



**Ontology-driven
Rule-based
Service-Oriented
Business Abstraction Tier**

Bahadır Baran Odevci (bahadir.odevci@finansbank.nl)

Enterprise Architect

Finansbank (Holland) N.V.

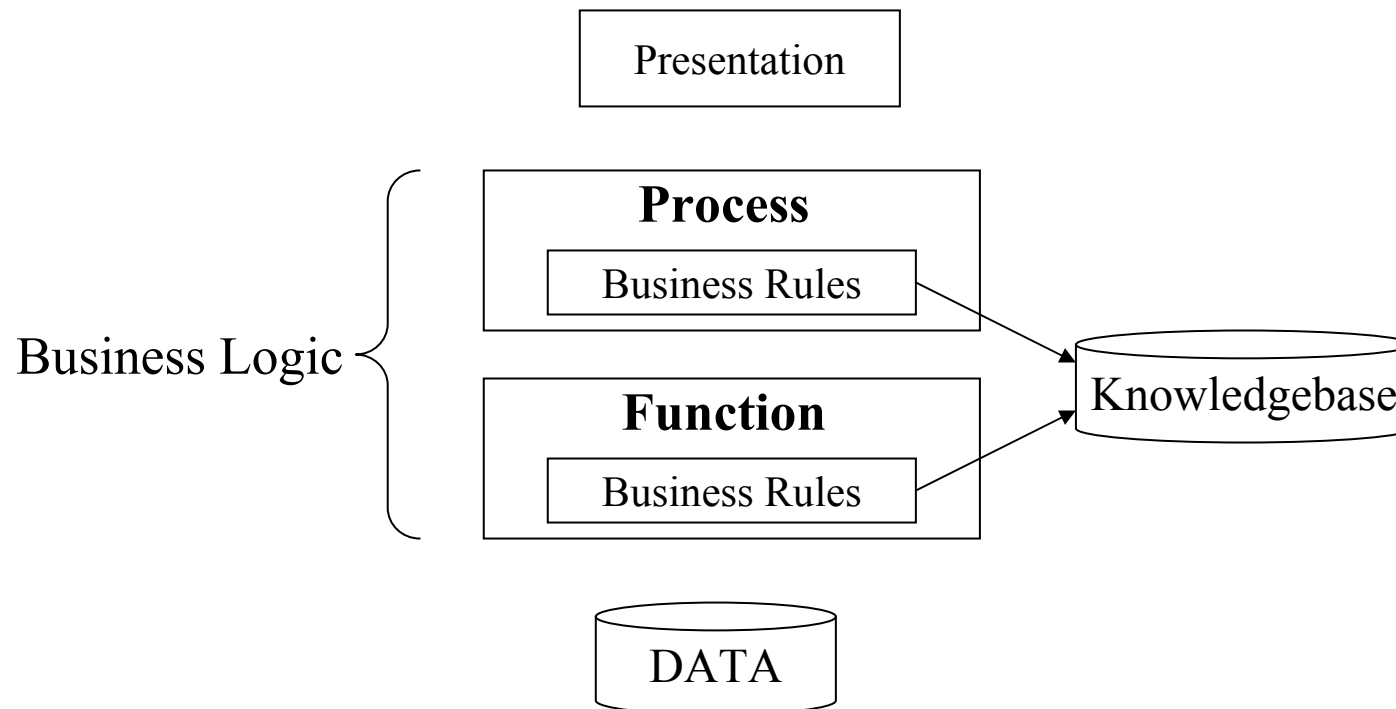
Separate the *know* from the *flow*.

