Building Test Harness from Service-based Component Models

MoDeVVa’13

Pascal André, Jean-Marie Mottu, and Gilles Ardourel

AeLoS Team, University of Nantes, France

Building Test Harness from Service-based Component Models

Outline

- **Context**: Service-based component models
- **Motivation**: Testing at the model level
- **Test Harness**: Used to build and run the tests
- **Problematic**: Issues on testing component assemblies
- **Contribution**: Test harness construction assistance

Context

- Model Driven Engineering
- We develop Service-based Component Models
- Components are rich, embedding behavior and communications

- We already apply formal verification on them

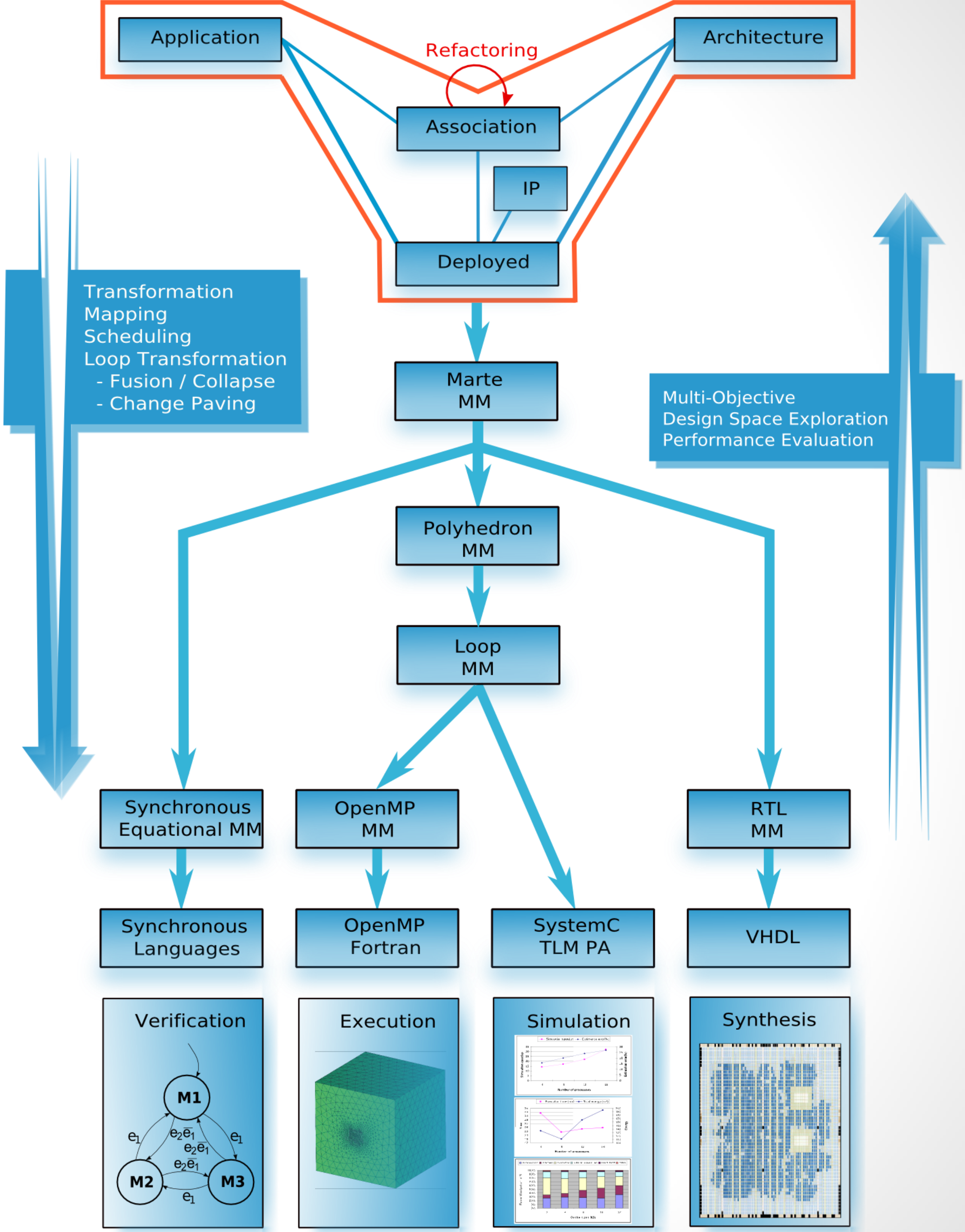
Motivation

- We would like to test at the model level:
  - Finding bugs as soon as possible.
  - Independent from the deployment platforms.
    - Limit the complexity.
    - At the code level, generated code
      - is dependent from the platform,
      - based on and dependent from a framework.

- At the code level, tests should consider too many elements.
- At the model level, bugs can already be corrected and are not disturbed by any platform consideration.

Motivation

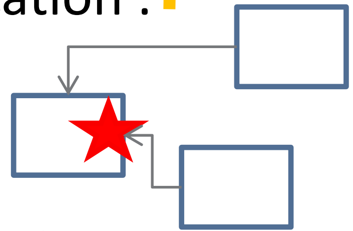
- Gaspard2 transformation chain leading to several platform dependent implementations



Motivation

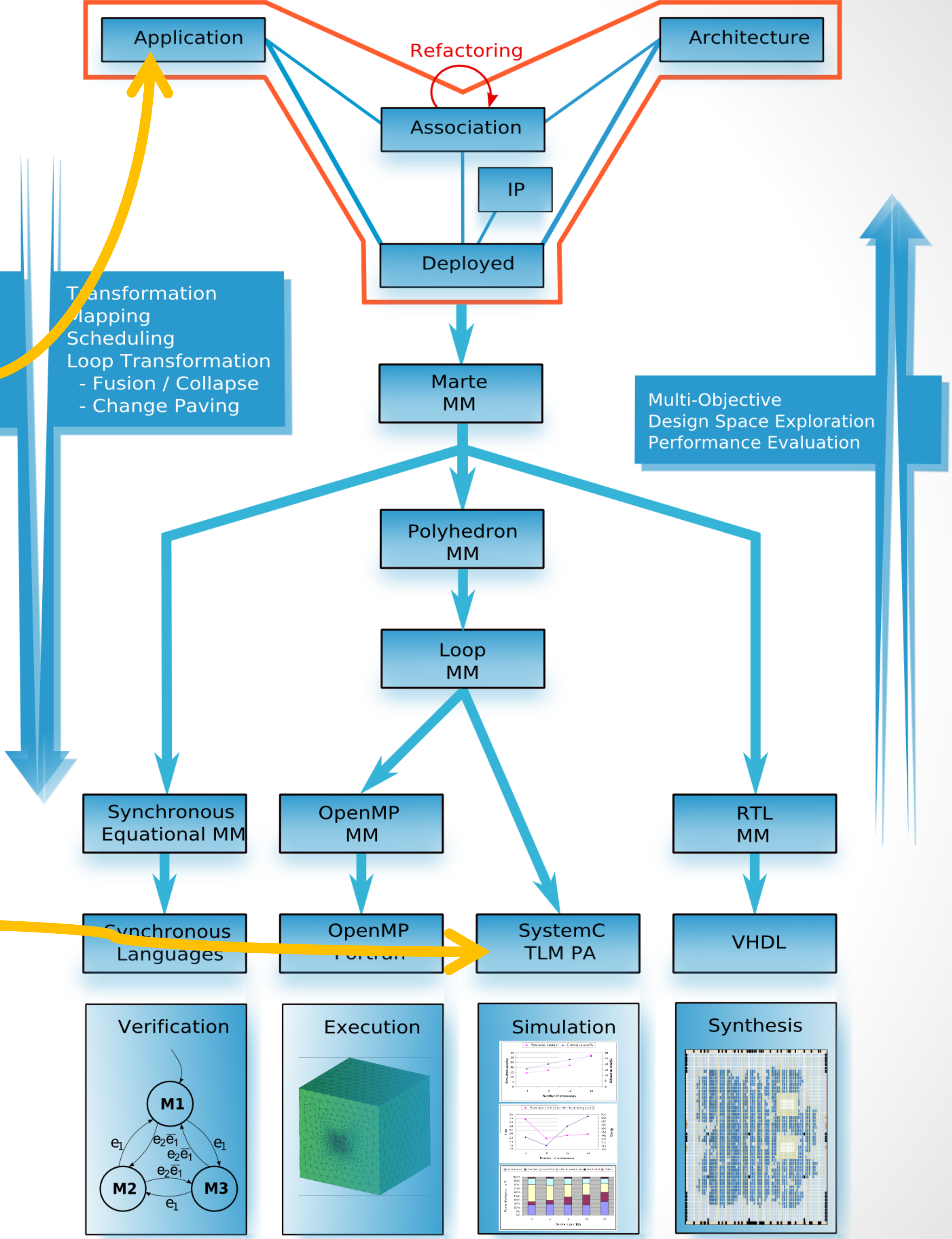
- Gaspard2 transformation chain leading to several platform dependent Implementations

Application :



SystemC :

CodeCodeCodeCodeCode
CodeCodeCodeCodeCode
CodeCodeCodeCodeCode
CodeCodeCodeCodeCode
CodeCodeCodeCodeCode
CodeCodeCodeCodeCode



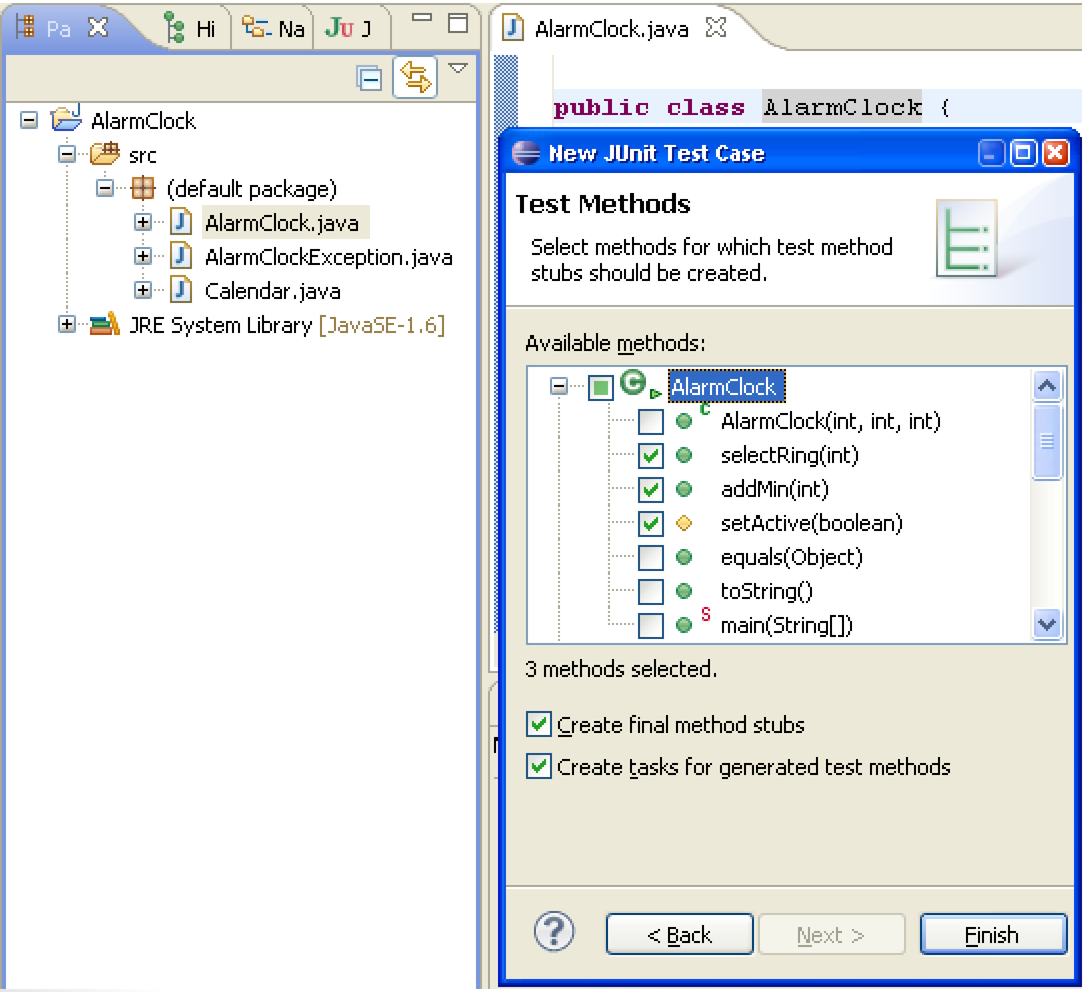
Test Harness

- Test harnesses are used
  - to provide test data,
  - to run the test,
  - to get the verdict (fail or pass).

- Additionally they are used
  - to initialize the System Under Test,
  - to configure it to be in the targeted state.

Test Harness

- Using Eclipse, a Junit test harness is partially generated.



- AlarmClock
  - src
    - (default package)
      - AlarmClock.java
      - AlarmClockException.java
      - Calendar.java
  - JRE System Library [JavaSE-1.6]

AlarmClock.java

```
public class AlarmClock {
```

New JUnit Test Case

Test Methods

Select methods for which test method stubs should be created.

Available methods:

- ■ AlarmClock
  - □ AlarmClock(int, int, int)
  - ☑ selectRing(int)
  - ☑ addMin(int)
  - ☑ setActive(boolean)
  - □ equals(Object)
  - □ toString()
  - □ main(String[])

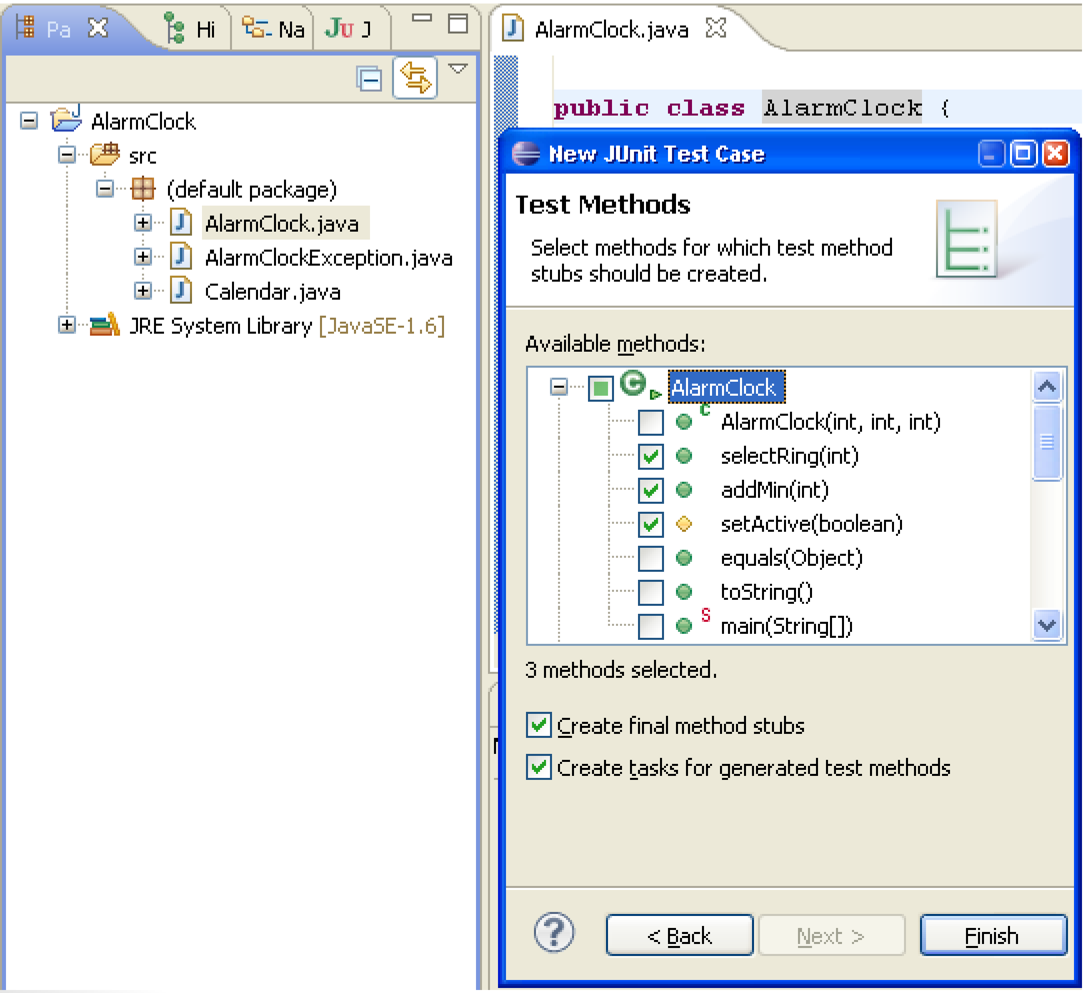
3 methods selected.

