An Agile MDA Approach for Service-Oriented Components

S. Motogna, I. Lazăr, B. Pârv, I. Czibula

Department of Computer Science
Babeş Bolyai University
Cluj-Napoca, Romania

ComDeValCo

- **Com**ponent **De**finition, **Va**lidation and **Co**mposition framework
- Includes:
  - Action language
    - UML structured activities
    - Graphical notation
  - Dynamic execution environment: iComponent

The context

Agile MDA approach:

- MDA: OMG,2003
  - PIM, PSM, transformations
  - heavy-weight processes
- Agile – Mellor, 2005
  - light-weight process
  - test-first
  - immediate execution
- Executable UML
  - Non-standard action languages: Mellor&Balcer, 2002;
  - fUML: OMG standard, 2008

Service-orientation:

Approach:

1. Decompose ⇒ a collection of interacting services.
2. Define components implementing application services.
3. Define composite components for execution - as service specifications

Platform dependent:

- SCA,2007; iPOJO, 2008;

Platform independent:

- iComponent, 2008

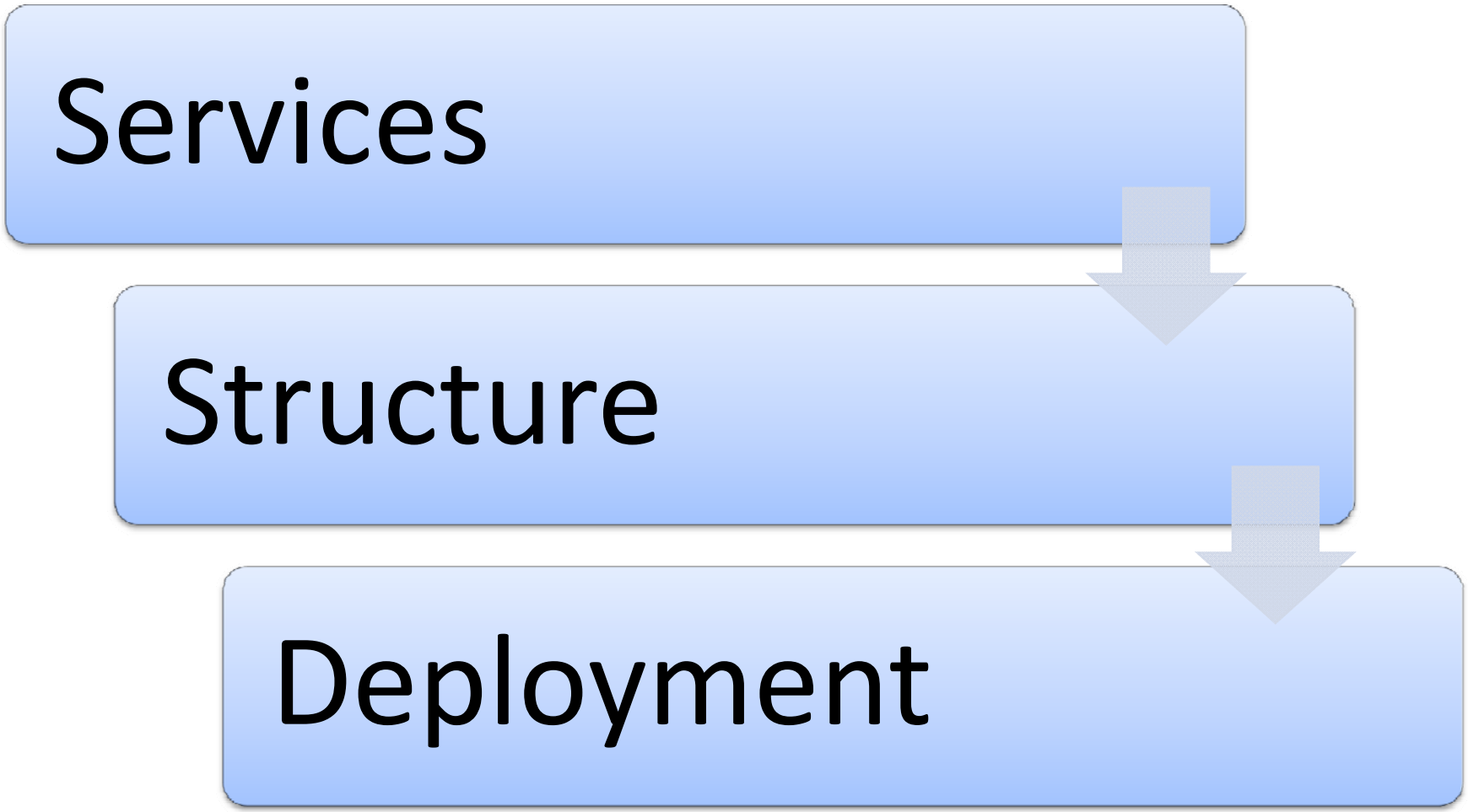
Our approach

- Any UML case tool can be used to construct the models
- conform to fUML specification for executable models

➢ consists of applying the following steps in the specified order:

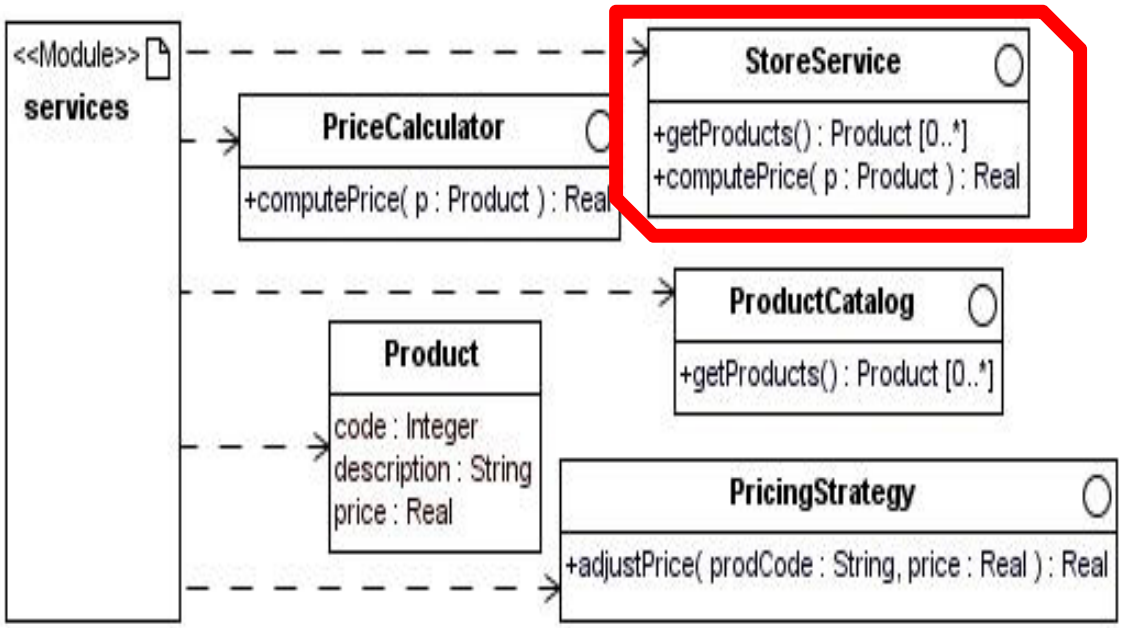
1. the model is described on different layers:
2. for simple components proceed with test-first component development.

Model description layers



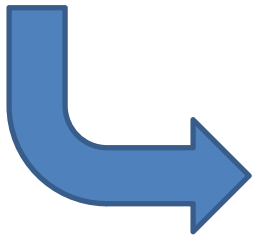
Service Model

- Defined by the system analyst
- Describes the services provided by the system.
- Corresponding modules may include any data type:
  - classes,
  - interfaces, or
  - components.



