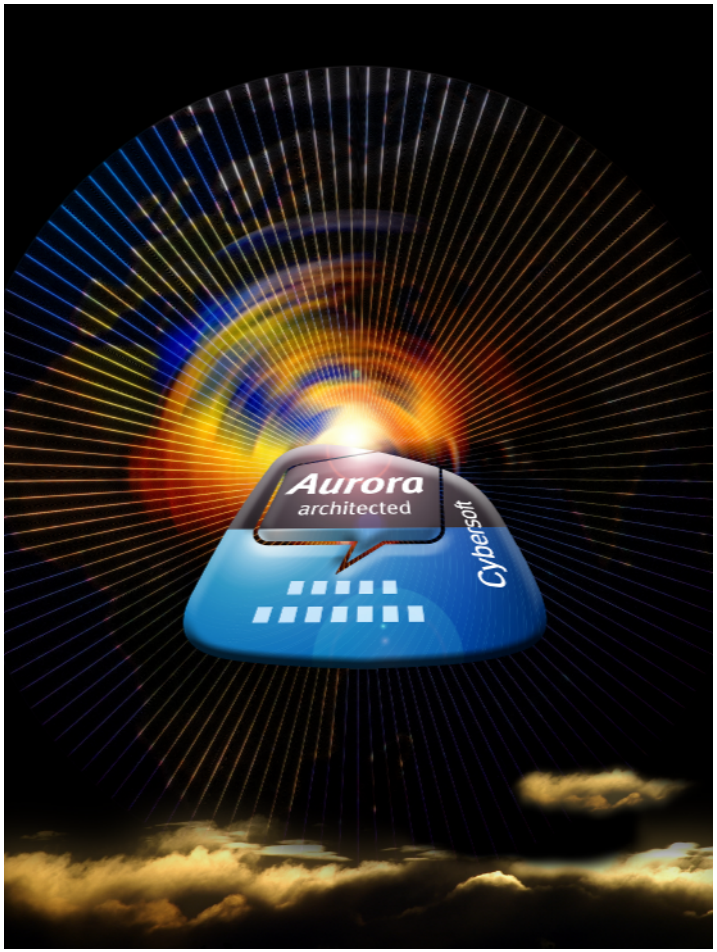


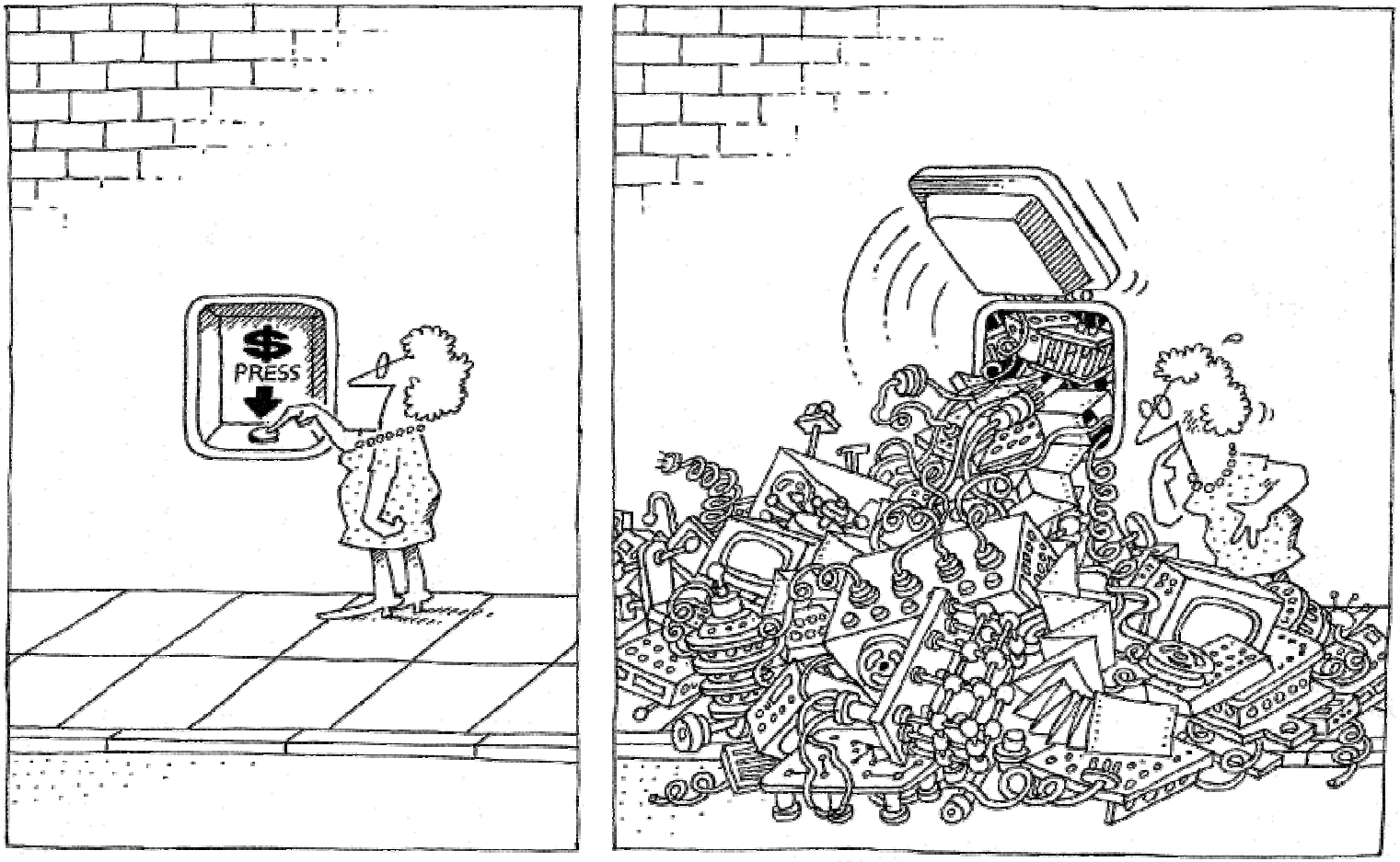
Business Rules Segregation with Aspects

Cybersoft

Dr. Semih Çetin

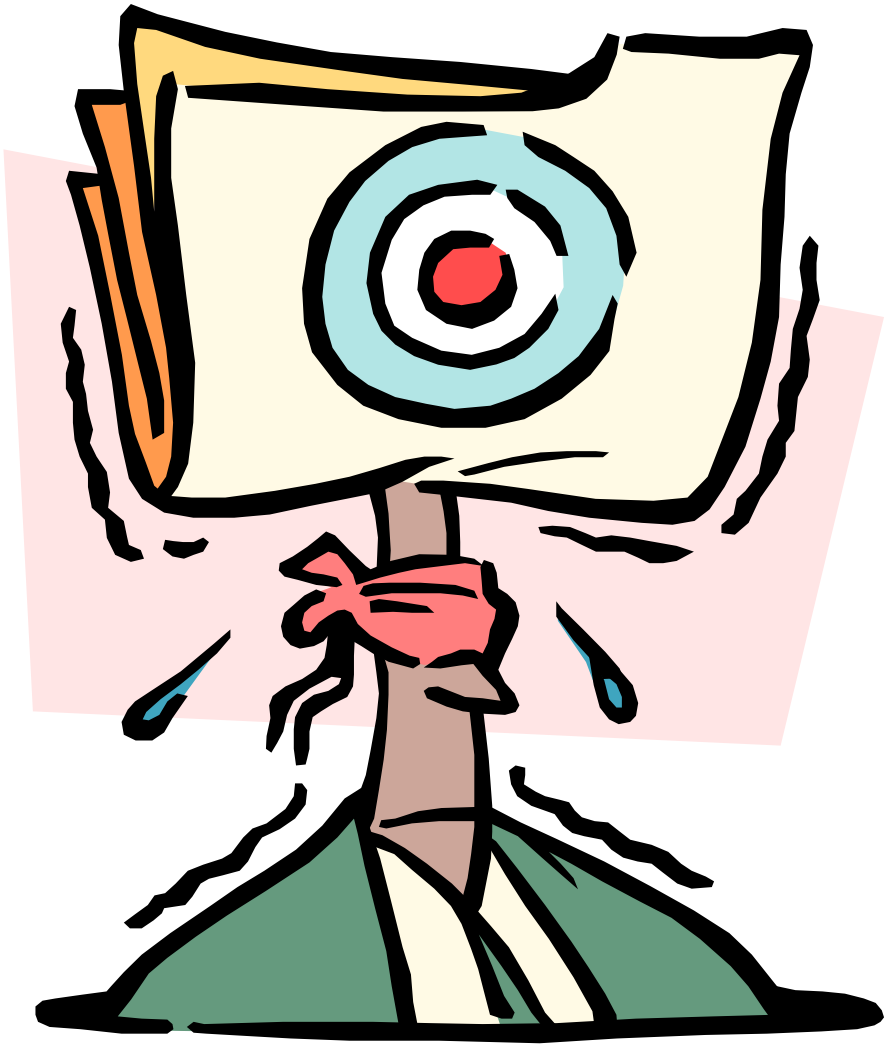
Content

- Motivation
- Separation of Concerns and Aspects
- Business Rules as a Separate Concern
- RUMBA: **RU**le-based **M**odel for **B**asic **A**spects
- Conclusions



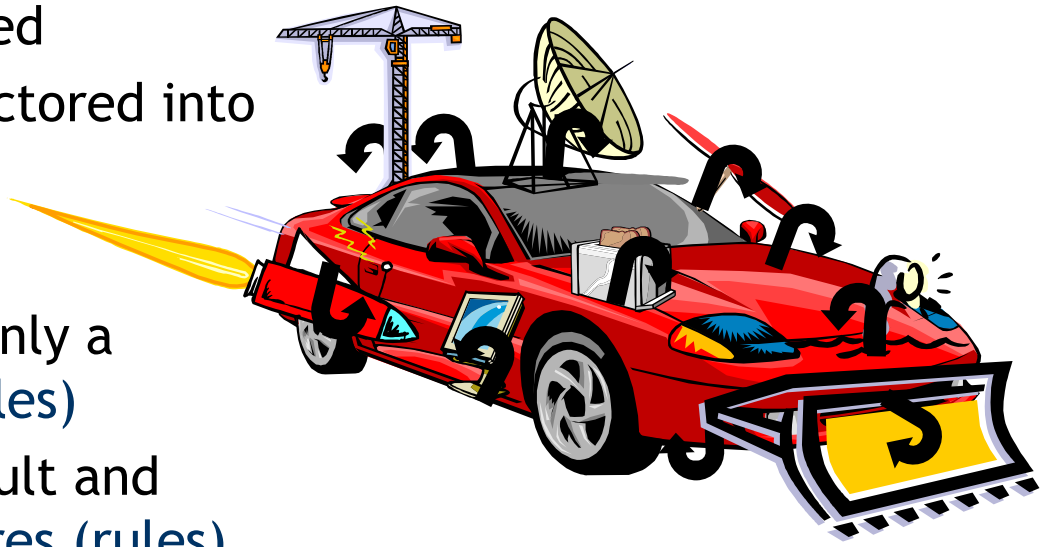
Motivation

What is the "Most Challenging" in Software?