—Roger T. Burlton

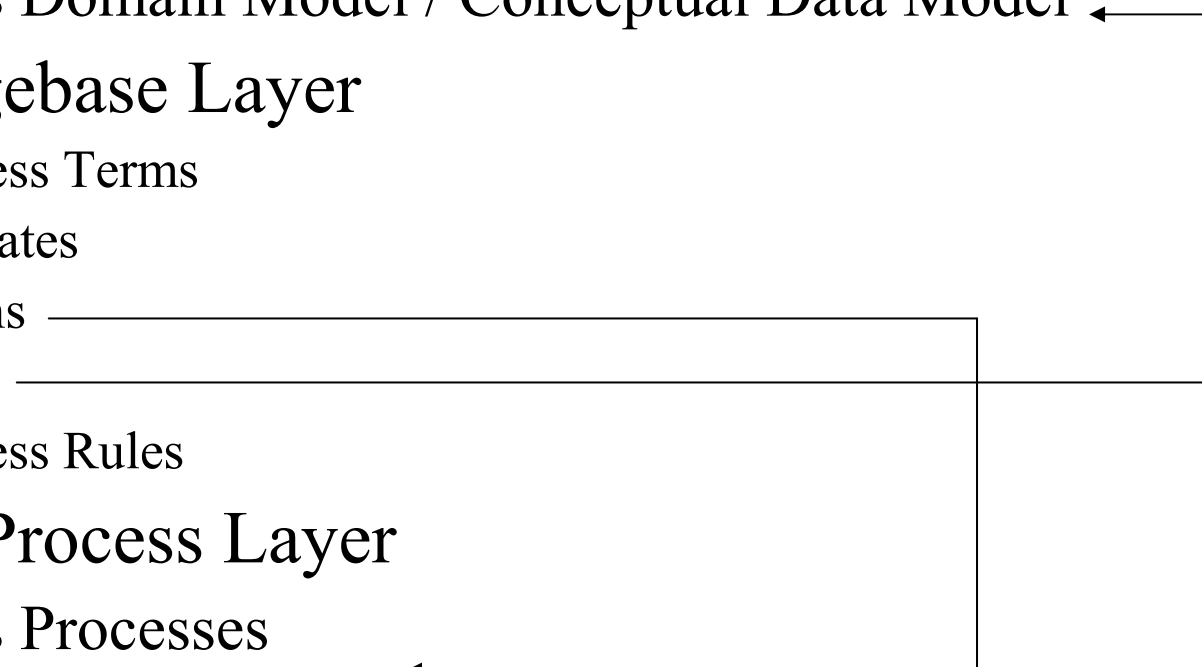
- How to form a knowledgebase starting from an ontology model, and how to perform reasoning on the knowledgebase in an enterprise-scale Service-Oriented Architecture

Primary Goal

- Extend the SOA model by;
“Embracing business rules by both programming in the large and programming in the small”



- **Finansbank Turkey**
 - In-house Core Banking Application, launched in Q2, 2002 (>1,500 man-months, 40,000 TPM)
 - 2000 business transactions
 - 5000 transactional screens
 - 300 branches
 - Proprietary software infrastructure solutions
 - Application Server
 - Enterprise Service Bus
 - O2R Mapping software
 - Front-end technology
 - » Java-based thin browser (Core Banking)
 - » Templating language (Internet Banking)
 - RAD Suite
- **International Business Technologies (IBTech)**

- **Ontology Layer**
 - Business Domain Model / Conceptual Data Model
 - **Knowledgebase Layer**
 - Business Terms
 - Predicates
 - Actions
 - Facts
 - Business Rules
 - **Business Process Layer**
 - Business Processes
 - Business Sub-Processes
 - Business Functions
- 

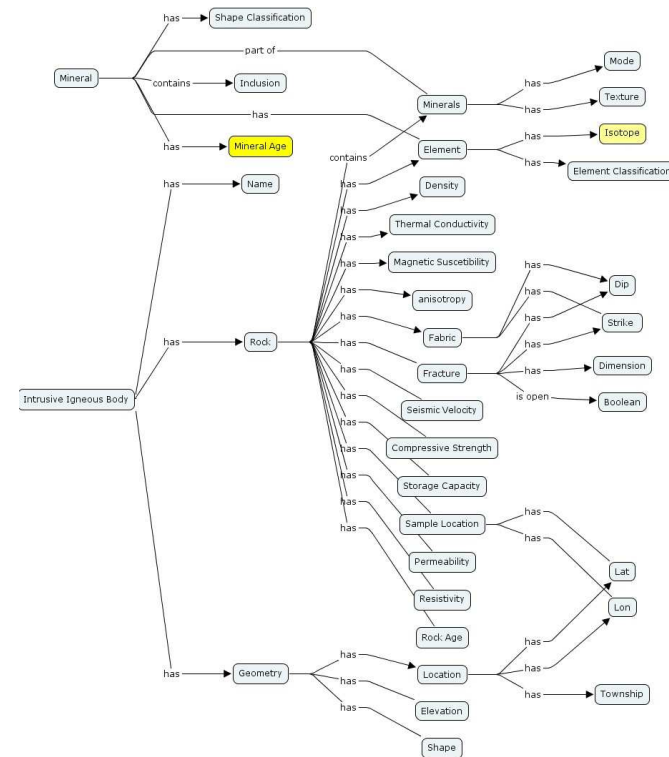
"representation of a shared conceptualisation
of a particular domain"