Structural Model

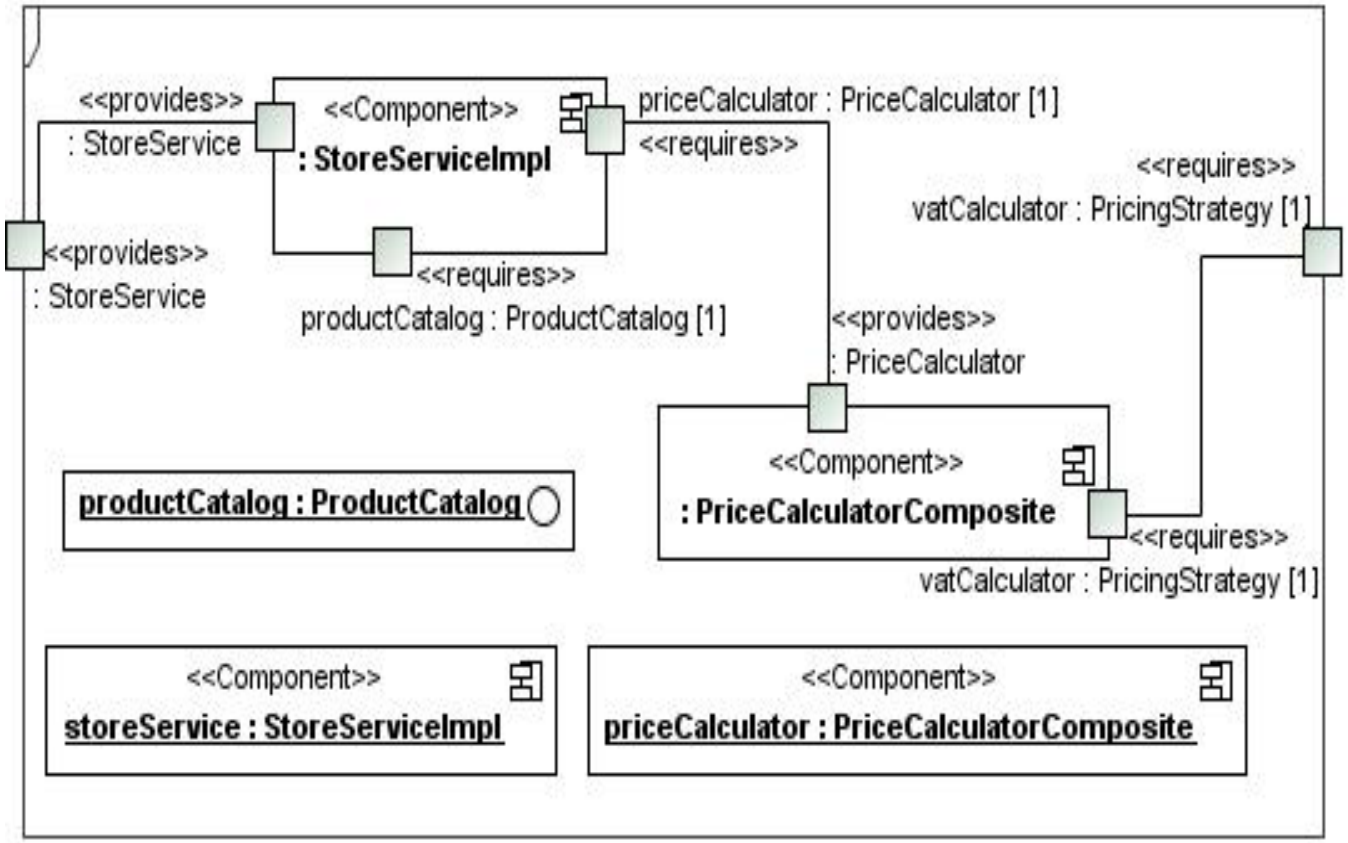
- typically defined by the system architect,
- indicates component instances that will implement the services ⇒ decomposition in simple and composite components.



