Automatic Generation of Platform Independent Built-in Contract Testers

Helton Souza Lima

Franklin Ramalho

Patricia Machado

Everton Alves

Grupo de Métodos Formais - GMF

Universidade Federal de Campina Grande

Brazilian Symposium on Software Components, Architectures and Reuse – SBCARS

Campinas - SP

Introduction
The problem

- Component-Based Development
  - High quality reusable software
  - Reliable components = well tested components
- Model Based Testing
  - Tests are in conformance to the specification
  - Tests are constructed as soon as specifications
- Automatic Model Based Testing
  - Changes on models are instantly reflected on the tests
  - Tests are more reliable – no human interference
- Problem:
  - We need tests fully and automatically generated from models and with a high reliability and reusability

Introduction
The solution

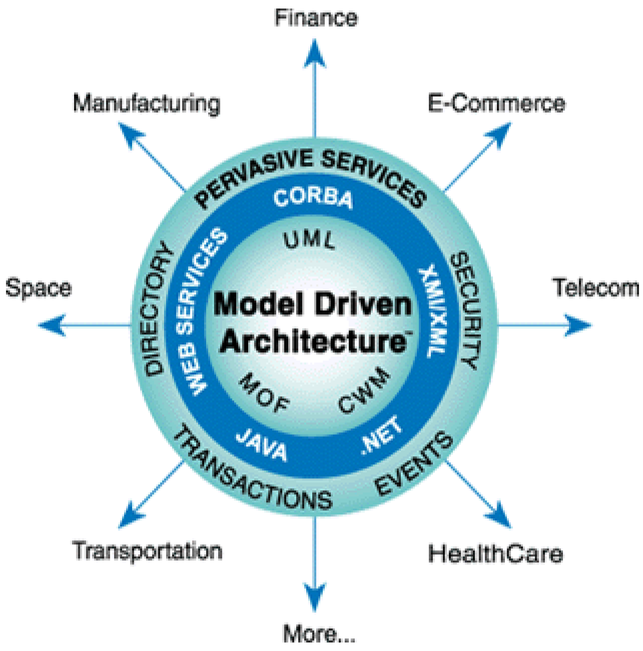
- To achieve these benefits we need:
  - (1) A component-based methodology
  - (2) The methodology should deals with models
  - (3) A model-based testing method inside the methodology
  - (4) An approach able to make the testing method automatic

Our Solution
MoBIT

- **MoBIT (Model-driven Built-In contract Testers)**
  - (1) and (2): The component and model-based methodology: **KobrA**
  - (3): Model-based testing method inside KobrA: **BIT**
  - (4): The approach to build the automatic BIT method: **MDE**

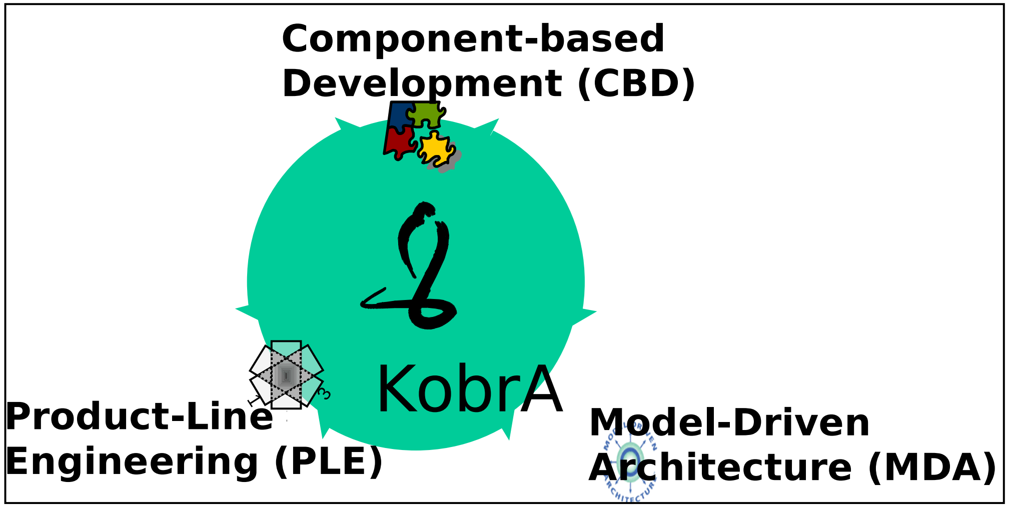
Background
The approach

- **MDE (Model-Driven Engineering)**
  - New approach of software development
  - Focus on models (UML and OCL models)
  - Steps:
    - Build the models in conformance to meta-models
    - Create the transformation rules between pairs of meta-models
    - Use engines to process the transformations among source and target models
  - Benefits:
    - Maximize software reuse (including between different platforms)
    - Speed-up software development



Background
The methodology

- **KobrA**
  - Three foundations:
    - Component-based – modularity
    - Product-line – reuse
    - Model-Driven – UML models
  - Simple
    - Minimal set of concepts
  - Systematic
    - Methodological
  - Prescriptive
    - Step-wise process
  - Flexible
    - Wide range of circumstances
  - Scalable
    - Large and small systems


Component-based Development (CBD)
KobrA
Product-Line Engineering (PLE)
Model-Driven Architecture (MDA)

Background
The testing method

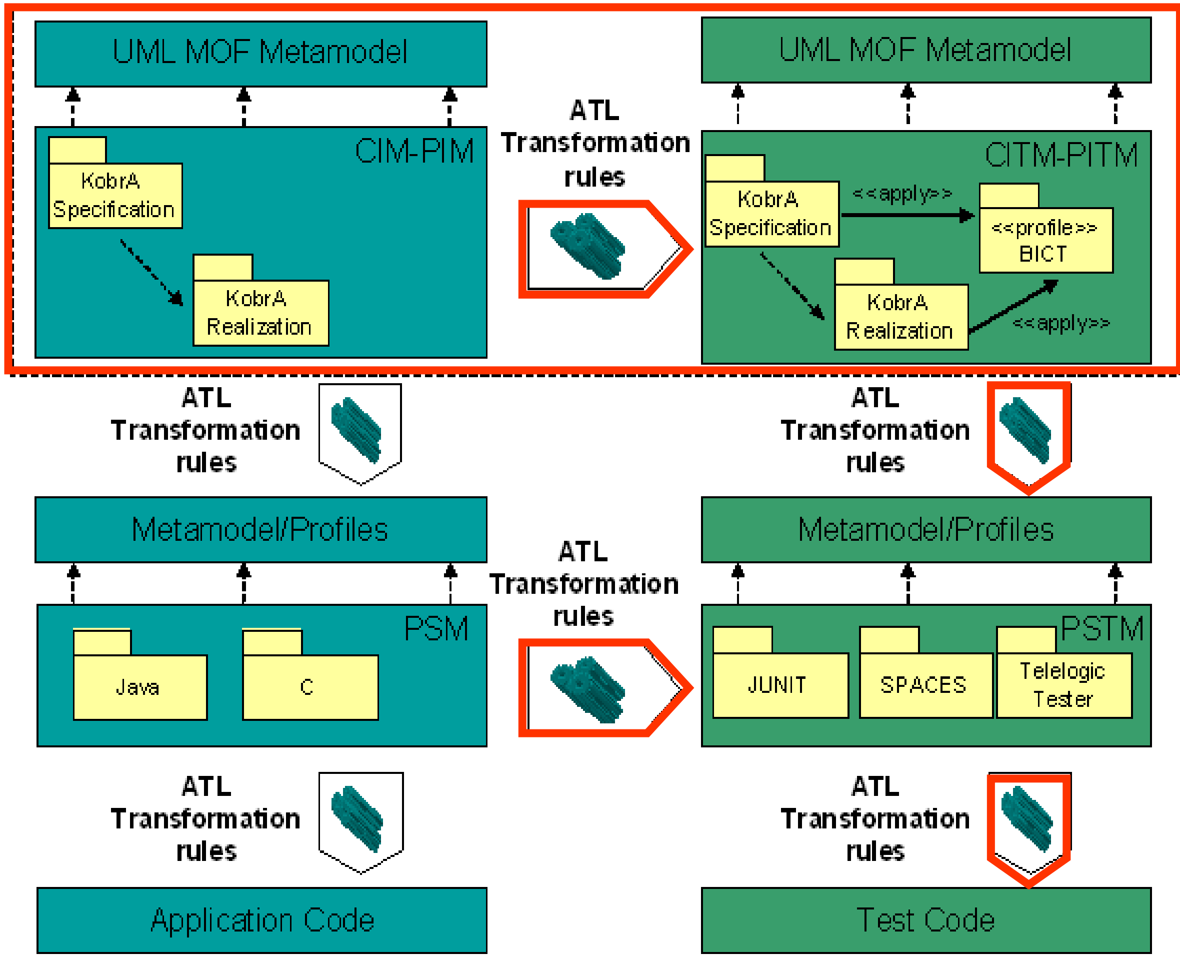
- **BIT (Built-In contract Testing)**
  - Standard testing method inside KobrA methodology
  - Functional testing
    - Based on behavior models defined in KobrA
  - Client-Server integration testing
    - Component-based
  - Step-by-step method
  - Defines a standard structure to perform tests on the server
    - Based on the behavior and interfaces specified in the models
  - Our goal is make this method an automatic step

