Pricing Engine Build Above Rule Engine

Wei Zhang
zhang.wei.ny@gmail.com
Up Up Consultant, LLC

As developers, how we implement business rules?

- No matter what programming language we use, C/C++, JAVA, Scala etc, most likely we implement business rules using “IF-ELSE” or “CASE” or “SWITCH”, etc.

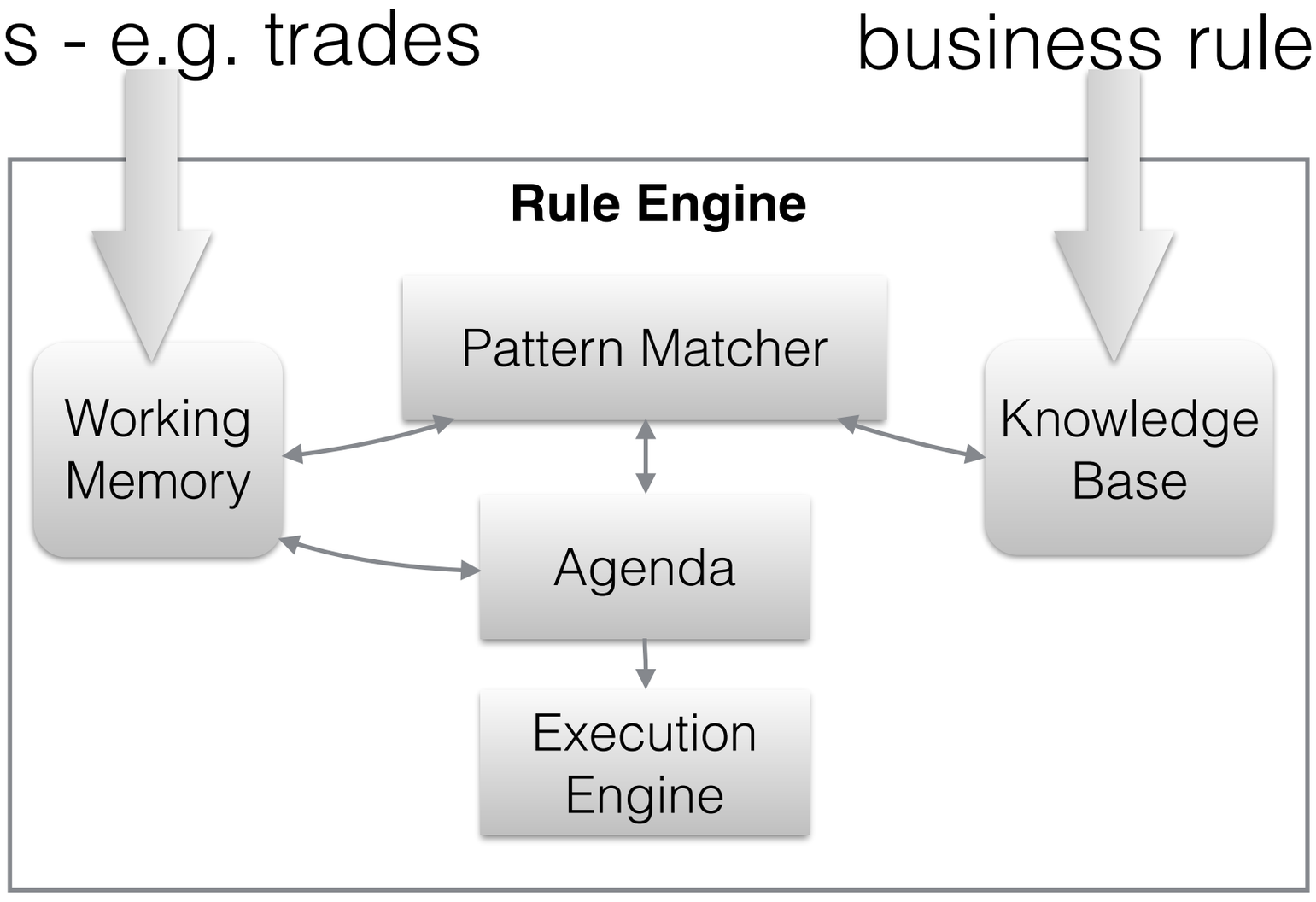
What is wrong to use“IF-Else” for business rules?