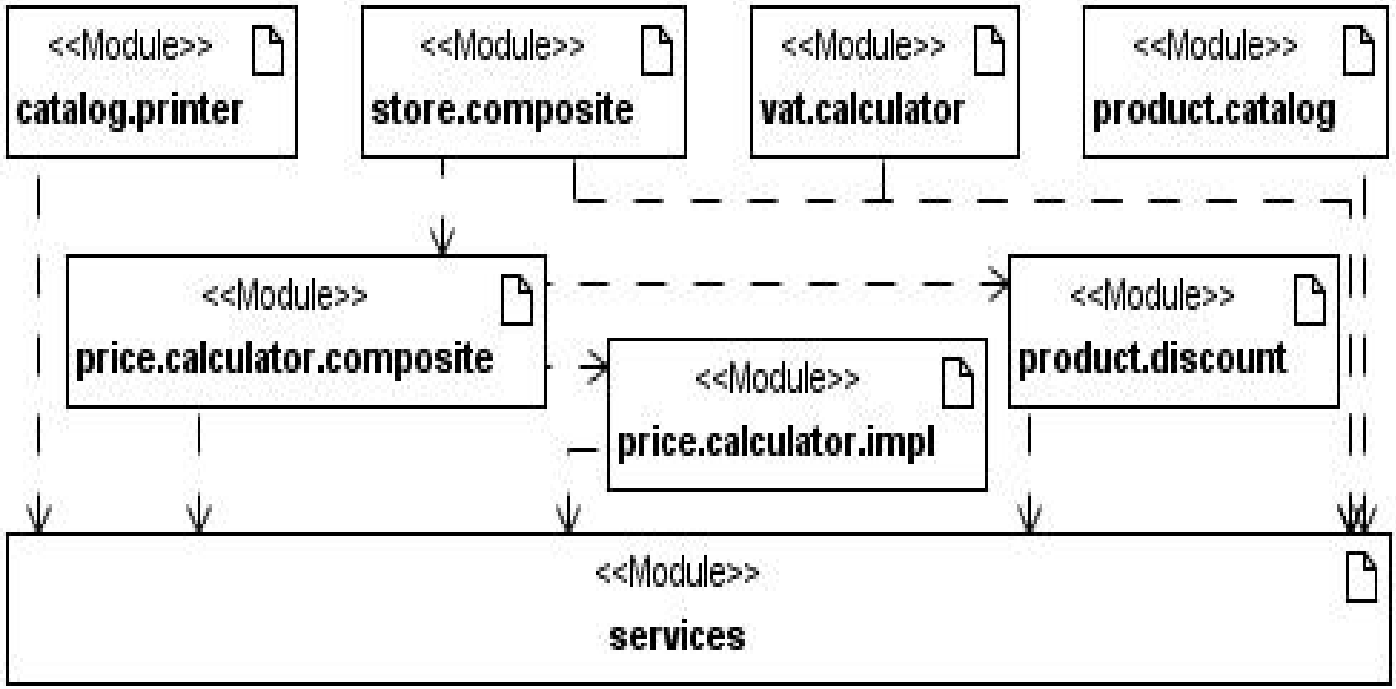
Rules

- the internal structure of a component uses instances of other components, connected through ports;
- Specify the provided and required ports
- To select a certain service implementation satisfying some criteria – attributes:
- ***property*** – provides -to export service properties
- ***filter*** – requires – to filter required services
- InstanceSpecification objects indicate which components should be created