Purpose

To provide a shared and common understanding of a domain that can be communicated across people and application systems

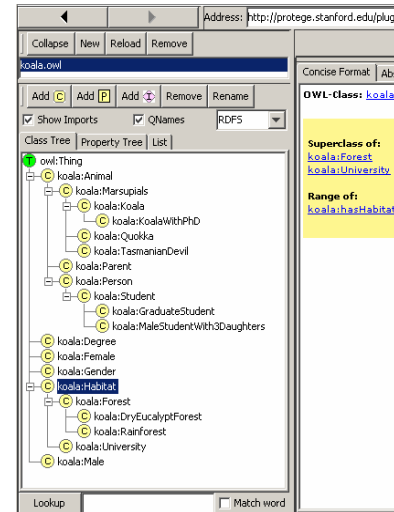
- ▼ person
 - ▶ patient
 - ▶ relative
 - ▶ physician
 - ▼ inanimate physical object
 - ▶ medical instrument
 - ▼ event
 - ▶ pt. encounter
 - ▼ location
 - ▶ location in body
 - ▼ area
 - ▶ area in body
 - ▼ process
 - ▼ intended process
 - ▼ iatrogenic process
 - ▼ iatrogenic action on object
 - ▶ remove
 - ▶ install
 - ▶ destroy
 - ▶ open
 - ▶ close
 - ▶ create
 - ▶ measure
- ▼ state
 - ▶ iatrogenic state
 - ▶ disposition
- ▶ time point
- ▶ time interval

- Domain level constraint definitions per application
 - Referential integrity, uniqueness and other constraints
- Editing should support
 - Difference and Merging
 - Versioning



Ontology Editor

Ontology meta-data



<domain>

```
<class isa="Thing" native="a customer" type="Customer">  
  <attribute name="age" native="a customer's age" relation="hasAge"/>  
  <attribute name="customerNo" relation="identifiedBy"/>  
  <attribute name="customerType" relation="isTypeOf"/>  
  <attribute name="loans" type="Loan" relation="hasLoans" cardinality="0-*"/>  
  <attribute name="maritalStatus"/>  
  <attribute name="ConsumerCredit" cardinality="0-*" type="ConsumerCredit"/>  
  <attribute name="accounts" type="Account" relation="hasAccounts" cardinality="0-*"/>  
</class>
```

</class>

```
<class isa="Thing" type="Credit"/>
```

```
<class isa="Credit" type="ConsumerCredit">
```

```
  <attribute default="0" name="creditScore" native="customer's credit score" type="Number"/>
```

```
  <attribute name="creditDecision" native="decision on customer's credit application"/>
```

```
</class>
```

Business Terms

- Forms the base set of parameter types in Business Function signatures
- Terms match to fields existing on the User Interface
- Three type of business terms
 - Variable Terms
 - Compound Terms (Business Functions)
 - Constant Terms

Variable Terms

- Entities and Attributes in business domain model

```
<class isa="Thing" native="a customer" type="Customer">
  <attribute name="age" native="a customer's age"/>
  <attribute name="householdSize" native="a customer's household size"/>
  <attribute name="occupationType"/>
  <attribute name="customerType"/>
  <attribute name="accounts" type="Account" relation="0-*"/>
  <attribute name="loans" type="Loan" relation="0-*"/>
  <attribute name="maritalStatus"/>
  <attribute name="ConsumerCredit" relation="0-*" type="ConsumerCredit"/>
</class>
<class isa="Thing" type="Credit"/>
<class isa="Credit" type="ConsumerCredit">
  <attribute default="0" name="creditScore" native="customer's credit score"
  type="Number"/>
  ...
```

Compound Terms

```
<!-- compound terms -->
<compoundTerms>
  <compoundTerm id="1" name="CustomerTotalTransactions" native="number of all transactions of
[entity1]">
    <function ID="totalTransaction"/>
  </compoundTerm>
  <compoundTerm id="4" name="nextWorkingDay" native="next working day" >
    <function ID="nextWorkingDay"/>
  </compoundTerm>
```

Business Functions (Utility methods, Web Services, Bfel...)

```
<function description="returns next working day" id="1" name="nextWorkingDay" type="Date">
  <method class="com.cb.smg.SystemDate" name="getNextWorkingDay()"/>
</function>
<function description="returns total number of transactions of a customer" name="totalTransaction">
  <bfel name="totalTransactions.bfel"/>
  <wsdl url=""/>
</function>
```