Traditional Software Development

- First release
  - Clean and well decomposed design – minimal set of features (rules)
- Subsequent releases
  - More features (rules) added
  - New functionality is re-factored into new features (rules)
- Challenges
  - Programs typically need only a subset of the features (rules)
  - Re-factoring can be difficult and time consuming for features (rules)
  - Hooks need to be scattered throughout the code to manage the new features (rules)



BPM: The Promise of Agility

- We invented BPM to achieve three things:
  - Shorten processing time
  - Increase revenue
  - Enable business users to change themselves

- We have achieved the first two, but failed on the third.

Managing The Change

- Claims processing in a major insurance company
- 12 process steps
- > 5.000 business rules

What do you think changes faster???

How many rules exactly?

- > 90 product types and their associated **product rules**
- > 700 **data edit** rules
- > 70 **claim pending** rules
- > 200 types of **correspondence letters**
- > 250 **payment authority** rules
- > 70 claim **quality review sampling** rules
- > 1000 **special client claim handling** rules
- > 2000 **state regulatory** rules
- > 850 **accounting** rules
- > 600 published **claim processing** guidelines

Business Rules Management

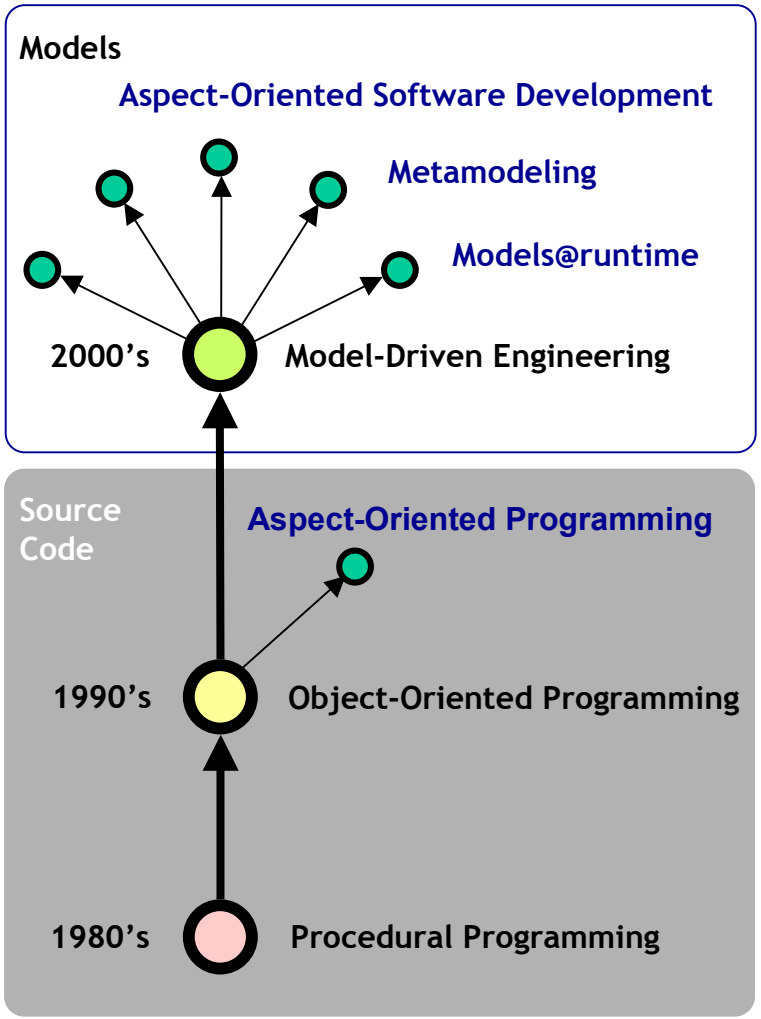
- **Business Process Management (BPM)** mainly organizes:
  - **workflows**
  - **rules**
  - **other business entities (services, transactions, etc.)**
- BPM **orchestrates** and **monitors** workflows among **people** and **business entities** under the supremacy of **Business Rules (BRs)**.
- BRs are **critical decision points** defined and implemented:
  - **within** (**tangled in**) a given business process,
  - **among** (**scattered through**) business processes ruled by choreography.
- **Tangled** and **scattered** aspects are **crosscutting** concerns that should be avoided for a **modular design**.
- BRs crosscut BPM **multi-dimensionally** in:
  - **functional design**
  - **architectural (non-functional) design**
- BRs are not simple enough to be **equalized** everywhere in BPM.

Dynamic Business Rules

- **Dynamism** requires the major quality attributes:
  - modularity
  - flexibility
  - adaptability
  - composability
- **Dynamism** can be achieved systematically with:
  - commonality/variability management
  - separation and composition of concerns
  - configurable parts
  - declarative approaches
- **Dynamism** has multi-dimensional concerns in:
  - functional design
  - architectural (non-functional) design

Separation of Concerns and Aspects

Managing Complexity in Software



Aspects – Quite Not New (1)

“Divide et Impera”

Julius Caesar

“When faced with any large task, it is usually best to put aside some of its aspects for a moment and concentrate on others“

David Gries

Aspects – Quite Not New (2)

We know that a **program** must be **correct** and we can study it from that viewpoint only; we also know that it should be **efficient** and we can study its efficiency on another day...

But nothing is gained – on the contrary – by tackling these **various aspects** simultaneously. It's what I've sometimes called "**the separation of concerns**".

Edsger Dijkstra, 1971

Aspects – Quite Not New (3)

Modularity of software is a vital concept in several applications. It helps **separate many concerns** so that they can be analyzed, modeled, designed, implemented and tested **independently**.

Professor David Lorge Parnas, 1971

Information Distributions Aspects of Design Methodology, IFIP Congress

Life: All about "Separation of Concerns" (1)

Life: All about "Separation of Concerns" (2)

Thesis for Better Programming

- The Law of **Demeter for Concerns (LoDC)** helps you to better apply, explain and understand **Aspect-Oriented Software Development (AOSD)**:
  - LoDC: Talk only to your friends who contribute to your concerns.
  - AOSD: Modularizing crosscutting concerns.
  - Concern: Any issue the developer needs to deal with: a use case, a caching policy, a business rules execution policy, whatever encapsulated...

Then, What are Programmatic Concerns?

- AOP is based on the idea that computer systems are better programmed by **separately specifying** the **various concerns** of a system.
- **Separation of concerns** is an important software engineering principle guiding all stage of a software development methodology.
- Concerns:
  - are **properties** or **areas of interest**
  - can range from **high-level** notion to **low level**-notion
  - can be **functional** or **nonfunctional (quality-based)**

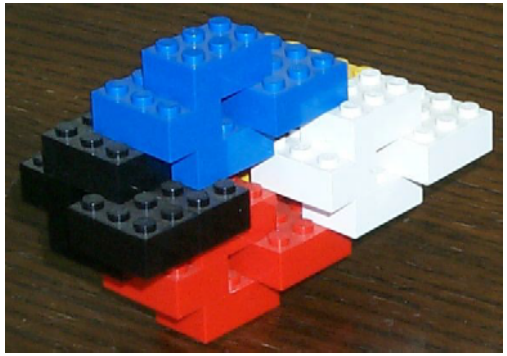
The AOSD Idea

- Provides better separation of concerns by explicitly considering crosscutting concerns (as well)
- Does this by providing explicit abstractions for **representing** crosscutting concerns, i.e. **aspects**
- and **composing** these into programs, i.e. **aspect weaving** or **aspect composing**.
- As such AOSD improves modularity
- and supports quality factors such as
  - maintainability
  - adaptability
  - reusability
  - understandability
  - ...
- Slogan: **Modularize Crosscutting Concerns**.

What are Aspects? (1)

- Current working definition is (Gregor Kiczales, May 1999):
  - modular units that cross-cut the structure of other modular units
  - units that is defined in terms of partial information from other units
  - exist in both design and implementation
- In architectural concern:
  - the aspects serve as **building blocks**.
  - in the case of more than one aspect corresponding to a concern, we call them **sub-aspects**.
  - **sub-aspects** are usually shared by more than one concern but can be composed when needed to form a composite aspect matching a single concern.

What are Aspects? (2)

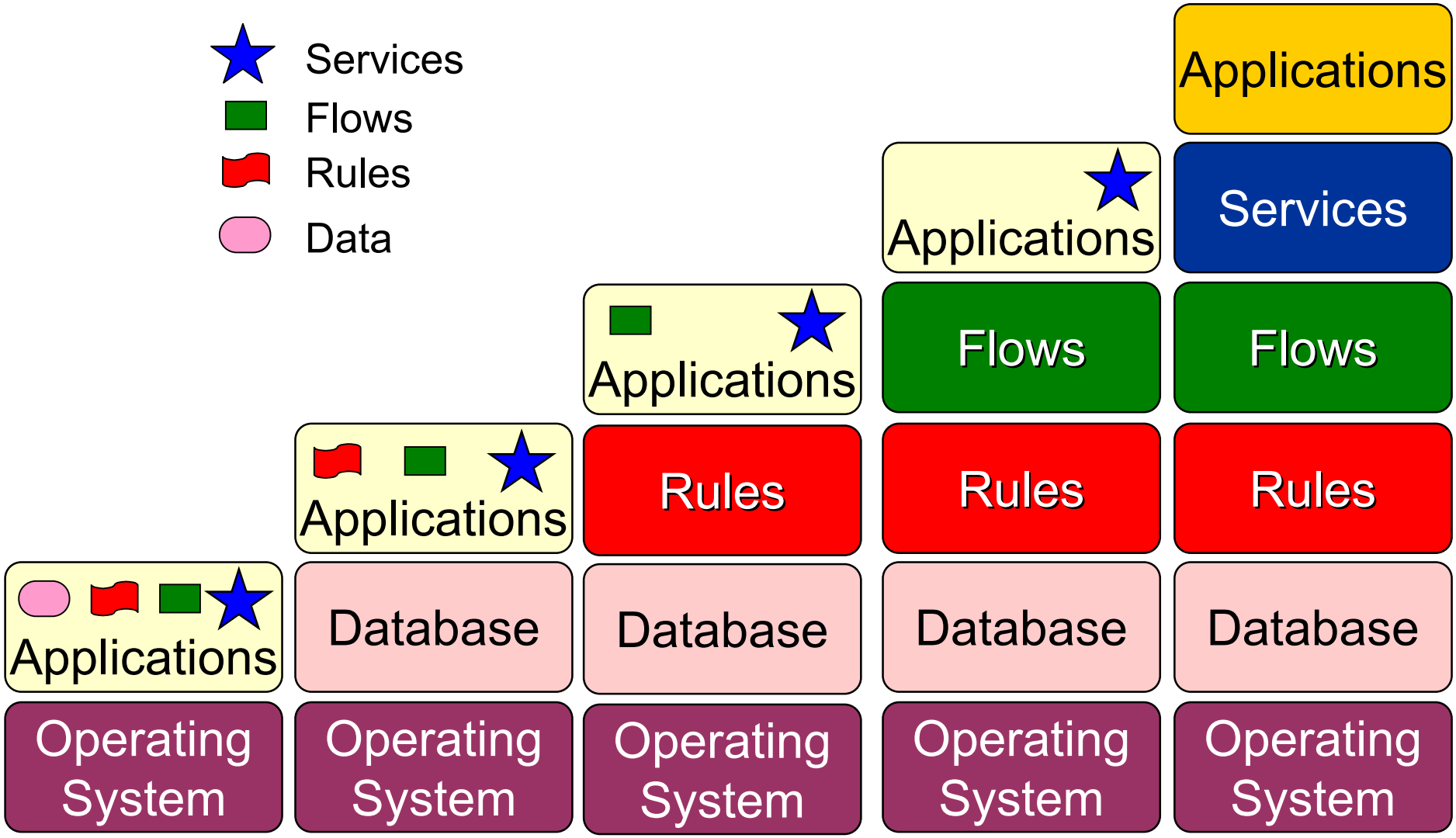


- Aspects can be seen as **LEGO bricks** that can be **assembled incrementally** to form more **complex aspects** from simpler **sub-aspects**.
- Each **concern** is addressed in a modular way by a **collection of aspects**, which can be **composed when needed**.
- The **overlapping parts** of the different **concerns** are given **explicitly** by the **aspects** common to them.
- Ideally, each aspect describes only a **small increment**. This allows **on-demand creation** or **remodularization** of aspects by **composing different collections of aspects** to match **whatever concern** is needed.