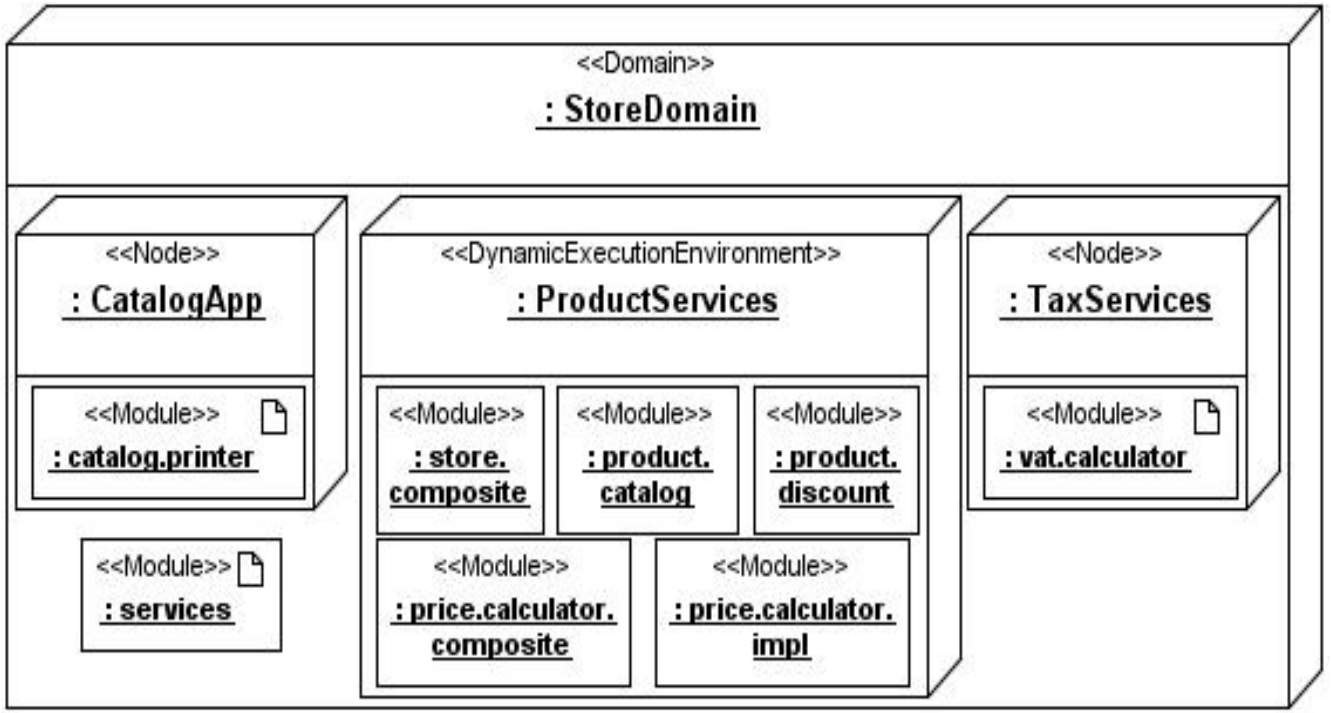
Structural Model and component instances