Constant Terms

Parameter tables (2-dim)

```
<referenceData description="organisation branch table"  
  id="PRM_ORGANISATION_BRANCH">  
  <unique name="BRANCHCODE" native="branch's code" type="Text"/>  
  <column name="BRANCHNAME" native="branch's name" type="Text"/>  
  <column name="CITYCODE" native="branch's city code" type="Text"/>  
  <column name="DISTRICTCODE" native="district code" type="Text"/>
```

One-dim parameters

```
<constants>  
<constant description=" credit score of the application is constant"  
  name="CREDIT_SCORE_HI" native="credit score is HIGH" type="Number" value="983"/>  
<constant name="DEFAULT_CURRENCY" native="system default currency"  
  type="Number" value="EUR"/>
```

Predicates

<predicates>

**<predicate name="add" native="add [entity1] to [entity2]"
notation="+=" priority="1"/>**

<predicate description="" id="2" name="is" native="[entity1] is [entity2]"
notation="="/>

**<predicate name="in-between" native="[entity1] is in-between(incl.)
[entity2]" notation="[]" priority="1"/>**

<predicate description="" id="4" name="matchRange" native="[entity1]
matches the range in [entity2]" notation="[]"/>

<predicate name="NEQ" native="[entity1] is not equal to [entity2]"
notation="!=">

...

Business Rules by example



source code

```
If (0 < customer.getAge() < 17){  
    int currentScore = customer.getCreditScore();  
    //Fetch the points mapped to the given age range  
    ReferenceData ageScore = new ReferenceData("PRM_AGE_SCORE");  
    ageScore.put("AGE_RANGE", "0-17");  
    int score = ((Integer) rdsektor.getValue(System.getInstance().getDate(), ageScore,  
    "SCORE")).intValue();  
    customer.setCreditScore(currentScore + score);  
}
```

Business Rules by example



declaration

```
<rule id="5">  
  <precondition>  
    <if entity="Customer.age" predicate="in-between" operand="0-17"/>  
    <then>  
      <action entity="-500" predicate="add" operand="Customer.ConsumerCredit.creditScore"/>  
    </then>  
  </precondition>  
</rule>
```

Business Rules by example



native language representation

if

customer's age *is in-between(incl.)* 0-17

then

add -500 *to* customer's credit score

How flexible ?

```
<!--
```

```
If
```

```
  [score age points of a customer's age] is less than minimum age score
```

```
then
```

```
  credit decision is DECLINED
```

```
  exit the application
```

```
-->
```

```
<rule name="scoringKnockout1" description="">
```

```
  <if description="">
```

```
    <var id="scorecard_age"/>
```

```
    //ScoreCard type
```

```
    <predicate id="getPoints"/>
```

```
    //ScoreCard type operator
```

```
    <var id="Customer.age"/>
```

```
    //Operand
```

```
    <predicate id="lt"/>
```

```
    //Lower priority operator wrt getPoints operator
```

```
    <constant id="MINIMUM_AGE_SCORE"/>
```

```
  </if>
```

```
  <then>
```

```
    <expression>
```

```
      <var id="decision"/>
```

```
      <predicate id="is"/>
```

```
      <constant id="CREDIT_DECLINED"/>
```

```
    </expression>
```

```
    <function name="exitTheCreditApplication"/>
```

```
  </then>
```

```
</rule>
```

Internationalized Rule Editing



```
<compoundTerm id="2" name="branchCityCode">
  <function ID="getBranchCityCode"/>
</compoundTerm>
<function description="return branch city code" name="getBranchCityCode" native="bundle:1">
  <method class="com.cb.smg.Utility" name="getReferenceData">
    <param id="1" map="entity1" type="Product.Organisation.branchCode"/>
    <param id="Product.Organisation.branchType" map="entity2"/>
  </method>
</function>
```

Rule:

If type of branch with code [Product.Organisation.branchCode] is
BRANCH TYPE CORPORATE

Then ...

