

# UML-based Integration Testing for Component-based Software

Ye Wu and Jeff Offutt

Information and Software Engineering  
Department  
George Mason University

Mei-Hwa Chen

Computer Science Department  
SUNY Albany

## Outline

- Introduction
- Problems and objective
- Methodology
- Conclusion

# Component

- *A component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.[Szyperski ]*
- *An independently deliverable piece of functionality providing access to its services through interfaces. [Alan Brown]:*

# Component-based software

A Component-based system (CBS) is an integration of software components which conform to component-based software engineering principles.

Three key elements:

- Software components. (COTS, or In-house developed software components)
- Component model.(COM+, EJB, CORBA)
- A process and architectures that support component-based development.

## Characteristics of CBS

- Source code availability (COTS components)
- Heterogeneity
- Evolvability
- ... ..

## Problems

When combining components, how do we assure the quality of the integrated system?

- Black-box testing
- White-box testing
  - Source code availability?
  - Heterogeneity?
  - Solution? UML(Unified Modeling Language)

## Why UML?

- Implementation transparency – UML provides high-level information that characterize component internal behavior
- Heterogeneity and Availability – UML has emerged as industry standard for software modeling notations
- Evolvability – Extensibility
- Feasibility – Provides different levels of capacity and accuracy for component modeling

## Test Model for CBS

- Interface
- Event
- Context dependent relationships
- Content dependent relationships

## Component Behaviors

- Stateless behavior
- Stateful behavior

## Simple(Stateless) Behavior

Component behavior only depends on current client invocation.

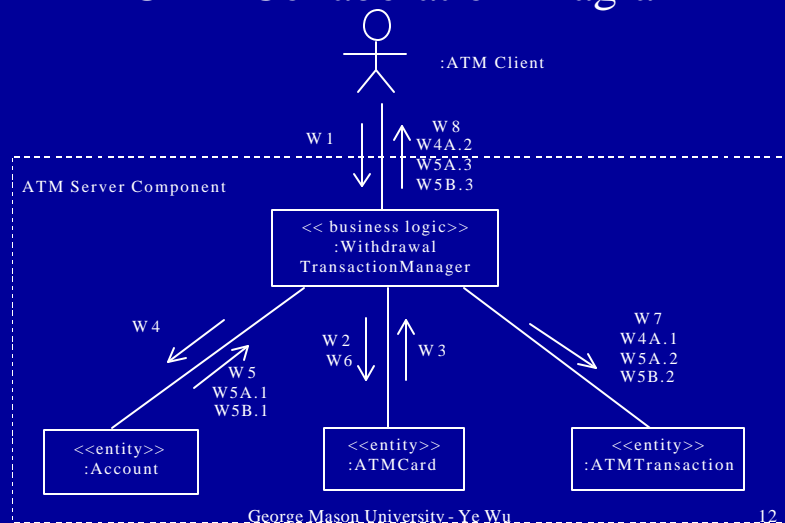
For example, a data management component, which query a database with various criteria, may expose a simple behavior.

# Stateful Behavior

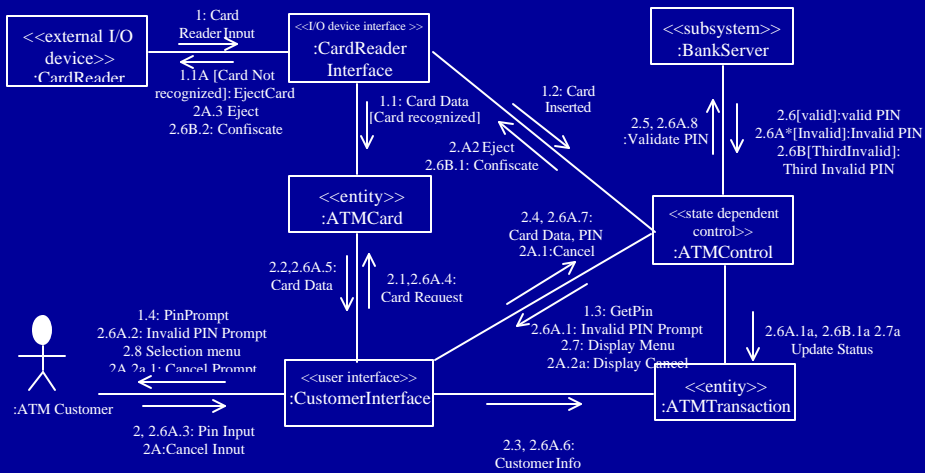
Component behavior depends on the history of client invocations.