Deployment Model – Modules dependencies



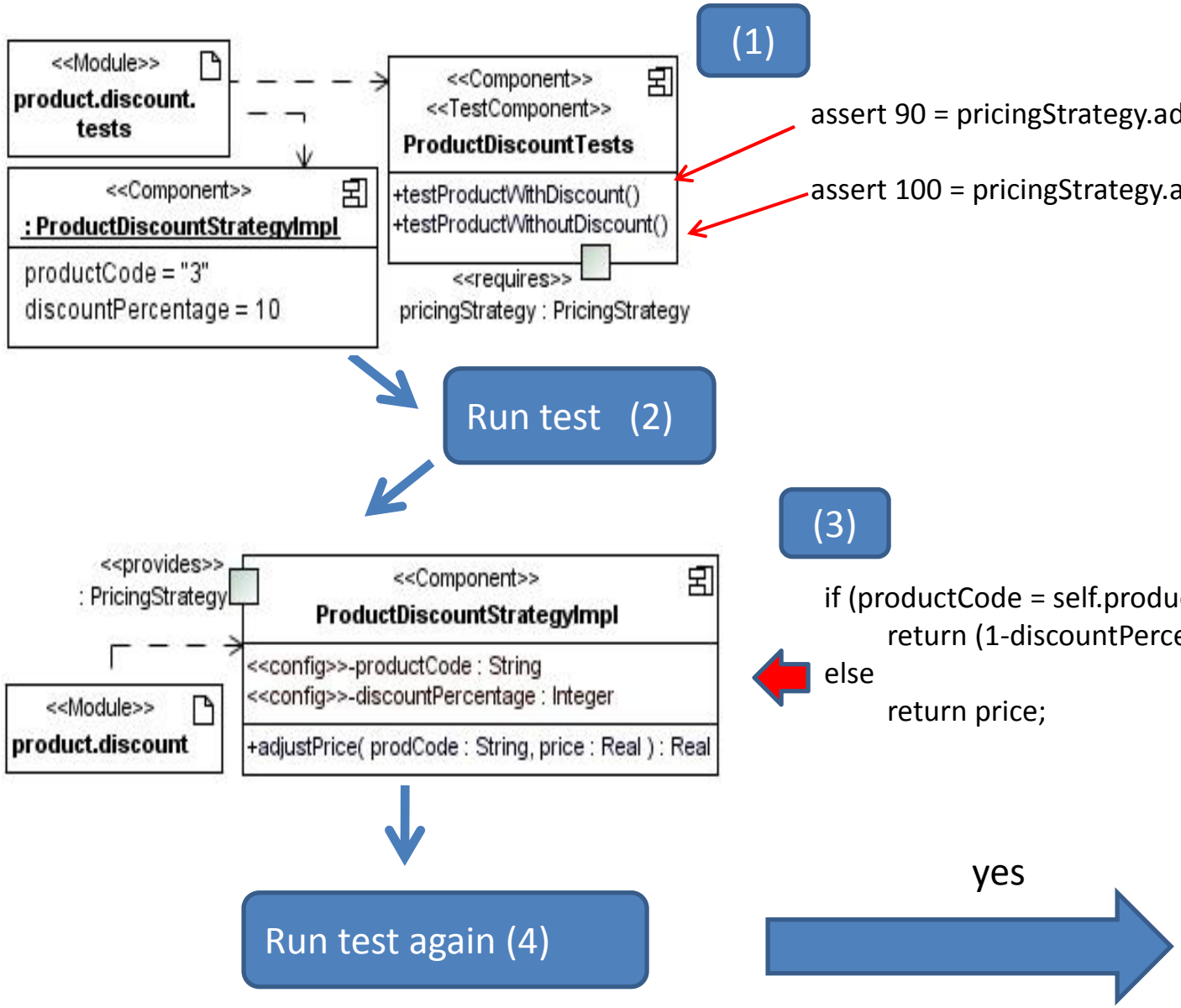
Deployment Model – Domain and nodes



Deployment Model – Component instances

Agile MDA process

1. Add a test
2. Run the test
3. Add production code
4. Run the test


(1)
<<Module>>
product.discount.
tests
<<Component>>
<<TestComponent>>
ProductDiscountTests
+testProductWithDiscount()
+testProductWithoutDiscount()
<<requires>>
pricingStrategy : PricingStrategy
assert 90 = pricingStrategy.adjustPrice(''3'',100);
assert 100 = pricingStrategy.adjustPrice(''1'',100);
<<Component>>
: ProductDiscountStrategyImpl
productCode = "3"
discountPercentage = 10
Run test (2)
(3)
<<provides>>
: PricingStrategy
<<Component>>
ProductDiscountStrategyImpl
<<config>>-productCode : String
<<config>>-discountPercentage : Integer
+adjustPrice(prodCode : String, price : Real) : Real
<<Module>>
product.discount
if (productCode = self.productCode)
return (1-discountPercentage/100)* price;
else
return price;
Run test again (4)
yes

Related work

iComponent	iPOJO	SCA
Domain	-	Domain
Node	-	Node
DynamicExecutionEnvironment	OSGi implementation	-
Module	Bundle	Contribution
Component	Component	Component
Composite component	Composite	Composite
provides	provides	Service
requires	requires	Reference
validate and invalidate	validate and invalidate	-
controller	controller	-
config	Property	Property

Conclusions

- Agile MDA development approach for the development of service-oriented components.
- Distributed service architecture
- Separation between business logic and non-functional aspects
- Platform independence
- Rapid development - graphical