- It is good for simple, static rules, the rules are coded in applications as “if - else - then”. If you find your codes have too many “if - else - then”, and business asks you to change the logic often, then you have trouble. Any logic change, even single line of codes, is required **IT code changes, test, QA and code release**

- Yes, we can build data driven applications, most likely, rules are stored as data and coded as stored procedures inside database. But when new line of business added or changed, **IT must remodel the database and implement new logic in stored procedures, which need IT code development and release. The programming language is PL/SQL or T-SQL and the performance is depending on data volume and database load**

What is Rule Engine?

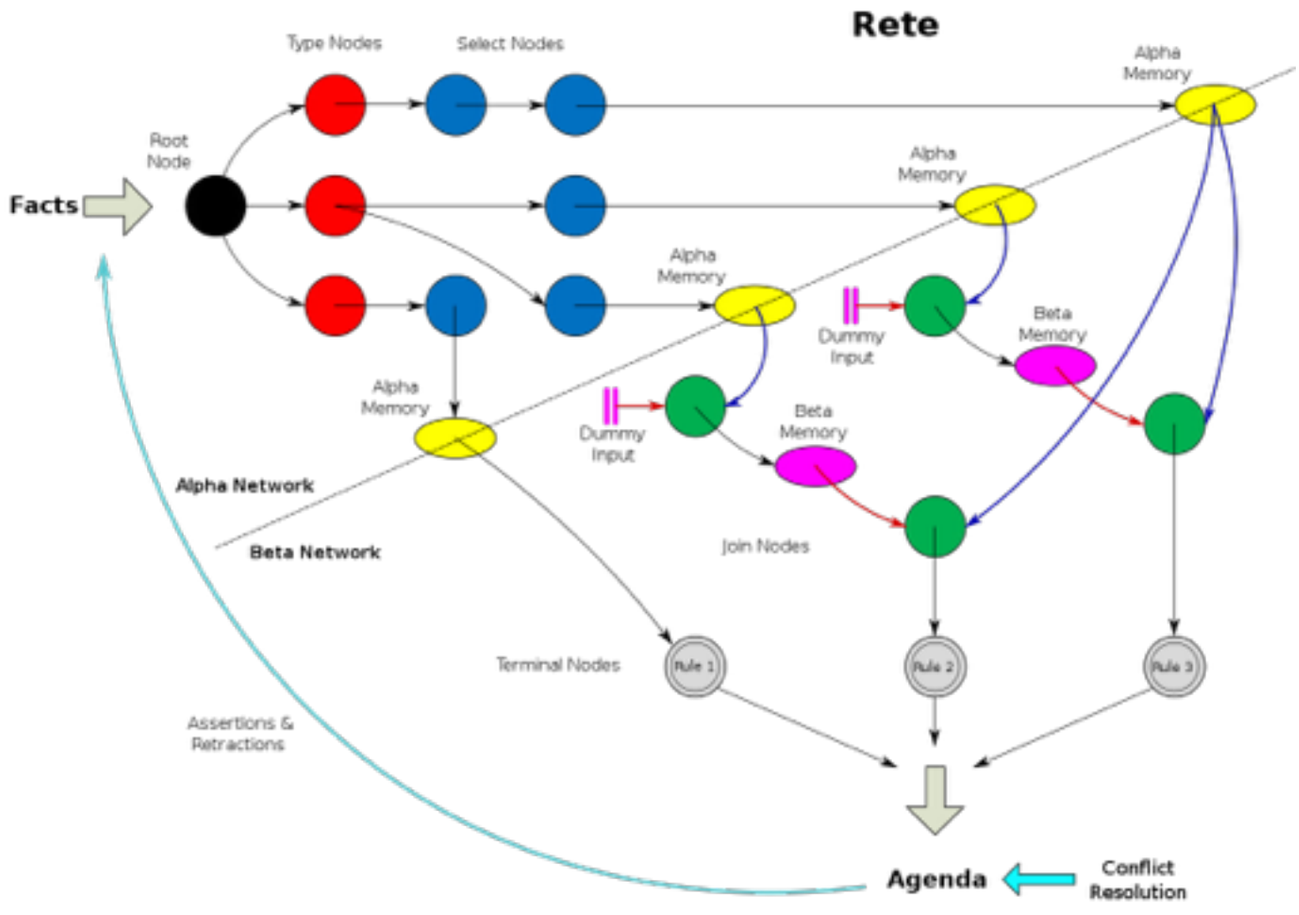
- The rule engine is an application which executes one or more actions based on fact and rules