☑ Create final method stubs

☑ Create tasks for generated test methods

Test Harness

- Using Eclipse, a Junit test harness is partially generated.



```
public class AlarmClock {
```

New JUnit Test Case

Test Methods

Select methods for which test method stubs should be created.

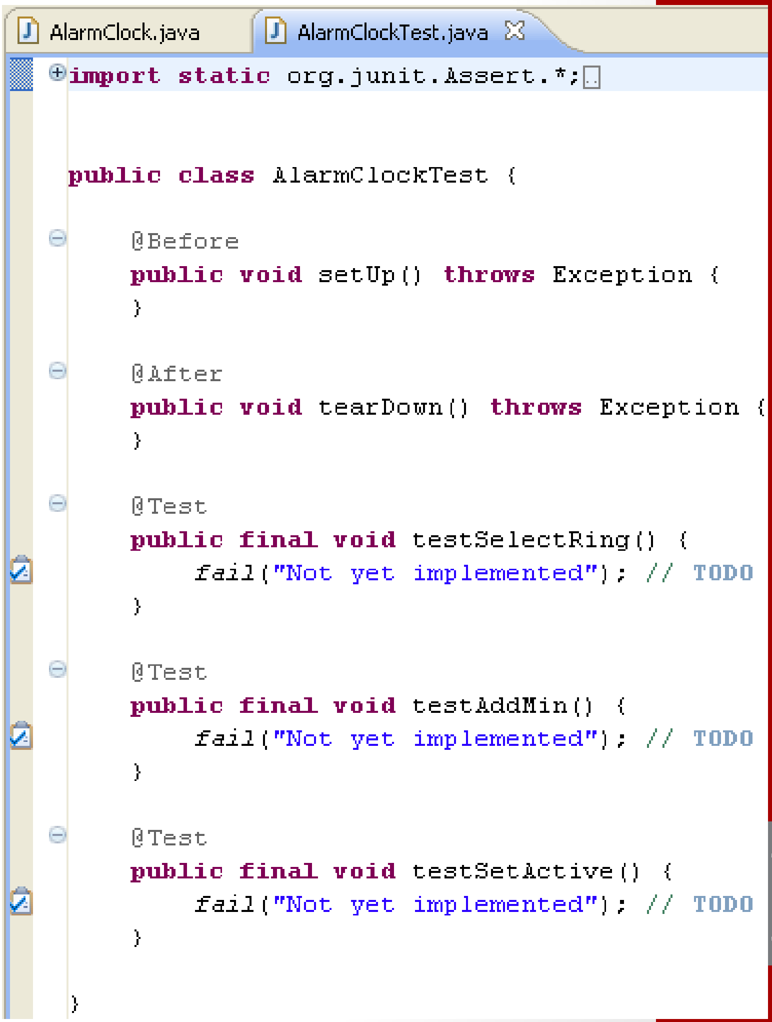
Available methods:

- ■ AlarmClock
  - ☐ AlarmClock(int, int, int)
  - ☑ selectRing(int)
  - ☑ addMin(int)
  - ☑ setActive(boolean)
  - ☐ equals(Object)
  - ☐ toString()
  - ☐ main(String[])

3 methods selected.

☑ Create final method stubs

☑ Create tasks for generated test methods

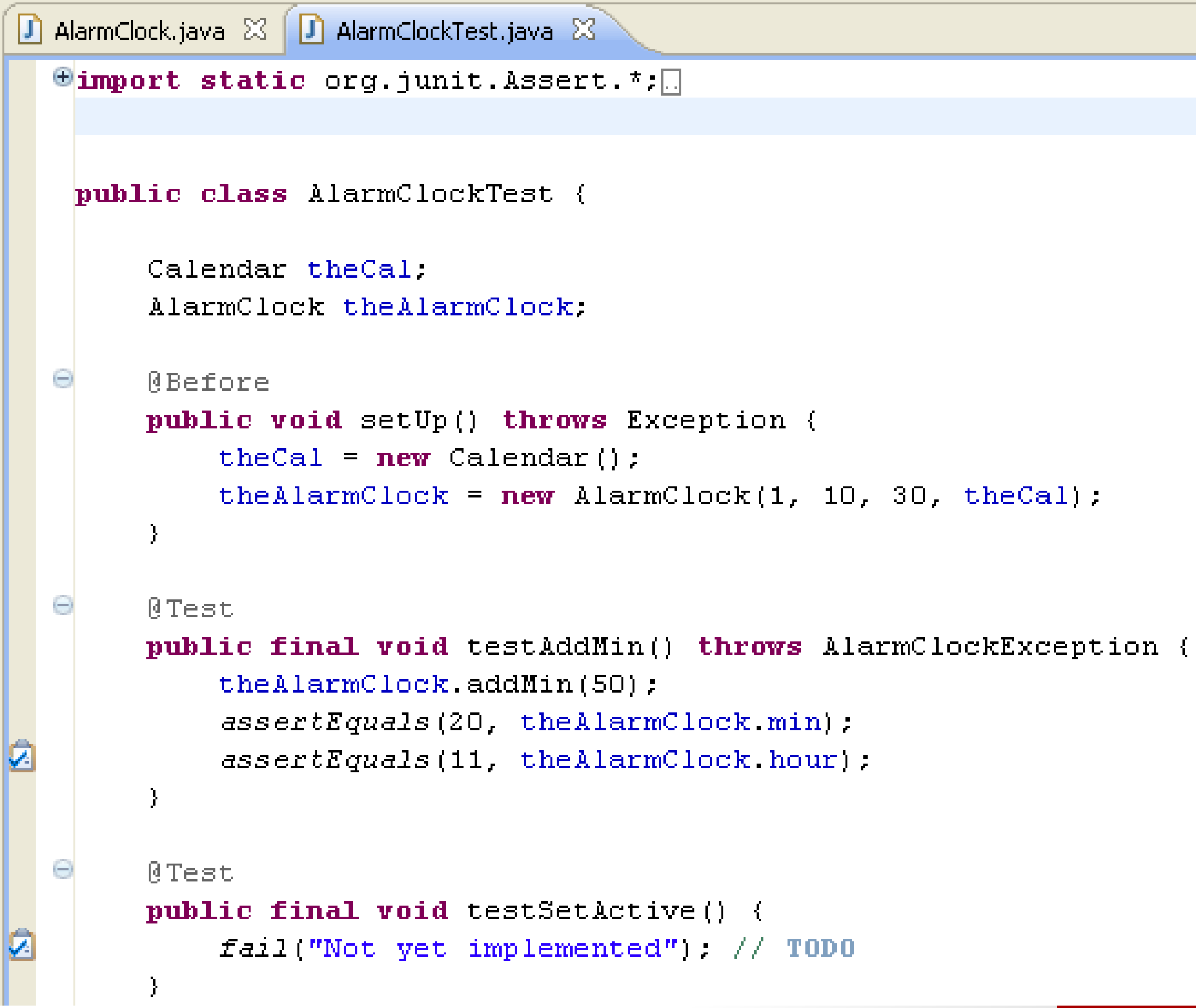


```
import static org.junit.Assert.*;

public class AlarmClockTest {

    @Before
    public void setUp() throws Exception {
    }

    @After
    public void tearDown() throws Exception {
    }

    @Test
    public final void testSelectRing() {
        fail("Not yet implemented"); // TODO
    }

    @Test
    public final void testAddMin() {
        fail("Not yet implemented"); // TODO
    }

    @Test
    public final void testSetActive() {
        fail("Not yet implemented"); // TODO
    }

}
```

Test Harness

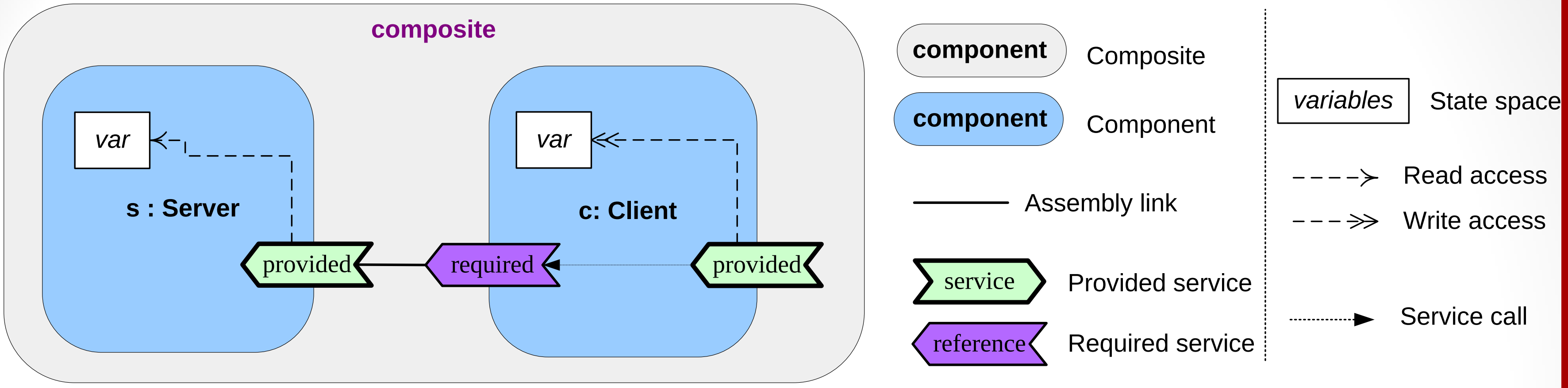
- Using Eclipse, a Junit test harness is partially generated.

- Then the tester
  - instantiates the objects,
  - links them,
  - runs the tests with test data,
  - gets the verdict from test oracles.



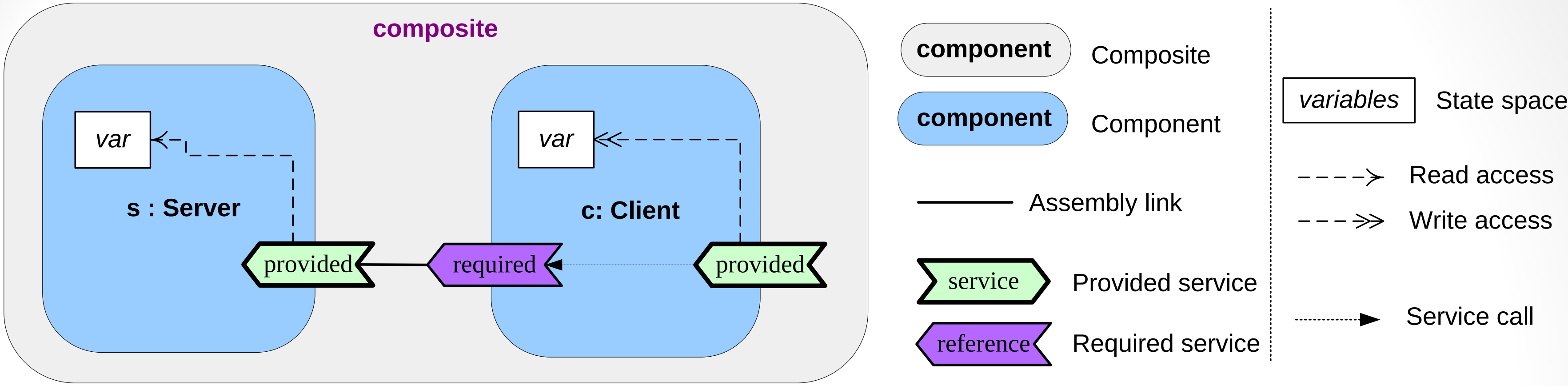
```
import static org.junit.Assert.*;


public class AlarmClockTest {

    Calendar theCal;
    AlarmClock theAlarmClock;

    @Before
    public void setUp() throws Exception {
        theCal = new Calendar();
        theAlarmClock = new AlarmClock(1, 10, 30, theCal);
    }

    @Test
    public final void testAddMin() throws AlarmClockException {
        theAlarmClock.addMin(50);
        assertEquals(20, theAlarmClock.min);
        assertEquals(11, theAlarmClock.hour);
    }

    @Test
    public final void testSetActive() {
        fail("Not yet implemented"); // TODO
    }
```

Problematic



- Service-based Component Models are assembled into complex architectures

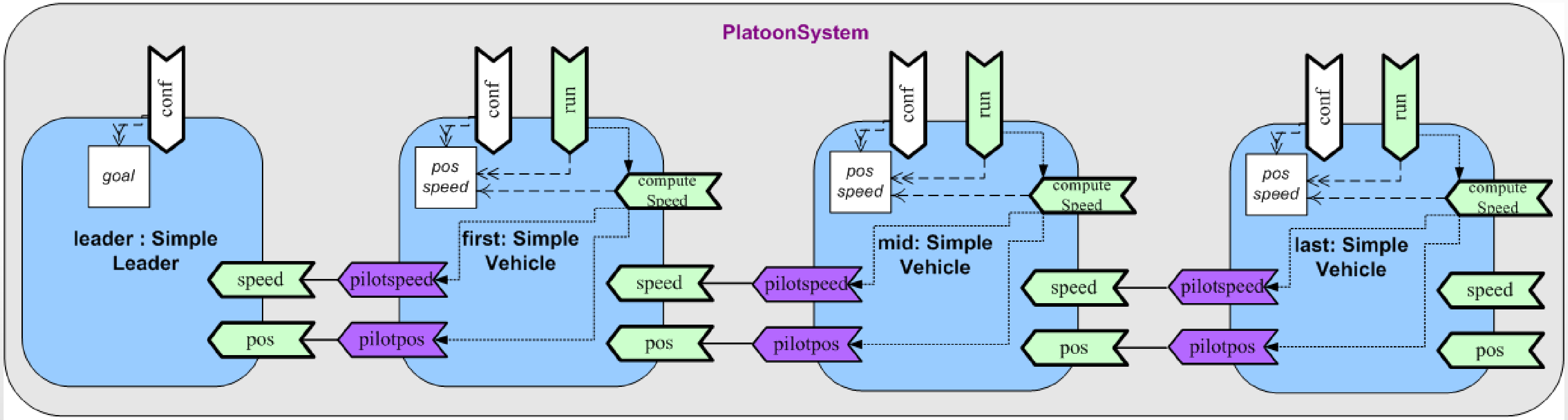
Problematic



- Service-based Component Models are assembled into complex architectures
- [Gross04] identifies issues:
  - Testing in a new context
  - Lack of access to the internal working of a component
- [Ghosh99] identifies problem :
  - Selection of subsets of components to be tested
  - Creation of testing components sequences