Our Solution
MoBIT

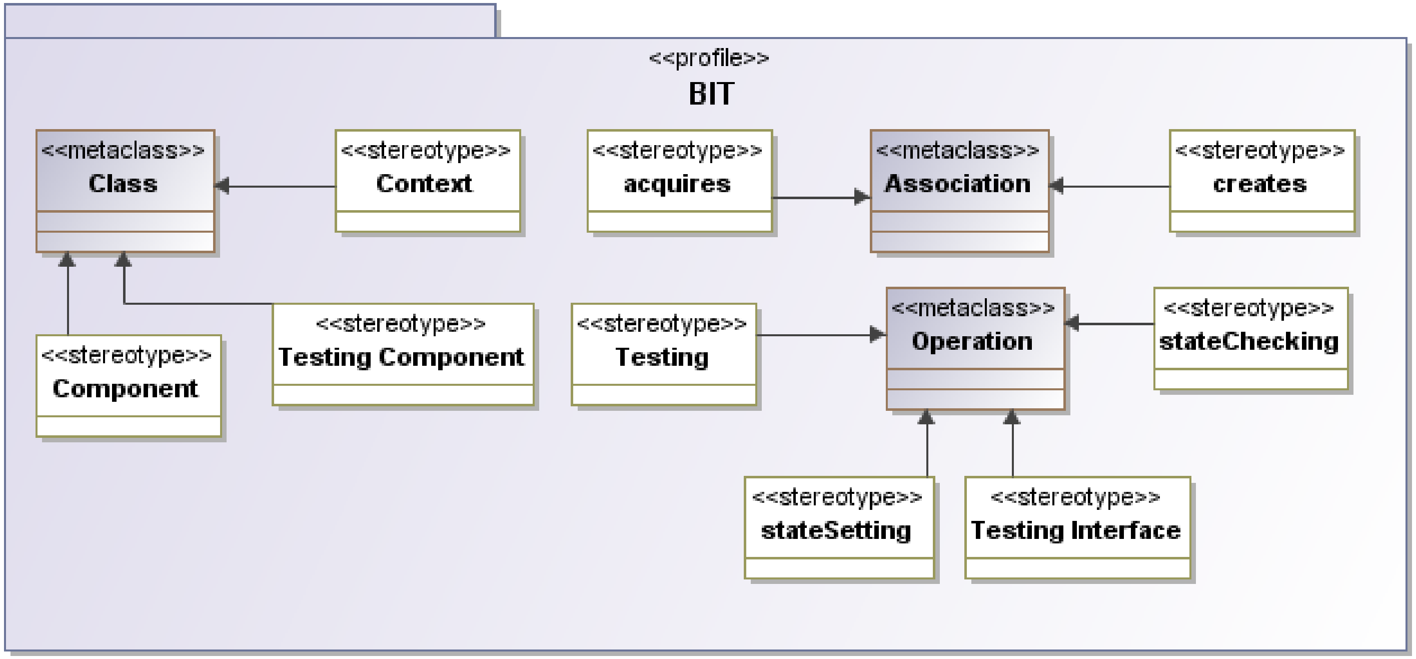
- **MoBIT**
  - Another term: Model-Driven Testing (MDT)
  - Is an automatic MBT approach
  - In conformance to the MDE principles
    - meta-models, models and transformations
  - Tests infrastructure is generated through the transformations
  - After the rules are implemented, there is no cost to generate the tests
  - An architecture was created...

MoBIT Architecture

- Follows the MDE general architecture
- Meta-models and models:
  - Development and Testing
- Horizontal and vertical transformations – refinements
- This work focus on:
  - CIM-PIM to CITM-PITM
  - From structural and behavioral diagrams
  - To structural diagrams refined with BIT concepts