Rete Algorithm

Designed by Dr. Charles L. Fogy

- Rules are loaded into memory as network of nodes, each note is a pattern of left-hand-side rule (condition). when a fact is inserted into working memory, the root node passes it to its child nodes and then propagates though the network until it reaches the terminal node …



Who needs pricing engine?

- Financial Company - Commission calculation
- Insurance Company - Claim adjudication and payment
- Service provider - Pharmacy Benefit Management, pharmacy claim adjudication, billing and payment
- Any company needs to dynamically price the product based on complex business rules

Benefits Of Using Rule Engine

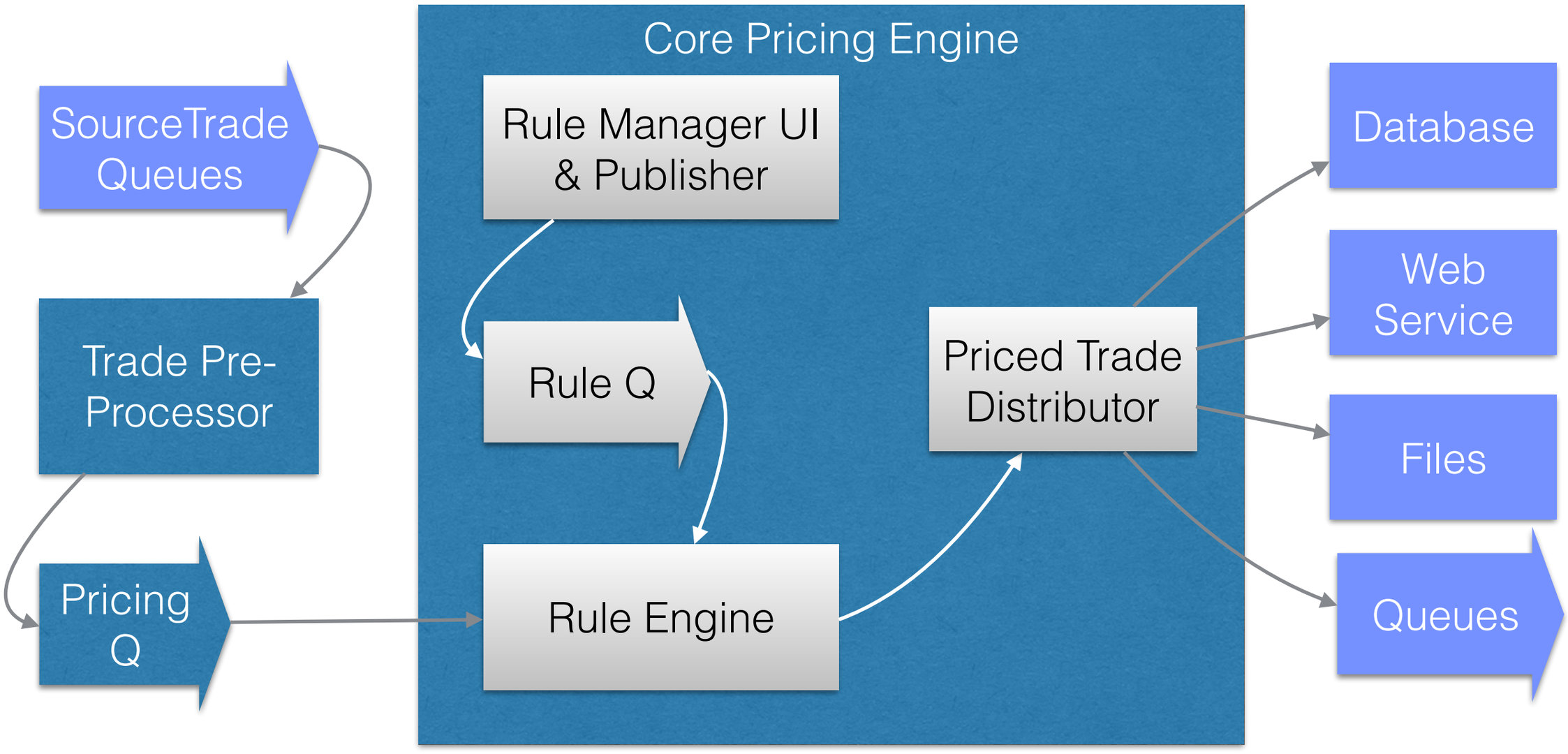
- Separate business logic from application codes
- Dynamically change rules without compiling
- Significantly Performance gain for a system with large set of complex rules (10K against 100 rules, 500 transaction/s, 30 - 50 ms difference)
- DSL for business users to read and write rules

