Rules and BPMN concepts:

"Reactive programming is based on propagating and responding to incoming events over time, declaratively."

RxJS is a library for reactive programming using Observables, to make it easier to compose asynchronous or callback-based code. This project is a rewrite of [Reactive-Extensions/RxJS](https://github.com/Reactive-Extensions/RxJS) with better performance, better modularity, better debuggable call stacks, while staying mostly backwards compatible, with some breaking changes that reduce the API surface.

“An event is a **significant change of state** at a particular **point in time**”

“**Complex Event**, is an abstraction of other events called its members.”

“An event is an observable occurrence.”

“An event in the Unified Modelling Language is a notable occurrence at

a particular point in time.”

http://www.wikipedia.org

“Anything that happens, or is contemplated as happening.”

“An object that represents, encodes or records an event, generally for

the purpose of computer processing”

*A workflow engine combining processes and rules*

Overview

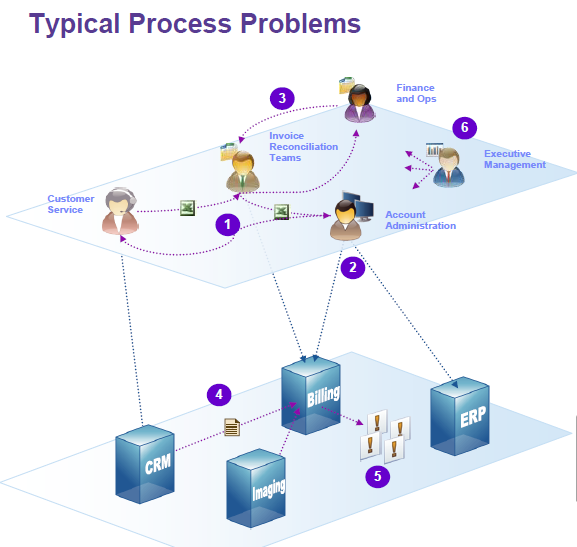
– What is a rule?

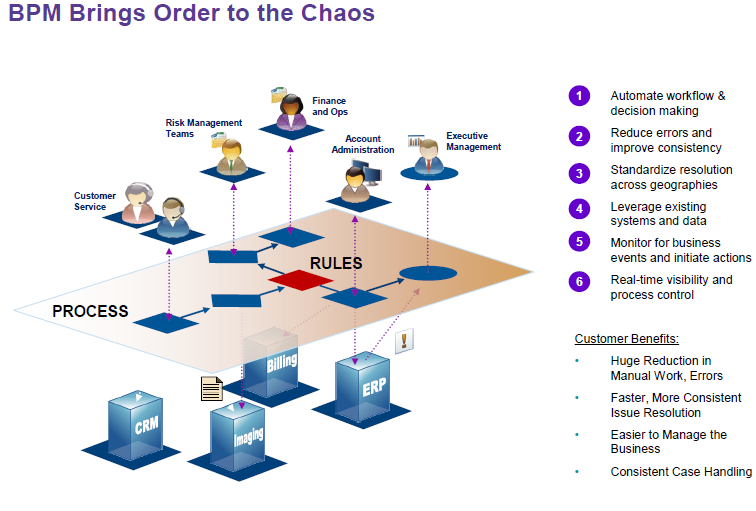
– Why rules?

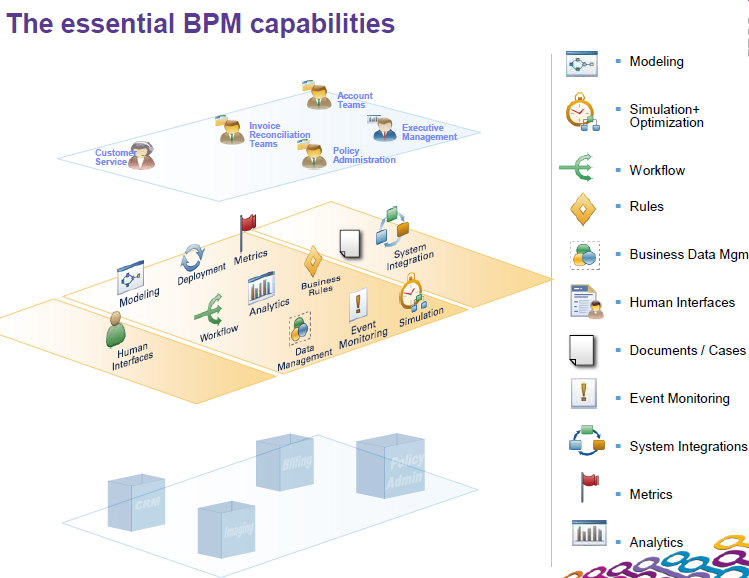
– How does a rule engine work?

<https://www.sitepoint.com/react-for-angular-developers/>

<http://blog.athico.com/>







**rule:**

Rules tell the system to look for certain patterns, and when found to perform certain actions.

" Whe n" = Look for certain patterns

" The n" = Perform certain actions

### What is React?

React is a declarative, efficient, and flexible JavaScript library for building user interfaces.

[**Drools**](http://www.jboss.org/drools)is an Object-Oriented Rule Engine for Java.

BPM:

A ***business process*** is a process that describes the order in which a series of steps need to be executed, using a flow chart.