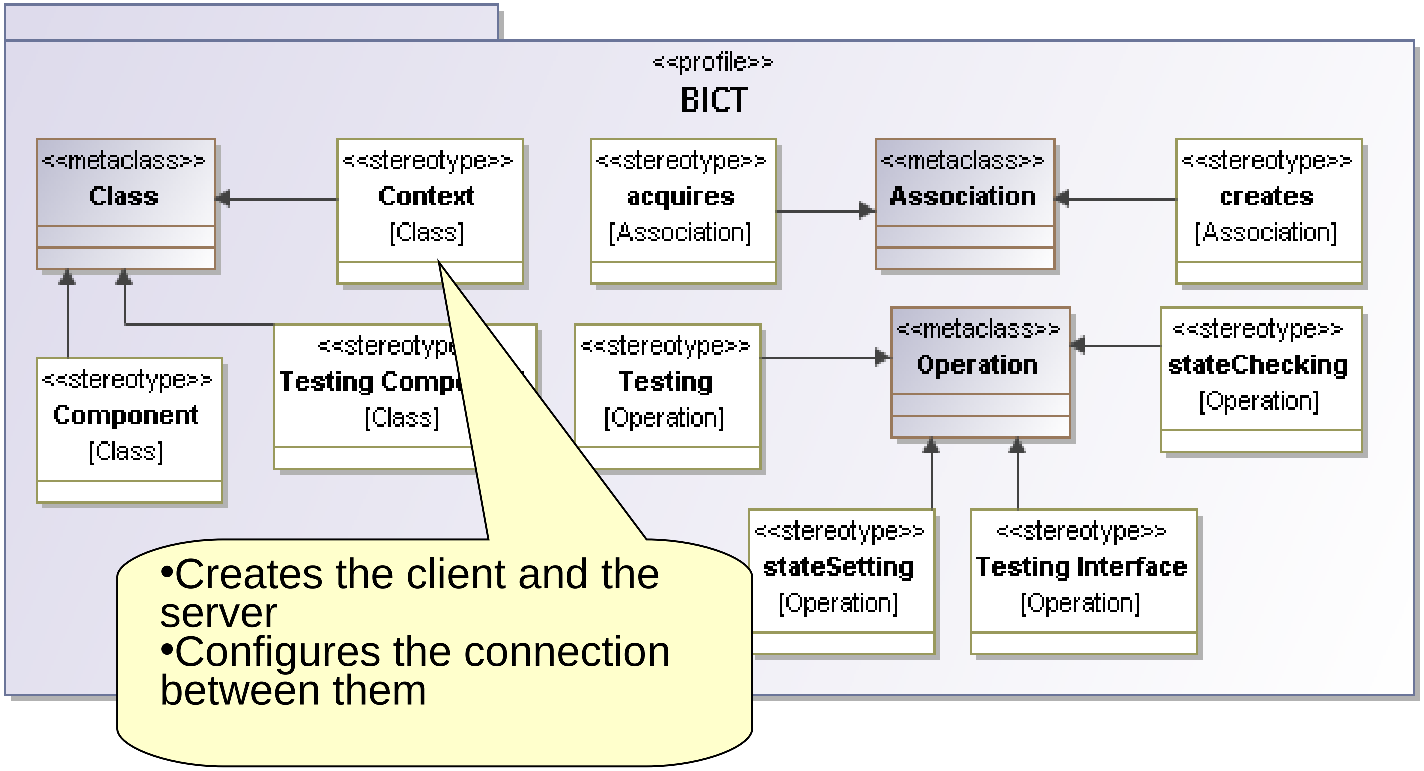


BIT Profile



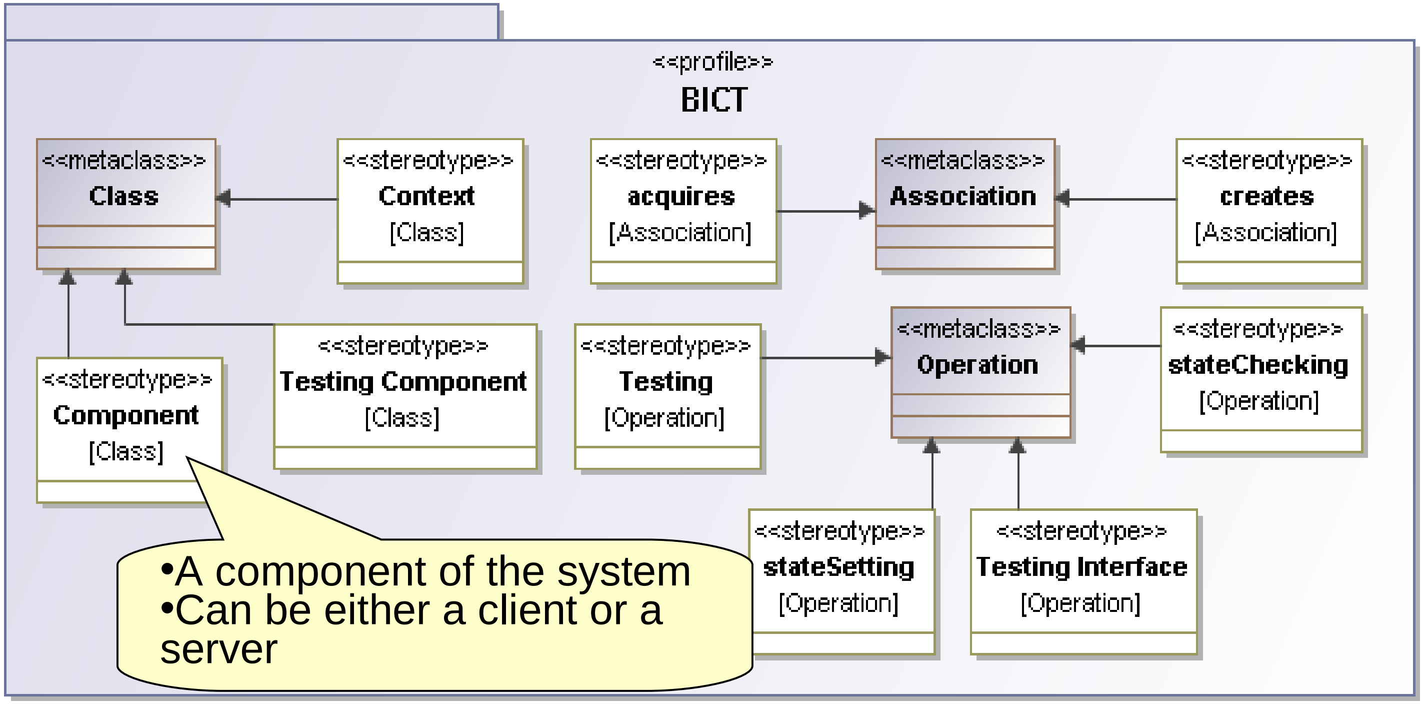
- UML extension mechanism: adds specific semantic to UML elements
- A set of stereotypes
- Each one corresponds to a BIT concept

BIT Profile



- Creates the client and the server
- Configures the connection between them

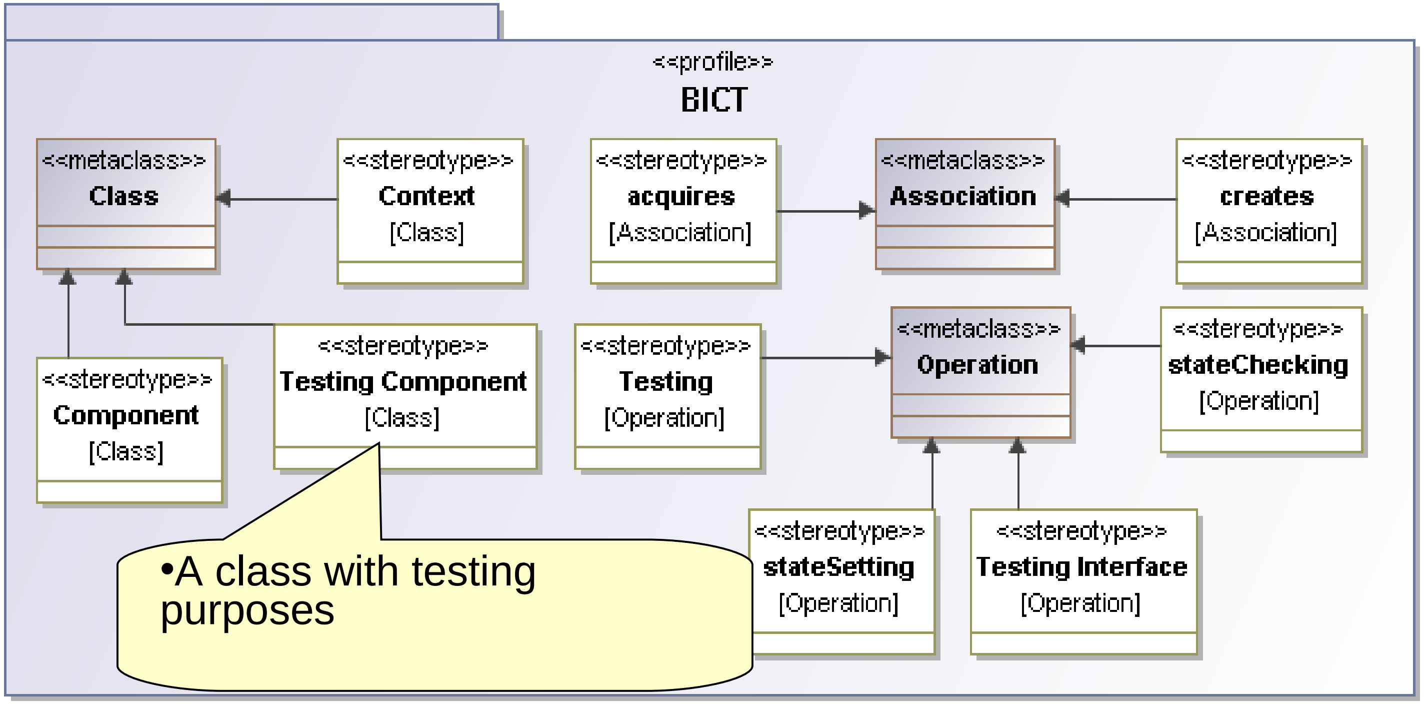
BIT Profile



<<profile>>
BICT
<<metaclass>> Class
<<stereotype>> Context [Class]
<<stereotype>> acquires [Association]
<<metaclass>> Association
<<stereotype>> creates [Association]
<<stereotype>> Component [Class]
<<stereotype>> Testing Component [Class]
<<stereotype>> Testing [Operation]
<<metaclass>> Operation
<<stereotype>> stateChecking [Operation]
<<stereotype>> stateSetting [Operation]
<<stereotype>> Testing Interface [Operation]