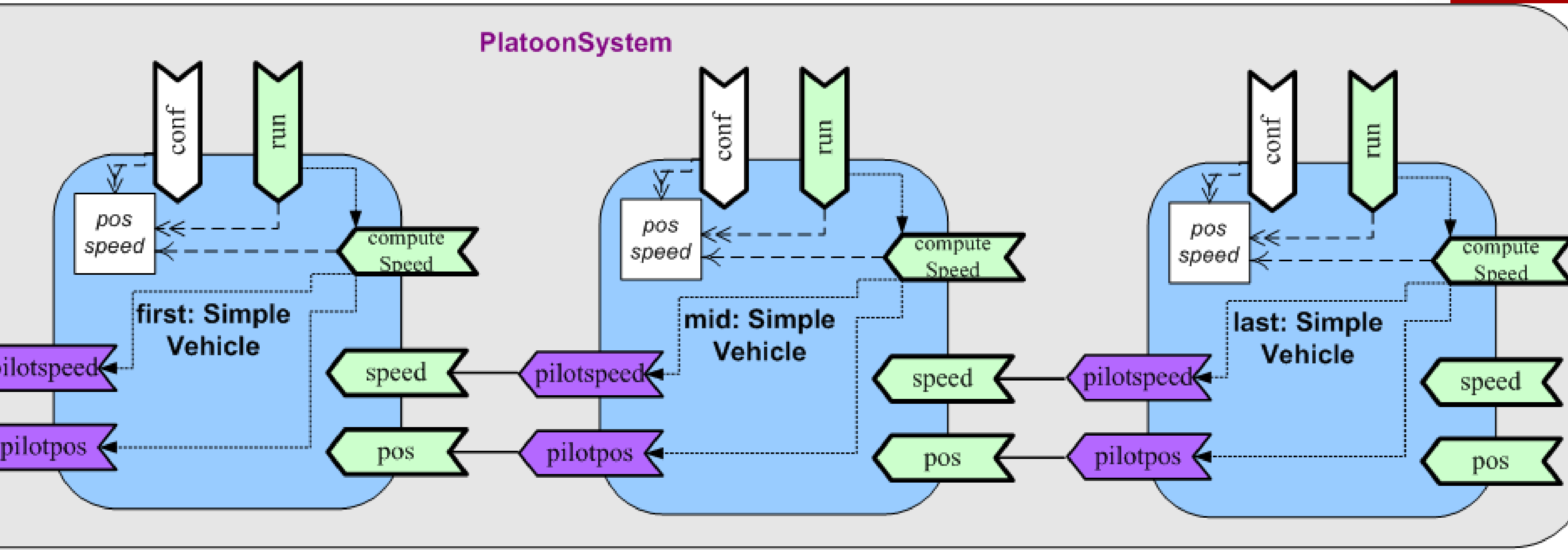
Motivating Example

- Assembly of components: platoon of vehicles



Motivating Example

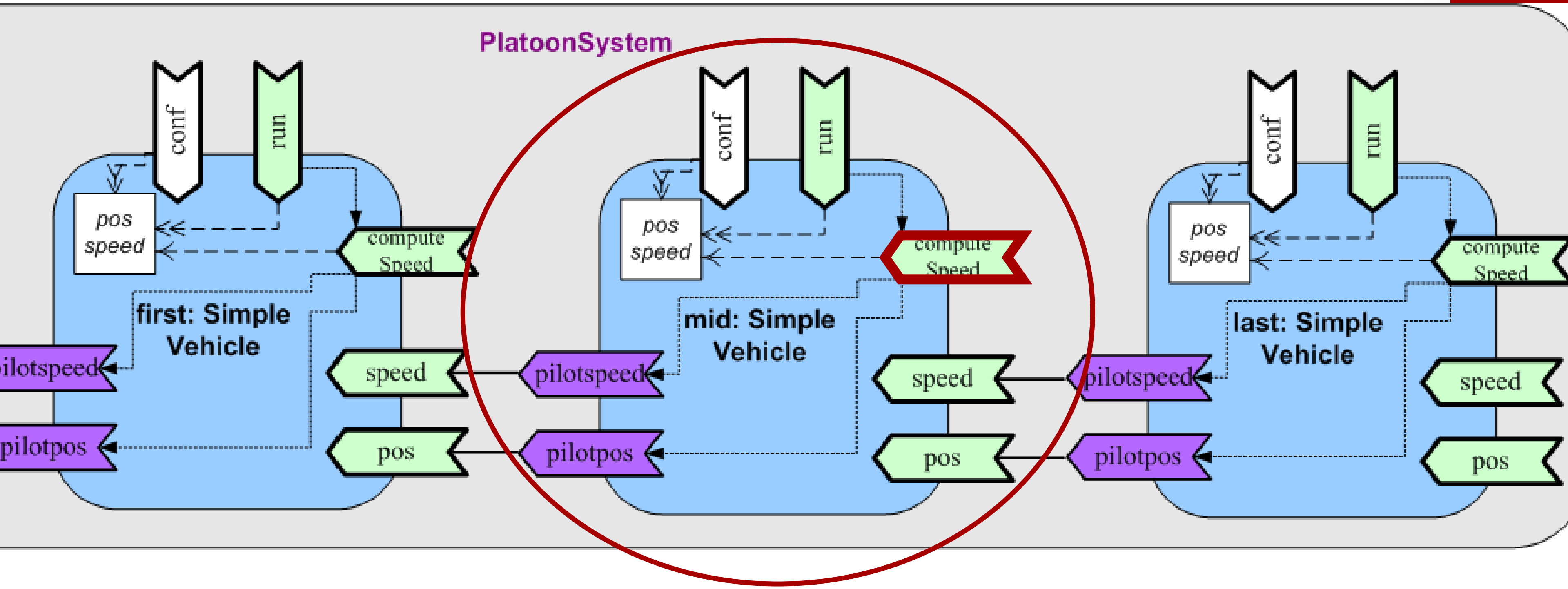
- Assembly of components: platoon of vehicles



Building Test Harness from Models

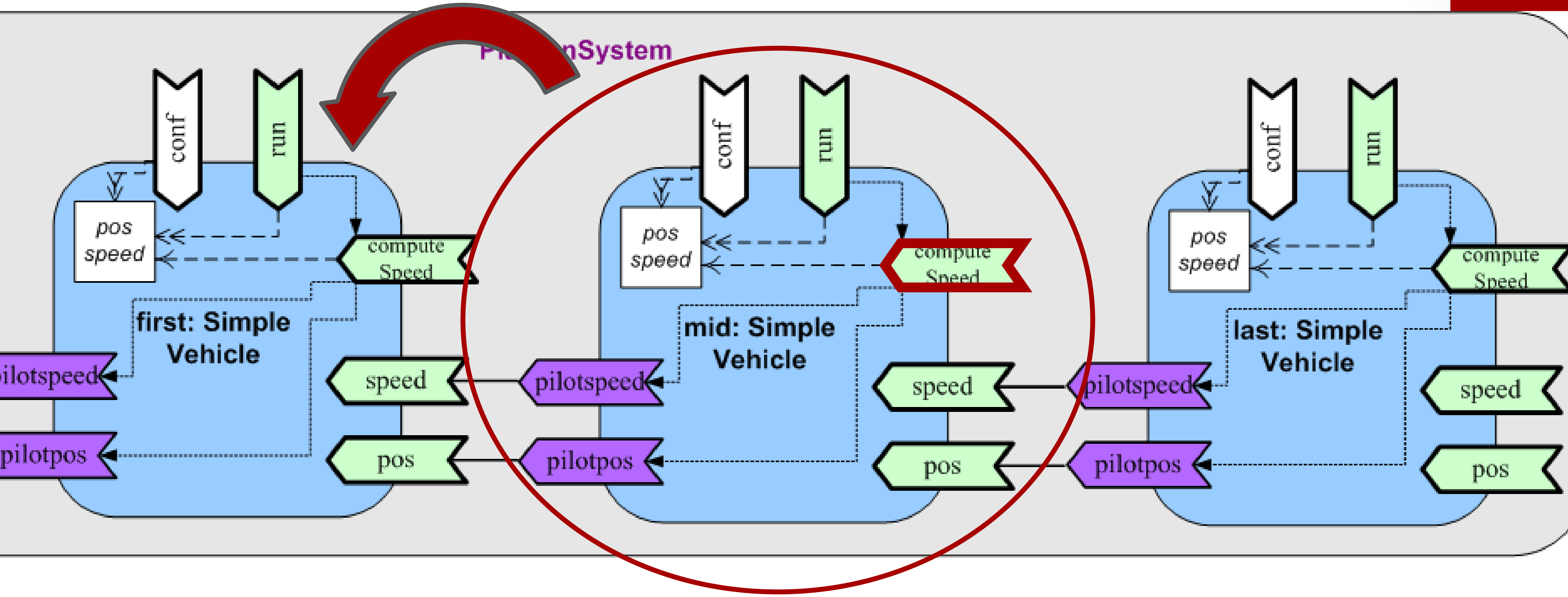
Motivating Example

- Assembly of components: platoon of vehicles
- A service of one Component Under Test



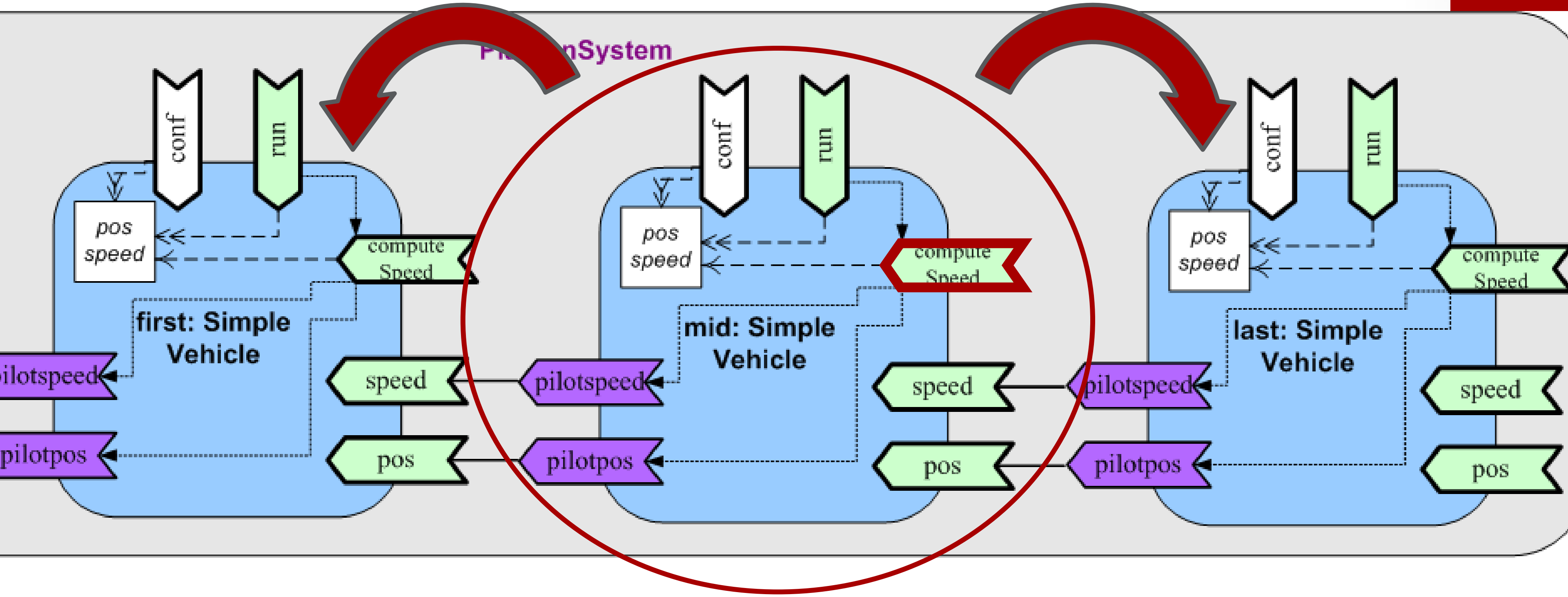
Motivating Example

- Assembly of components: platoon of vehicles
- A service of one Component Under Test
- Required services to be provided



Motivating Example

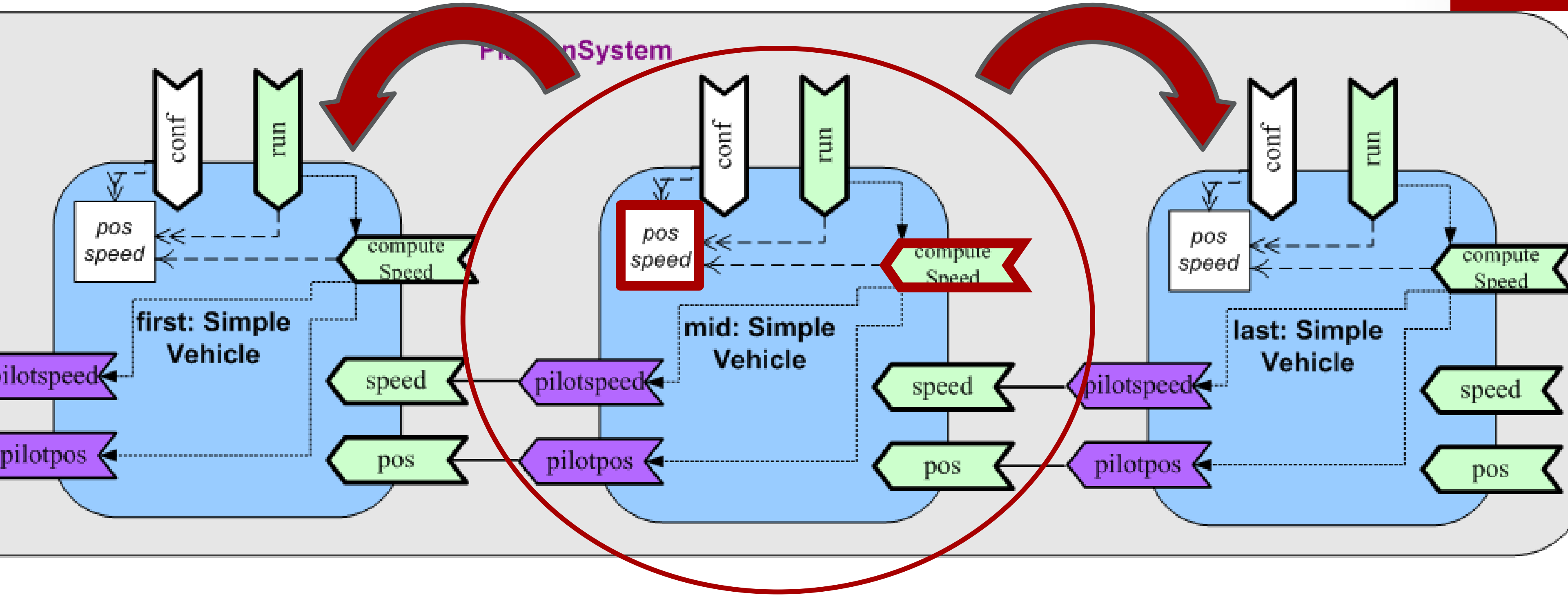
- Assembly of components: platoon of vehicles
- A service of one Component Under Test
- Required services to be provided
- Call and request the service under test