What are Aspects? (3)

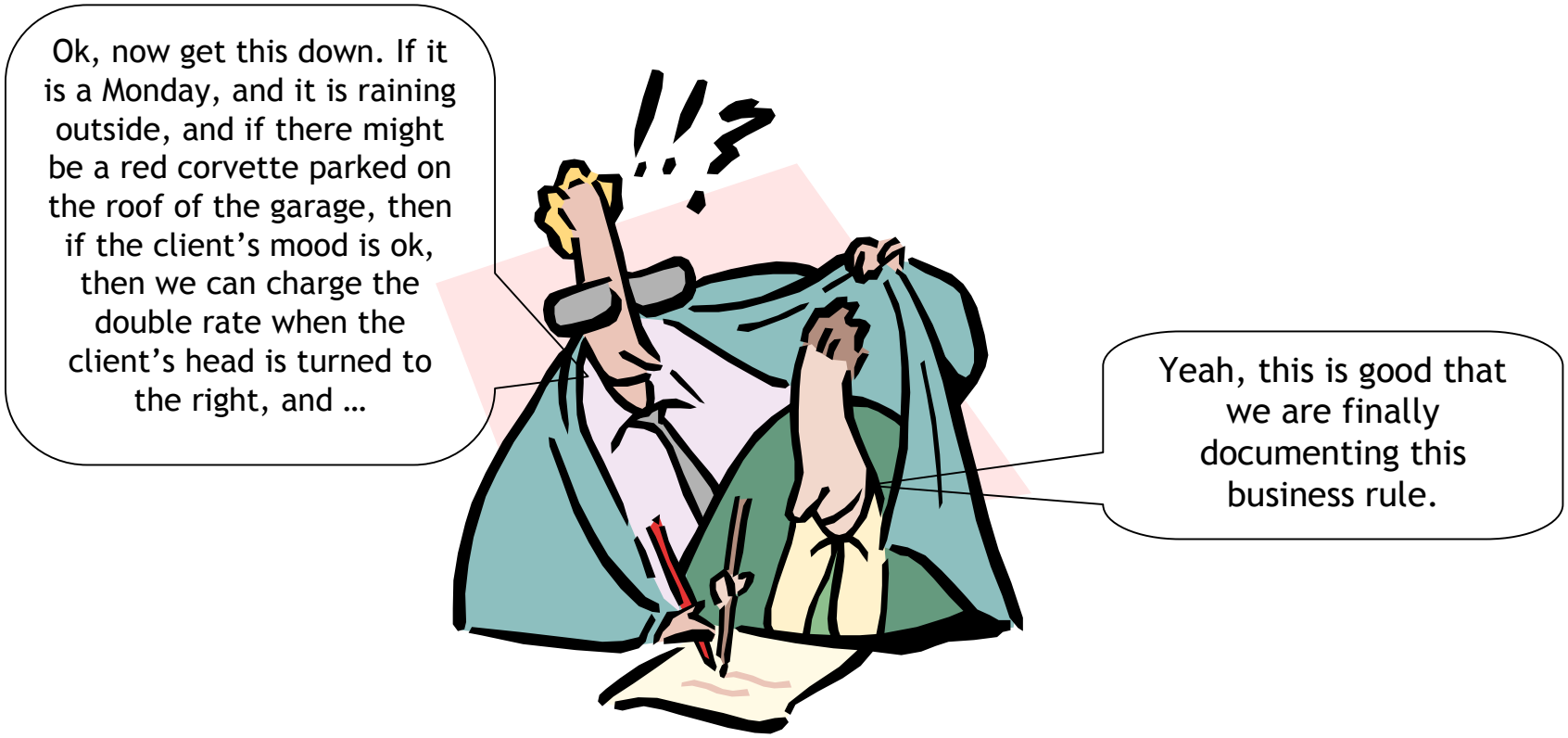
- Aspects are **similar to classes** since they
  - have **type**
  - can **extend** classes and other aspects
  - can **contain** other aspects
  - can be **abstract** or **concrete**
  - can have **fields**, **methods**, and **types** as members
- Aspects are **different than classes** since they
  - can additionally include as members **pointcuts** (picks out join points), **advice** (code that executes at each join point)
  - do not have **constructor** or **finalizer** and they cannot be created with the new operator.
  - can **access private members** of other types if they are privileged

What Are The Major Aspects of Software?



Business Rules as
A Separate Concern

Business Rules?



What is a Business Rule? (1)

"…a statement that defines or constrains some aspect of a business by asserting control over some behavior of that business"

What is a Business Rule? (2)

- A business rule is:
  - an atomic unit of business logic
  - susceptible to change
  - crosscutting application components
  - a constraint, derivation or reaction
- Avoid logical duplication of business rules to ensure:
  - independent development
  - independent evolution

  of business rules and applications

Business Rules Aims to

- Integrate IS concepts such as data, process, object and event.
- Define the actors and prescribe how they should behave by setting constraints.
- Relate business concepts in terms meaningful to business people.
- Bridge the communication gap between IS practitioners and the business people they serve.
- Help manage business change.

Business Rules Classification (1)

- **Definitions**: define business terms such as:
  - A CUSTOMER is defined as someone who uses our services or products.
  - A STUDENT is defined as someone admitted to the university.

Business Rules Classification (2)

- **Facts**: connect business terms in ways that make business sense such as:
  - A CUSTOMER may place an ORDER.
  - A CUSTOMER may have a CREDIT LIMIT.
  - A STUDENT may enroll in a COURSE.

Business Rules Classification (3)

- **Constraints**: constrain how business terms are connected such as:
  - A CUSTOMER may have no more than one UNPAID ORDER.
  - Each PROFESSOR may teach up to four COURSE SECTIONS.

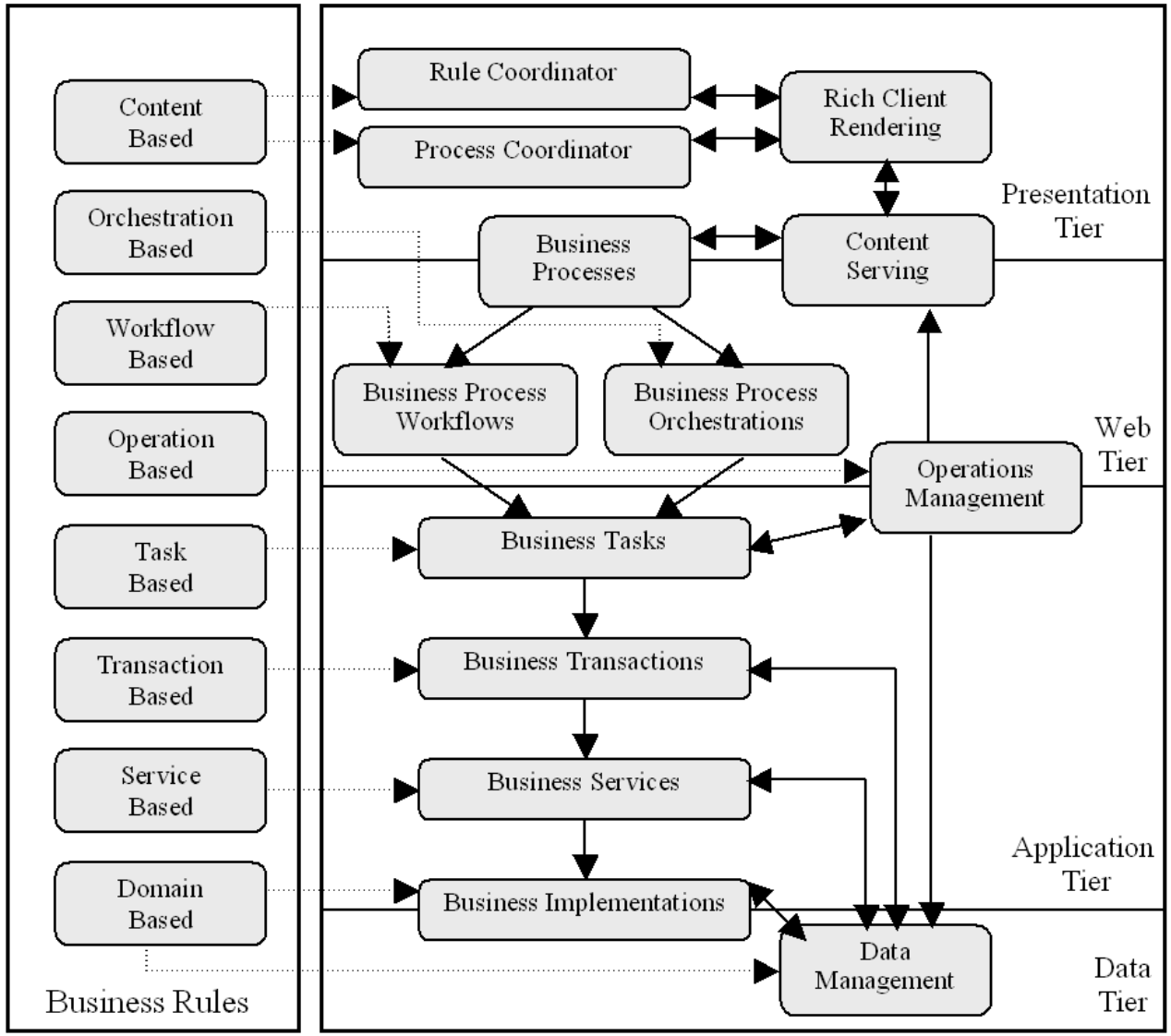
Business Rules Classification (4)

- **Derivations**: enable new knowledge or actions such as:
  - If a STUDENT has a GPA >= 3.0, then that STUDENT is an HONORS STUDENT.
  - A CUSTOMER's CREDIT AVAILABLE is his CREDIT LIMIT minus OUTSTANDING BALANCE.

Caveat: Who Owns The Business Rules?

- IS practitioners DO NOT create business rules; **business people do**!
- Your task is:
  - to help users **identify**, **formulate**, and **monitor** business rules,
  - to **implement** these business rules in an executable way,
  - to **facilitate** information systems evolving with changes in business rules.

Business Rules Crosscutting The BPM Tiers



- Business Process Workflow is the flow of internal processes within an organization.
- Business Process Orchestration is the supervision of internal and external processes.
- Business Task is an activity within an organization to be performed by or delegated to people. It may contain one or more Business Transactions.
- Business Transaction is the set of short term and long term actions to carry out a specific business activity.
- Business Service is the set of interrelated Business Implementations managed by a computational model.
- Business Implementation is the set of data processing commands under a computational model.