The biggest difference between **BPMN**and traditional flowcharting is the support for events.  An event is a signal that something happened, and BPMN lets you say how the process should respond.

interactions between **process**and **rules**.

Business processes and rules are two core concepts which should be stressed out:

We can invoke **business rules** from within BPM processes.

**Business processes:** Represent what the business does.  
**Business rules:**Represent decisions that the business does.

The various parts that compose a rule may be extended to lend domain-specific sense to the rule.

Wt is a rule?

*A principle or regulation governing conduct, action, procedure, arrangement, etc.*

Why rules ?

● Separate logic from application

● Understandability

– Declarative, higher-level

● Speed and scalability

– ReteOO

● Global enforcement, maintainability and agility

– Logic is centralized, embrace change

● Traceability

**What is a Rule Engine ?**

The term Rule Engine can be referred to any system that uses rules, in any form, that can be applied to data to produce outcomes; it may refer to simple systems like form validation or more complex systems like dynamic expression engines. In a few words a Rule engine allow you to say "What to do" and not "How to do it".

What is a rule engine ?

This question is often heard at interviews, usually when comparing business rules with business processes. So let's make a clarification:

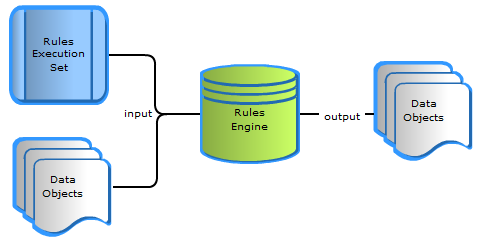
**Business processes**: Represent what the business does  
**Business rules**: Represent decisions that the business does

A rule engine may be viewed as a sophisticated if/then statement interpreter. The if/then statements that are interpreted are called **rules**.

The **if** portions of rules contain conditions such as account.getMoney() < 0. The **then**portions of rules contain actions such as sendWarning(account).

if (account.getMoney() < 0)  
sendWarning(account);

The inputs to a rule engine are a **rule execution set** and some **data objects**. The outputs from a rule engine are determined by the inputs and may include the original input data objects with possible modifications, new data objects, and side effects such as sendMail('Thank you for shopping').



--

Rules are stored in a forward-chaining rule engine, i.e. the engine implements an execution cycle that allows the action of one rule to cause the condition of other rules to become met. In this way, **a cascade of rules may become activated and each rule action executed**. Forward-chaining rule engines are suitable for problems that require drawing higher-level conclusions from simple input facts.

So, here's how you could rewrite the above check as a rule:

global AccountManager manager;  
  
rule "checkMoney"  
when  
 $account : Account( money < 0 )  
then  
   manager.warn($account);  
end

As you can see, a Rule file is based on two basic concepts:  
  
**Rules:** declarative statements that govern the conduct of business processes. A rule consists of a condition and actions. The condition is evaluated, and if it evaluates to true, the rule engine initiates one or more actions.

**Facts**: are the data upon which rules operate. In our example, the available money are facts.

--

If we start with a simple rule with a single pattern like:

rule "Match all the persons with name Salaboy"

when

$person: Person(name == "Salaboy")

then

…

end

2)Multiple Patterns

rule "Match Person and Address"

when

$person: Person(name == "Salaboy")

$address: Address(addressLine1 == "nowhere")

then

System.out.println("Hey I just find " + $person + " and an Address: " + $address );

end

**Constraints inside Join Nodes**

rule "Match Person and his/her Address"

when

$address: Address(addressLine1 == "nowhere")

$person: Person(name == "Salaboy", address == $address)

then

System.out.println("Hey I just find " + $person + "that lives in: " + $address );

end

**What is a Rule Engine ?**

The term Rule Engine can be referred to any system that uses rules, in any form, that can be applied to data to produce outcomes; it may refer to simple systems like form validation or more complex systems like dynamic expression engines. In a few words a Rule engine allow you to say "What to do" and not "How to do it".

## Why using a Rule Engine ?

Rule systems are capable of solving very, very hard problems, providing an explanation of how the solution was arrived at and why each "decision" along the way was made.  
Among the main benefit of a Rule engine can be mentioned:

* **Logic and Data Separation:**Breaking your domain objects from business rules can lead to a mach easier to maintain application as it can shield from changes in the future, since the logic is all laid out in rules.
* **Speed and Scalability:**Many times we apply "if" conditions that we didn't really need. The Rete algorithm, as implemented by Drools, replaces all of the if ... then statements with an optimized network.
* **Centralization of Knowledge:**By using rules, you create a repository of knowledge (a knowledgebase) which can be considered as a single point of truth, for business policy (for instance) - ideally rules are so readable that they can also serve as documentation
* **Tool Integration:**Writing rules means also to get accustomed with Rule language, validation and debugging. Eclipse provide an excellent tool to deliver and test your Rules.
* **Understandable Rules: JBoss Drools** rules can make it easy to express solutions to difficult problems and consequently have those solutions verified (rules are much easier to read then code).Often, business users are more comfortable with expressing things that they know to be true, than to express things in an if...then format. Examples of things that you might hear from a business expert are:

"We need to buy that estate if the price is not over 1000000 $ and estate agency doesn't claim more then 5000 $"  
"We buy shares when the price goes over 15 Euro before next week"

By focusing on what we know to be true, rather than the mechanics of how to express it in Java code,

q)How does a rule engine work ?

Production rule engine using forward chaining