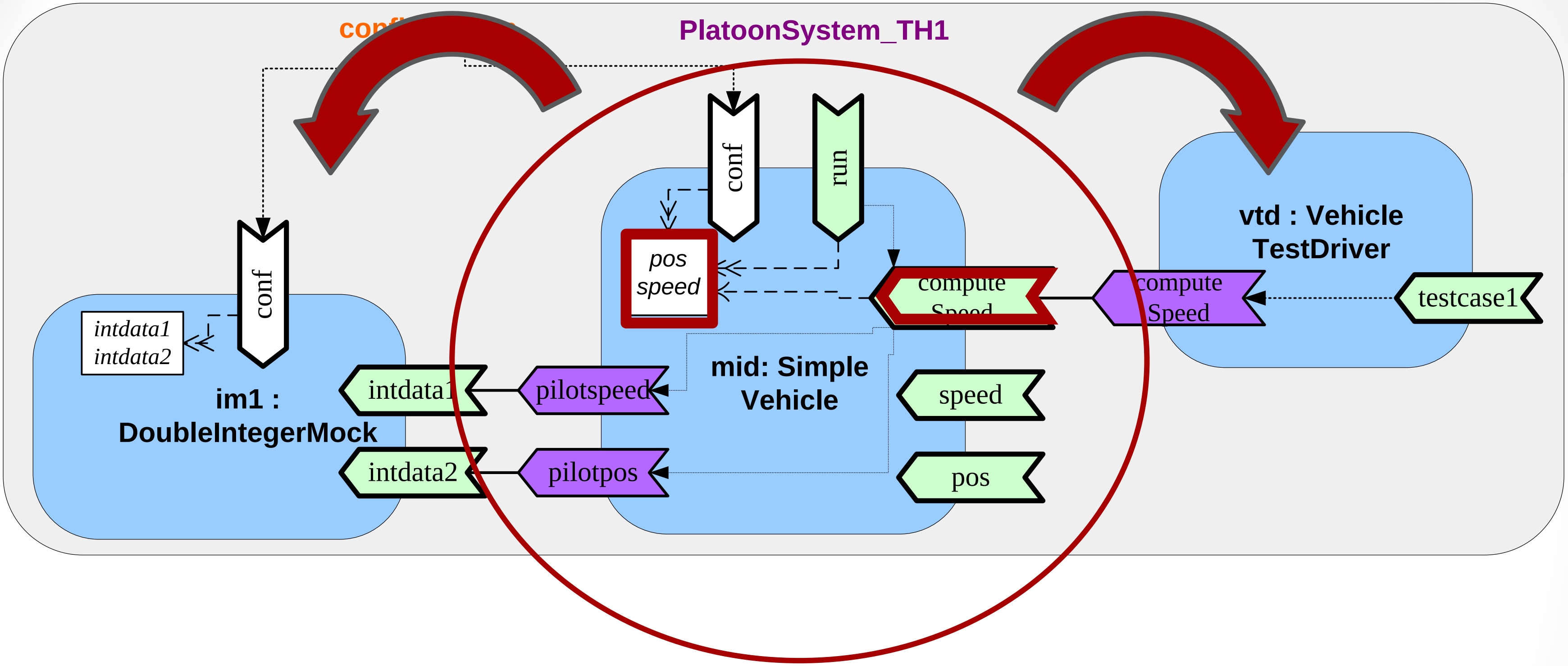
Motivating Example

Building Test Harness from Models

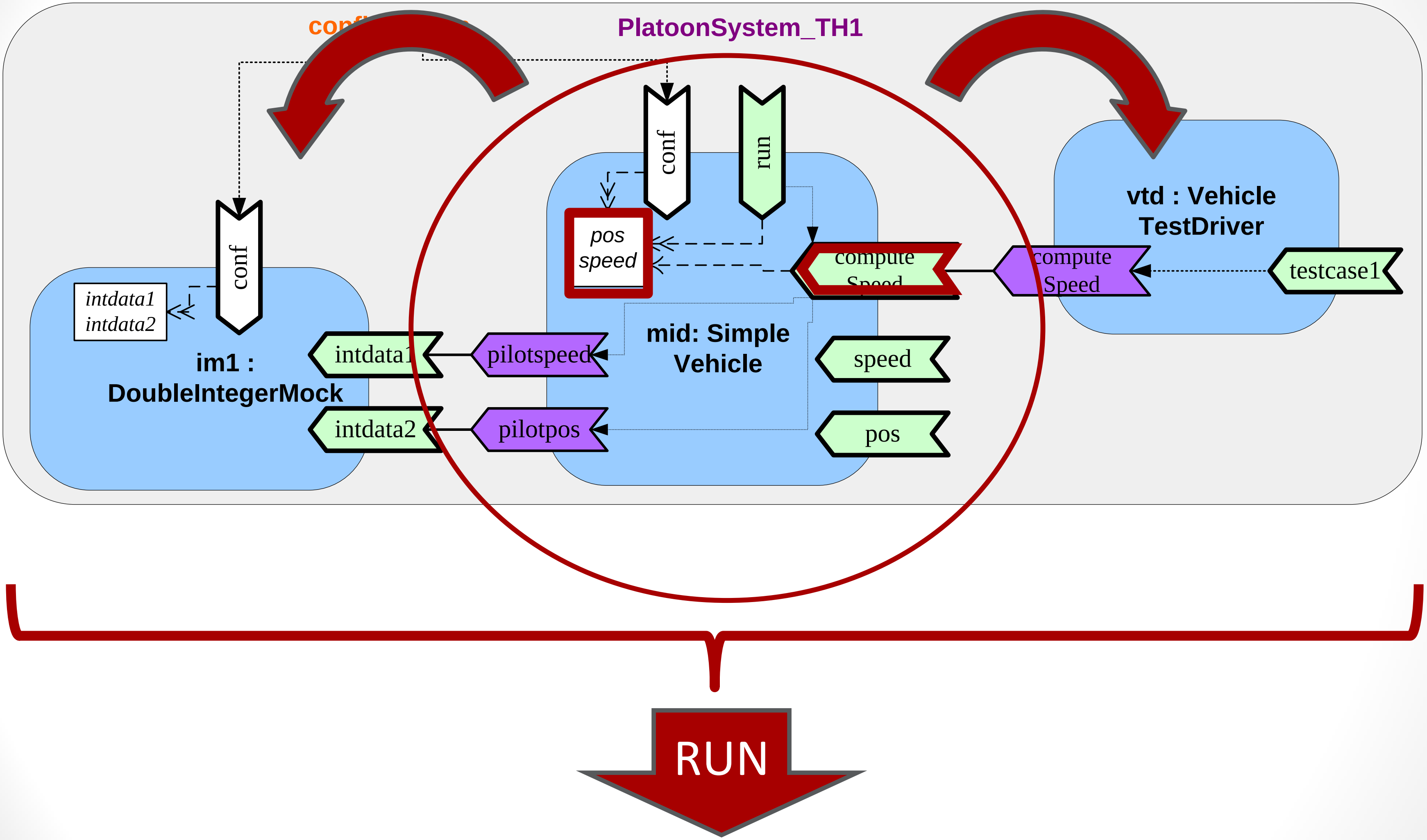
- Assembly of components: platoon of vehicles
- A service of one Component Under Test
- Required services to be provided
- Call and request the service under test
- Internal state initialisation



Motivating Example



Motivating Example



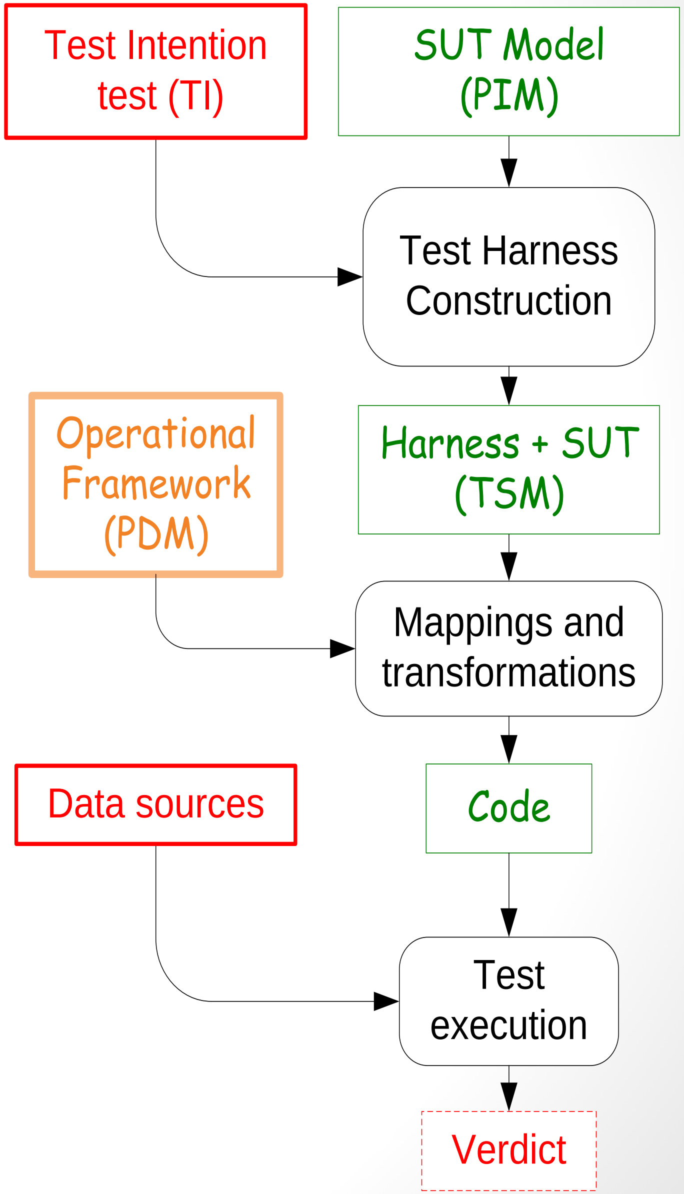
VERDICTS

Contribution

- Assisting the test harness building
  - MDD approach
    - Manipulating models with model transformations
  - Process in different steps
    - Automatic steps
    - Semi-automatic steps: the tester makes choices

- Running the tests
  - Specific Platform to the test
  - Automatic code generation

Assisting the test harness building



Test Intention test (TI)

SUT Model (PIM)

Test Harness Construction

Operational Framework (PDM)

Harness + SUT (TSM)