Resource Bundle in English

1=type of branch with code [Product.Organisation.branchCode]

Resource Bundle in Turkish

1=sube kodu [Product.Organisation.branchCode] olan subenin sube tipi

KDLC

- Talk
- Create (Amend) Domain Model
 - Add new predicate
 - Define new;
 - Compound Term (Mapping to a Business Functions)
 - Static Term
 - Action
 - Utility methods
 - Business Functions
 - Business Rules
- Create Process Descriptions
 - Use Domain-level constraints
 - Use Business Functions
 - Use Business Rules

- Ontology Model forms the base of the O2R Model
 - Ontology model conversion procedure -> normalized RDBMS model
 - Structured (inheritance and association), single O2R persistency (POM, Hibernate,..) throughout the component boundary
 - Ontology Editor has two perspectives (Analyst and Software Developer)
 - Based on business requirements, Software Developer adds finder methods that haven't been generated

- Ontology Model forms the base of the Object Model in the business tier
 - Ontology model conversion procedure -> Business Object Model
 - A “Business Function” is business-wise meaningful smallest piece of “behavior” of the corresponding Object in the Business Object Model
 - Behavior of the Object has its own metadata and is graphically modeled

- Interfacing Knowledgebase via the Reasoning Engine
- Boundaries (component, module) are defined based on conceptual object model, distributed call decision is taken on-the-fly based on boundaries
- Code generation
- Well-defined method signatures
 - Parameter types mapping to domain types
 - No two parameters point to same domain type