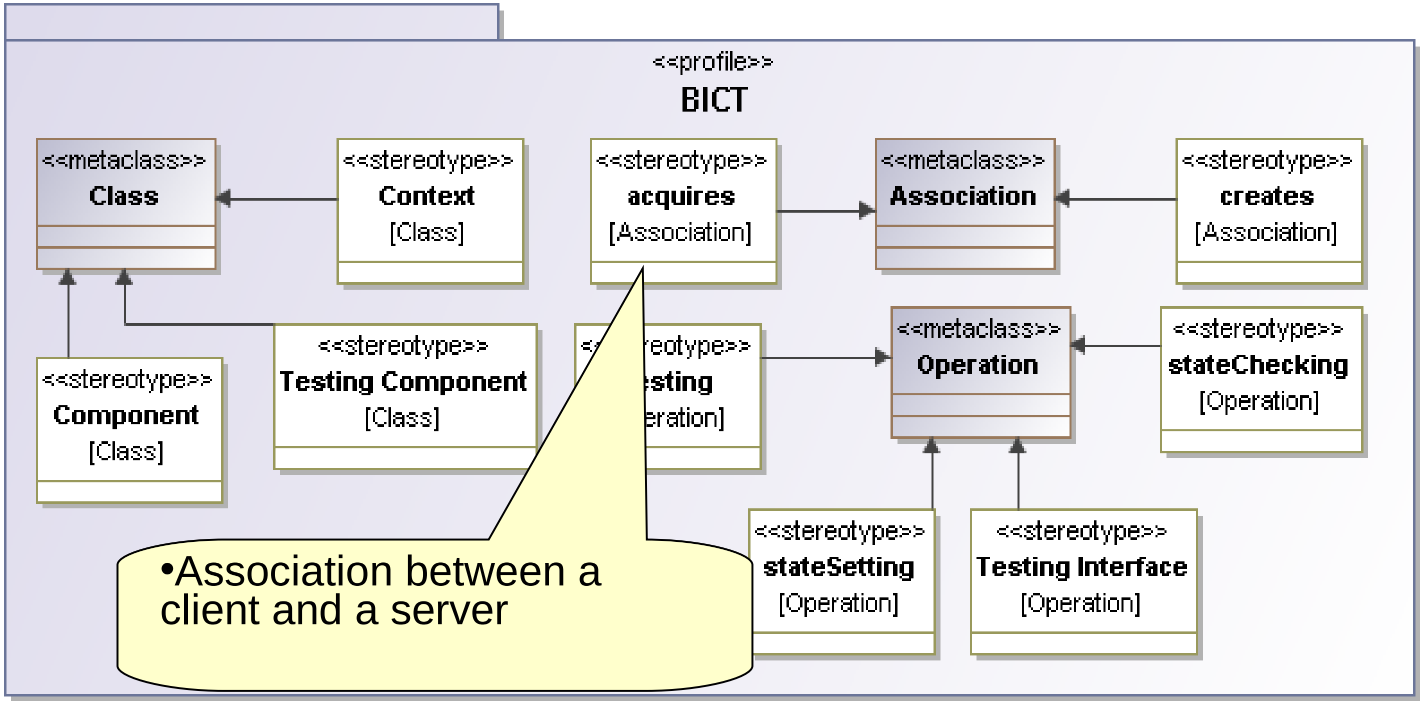
- A component of the system
- Can be either a client or a server

BIT Profile



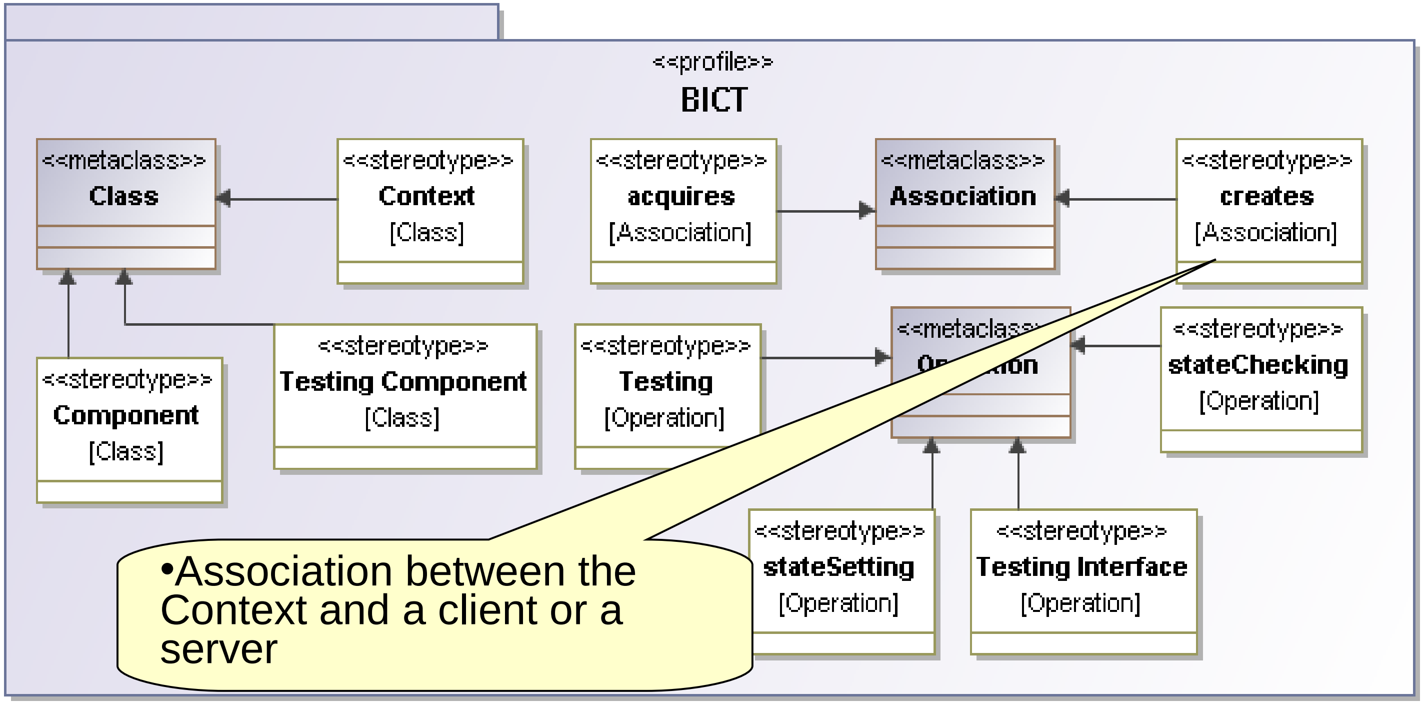
- A class with testing purposes

BIT Profile



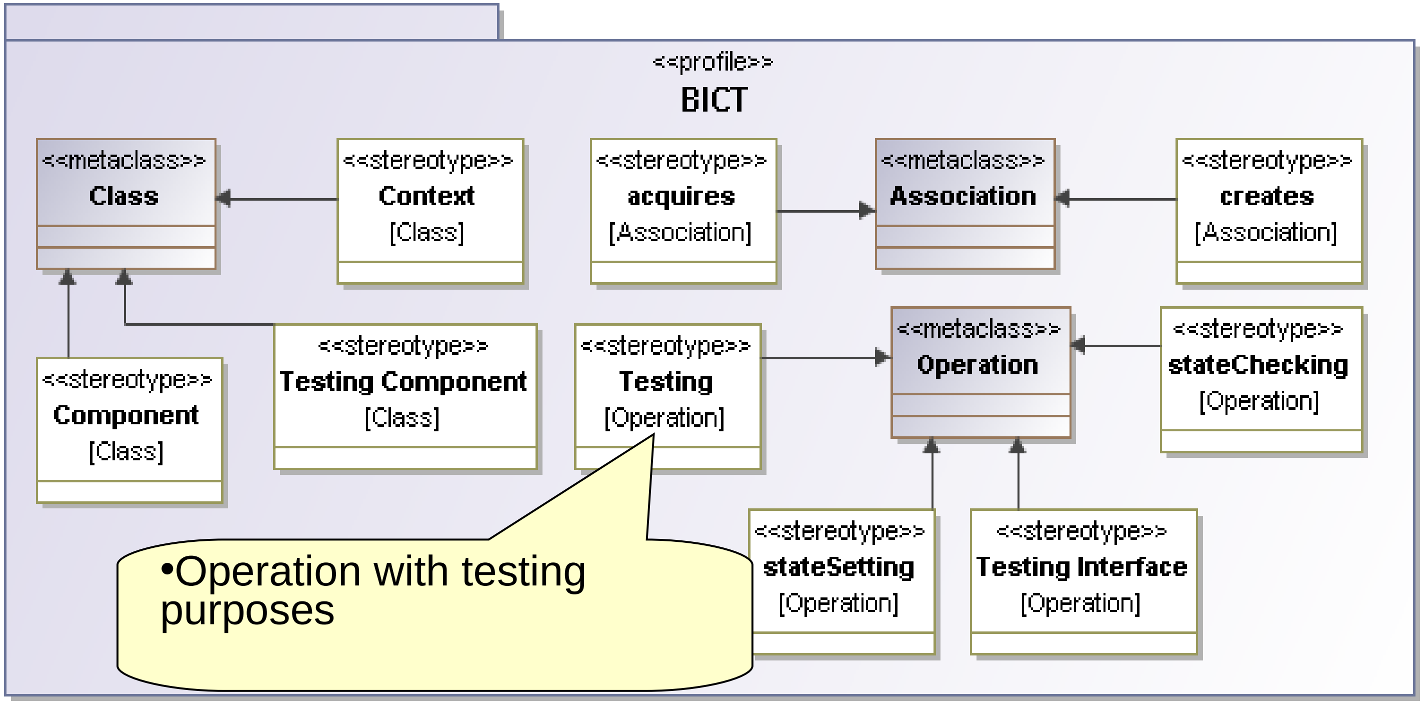
- Association between a client and a server