Mappings and transformations

Data sources

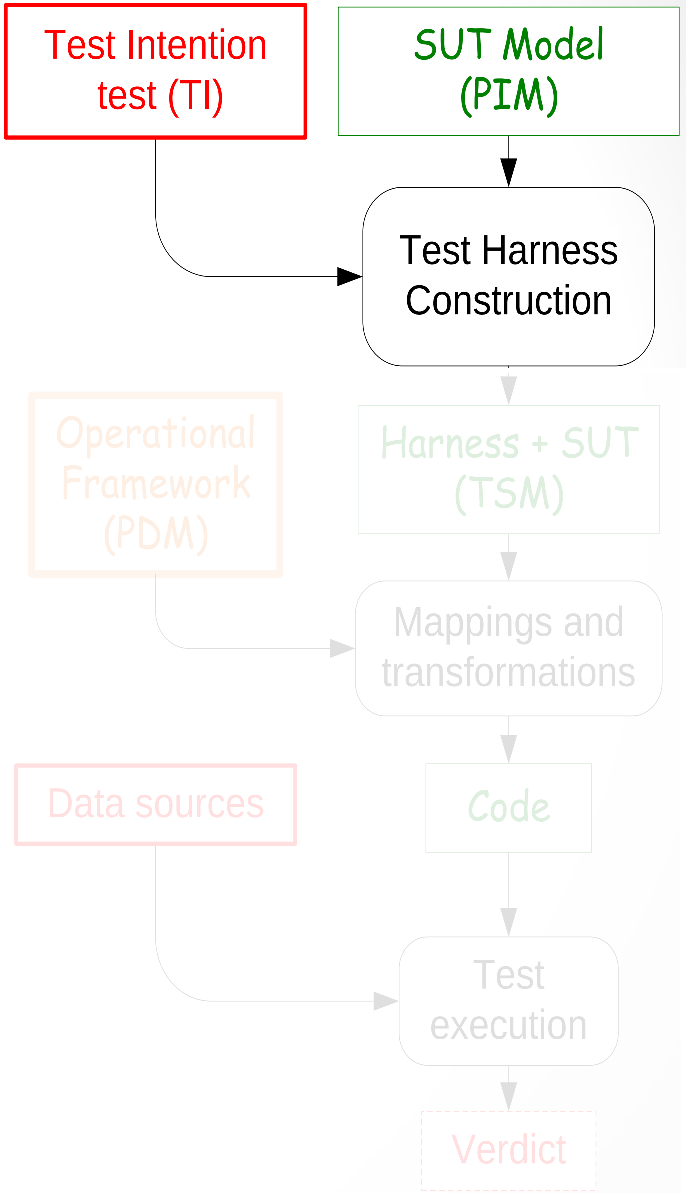
Code

Test execution

Verdict

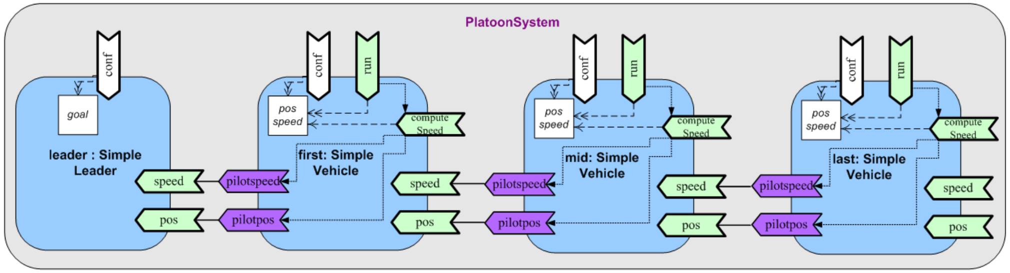
Assisting the test harness building

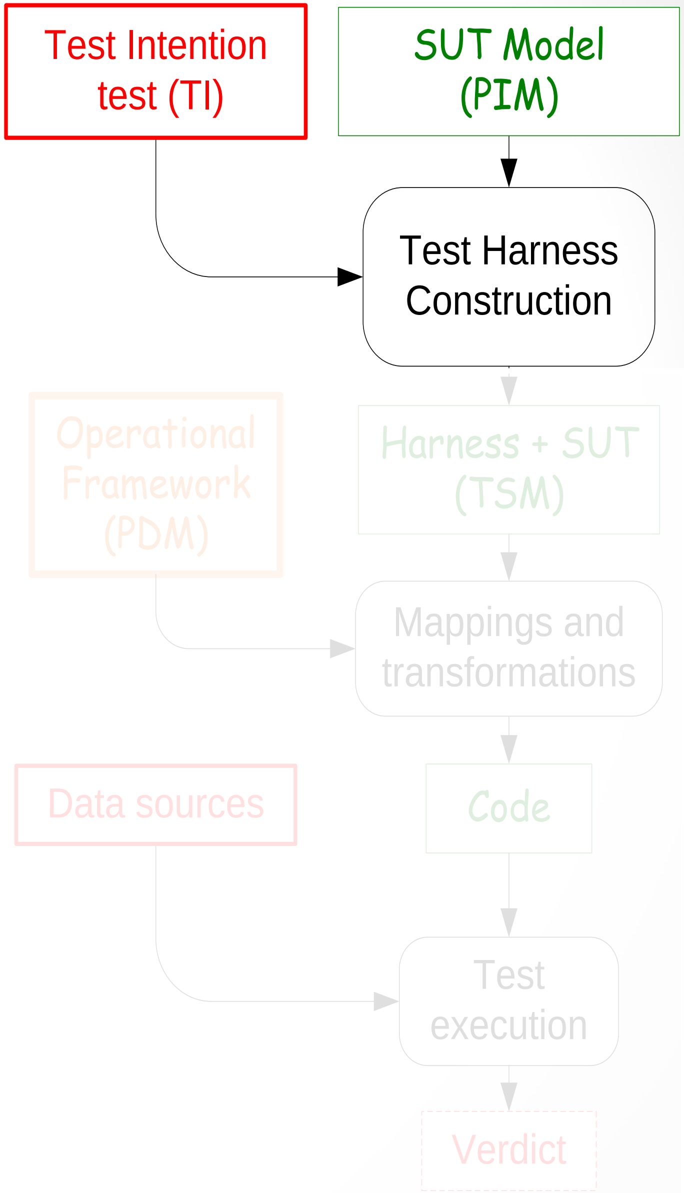
- Test harness construction



Assisting the test harness building

- Test harness construction
  - from SUT model





Test Intention test (TI)

SUT Model (PIM)

Test Harness Construction

Operational Framework (PDM)

Harness + SUT (TSM)

Mappings and transformations

Data sources

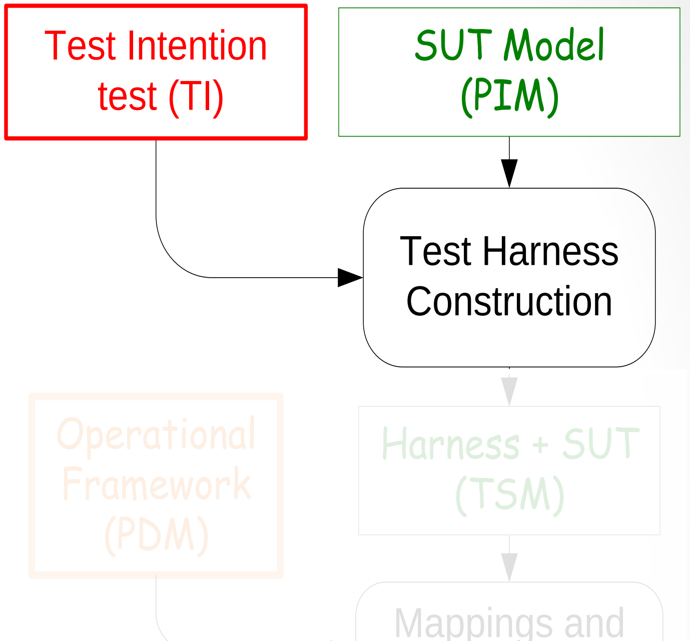
Code

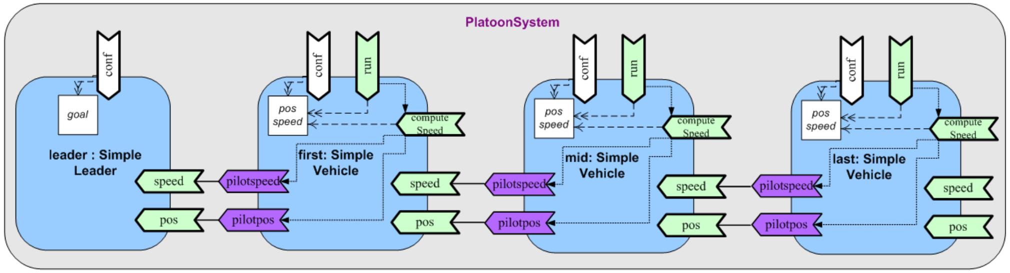
Test execution

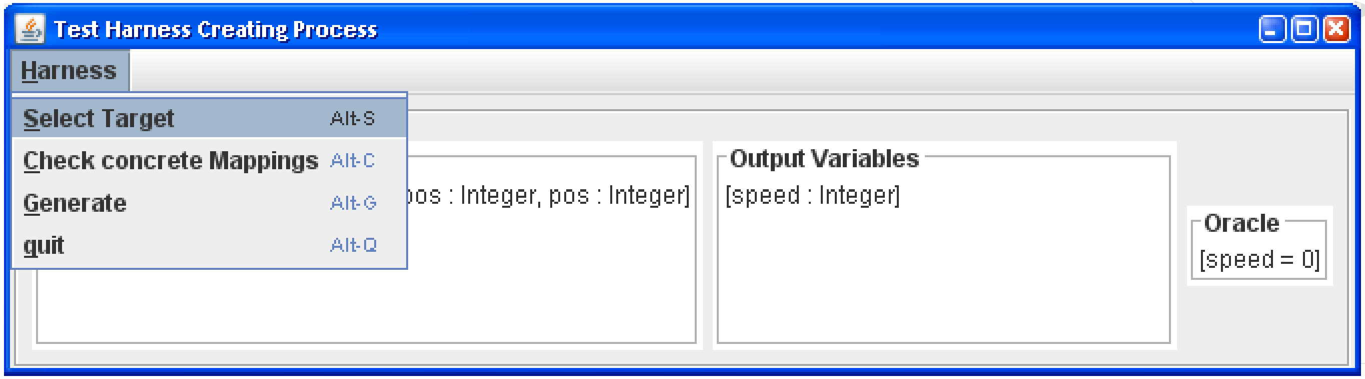
Verdict

Assisting the test harness building

- Test harness construction
  - from SUT model



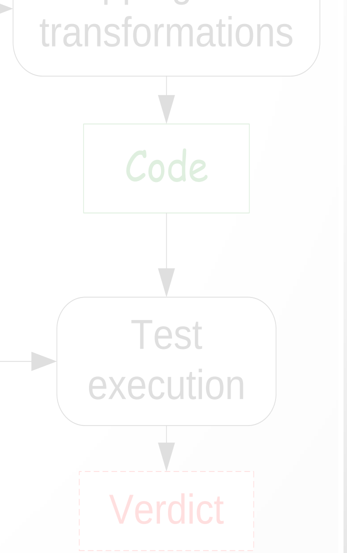




os : Integer, pos : Integer]

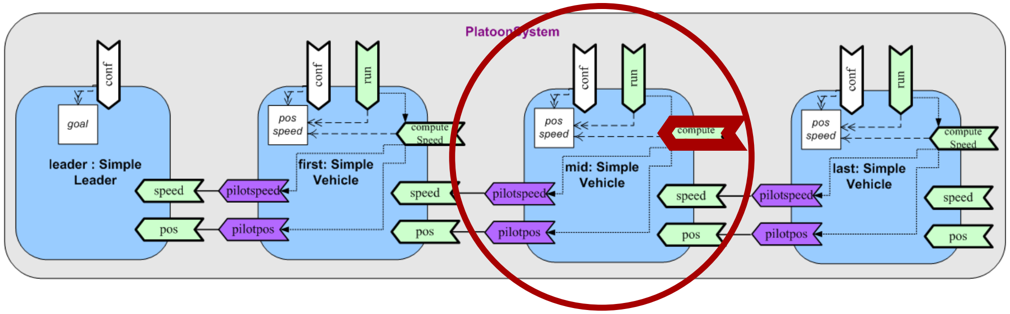
Output Variables
[speed : Integer]

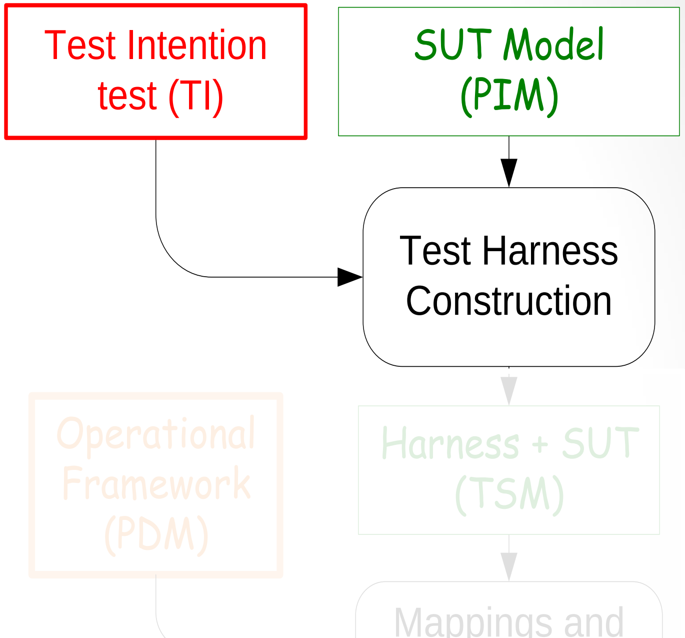
Oracle
[speed = 0]

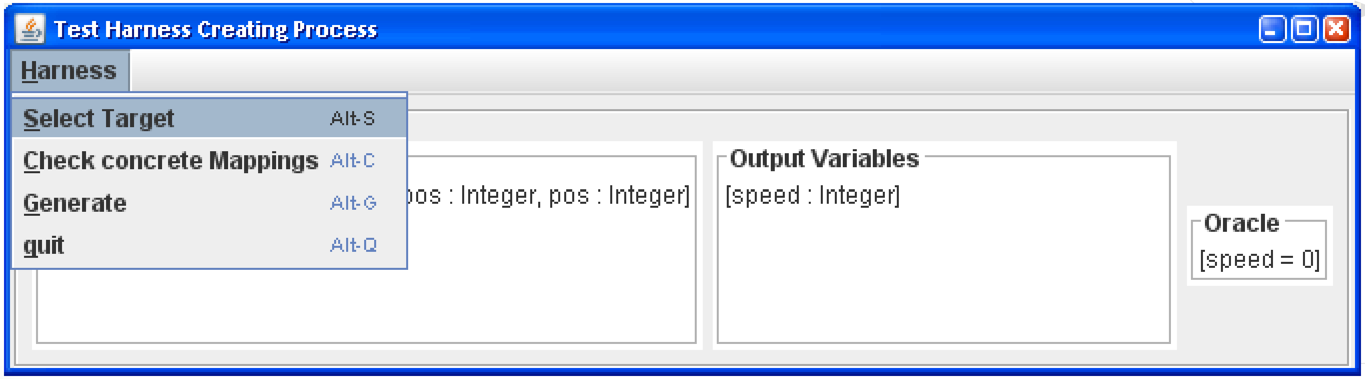


Assisting the test harness building

- Test harness construction
  - from SUT model







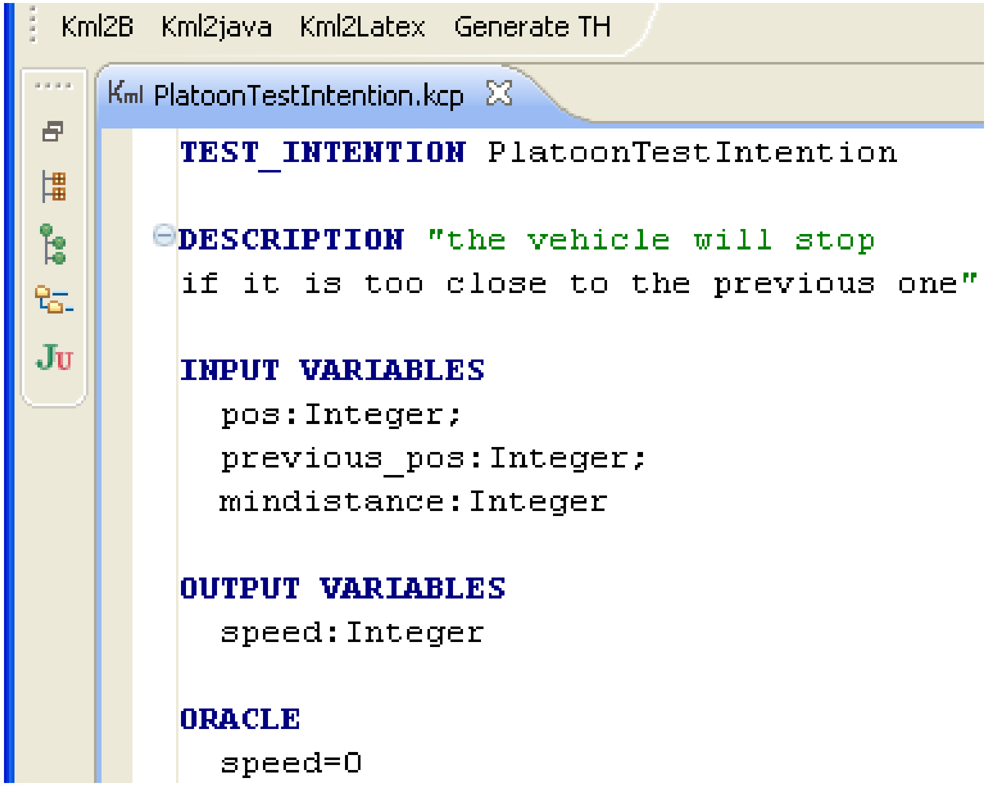
os : Integer, pos : Integer]

Output Variables
[speed : Integer]