Taxonomy of Business Rules

Business Rule Category	Complexity	Criticality	Frequency of Change	Order of Execution	Type of Access	Responsibility
Content-based	Simple	Low	Vibrant	Any	Internal	Business
Orchestration-based	Simple	Moderate	Low	First	Both	Business
Workflow-based	Composite	Moderate	Low	First	Internal	Business
Operation-based	Composite	Moderate	High	Any	Internal	IT
Task-based	Reasoning	Moderate	Vibrant	Any	Internal	Both
Transaction-based	Reasoning	Severe	Fair	Any	Internal	Both
Service-based	Reasoning	Severe	Vibrant	Any	Both	Both
Domain-based	Reasoning	Severe	Vibrant	First	Internal	Both

- Content-Based Rules are primarily used for data validation and verification.
- Orchestration-Based Rules supervise the composition of internal and external processes.
- Workflow-Based Rules govern the workflow of business tasks within an organization.
- Operation-Based Rules manage the operational concerns such as scheduled execution constraints.
- Task-Based Rules are circumstances under which the set of business transactions will be in coordination.
- Transaction-Based Rules govern the association of related business services under certain conditions.
- Service-Based Rules administer the circumstances for the combinatorial use of business implementations.
- Domain-Based Rules are the mostly referred ones to make a business model consistent.

Existing Approaches for BR Segregation

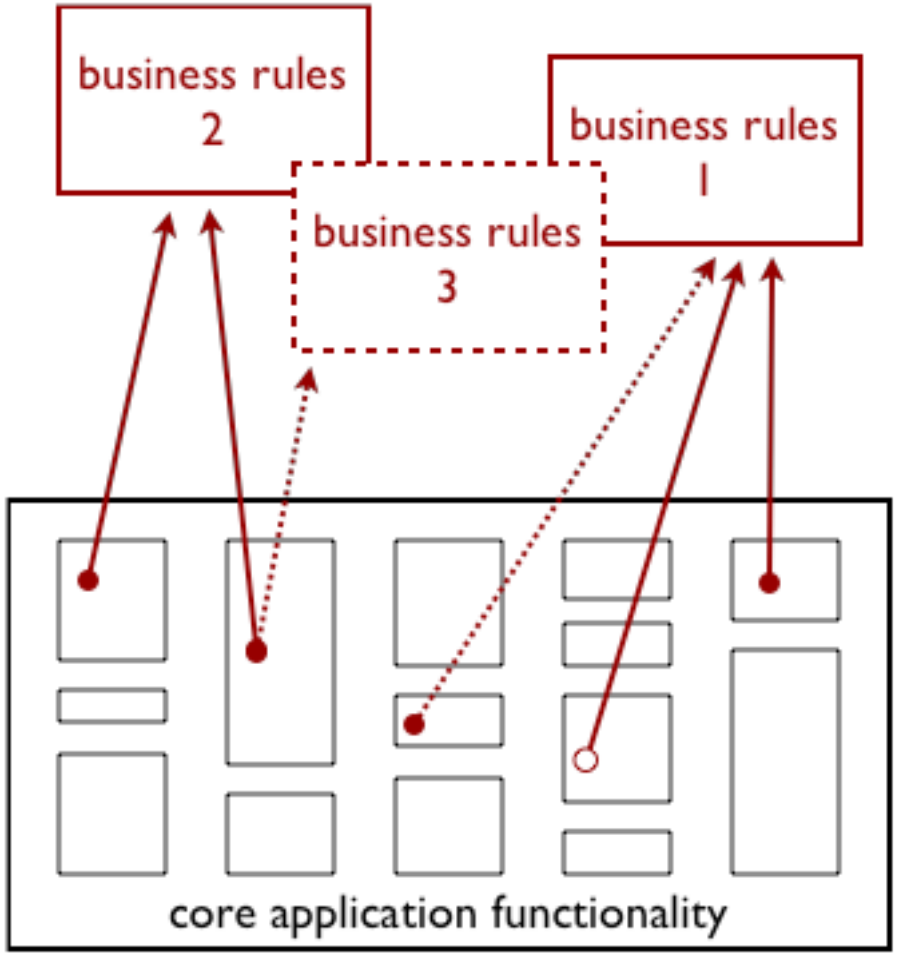
- Structured approach:
  - **modularity**
  - **information hiding**
- Object-Oriented approach:
  - **abstraction and encapsulation**
  - **design patterns (delegate, proxy, visitor, MVC, etc.)**
- Adaptive Object Model (AOM) approach:
  - **major help in "architectural dynamism"**
  - **BRs can be individually modeled in dynamic object hierarchies**
- Aspect-Oriented Programming (AOP) approach:
  - **major help in "functional dynamism"**
  - **BRs are first-class crosscutting concerns**
- Business Rules engines:
  - **versatile enough in server-side modeling**
  - **deployment drawbacks in multi-tier architectures (esp. for RIA)**

Separation of Business Rules with AOSD

- Decoupling business rules from core application
- Focus on **decoupling business rules connectors**: code that connects business rules with core application functionality
- **AOSD** is ideal to achieve the decoupling of business rules connectors

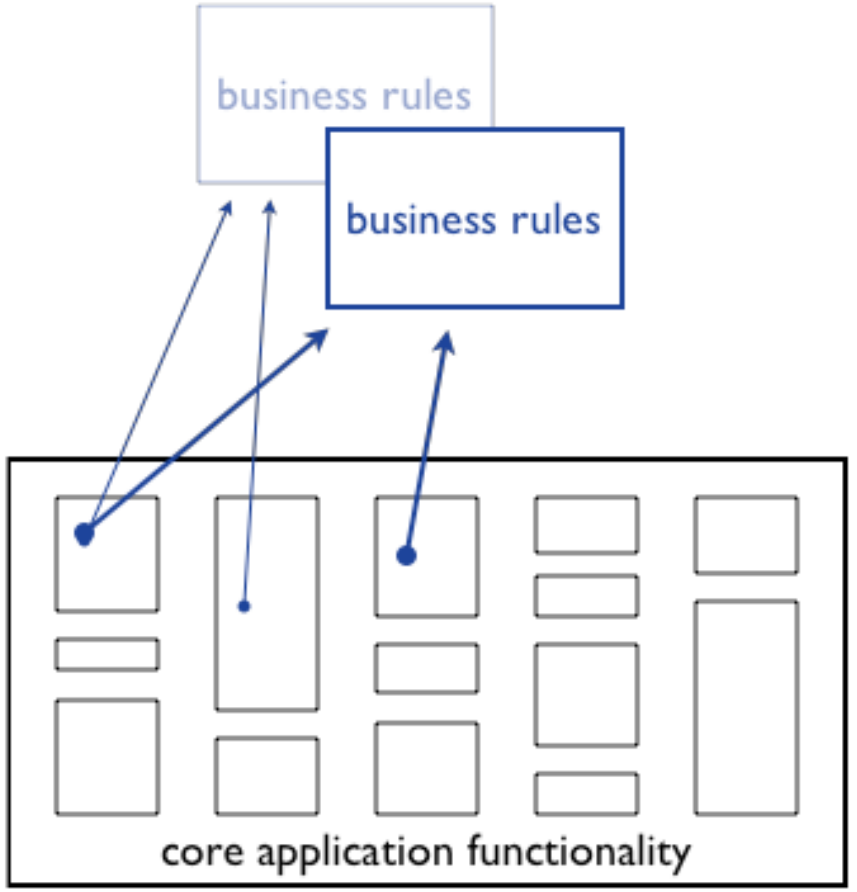
Triggering Business Rules

- Identification of events when to trigger the business rules:
  - static or dynamic
- Different events trigger same rules - code duplication
- unanticipated rules - invasive changes
- Unanticipated events - invasive changes
- Examples: discounts
  - when product price is obtained
  - when a customer checks out
  - when a customer checks out with one-click ordering



Dynamic Configuration of Business Rules

- Rules change at run-time
- Rule events change at run-time
- Rule connector should be dynamically reconfigurable
- Combination strategy deferred to run-time
- Introduction of Rule Inference Engines



Benefits of AOSD based Rule Management

- To non-invasively **adapt/configure** the business code:
  - Better modularity, reusability, and maintenance
  - Quicker reaction to customer needs
- To enable checking the **compliance of applications** with requirements:
  - Design by contracts
  - Goal-oriented requirements engineering
- To better model the **software architecture**
  - Multiple views and multi-dimensional separation of concerns
  - Association of functionality with software quality attributes

RUMBA: RUle-based Model for Basic Aspects

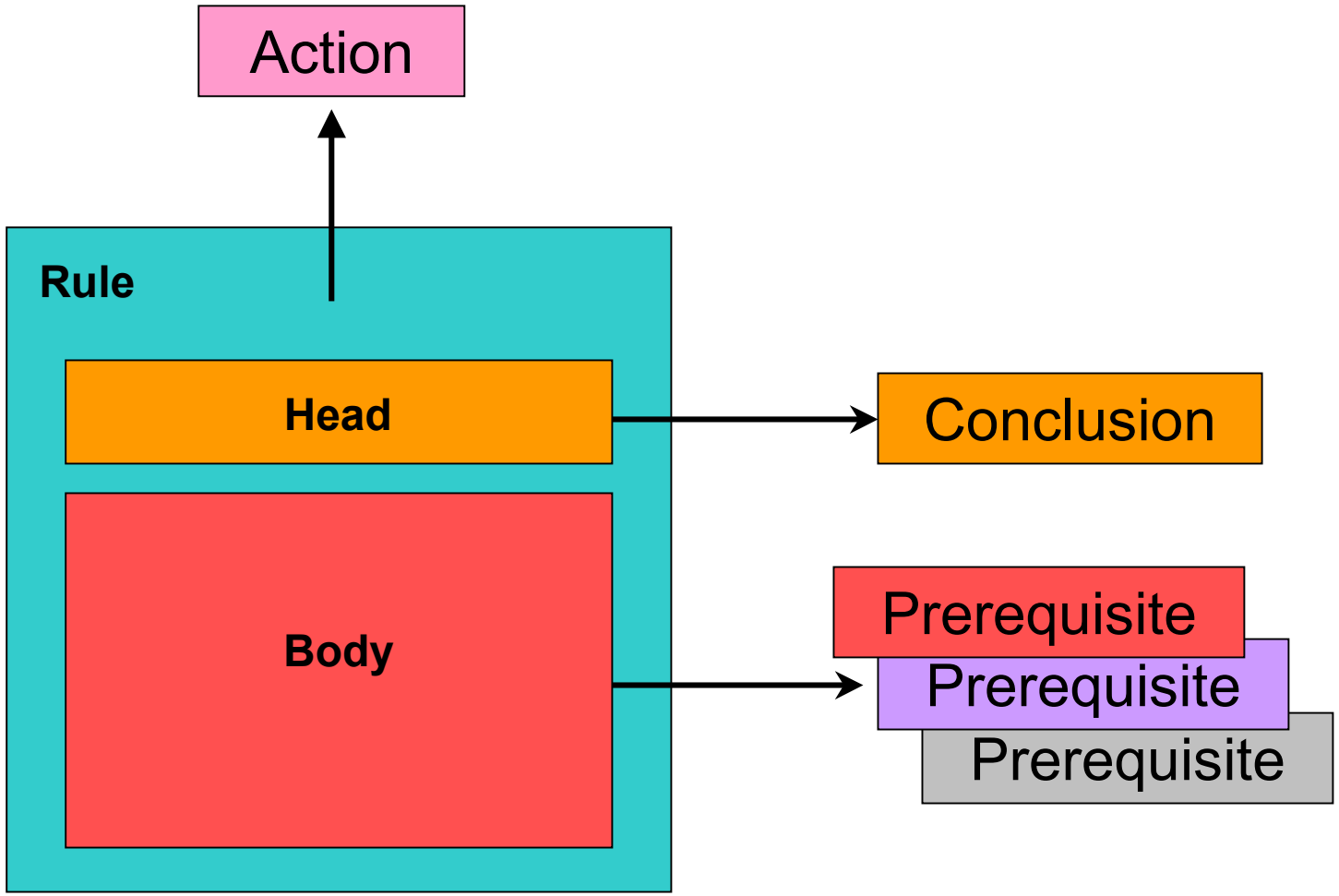
What is RUMBA? (1)

- stands for **RU**le-based **M**odel for **B**asic **A**spects.
- is a dynamic **Aspect-Oriented Framework (AOF)**.
- can model any business domain with **dynamic aspects**.
- may contain **methods**, **attributes** and **sub-aspects**.
- has a **recursive** manner for **aspect** and **rule** definitions.
- has a **visual environment** for business rules.
- has an infrastructure to model **composite rules**.
- is extensible with basic Java **method implementers**.
- provides **dynamism** with **deferred encapsulation**.
- can be used as a **standalone** Rule Execution engine.
- is **compact (28K)** enough to be used at every BPM tier.

