-pump
- https://freesvg.org/chart-report
- https://freesvg.org/1471618969
- https://freesvg.org/polygons-from-triangles
- https://freesvg.org/nested-hexagram
- https://freesvg.org/thank-you-note-vector-drawing
- https://www.wallpaperflare.com/question-mark-graphics-help-response-symbol-icon-characters-wallpaper-zwyne

endava

The code is available at: https://github.com/cmihalache/openl-intro