BIT Profile



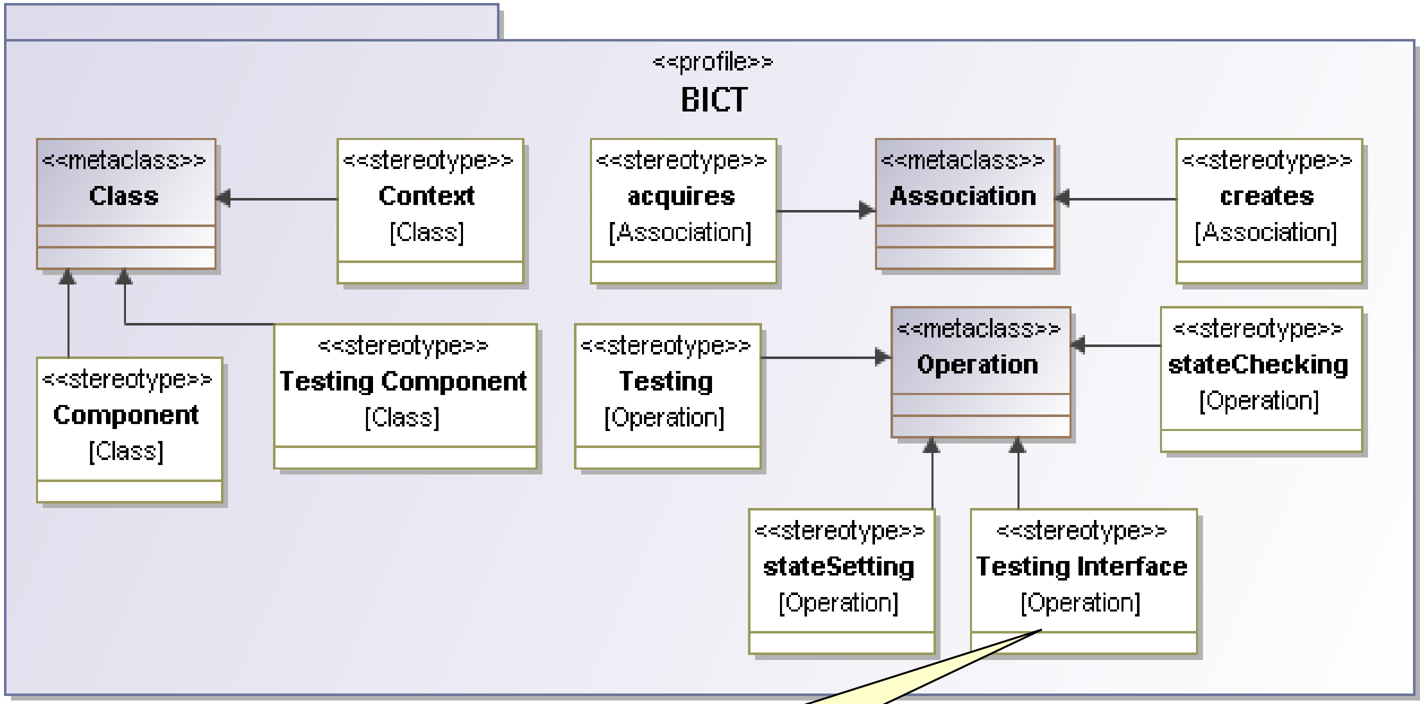
- Association between the Context and a client or a server

BIT Profile


<<profile>>
BICT
<<metaclass>> Class
<<stereotype>> Context [Class]
<<stereotype>> acquires [Association]
<<metaclass>> Association
<<stereotype>> creates [Association]
<<stereotype>> Component [Class]
<<stereotype>> Testing Component [Class]
<<stereotype>> Testing [Operation]
<<metaclass>> Operation
<<stereotype>> stateChecking [Operation]
<<stereotype>> stateSetting [Operation]
<<stereotype>> Testing Interface [Operation]