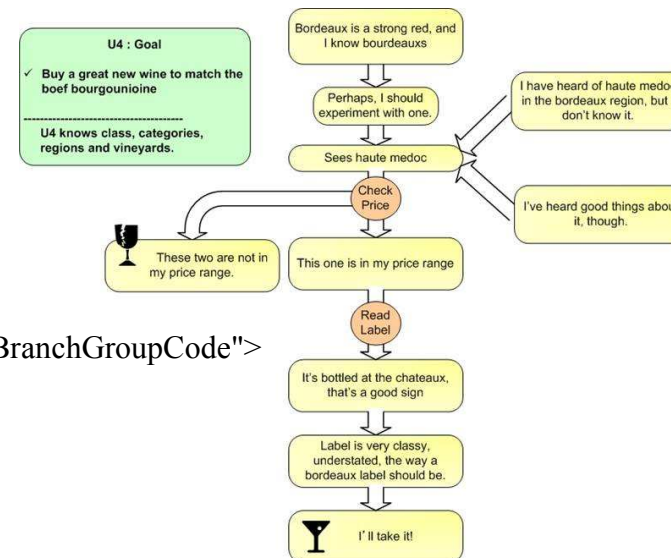
BFEL
BFEL

BFEL

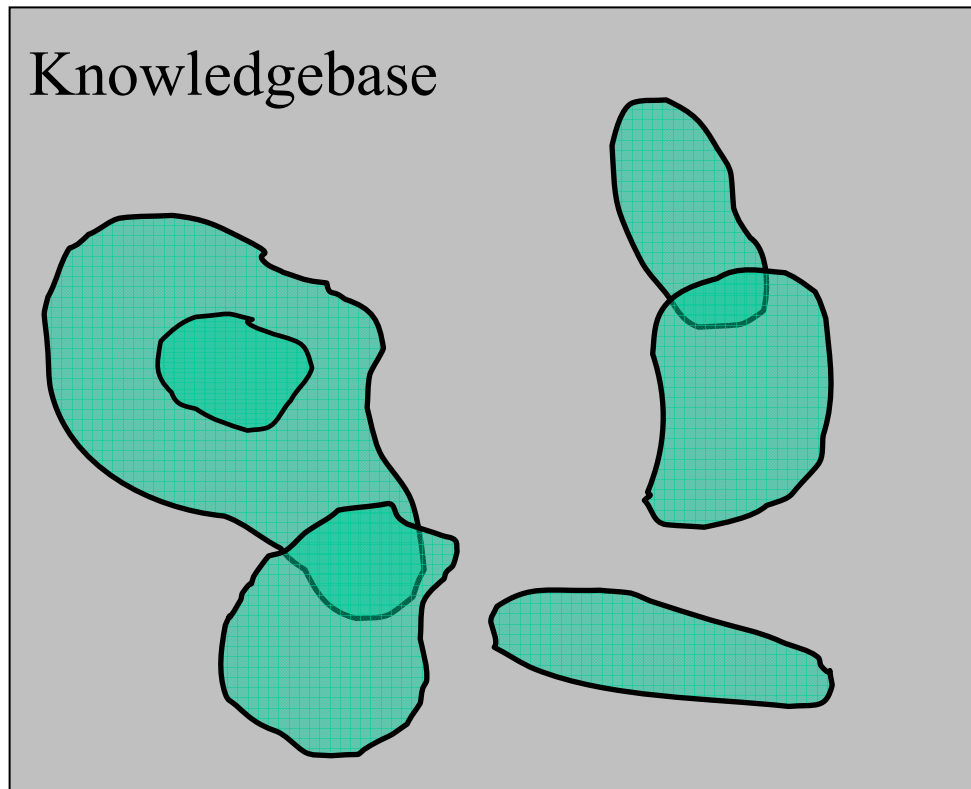
- 4GL, end-to-end, RAD platform

–Business Software Developers don't use a 3GL IDE
(Java or .NET)

```
<sequence>
  <function name="CREDIT_CUSTOMER_GROUP" type="BFEL">
    <input>
      <param name="CUSTOMEROID"/>
    </input>
  </function>
  <scope>
    <o2r finderName="FindCreditLimitBranchGroupCode" name="FindBranchGroupCode">
      <param name="BRANCHID"/>
    </o2r>
    <rule id="1" name="CONVERTCURRENCYRULE">
      <param name="CURRENCYCODE"/>
      <param name="AMOUNT"/>
      <param name="BRANCHGROUPCODE" node="FindBranchGroupCode"/>
    </rule>
  </scope>
</sequence>
```



Processes floating in the knowledgebase



Bird's eye view;

- Green areas are process boundaries
- Process boundary defines the scope of Production rules & Business Functions

Reasoning on-the-fly

- Asserting the facts via O2R
- Forming the so called “working memory” on-demand
- Reasoning on the KB within the process’ transaction boundary
- Process container, orchestrating the inputs & outputs amongst the processes

- Expressed in native language of choice
- Business Processes are SOA building blocks, available as Web Services to enterprise-wide integration

Open Deposit Account Business Process

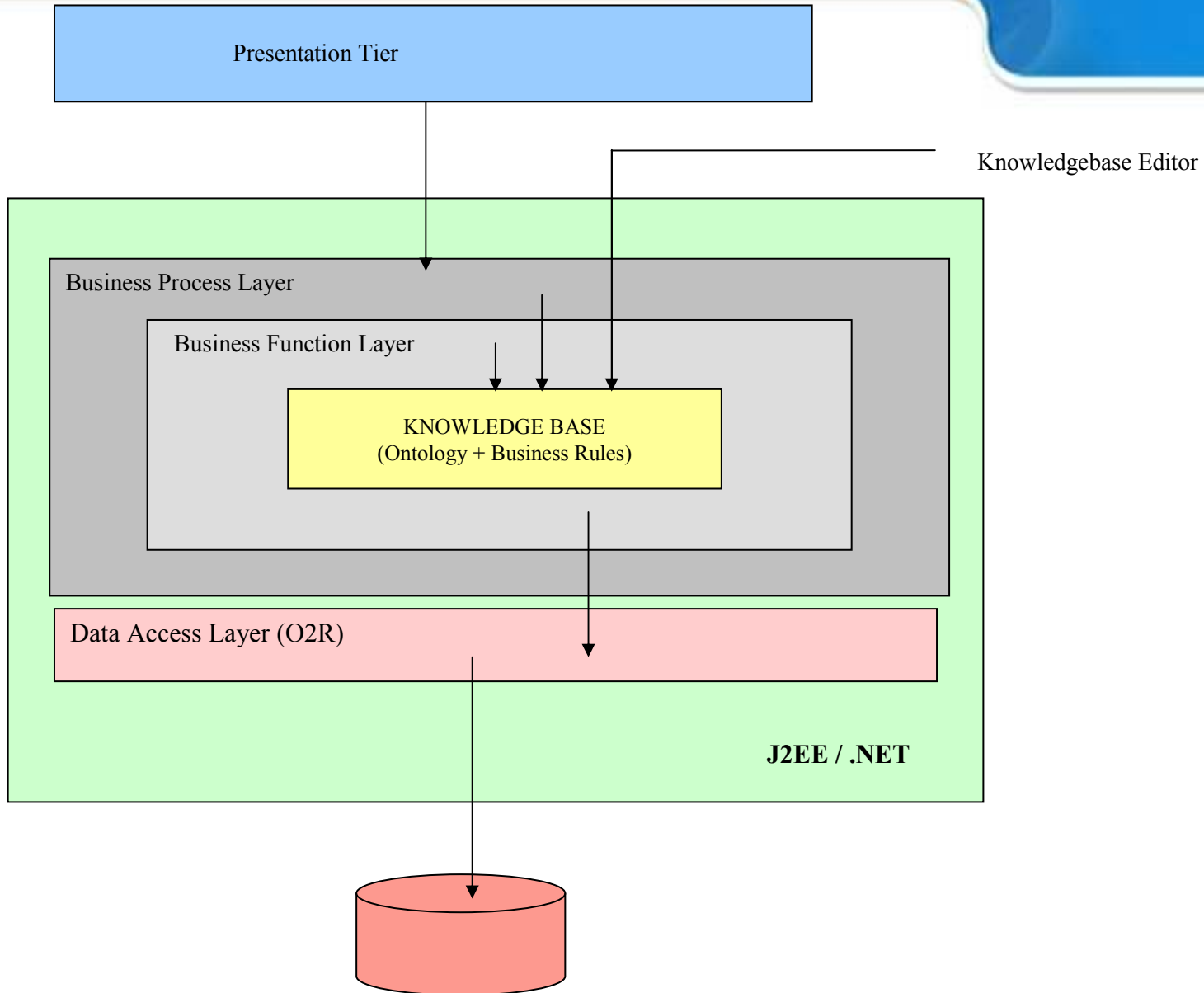


```
get customer accounting information
get deposit account's accounting GL account number
open account
if money transfer source is another account
    then
        transfer money from account
    else
        credit account
if transaction is from branch
    then print statement
if branch type is domestic branch
    then
        create foreign currency trade statement
        create statistic code for Central Bank
if transfer type is cash transfer and currency code is foreign currency
    get customer accounting information
    get product statistic code
    get rate of exchange
    if currency type is not equal to DOLLAR
        then
            calculate branch rate transaction
create product statistics movement
```