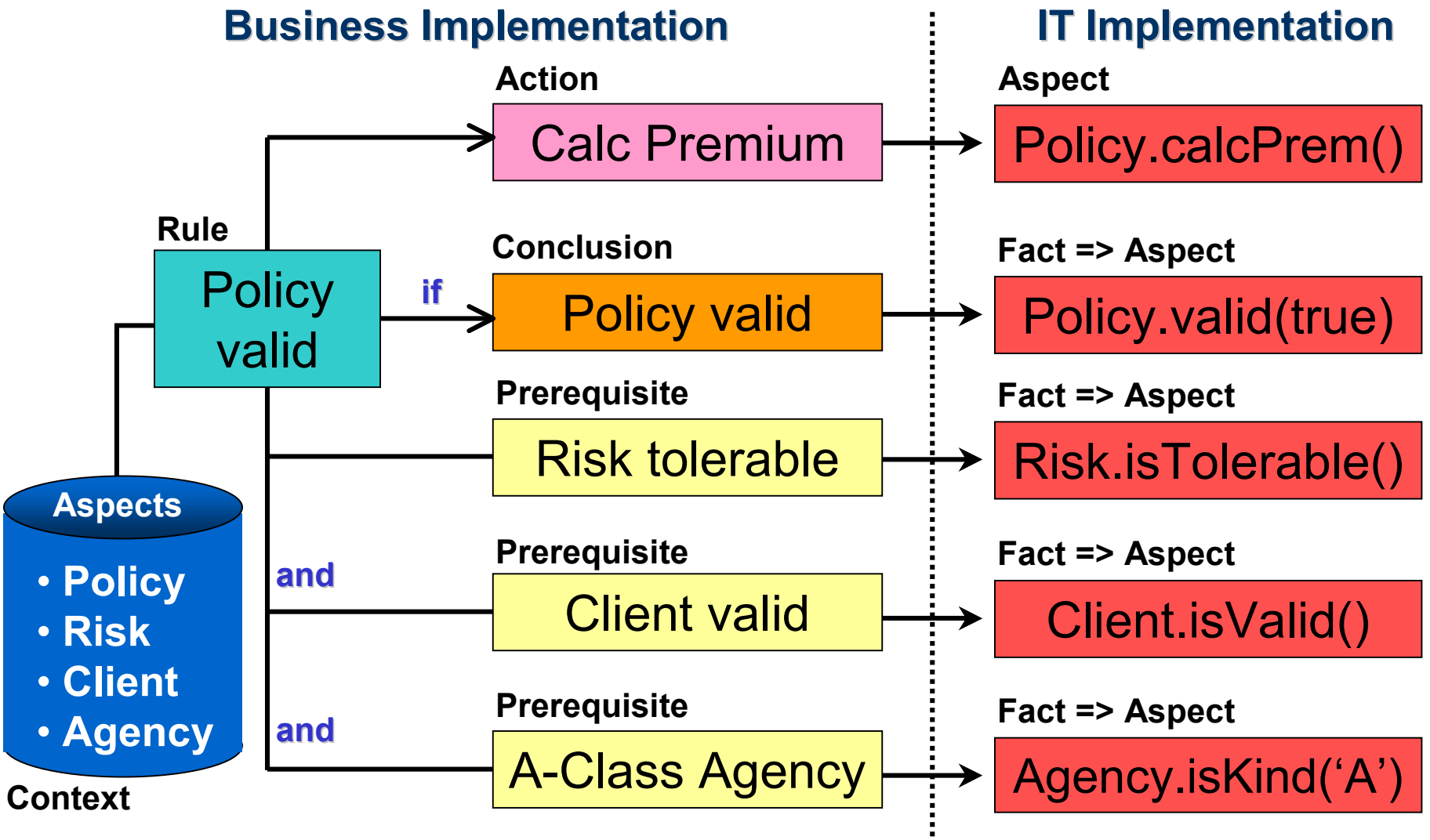
What is RUMBA? (2)

- inspired by the **Feature-Based Development (FDD)**.
- a **feature** in RUMBA is a **basic aspect**, which is the set of **data attributes, methods**, and other **basic aspects**.
- this recursive meta-structure enables the definition of **dynamic features** in a declarative environment.
- all **data types** are dynamic, thanks to **Data Face** pattern.
- all **methods** have the generic interface even for dynamic invocation.
- the only static part is the **Java method implementers**.
- even method implementers are **dynamically** bound to aspect **methods** in run-time with bytecode engineering, thanks to **Reflective Aspect** pattern.
- any business domain can be fully modeled with **dynamic features** and related **business rules**.
- simple and composite **business rules** are defined declaratively and later created **dynamically**, thanks to **Reflective Rule** pattern.

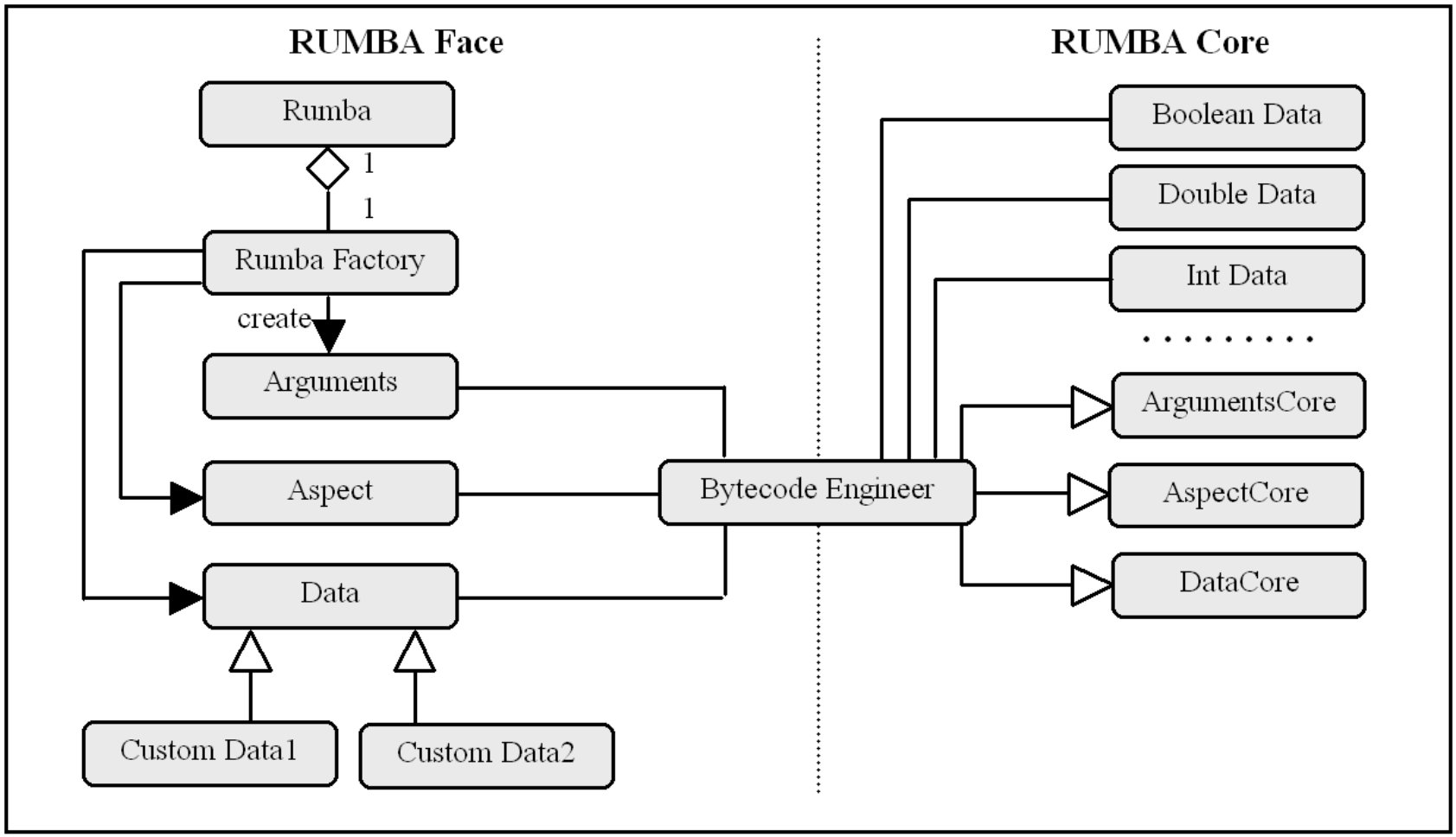
RUMBA Rules – RuleML Compliant Structure