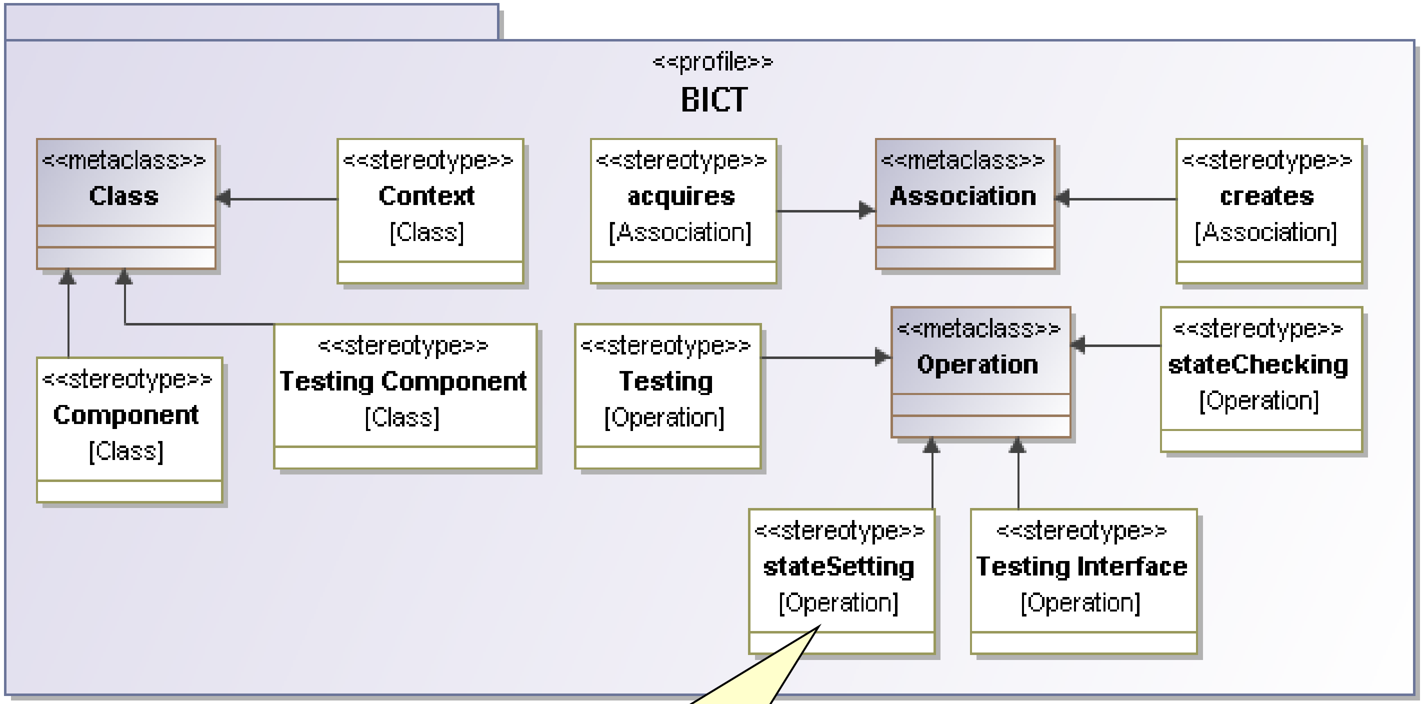
- Operation with testing purposes

BIT Profile



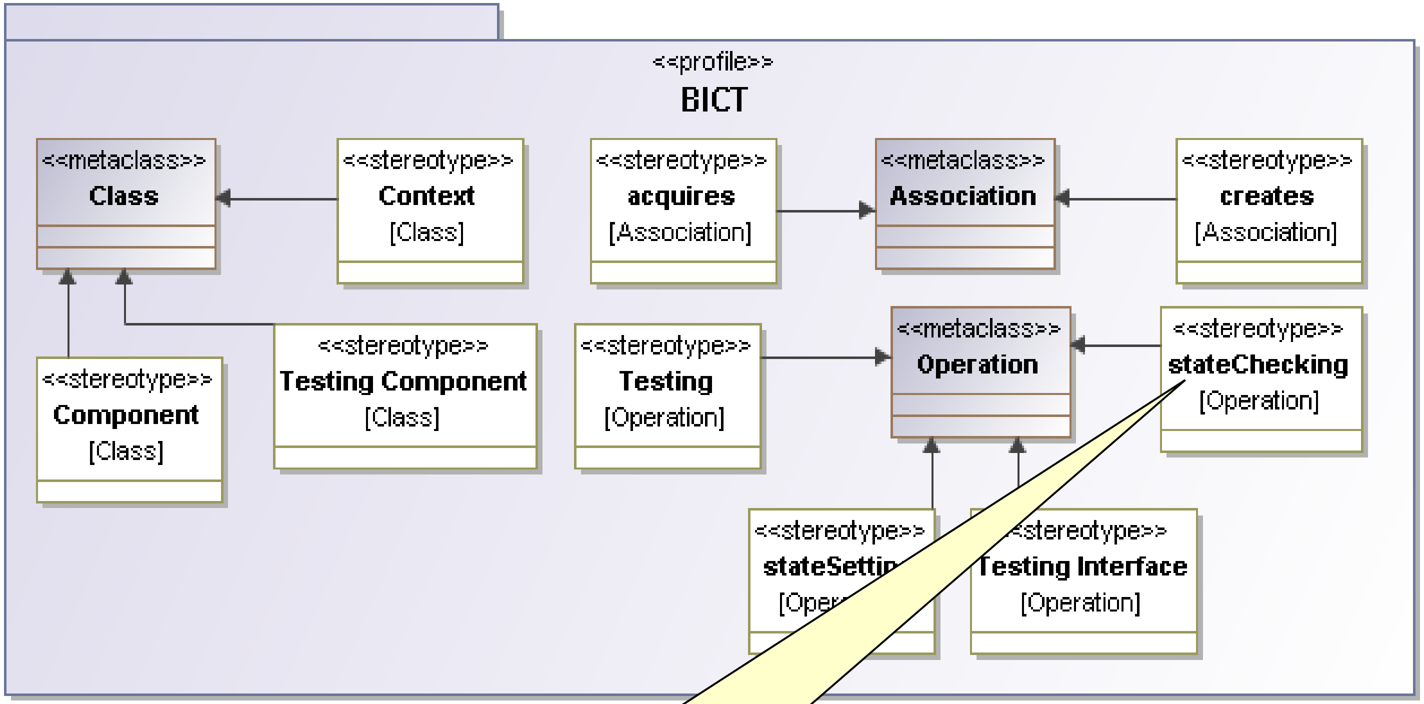
- A set of <<Testing>> operations

BIT Profile



- Sets the server to a specific state

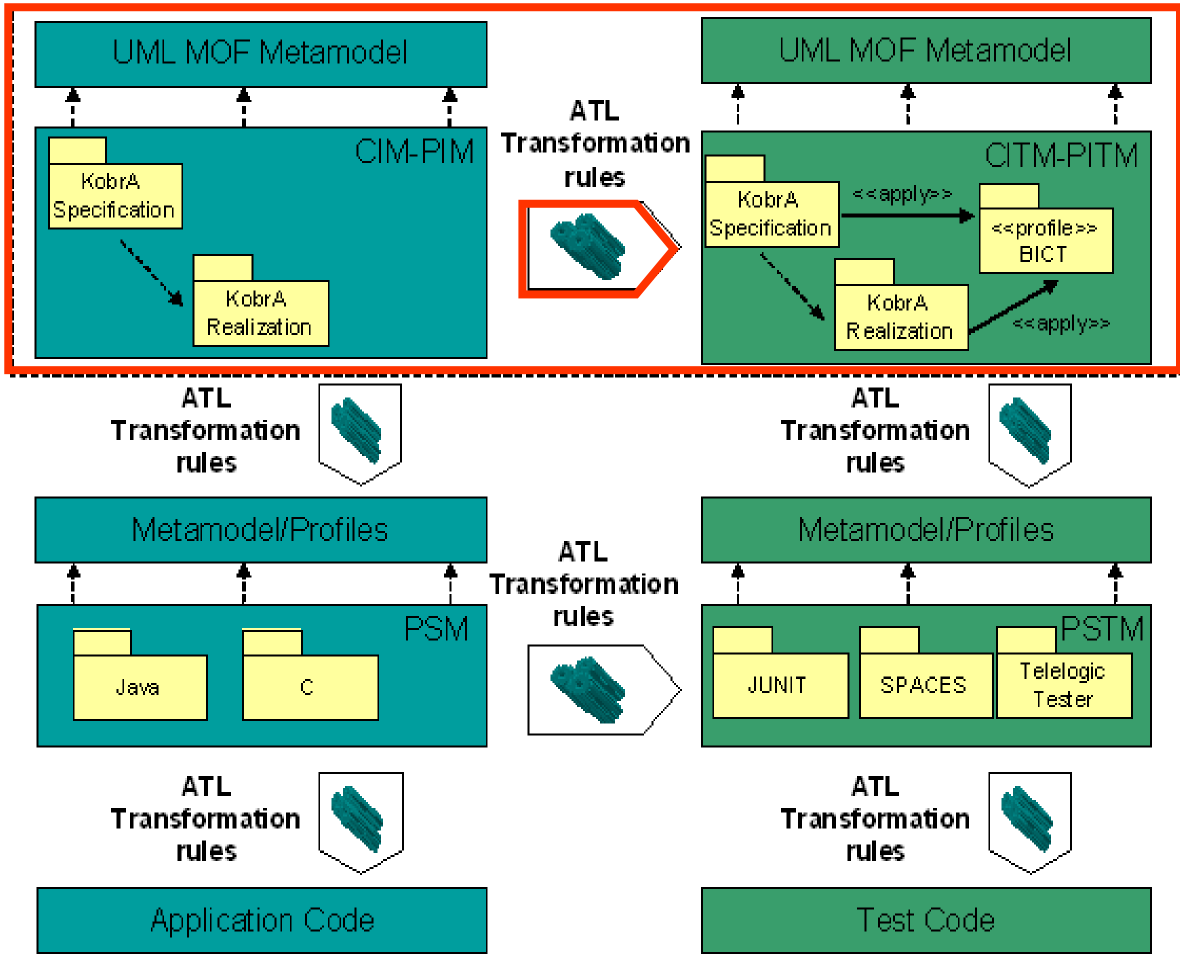
BIT Profile



- Informs if the server is in a specific state

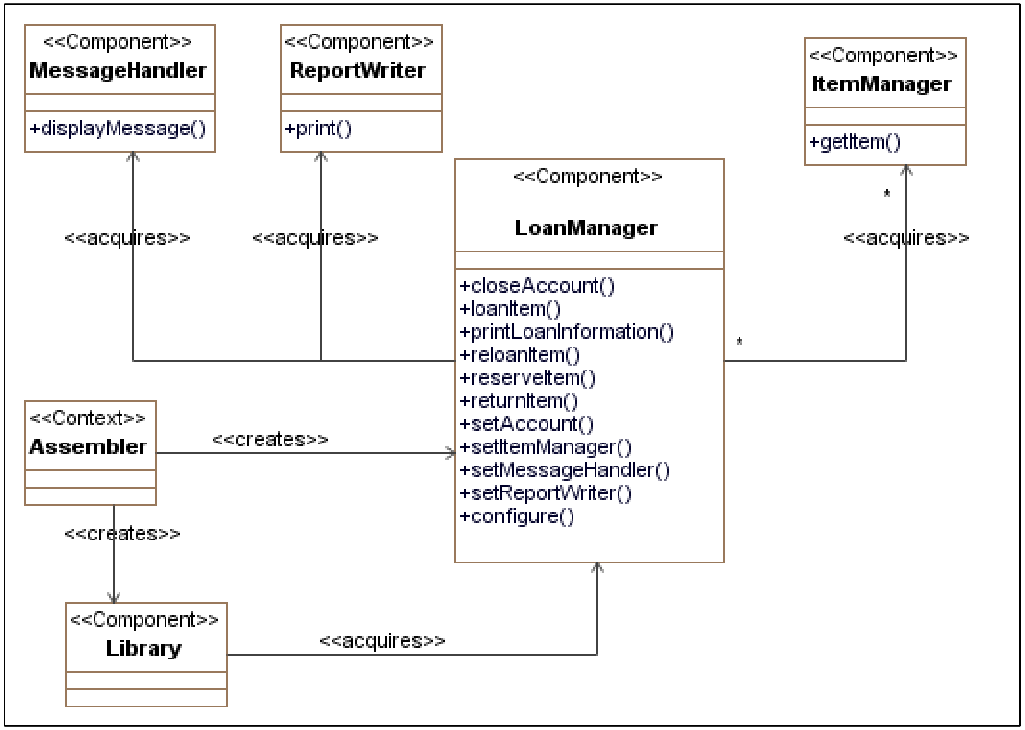
Transformations

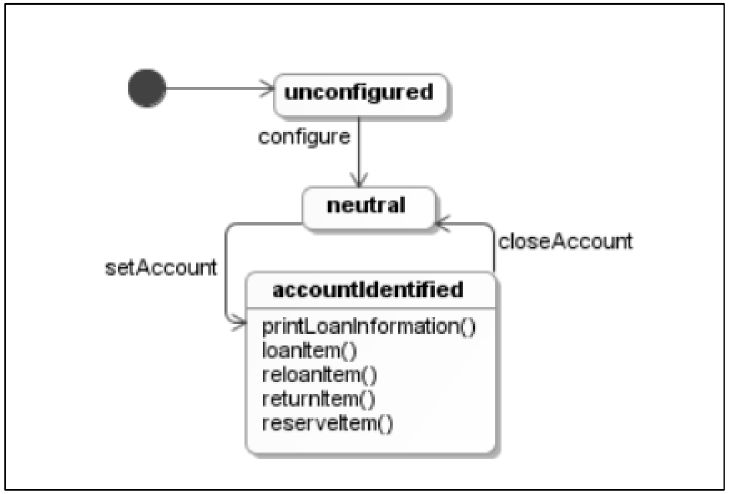
- From CIM-PIM
  - Source meta-model: class diagram and behavioral state machine diagram
- To CITM-PITM