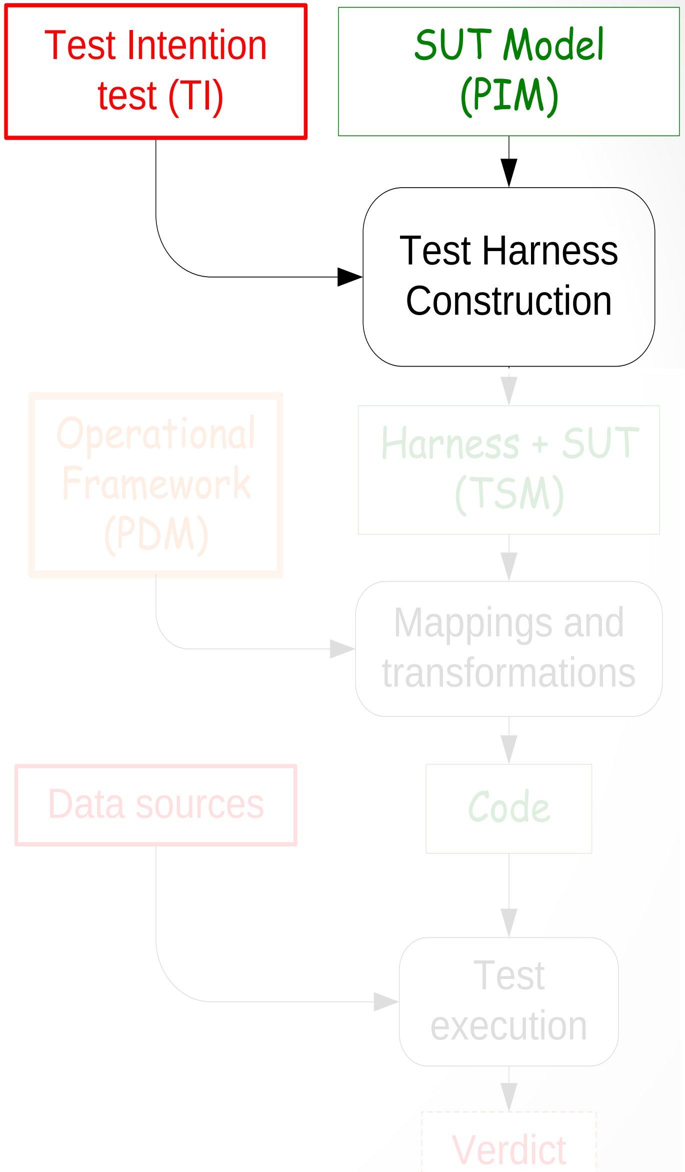
Oracle
[speed = 0]

Assisting the test harness building

- Test harness construction
  - from SUT model
  - from a test intention

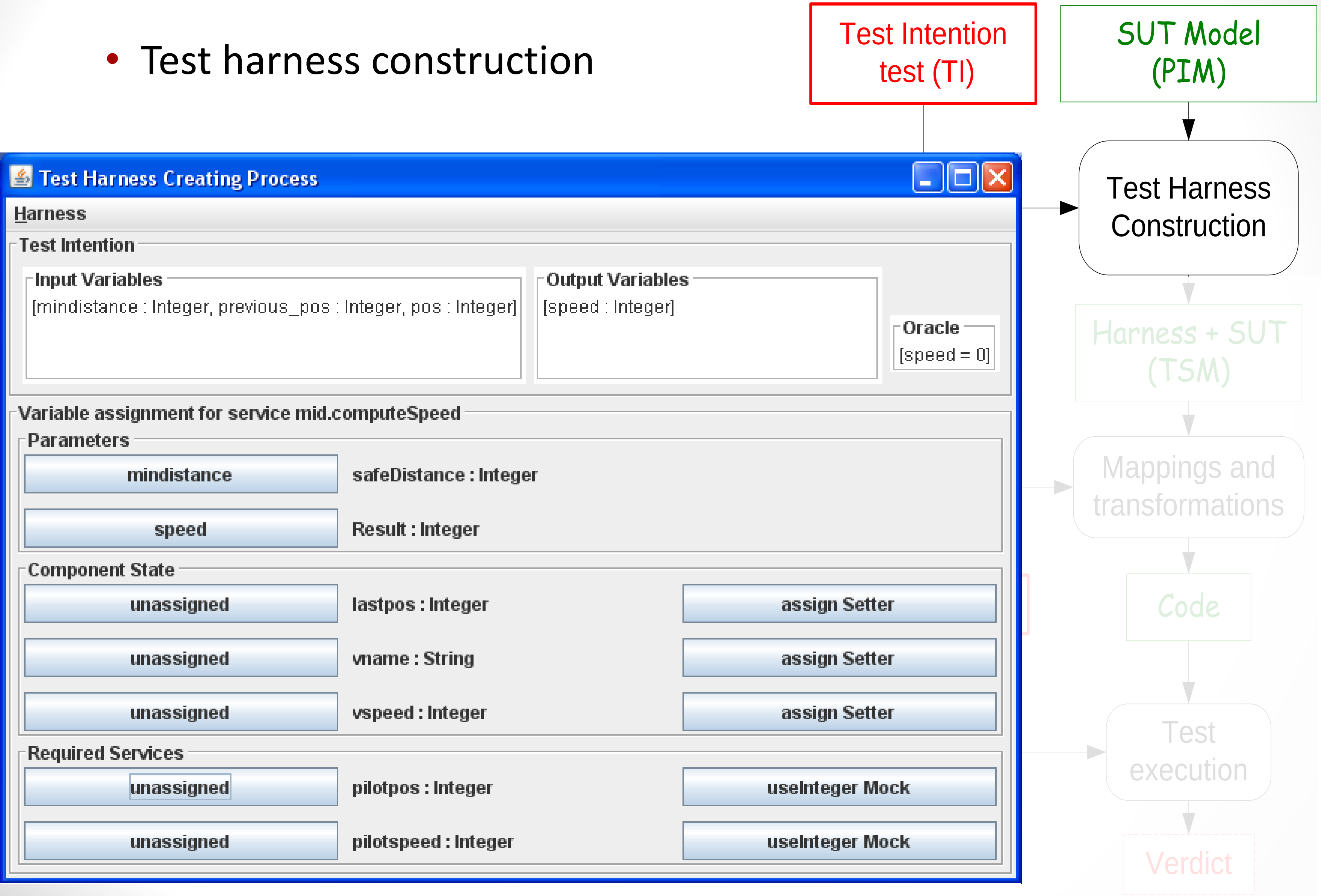


```
TEST_INTENTION PlatoonTestIntention

DESCRIPTION "the vehicle will stop
if it is too close to the previous one"

INPUT VARIABLES
  pos:Integer;
  previous_pos:Integer;
  mindistance:Integer

OUTPUT VARIABLES
  speed:Integer

ORACLE
  speed=0
```



Assisting the test harness building

- Test harness construction



Test Harness Creating Process

Harness

Test Intention

Input Variables
[mindistance : Integer, previous_pos : Integer, pos : Integer]

Output Variables
[speed : Integer]

Oracle
[speed = 0]

Variable assignment for service mid.computeSpeed

Parameters

mindistance	safeDistance : Integer
speed	Result : Integer

Component State

unassigned	lastpos : Integer	assign Setter
unassigned	vname : String	assign Setter
unassigned	vspeed : Integer	assign Setter

Required Services

unassigned	pilotpos : Integer	useInteger Mock
unassigned	pilotspeed : Integer	useInteger Mock

Test Intention test (TI)

SUT Model (PIM)

Test Harness Construction

Harness + SUT (TSM)

Mappings and transformations

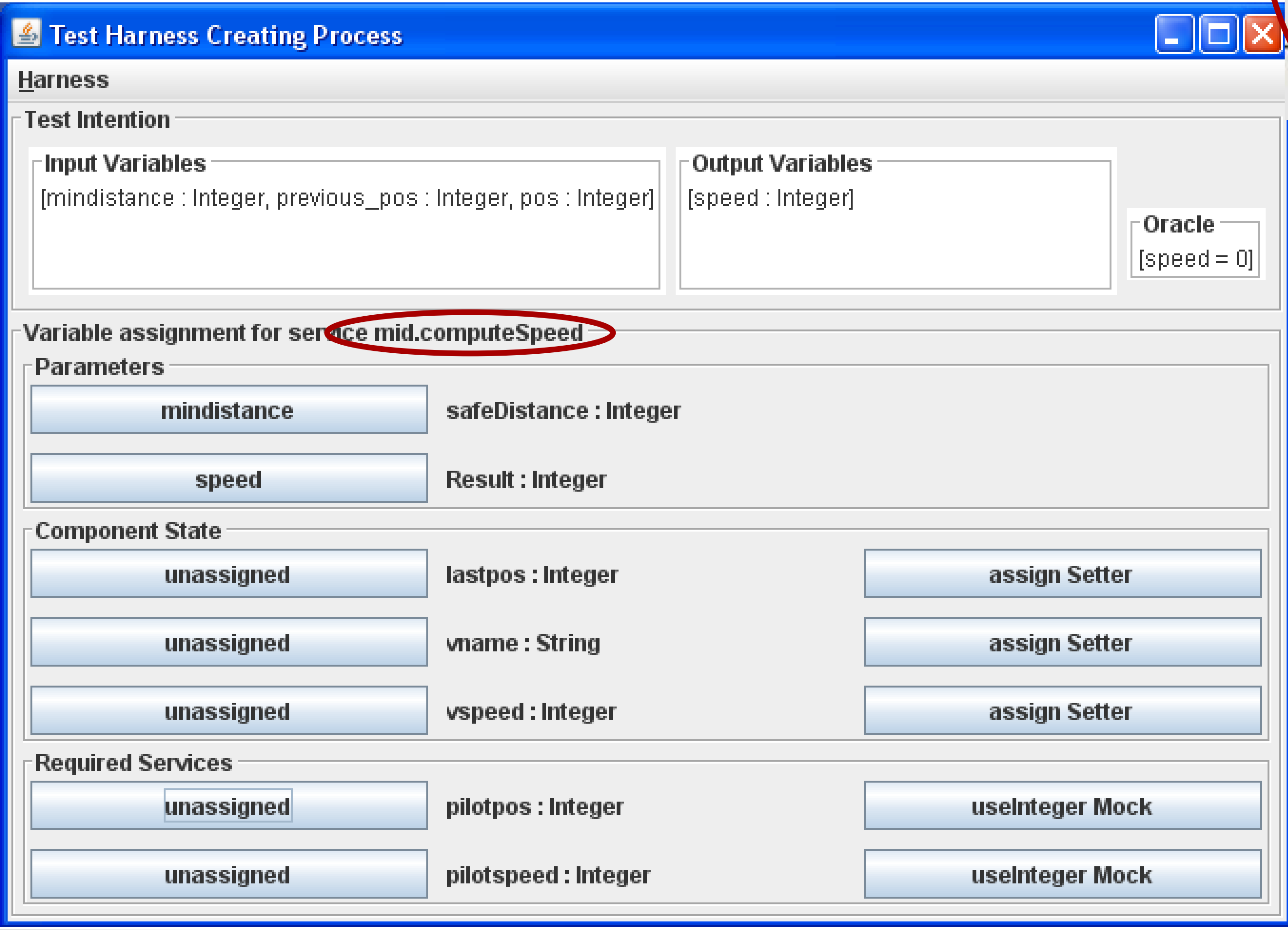
Code

Test execution

Verdict

Assisting the test harness build

- Test harness construction



Test Harness Creating Process

Harness

Test Intention

Input Variables
[mindistance : Integer, previous_pos : Integer, pos : Integer]

Output Variables
[speed : Integer]

Oracle
[speed = 0]

Variable assignment for service mid.computeSpeed

Parameters

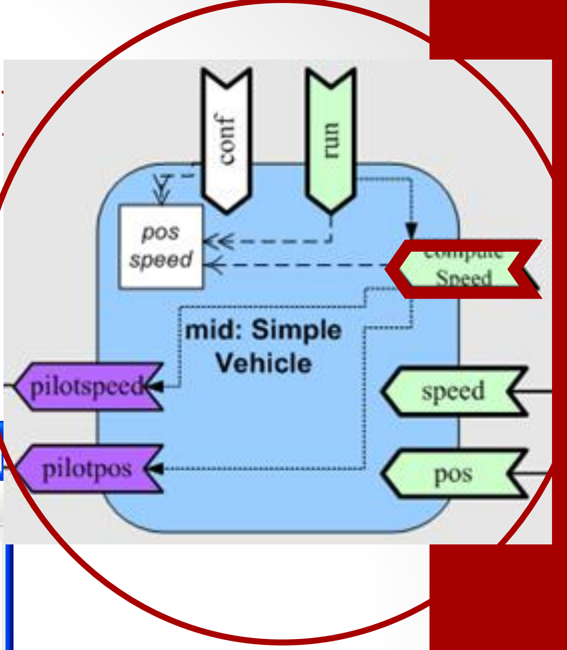
mindistance	safeDistance : Integer	
speed	Result : Integer	

Component State

unassigned	lastpos : Integer	assign Setter
unassigned	vname : String	assign Setter
unassigned	vspeed : Integer	assign Setter