Example 1 - Pricing Engine for trading commission

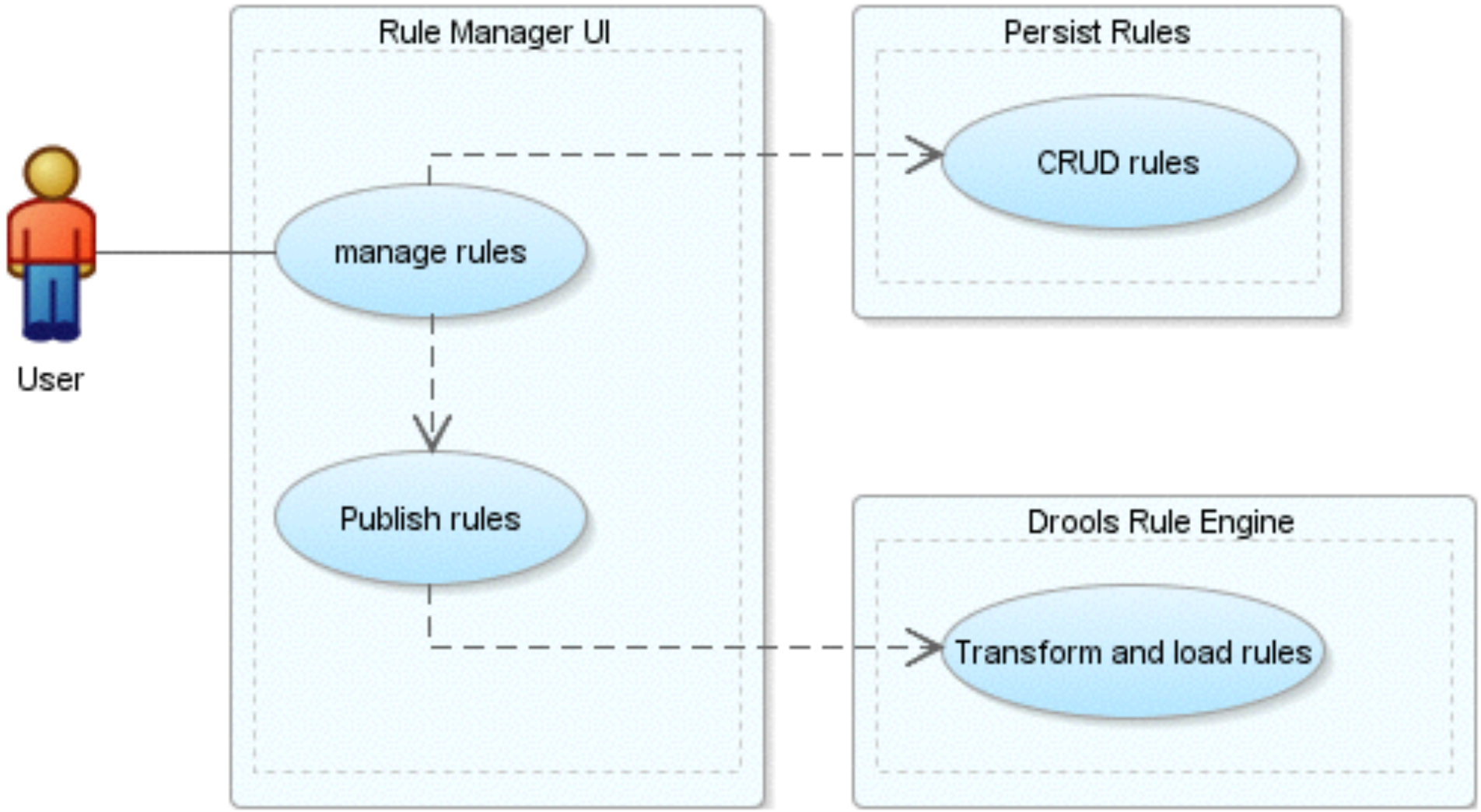
Business Requirements

- Realtime
- Multiple layers (regular pricing, risk pricing etc).
- Any combination of trade attributes can be part of business rule(s)