  - Target meta-model: class diagram annotated with the BIT profile
- Reuse UML metamodels
- Transformation Language: ATL
  - Atlas Transformation Language
- Framework: ATL-DT
  - Eclipse integrated
- Two different clauses
  - from
  - to



Case Study

- The Library System



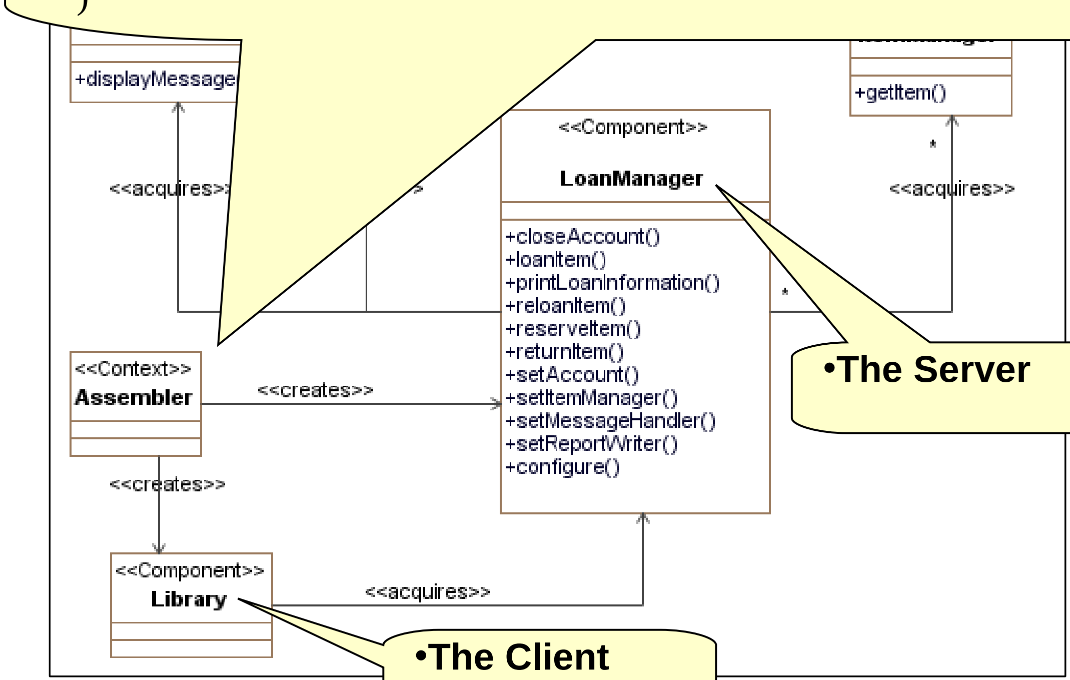


Case Study

- Identifying the source elements (from)

•The <<Context>>

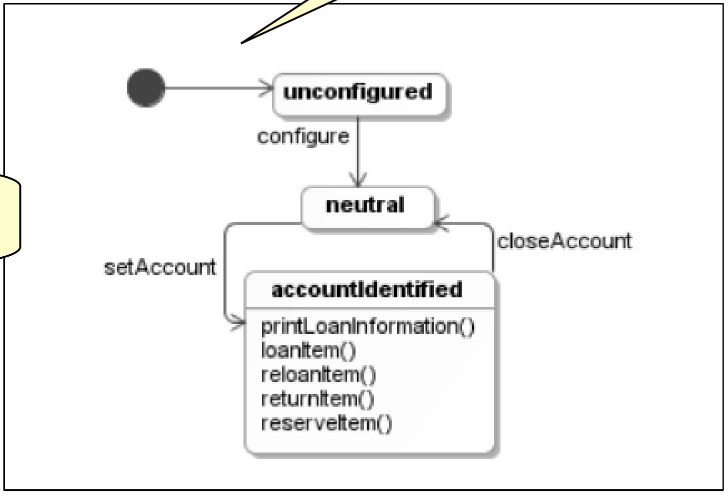
```
from cont:UML2!Class(
        cont.extension->exists(e | e.ownedExtension.stereotype.name = 'Context')
)
```