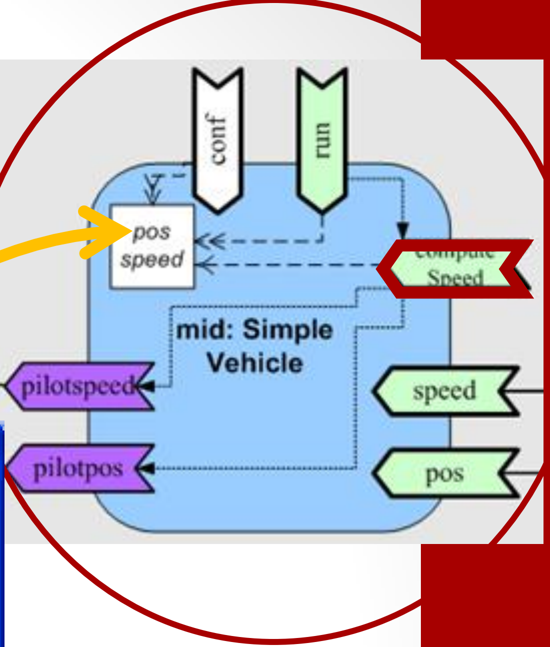
Required Services

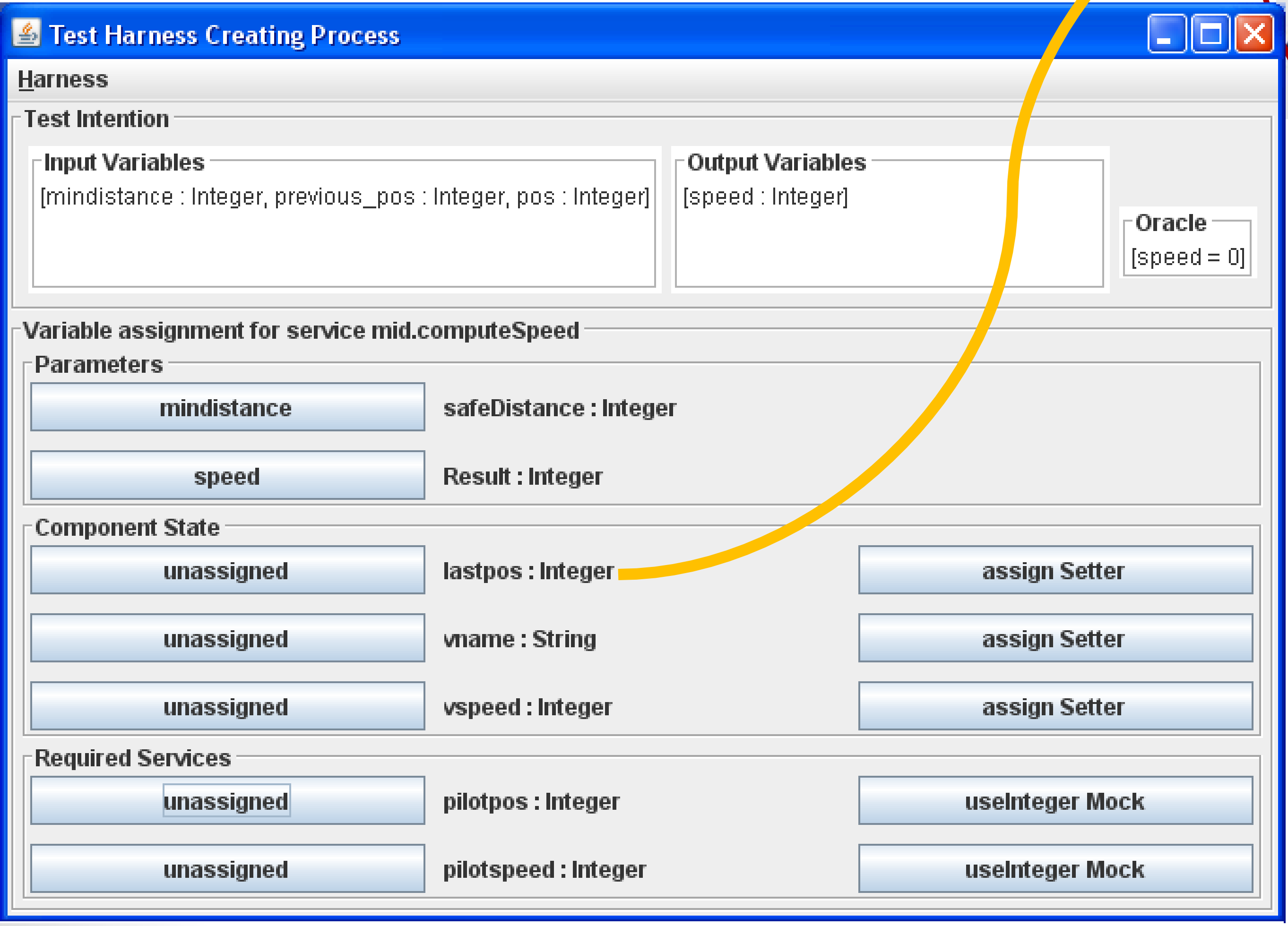
unassigned	pilotpos : Integer	useInteger Mock
unassigned	pilotspeed : Integer	useInteger Mock



Assisting the test harness build

- Test harness construction





Test Intention

Input Variables: [mindistance : Integer, previous_pos : Integer, pos : Integer]

Output Variables: [speed : Integer]

Oracle: [speed = 0]

Variable assignment for service mid.computeSpeed

Parameters

mindistance	safeDistance : Integer
speed	Result : Integer

Component State

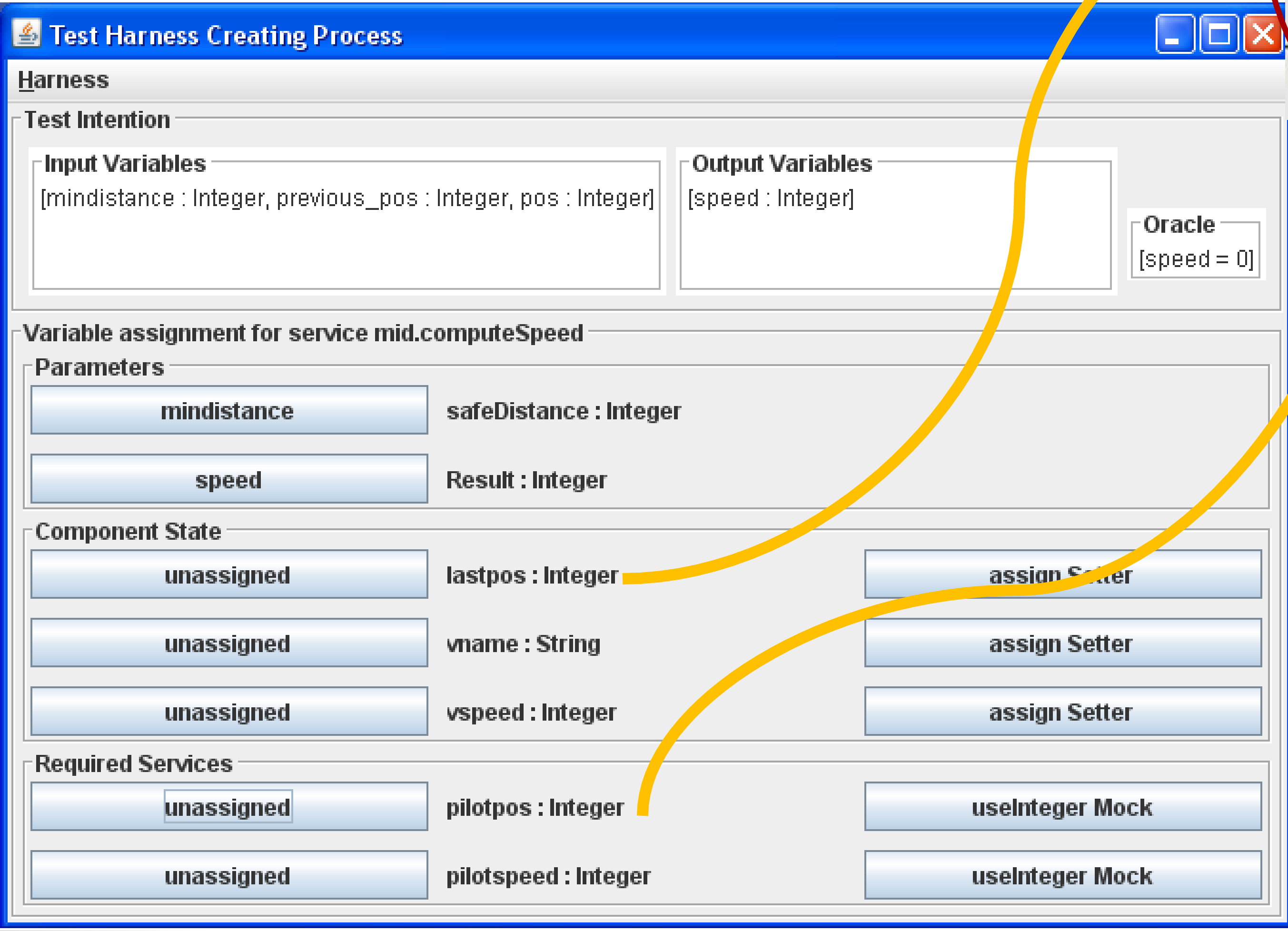
unassigned	lastpos : Integer	assign Setter
unassigned	vname : String	assign Setter
unassigned	vspeed : Integer	assign Setter

Required Services

unassigned	pilotpos : Integer	useInteger Mock
unassigned	pilotspeed : Integer	useInteger Mock

Assisting the test harness buildi

- Test harness construction



Test Harness Creating Process

Test Intention

Input Variables
[mindistance : Integer, previous_pos : Integer, pos : Integer]

Output Variables
[speed : Integer]

Oracle
[speed = 0]

Variable assignment for service mid.computeSpeed

Parameters

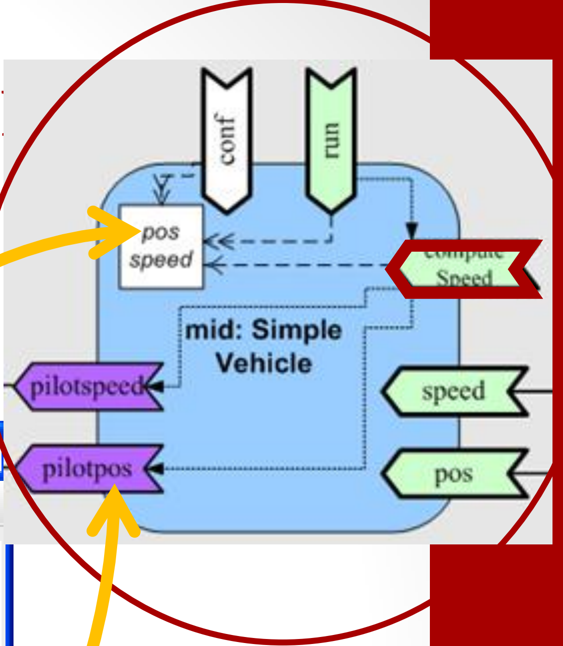
mindistance	safeDistance : Integer
speed	Result : Integer

Component State

unassigned	lastpos : Integer	assign Setter
unassigned	vname : String	assign Setter
unassigned	vspeed : Integer	assign Setter

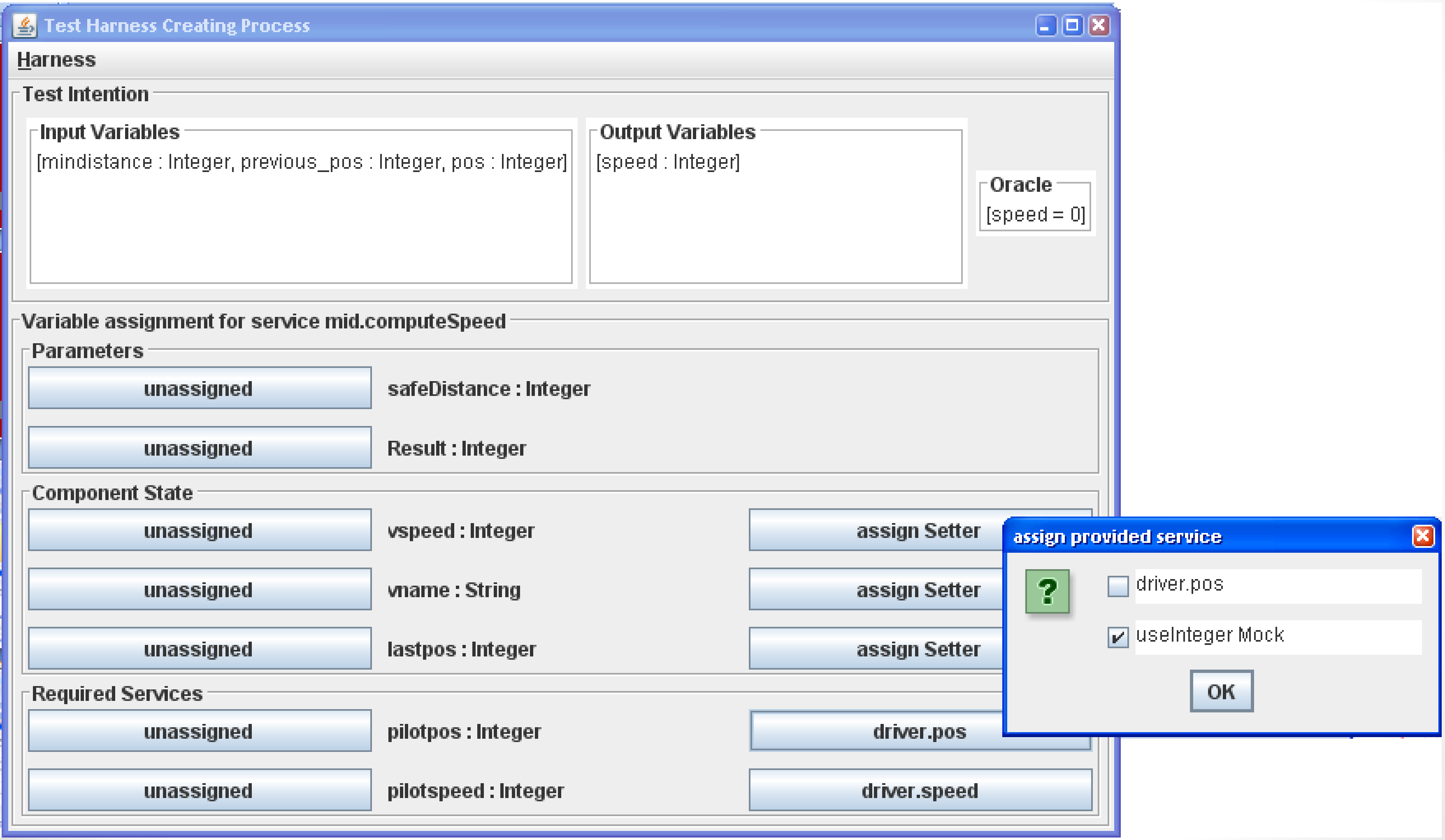
Required Services

unassigned	pilotpos : Integer	useInteger Mock
unassigned	pilotspeed : Integer	useInteger Mock



Assisting the test harness building

- Test harness construction



Test Harness Creating Process

Harness

Test Intention

Input Variables

[mindistance : Integer, previous_pos : Integer, pos : Integer]

Output Variables

[speed : Integer]

Oracle

[speed = 0]

Variable assignment for service mid.computeSpeed

Parameters

unassigned	safeDistance : Integer	
unassigned	Result : Integer	

Component State

unassigned	vspeed : Integer	assign Setter
unassigned	vname : String	assign Setter
unassigned	lastpos : Integer	assign Setter

Required Services

unassigned	pilotpos : Integer	driver.pos
unassigned	pilotspeed : Integer	driver.speed