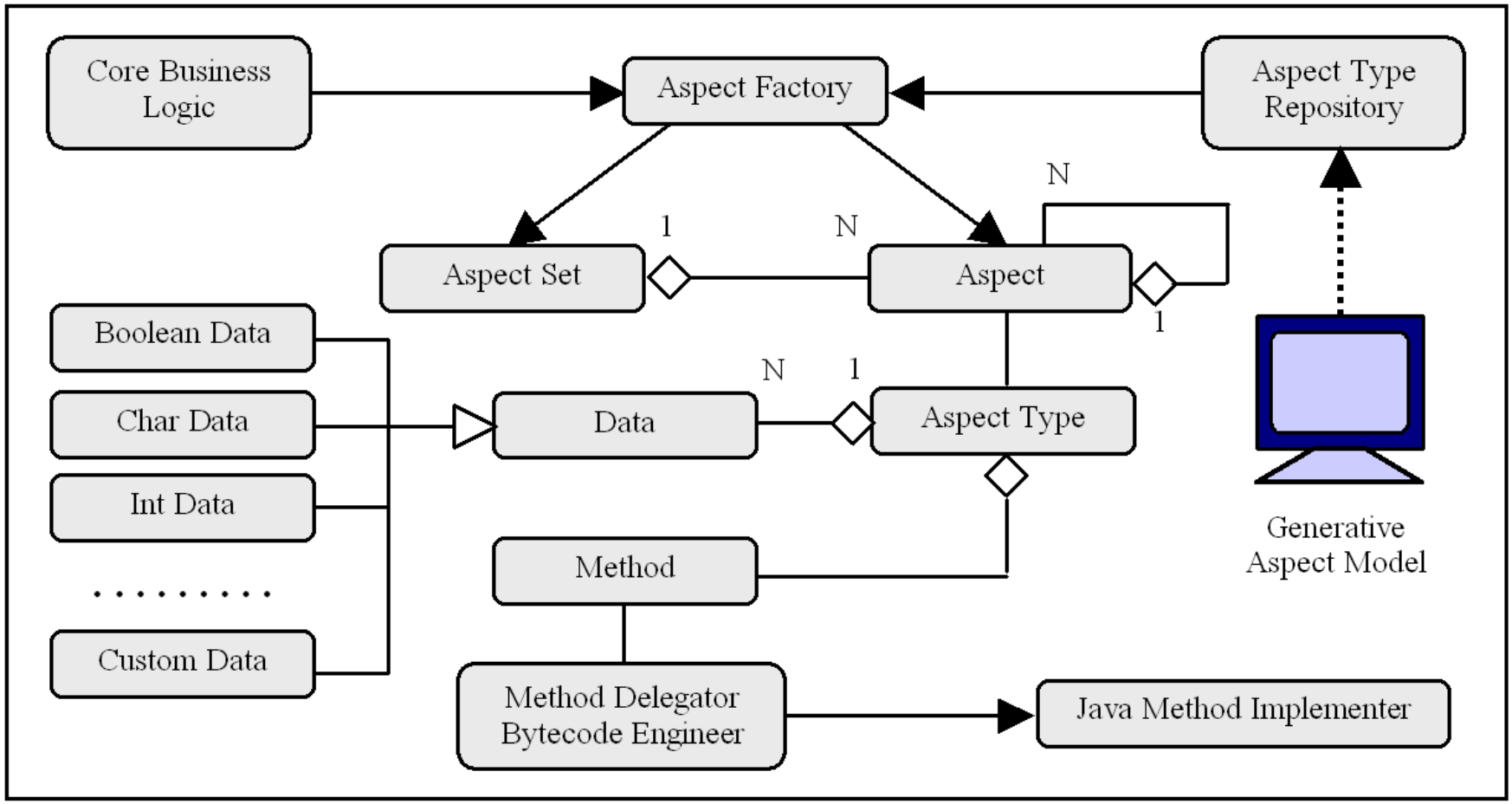
Business Rules in RUMBA



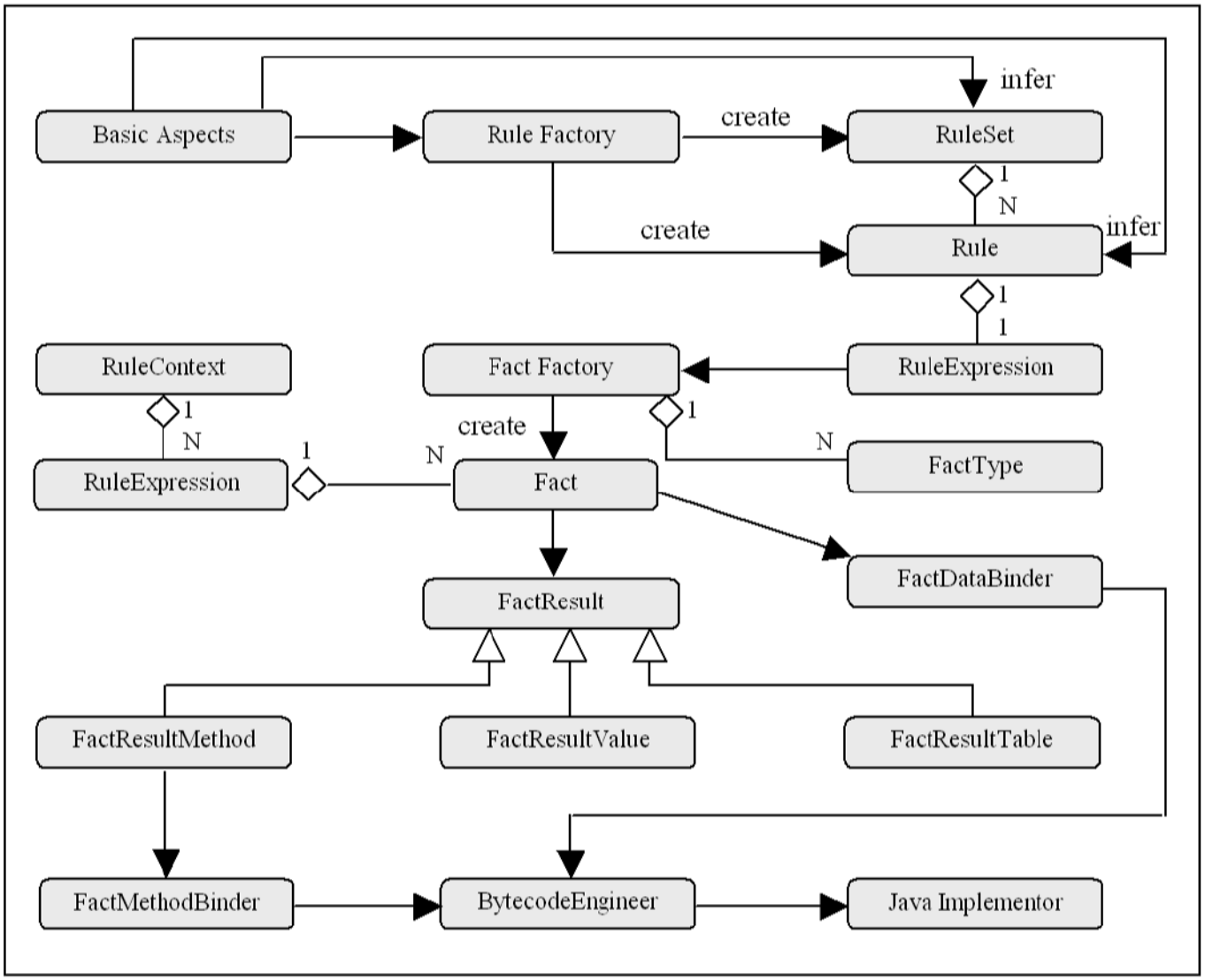
Data Face Pattern



Reflective Aspect Pattern

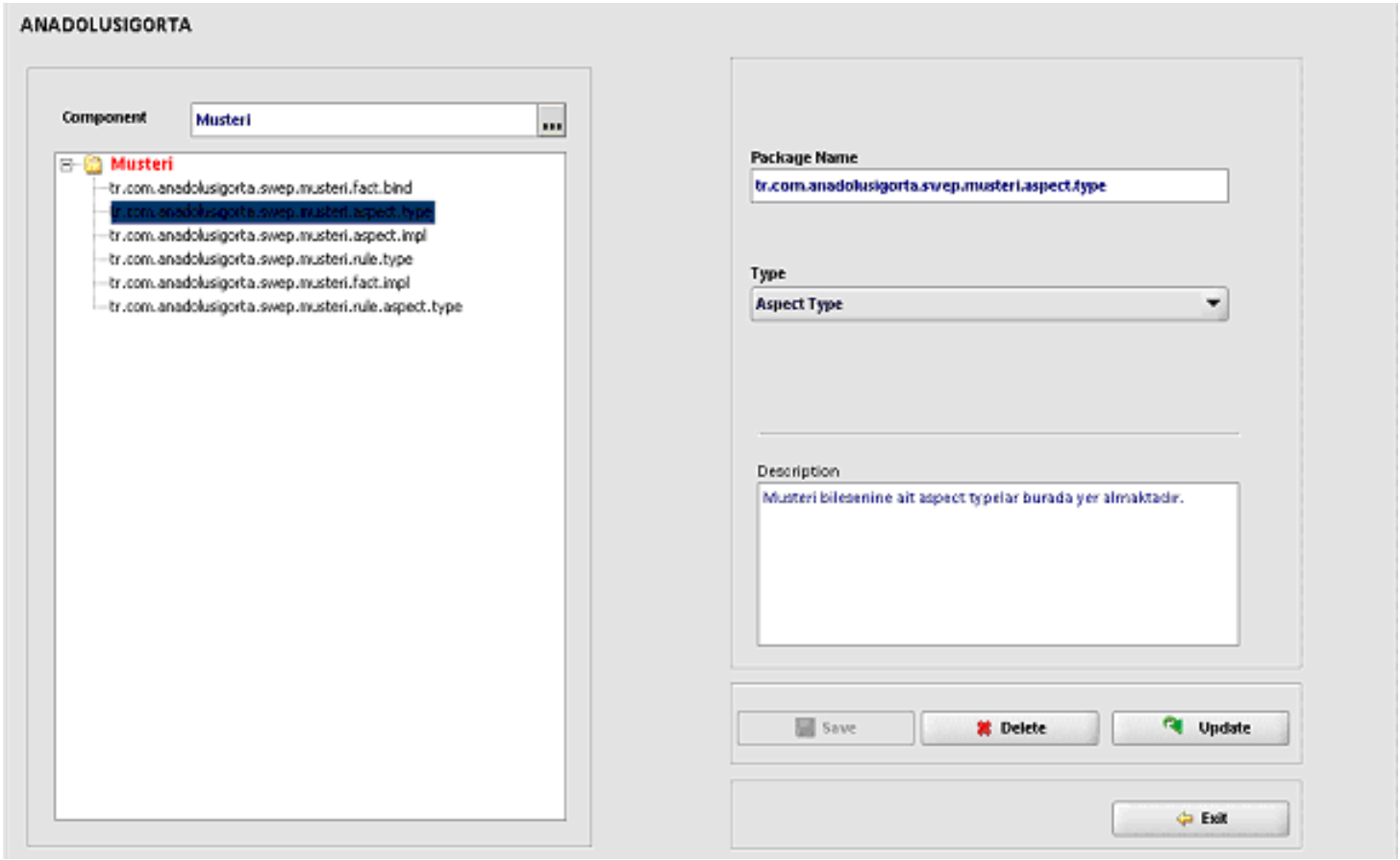


Reflective Rule Pattern



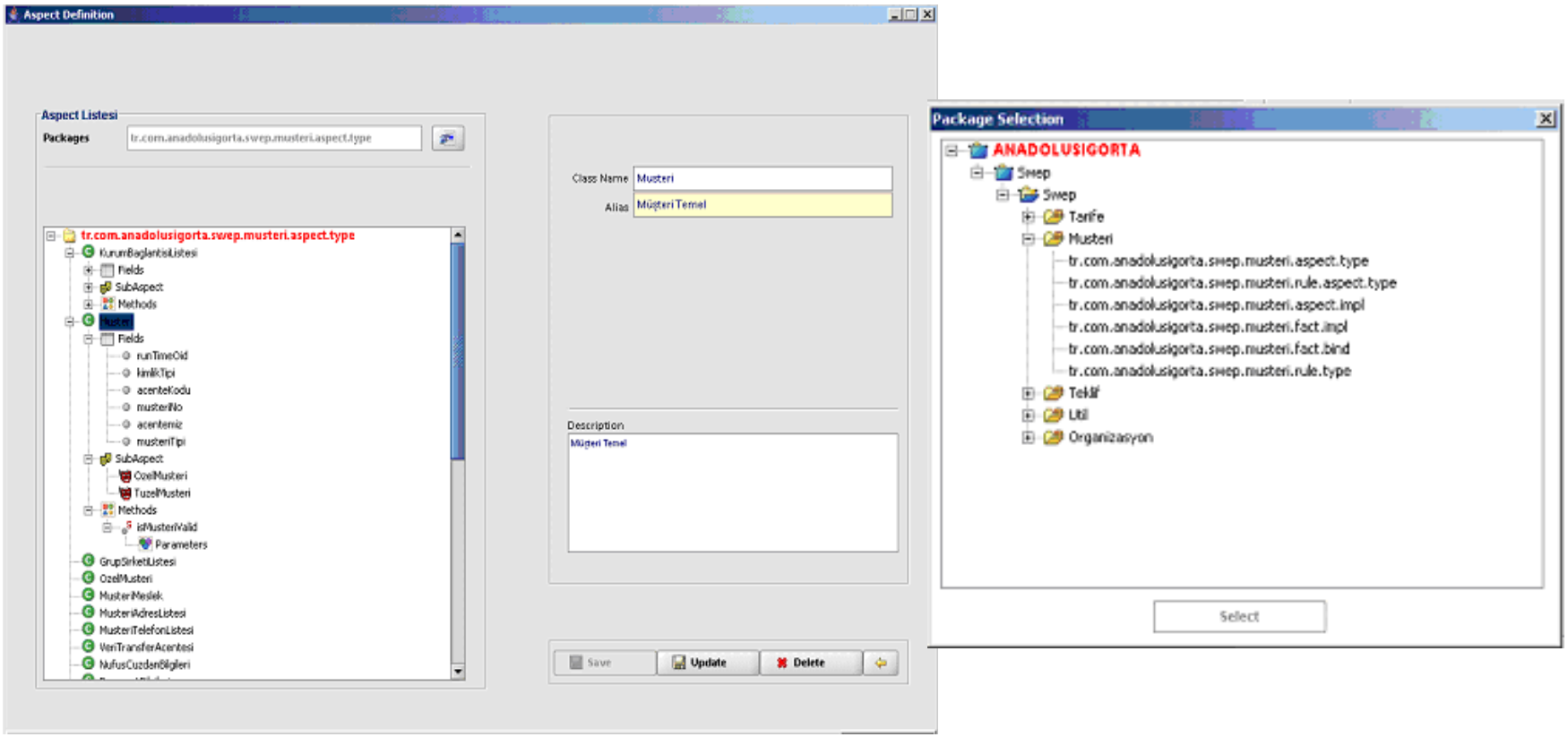
RUMBA – Defining Packages

- First, the Java packages for **Aspect Types** and **Aspect Method Implementers** are defined.



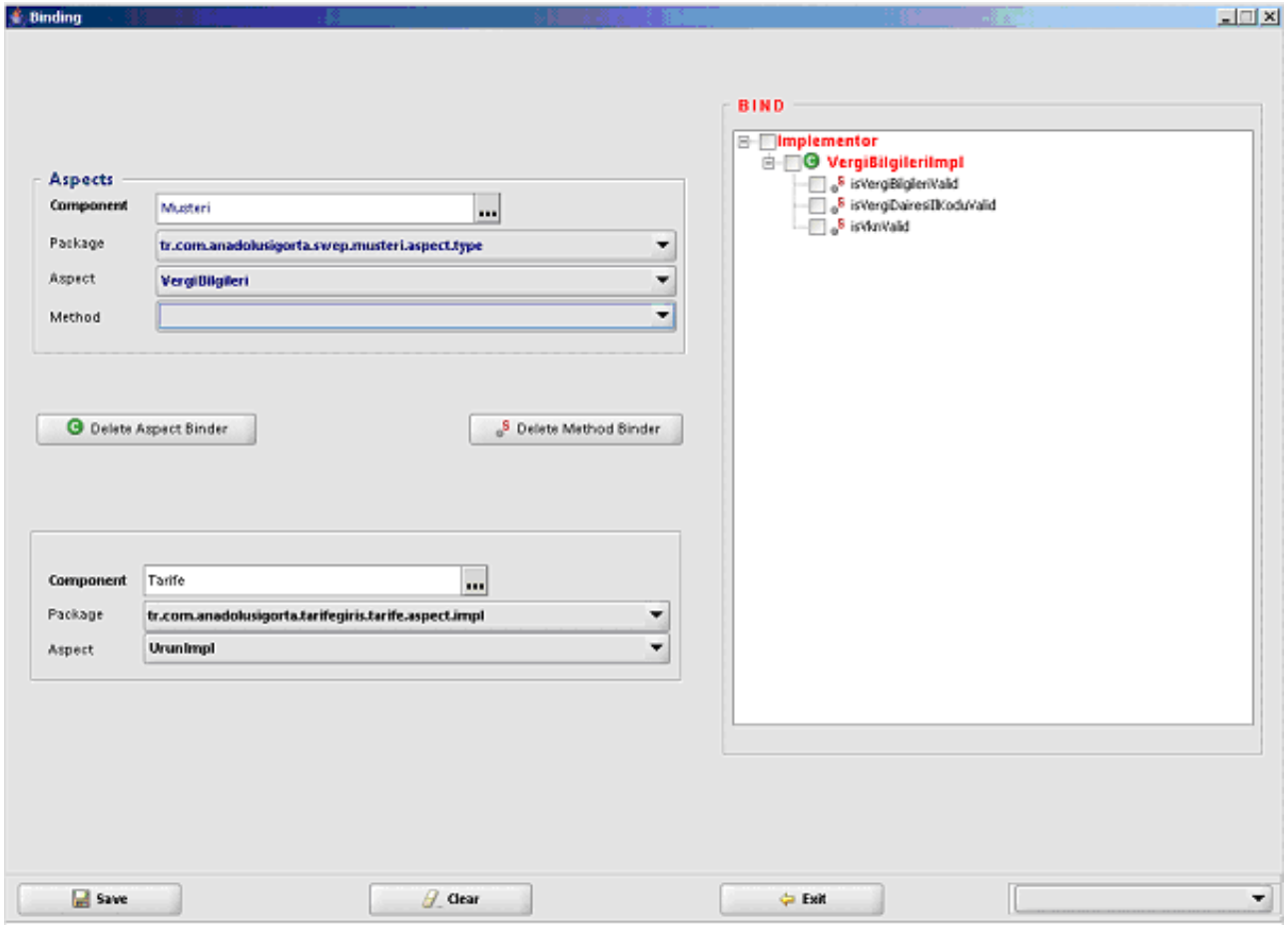
RUMBA – Defining Aspects

- **Aspects** and **Method Implementors** are created.