- Rules are managed by business users
- Support real time P & L calculation and real time client balance calculation

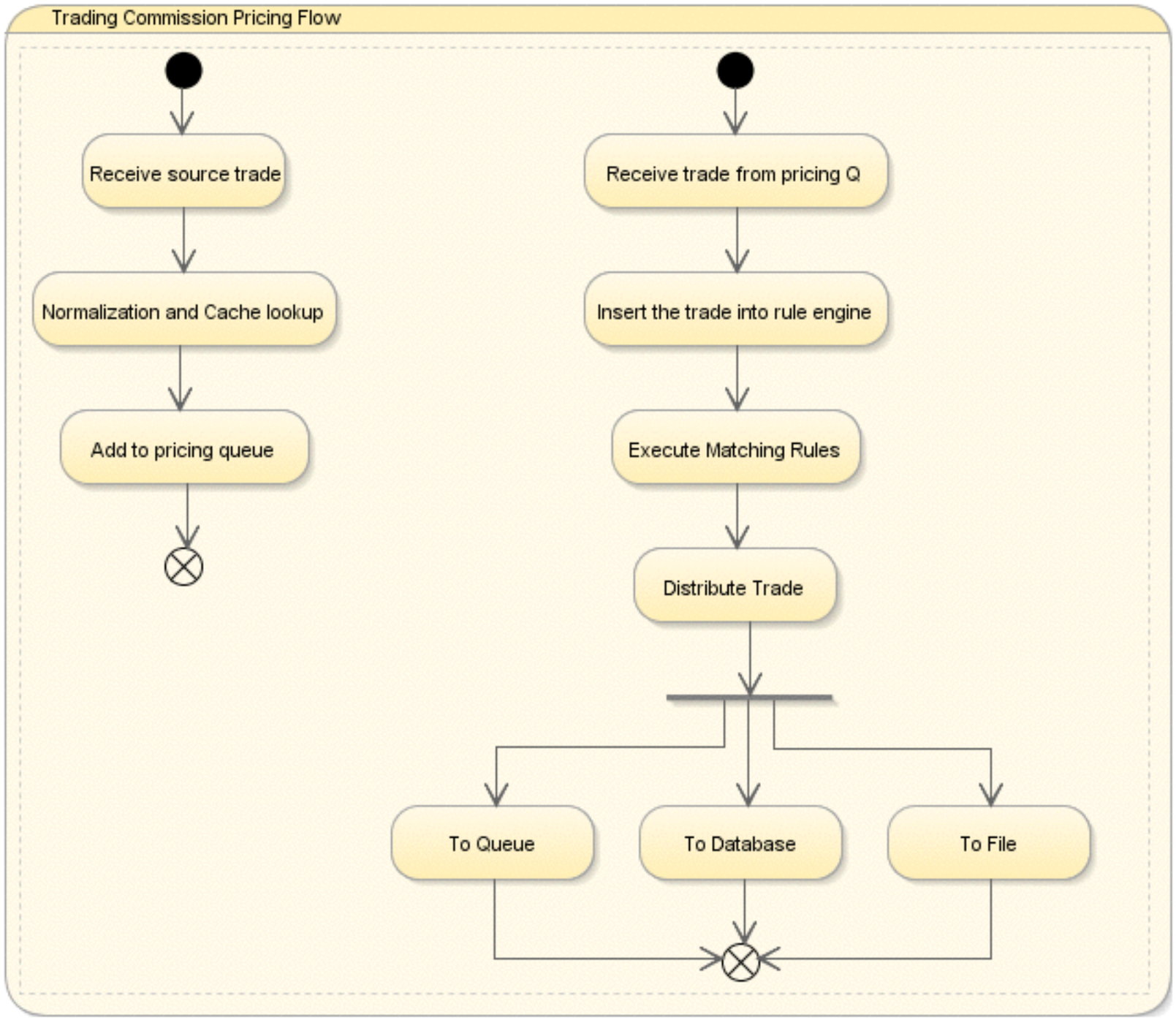
Trading Commission Pricing Engine Conceptual Architecture