What is needed?

- Enterprise-scale Software Run-time Platform
(J2EE, .NET, proprietary)
 - highly-scalable
 - highly-available
 - highly-reliable
 - highly-secure
 - transactional
 - distributed

Business Abstraction Tier



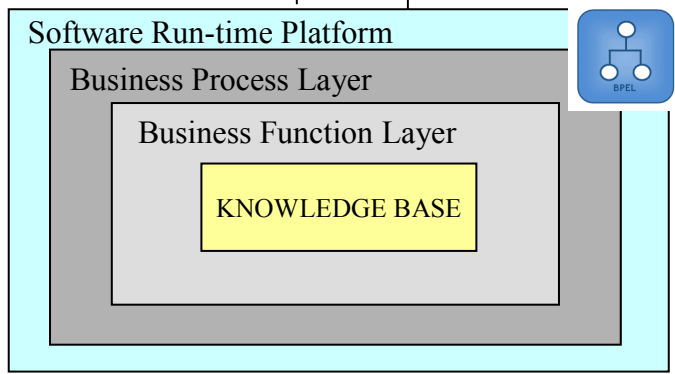
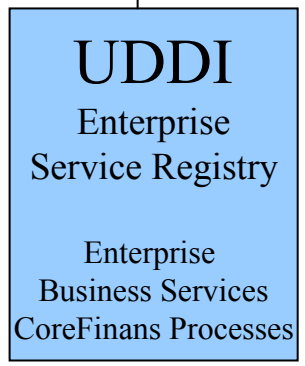
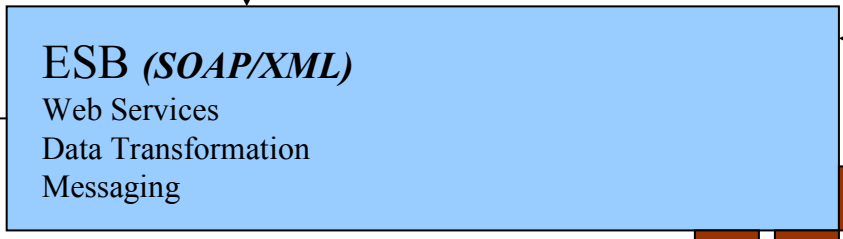
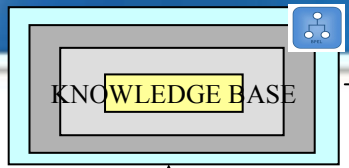
Reasoning Service Layer

Core Banking Ontology

OWL, RDF, OIL ...