- If needed, **Aspect Attributes** and **Sub-aspects** can be defined accordingly.



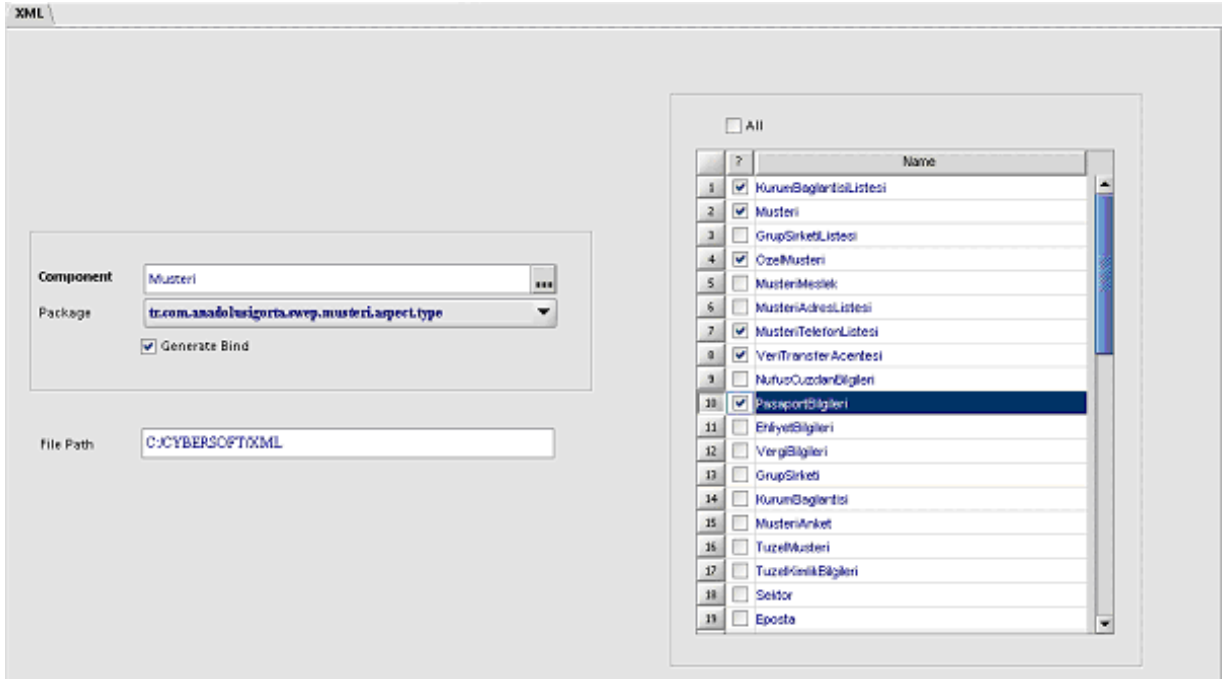
RUMBA – Associating Aspects with Methods

- Defined **Aspect Methods** are associated with corresponding **Aspects**.



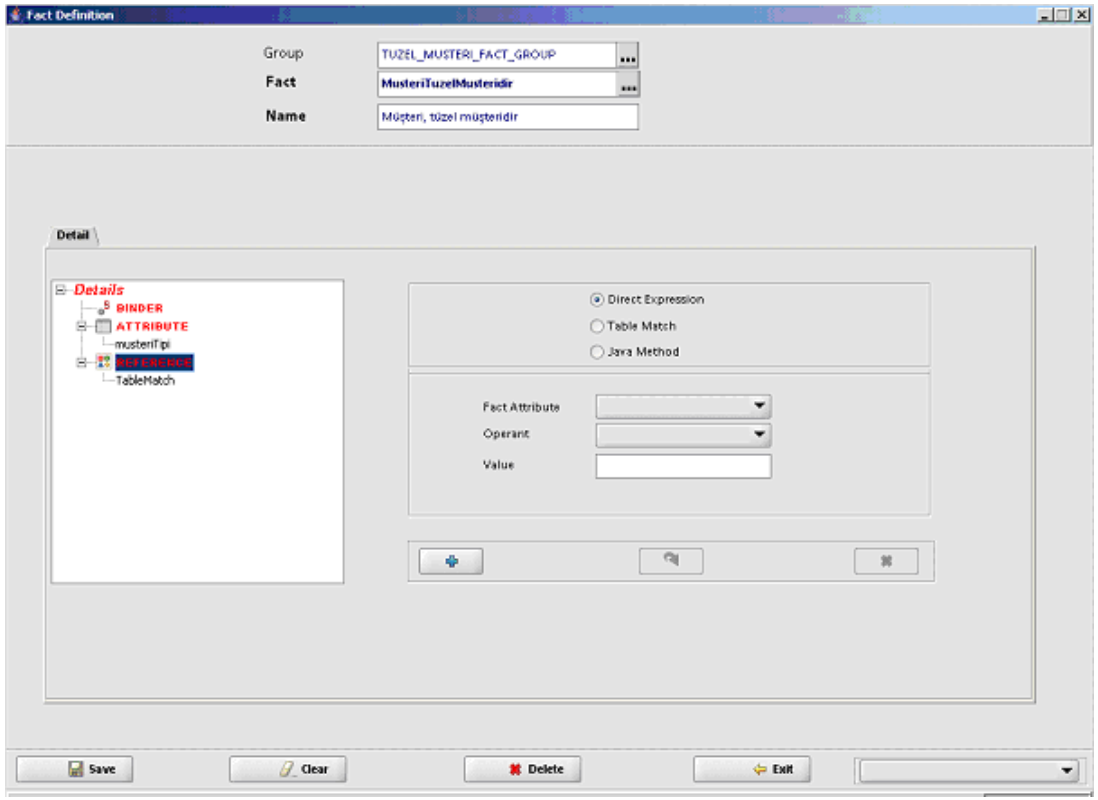
RUMBA – Generating the Aspect Frame

- Associated Java classes are generated for actual **Aspect Methods** behaviors only in “enclosed” format. Filling in the method bodies is, of course, left to developers.
- A **supplementary XML file** is also generated to demonstrate the binding association.