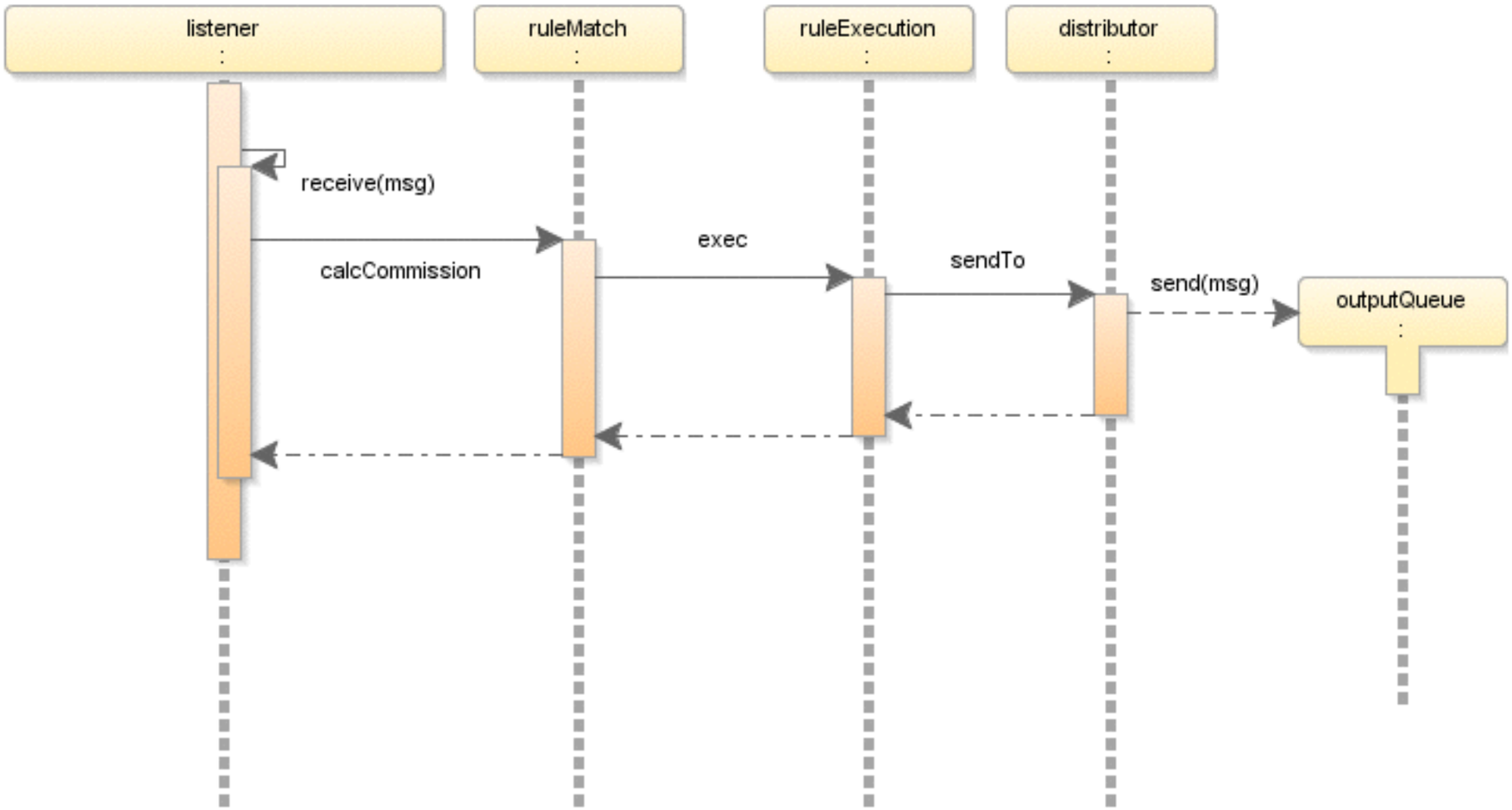
Rule Manager Use Case



Commission Calculation Flow Diagram



Commission Calculation Sequence Diagram



Before and After using Rule Engine

- Overnight batch processes
- Tiered pricing can't be handled automatically, like risk commission calculation, must be processes manually by business team.
- New type business contract needs to take weeks to be implemented by IT team.

- **Fully automated realtime pricing**
- **Tiered pricing with risk models are supported**
- **New business rules are created and managed effortlessly**

Data Model and Formats

- Source trade messages are serialized binary JSON
- Pricing rule data are saved in database modeled as relational tables. Rule data are published to rule engine in xml format then transformed into DRL.
- Trade messages are serialized java objects

Rule Examples

```
package com.upupconsultant.pricing.rule.client.Client123
import com.upupconsultant.pricing.model.Trade
import com.upupconsultant.pricing.model.BasicPricigInstruction
global com.upupconsultant.pricing.service.PricingService services
dialect "mvel"
declare Trade
    @role ( event )
    @expires( 2s )
end

rule “Regular Client123 99123478"
    salience 1000
    activation-group “REG GRP"
    agenda-group “REG PRICING"
when
    $t: Trade(
    clientId==“123”,
    country==“US”,
    currency==“USD”,
    ticker=“IBM”,
    ) from entry-point "trade stream"
then
    #BASE
    BasicPricingInstruction inst = new BasicPricingInstruction("PERCENT",0.4)
    services.reqularPricng($t,inst,kcontext.getRule().getName())
end
```

Rule Examples - continue

```
rule “Risk Client123 99123479"
    salience 1000
    activation-group “RISK GRP"
    agenda-group “RISK PRICING"
when
    $t: Trade(
    clientId==“123”,
    riskPercentage>=0.5

    ) from entry-point "trade stream"
then
    #RISK PRICING
    BasicPricingInstruction inst = new BasicPricingInstruction(“RISK”,”MODEL”,”1")
    services.riskPricng($t,inst,kcontext.getRule().getName())
end
```

Example 2 - Pricing Engine for medical claim payment

Business Requirements

- Multiple layers pricing (co-pay / co-insurance, in-network / out-network provider, maximum out of pocket expense etc).

- Eligibility rules are complex and managed by enrollment and service team.

- Pricing models are created or modified and needs to be tested by re-adjudicating the historical claims.