Forward reasoning
rule-engine

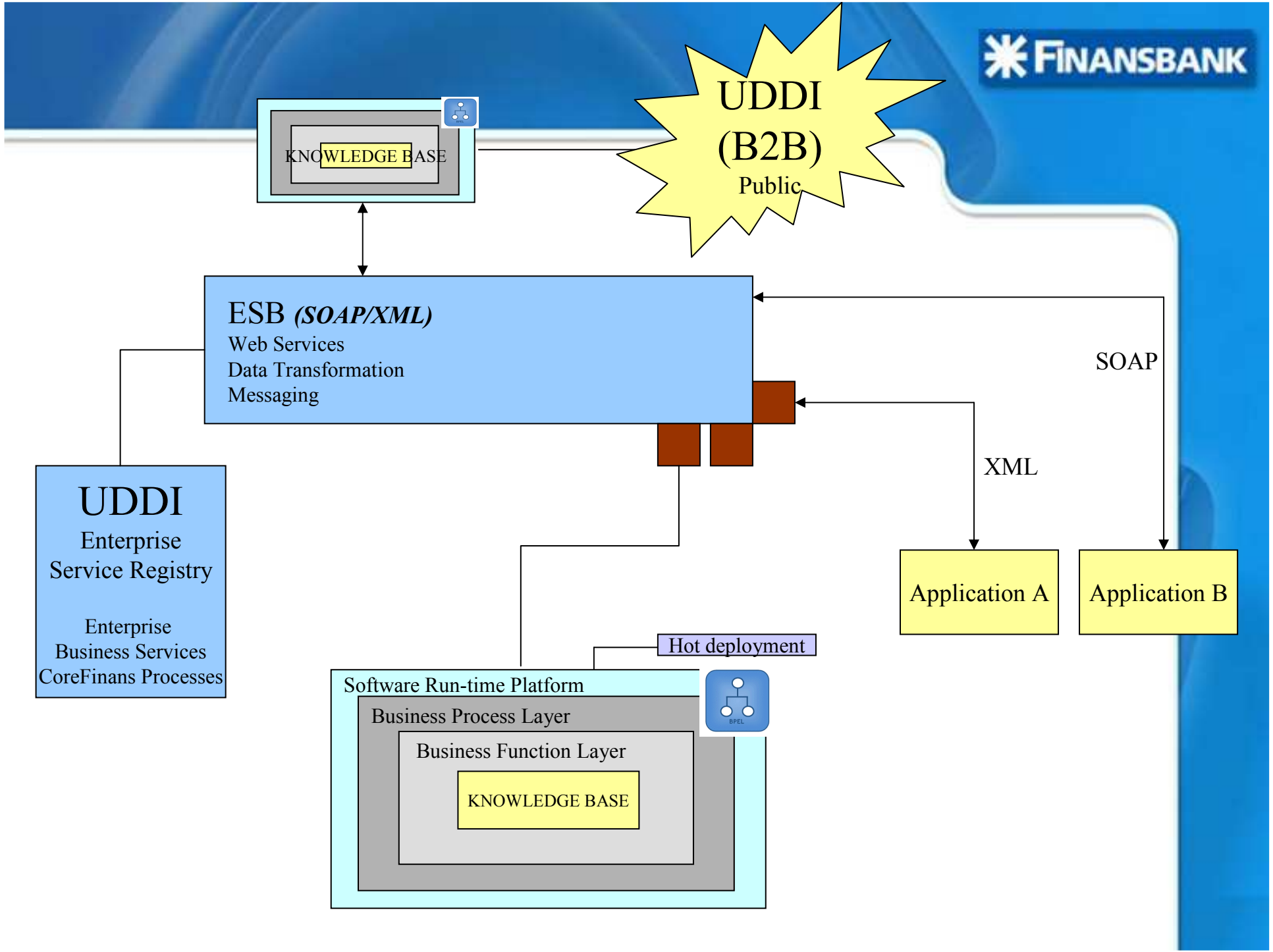
Forward Reasoning: An inference mechanism that starts with case data and executes production rules that use these data



SOAP

XML

Hot deployment



- Business analysts both determine and write the business logic
- The business side gets direct control over rules that govern how enterprise applications behave

- Much lower maintenance costs and much higher confidence that business rules are being implemented as intended
- When a BRMS is properly implemented and supported, IT can expect operating cost reductions of 10 percent to 15 percent

Gartner

- Enhancements on the “Talk” step of KDLC
- Speech to knowledgebase recognition

Q / A