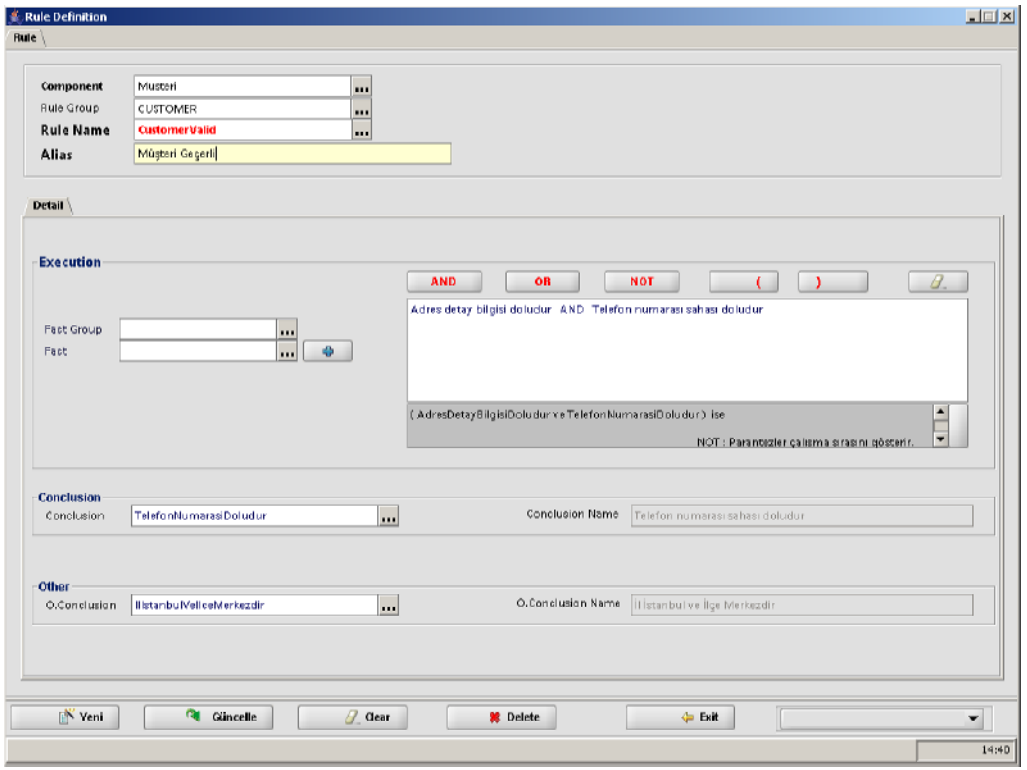
RUMBA – Defining the Facts

- Facts can be defined in three different forms:
  - **Basic Value Check** [e.g. $color == “brown”]
  - **Database Table Lookup** [e.g. $clientName in Client table]
  - **Java Method Call** [e.g. ClientRules.isColorBrown($color)]



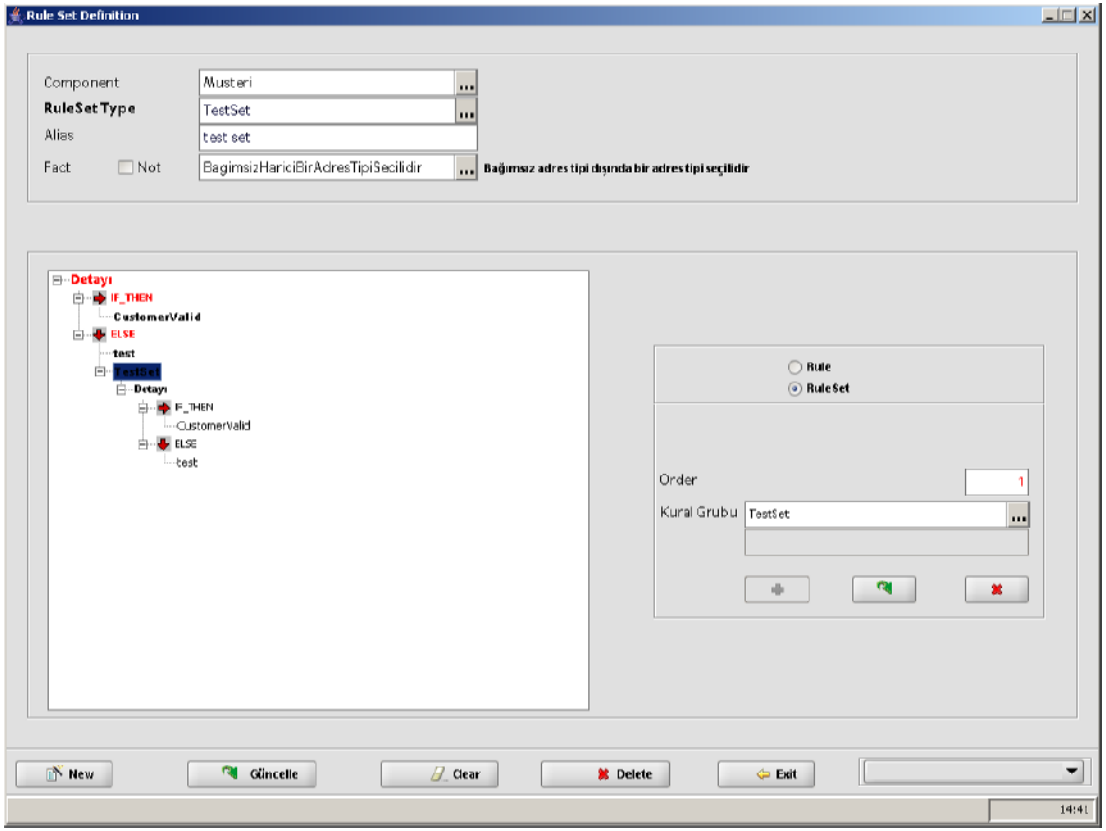
RUMBA – Defining the Rules

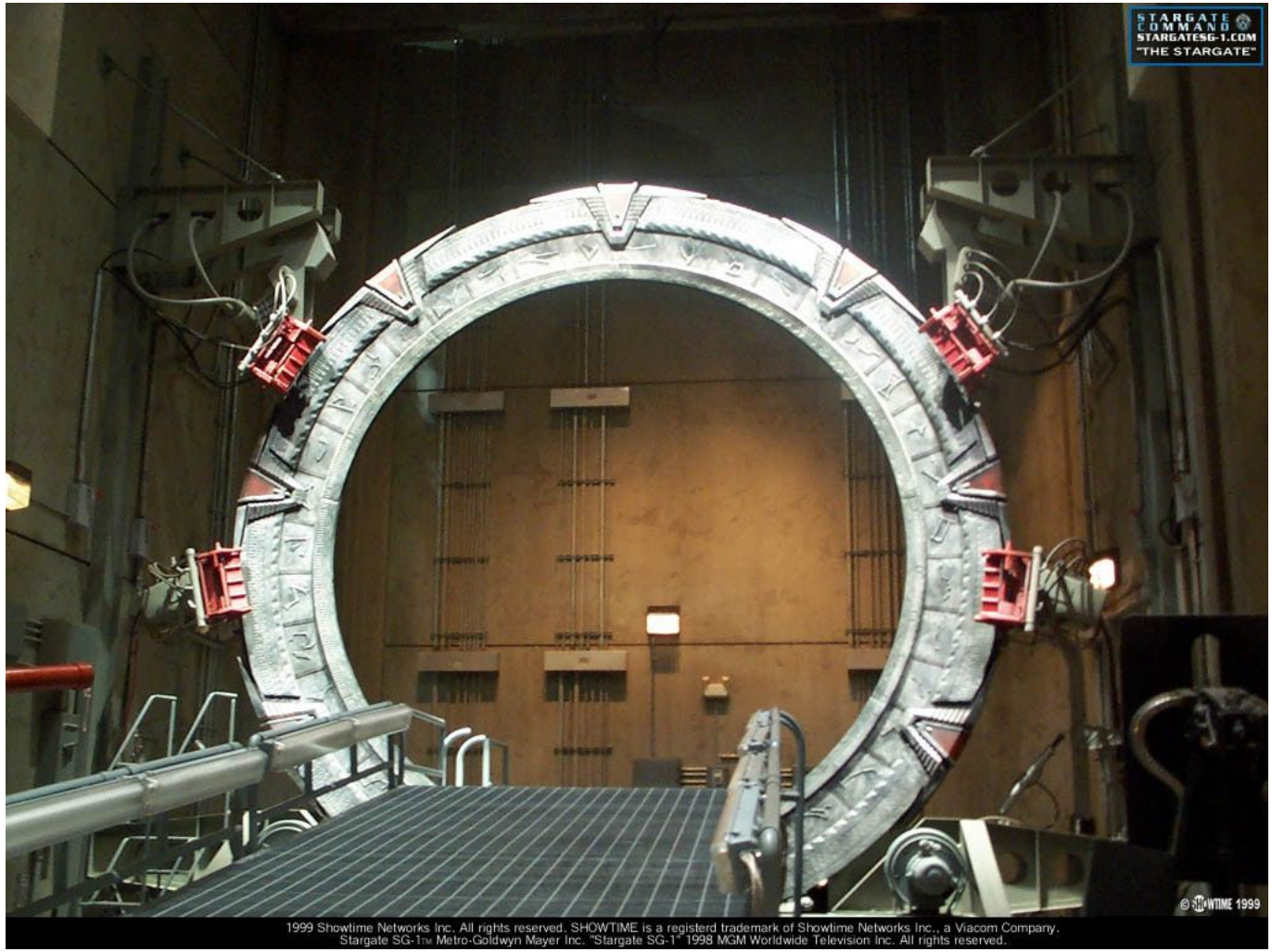
- Rules consist of facts combined by logical operators.
- The user should name the rule and define the **conditional** and **constitutional** parts accordingly.
- Rules and facts are constructed **on demand** by RUMBA.



RUMBA – Defining the Rule Sets

- Rule Sets include both **rules** and other **rule sets**.
- Rule Sets are defined by combining the **rules** and **rule sets** connected by logical operators.





Conclusions

Conclusions (1)

- RUMBA provides declarative environments for business people to define **features (basic aspects)** and **business rules** for any business domain.
- Later on, IT people implements **only the Java methods** for computational model and bind them with associated methods of basic aspects.
- Any method of basic aspects can be associated with different implementers for **polymorphic** dynamism.
- If needed, rule facts are also associated with Java **implementers** by IT people.
- Method implementers and rule fact associations can be changed **dynamically** even when the application is running.

Conclusions (2)

- All these definitions are kept in a **repository** and **dynamically managed** by graphical RUMBA interface.
- Business rules are completely segregated from the business processing model at every architectural tier, and this theoretically provides the **"if-then-else" free** implementation model to IT people.
- Both **business processing model** and **business rules model** have been separated and they can be managed independent from each other with declarative environments.

That's All Folks...

Cybersoft

vision@work to make IT happen