For instance, a ATM server component, a withdraw transaction may be affected by previous transactions.

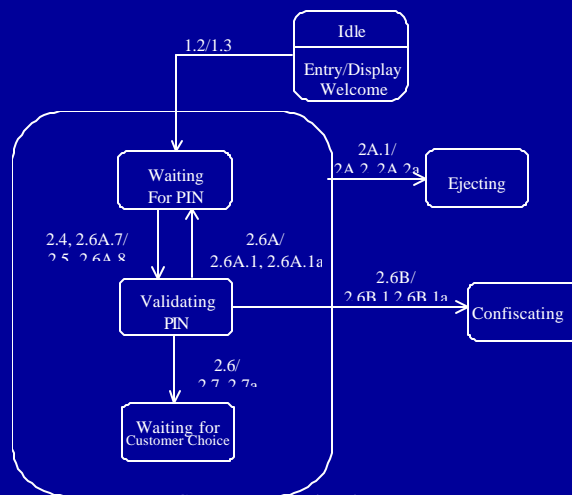
## Context Dependence Relationship — UML Collaboration Diagram



# Is collaboration diagram good enough?



# Context Dependence Relationship — Statechart Diagram



# Analysis of Content Dependence Relationships

- Program analysis based approach
- User specification based approach
- Collaboration diagram based approach
- Statechart diagram based approach

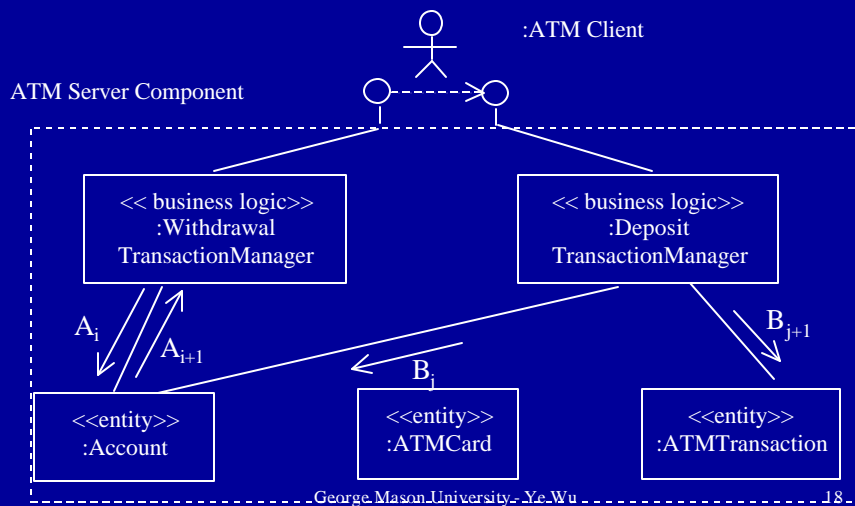
## Program Analysis Based Approach

- An operation  $o$  *uses* a variable  $x$  means that the value of the variable  $x$  is referenced in an expression or is used to decide a predicate in  $o$ .
- An operation  $o$  *defines* a variable means that the value of the variable is assigned when the operation is invoked.
- Operations  $o1$  depends on  $o2$  if and only if :  $o1$  uses a variable  $x$  that is defined in  $o2$ .

# User Specification Based Approach

- Component diagram
- Class diagram
- Naming convention, such as getXXX and setXXX.

# Collaboration Diagram Based Approach



## Statechart Diagram Based Approach

$o1$  depends on  $o2$ , if they do not fall into the following categories:

- If after the execution of  $o1$ , the state of the component remains in its original state, then  $o1$  does not depend on  $o2$ .
- If after the execution of  $o1$ , the state of the component is  $S$ , but from its original state, when executing  $o1$ , it can directly jump to state  $S$ ; this means the state transformation is not caused by the dependence relationships.

## Test Adequacy Criteria

- Test all interfaces and operations.
- Test all events in collaboration diagrams.
- Test all paths in collaboration diagrams.
- Test all states and transitions in statechart diagrams.
- Test all paths in statechart diagrams.
- Test all interface dependence relationships.

## Conclusion

- Test model for testing CBS
- Provide a way to overcome the difficulty introduced by implementation transparency
- Provide a guideline of component deliverables for component providers

## Contact

Ye Wu

Department of Information and Software  
Engineering

George Mason University  
Fairfax, VA, 22030, USA

Email – [wuye@gmu.edu](mailto:wuye@gmu.edu)