•The Server

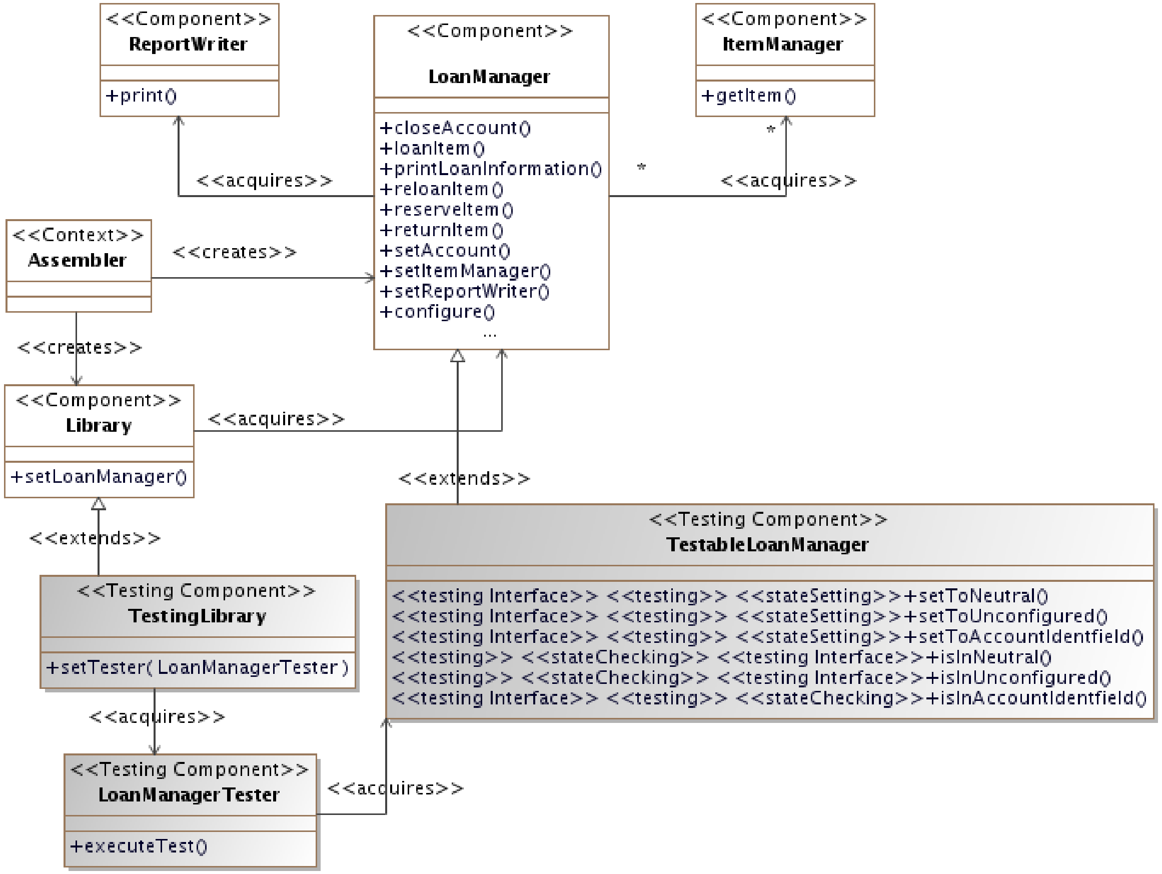
•The Client

•The states of the Server



Case Study

- Generating the target elements

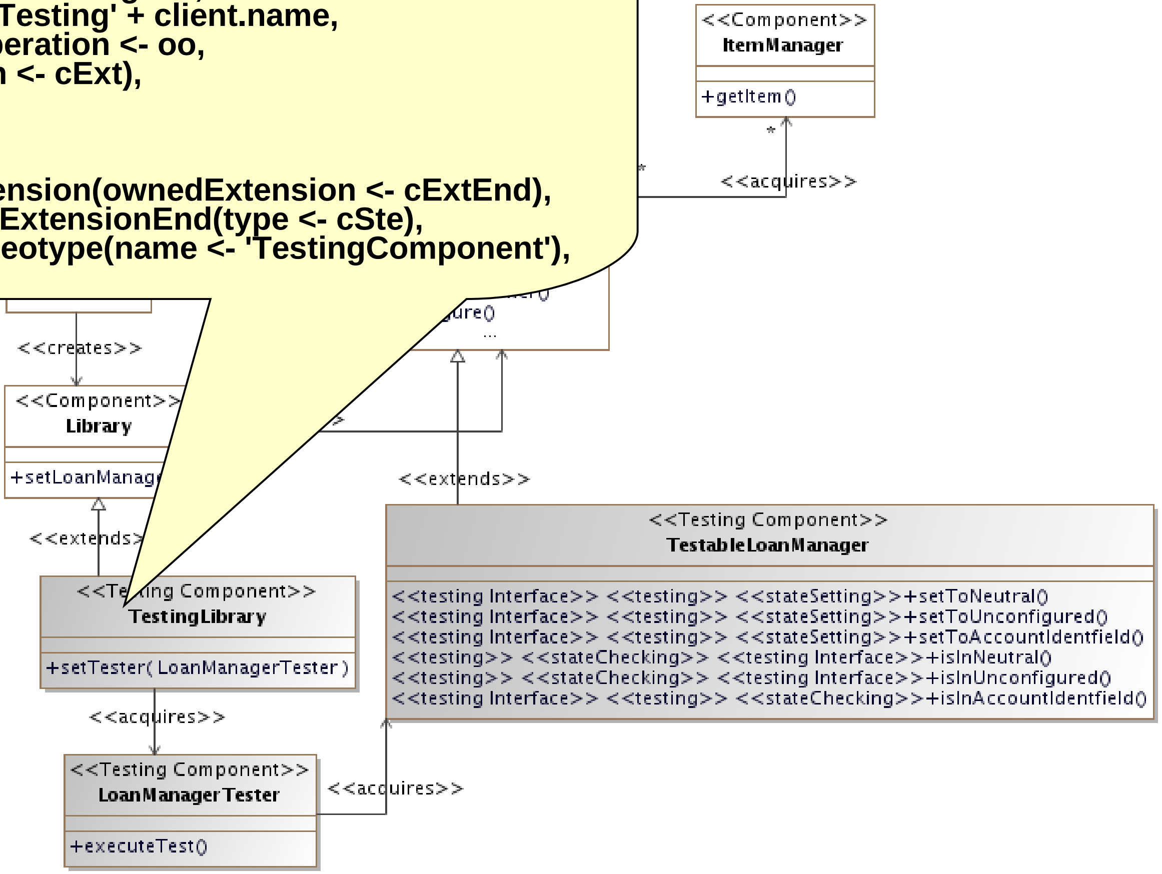


Case Study

- Generating the target elements

```
subc:UML2!Class(
                superClass <- Set{client},
                generalization <- genc,
                name <- 'Testing' + client.name,
                ownedOperation <- oo,
                extension <- cExt),
)
...

        cExt:UML2!Extension(ownedExtension <- cExtEnd),
        cExtEnd:UML2!ExtensionEnd(type <- cSte),
        cSte:UML2!Stereotype(name <- 'TestingComponent'),
```



<<Component>>
ItemManager
+getItem()
*
*
<<acquires>>
ure()
...
<<creates>>
<<Component>>
Library
+setLoanManag
<<extends>>
<<Te ing Component>>
TestingLibrary
+setTester(LoanManagerTester)
<<acquires>>
<<Testing Component>>
LoanManagerTester
+executeTest()
<<acquires>>
<<extends>>
<<Testing Component>>
TestableLoanManager
<<testing Interface>> <<testing>> <<stateSetting>>+setToNeutral()
<<testing Interface>> <<testing>> <<stateSetting>>+setToUnconfigured()
<<testing Interface>> <<testing>> <<stateSetting>>+setToAccountIdentfield()
<<testing>> <<stateChecking>> <<testing Interface>>+isInNeutral()
<<testing>> <<stateChecking>> <<testing Interface>>+isInUnconfigured()
<<testing Interface>> <<testing>> <<stateChecking>>+isInAccountIdentfield()

Concluding Remarks

- Results
  - A profile for the BIT method
  - A tool with ATL rules:
    - From pure KobrA structural and behavioral diagrams
    - To KobrA structural diagrams fully annotated with BIT concepts
    - Implemented as an Eclipse plugin
    - Integration with other tools through XMI

Concluding Remarks

- Discussion
  - MoBIT realizes MDT
  - Flexible approach and architecture
    - Although KobrA was choosen, another methodology can be incorporated
  - Future work
    - Other UML diagrams and OCL expressions
    - Other vertical and horizontal transformations