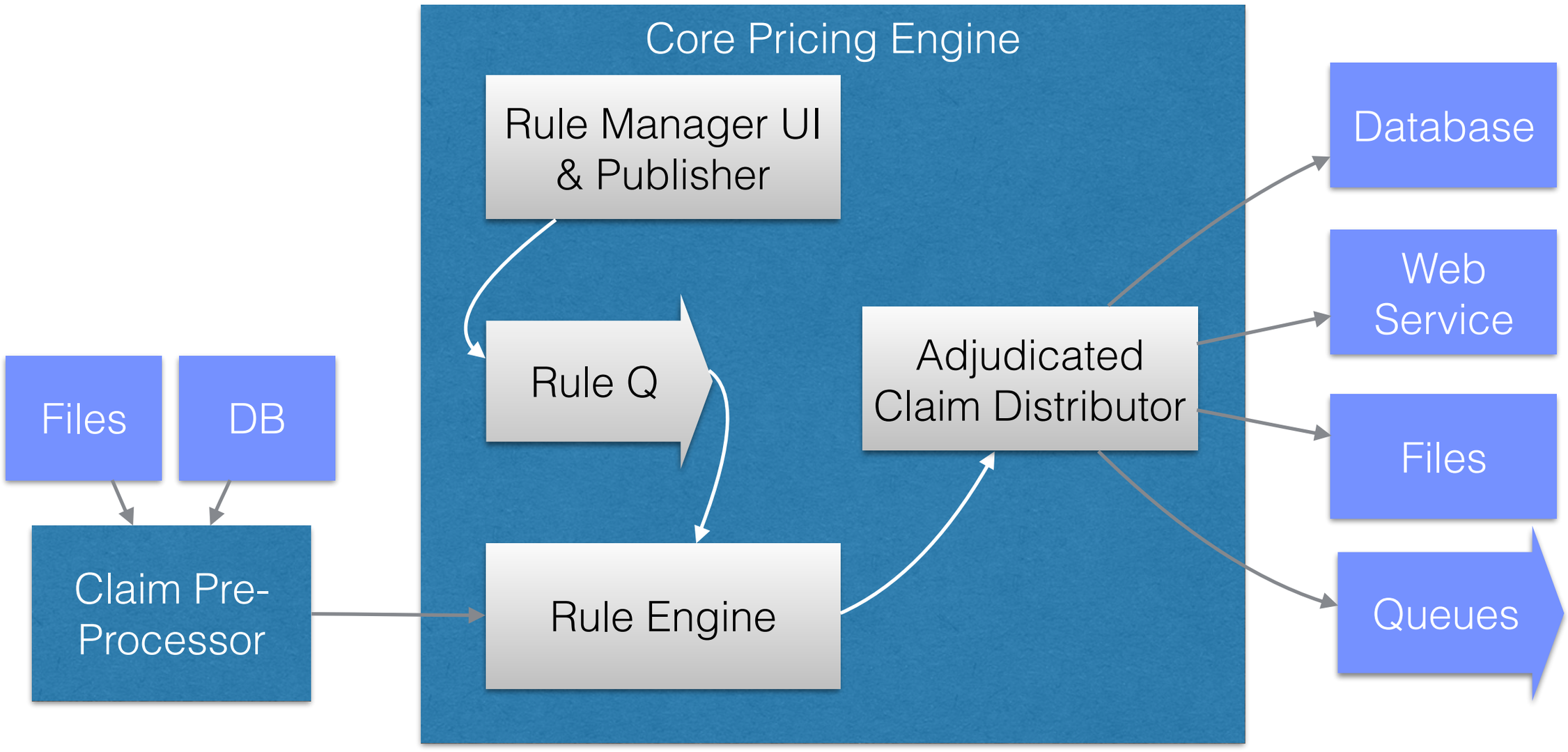
- Priced claims should be integrated with payment system with multiple protocols - SOA or MQ

Commercial Pricing Engine solution is very expensive …

- Tens of Millions dollars project …, and it is a blackbox.
- Doesn't support multi-tiered pricing out of box, customer solution is required …

- **Open source solution is much cheaper …**
- **All the functions and features needed are there or easy to be build …**
- **Performance is well approved by many companies cross industries …**

Medical Claim Pricing Engine High Level Architecture



Prototype and Demo

Q & A

Reference

- http://en.wikipedia.org/wiki/Rete_algorithm

- http://docs.jboss.org/drools/release/6.1.0.Final/drools-docs/html_single/index.html