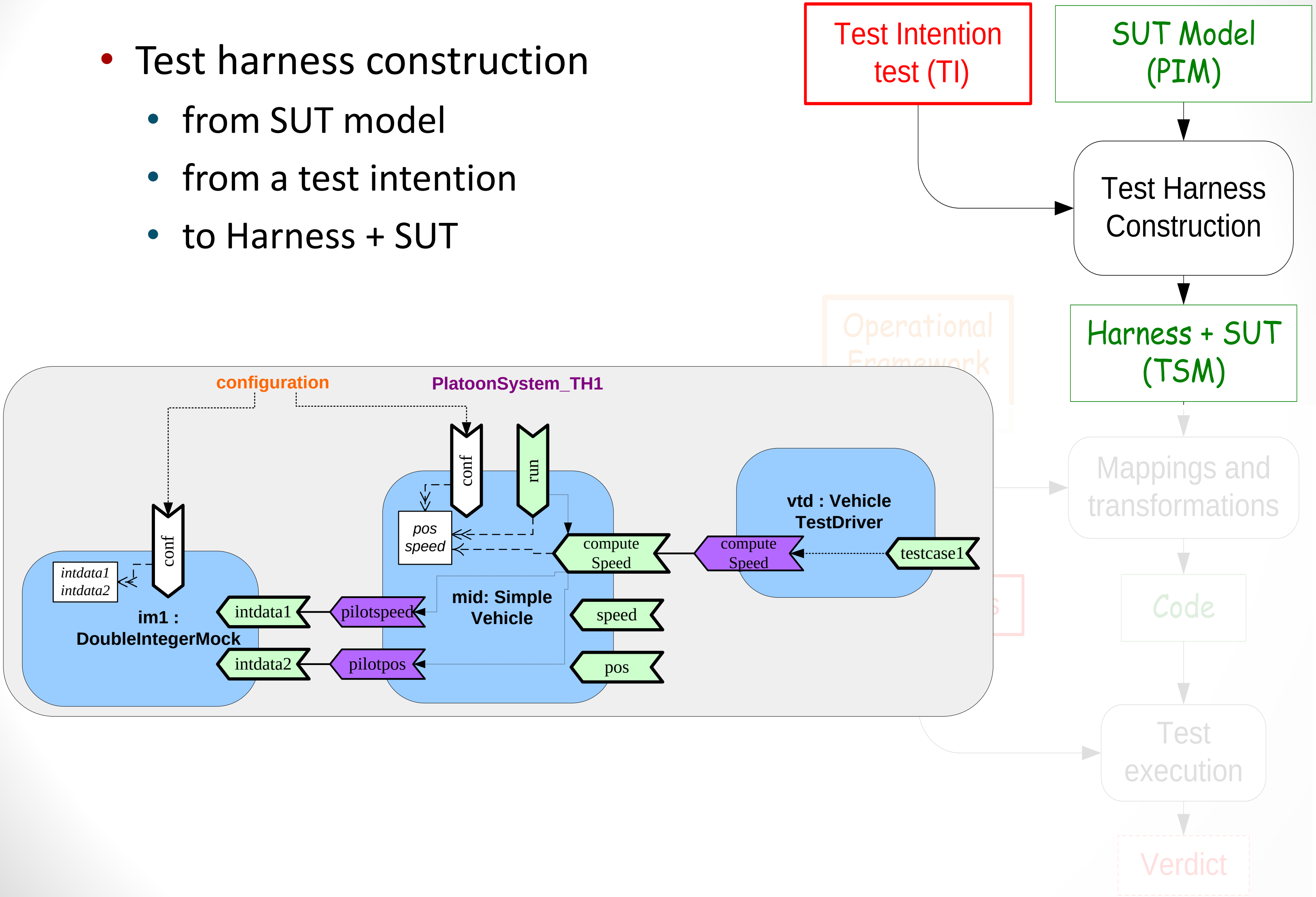
assign provided service

☐ driver.pos

☑ useInteger Mock

Assisting the test harness building

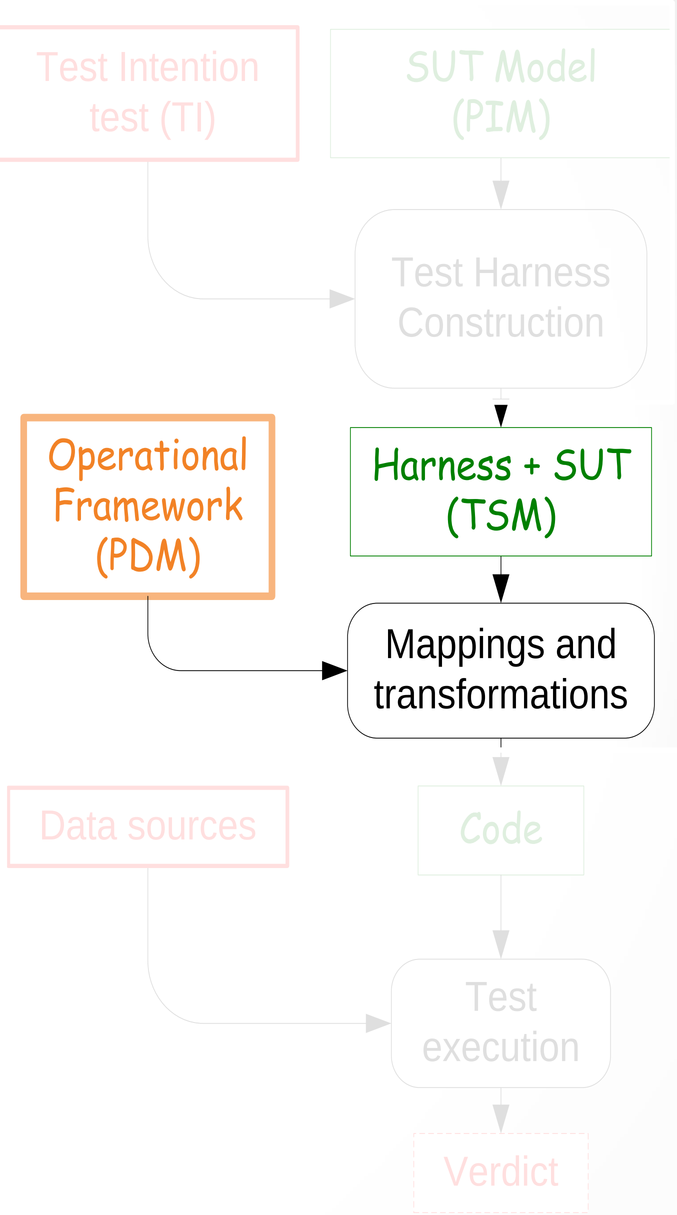
- Test harness construction
  - from SUT model
  - from a test intention
  - to Harness + SUT



Assisting the test harness building

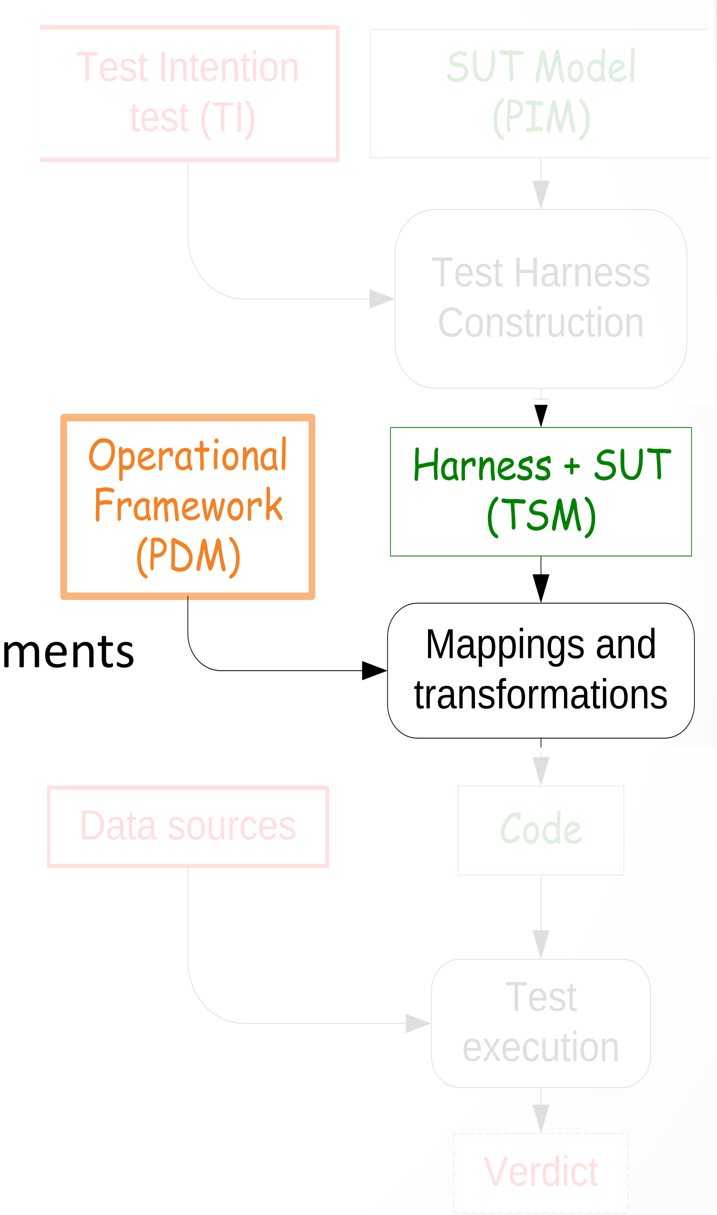
Running the tests

- Test harness construction
  - from SUT model
  - from a test intention
  - to Harness + SUT
- Mappings and transformations



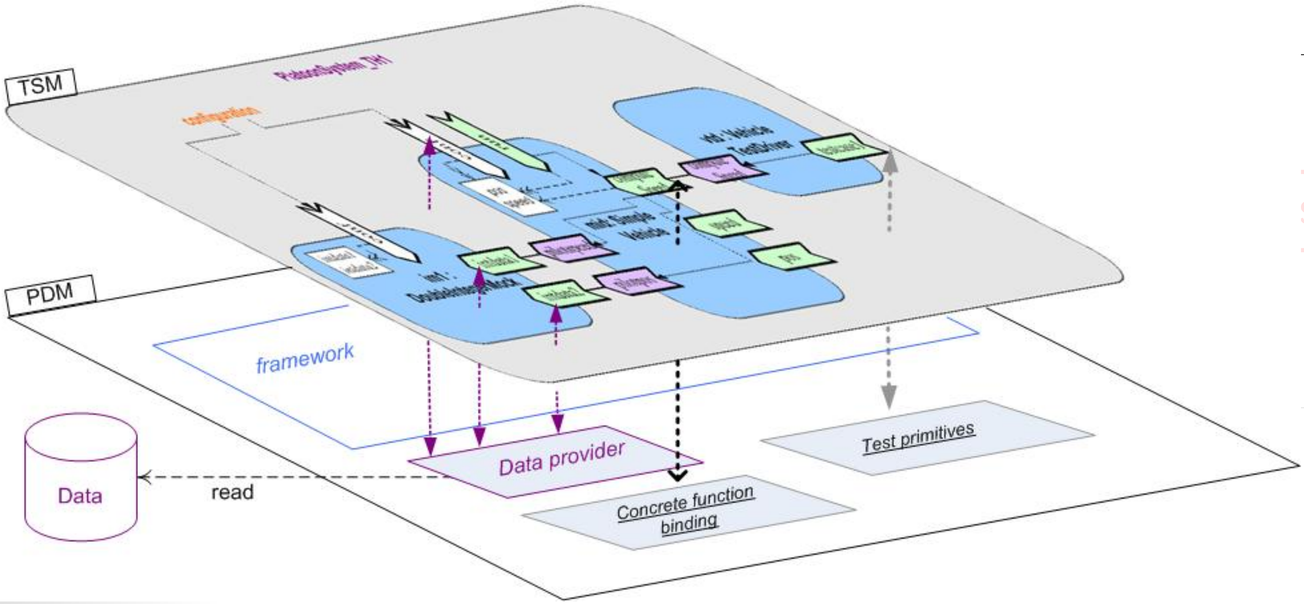
Assisting the test harness building
Running the tests

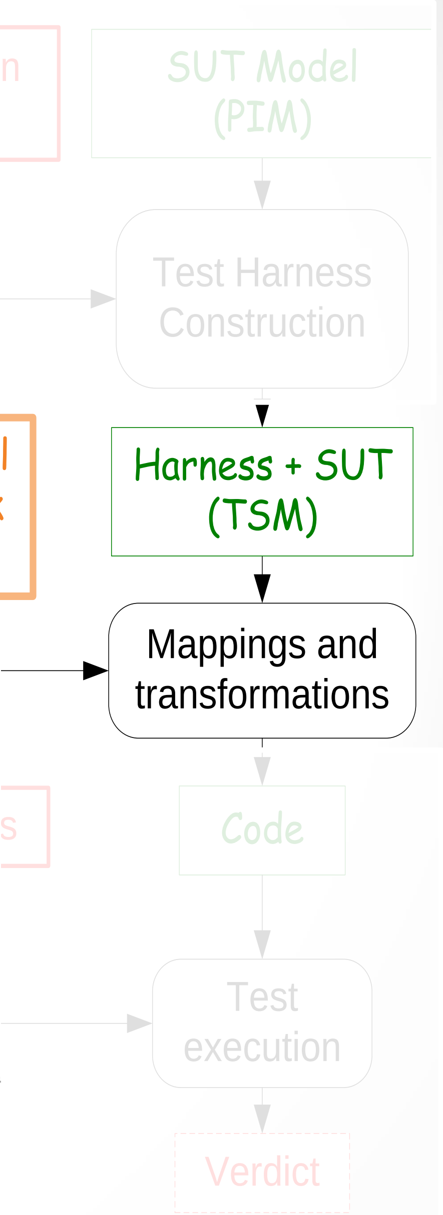
- Test harness construction
  - from SUT model
  - from a test intention
  - to Harness + SUT
- Mappings and transformations
  - from operational framework
    - Libraries used to describe the behavior of different model elements
    - costo_java_framework



Assisting the test harness building
Running the tests

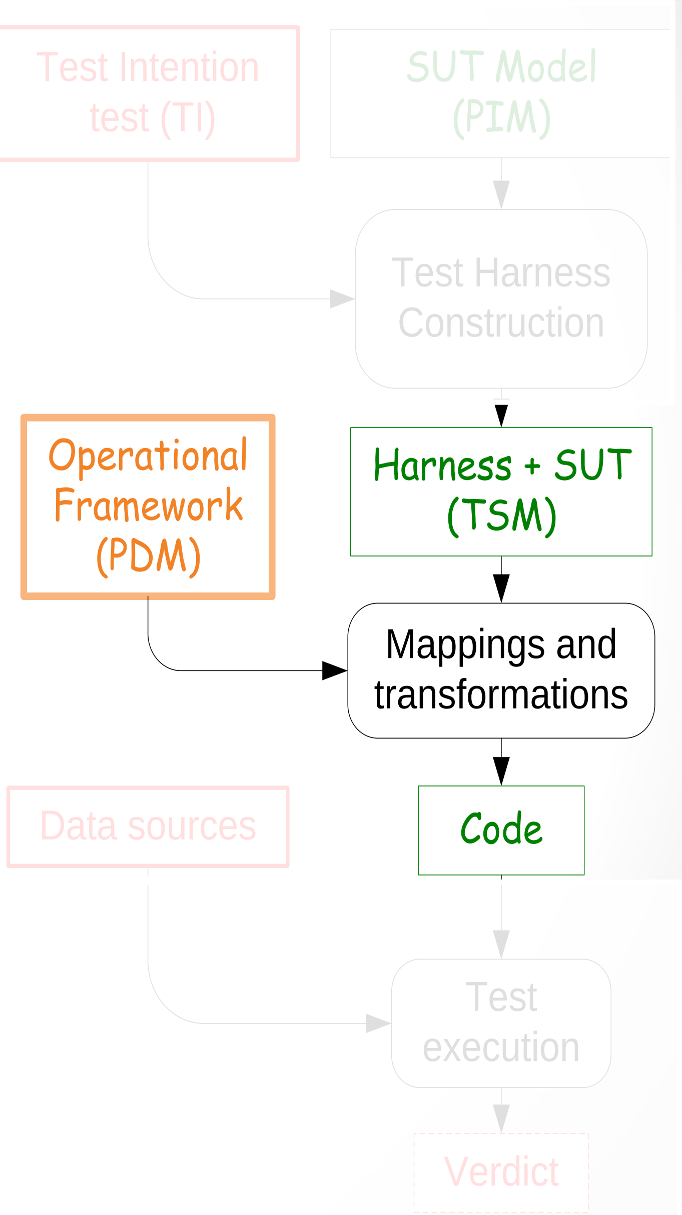
- Test harness construction
  - from SUT model
  - from a test intention
  - to Harness + SUT
- Mappings and transformations
  - from operational framework
  - Mappings





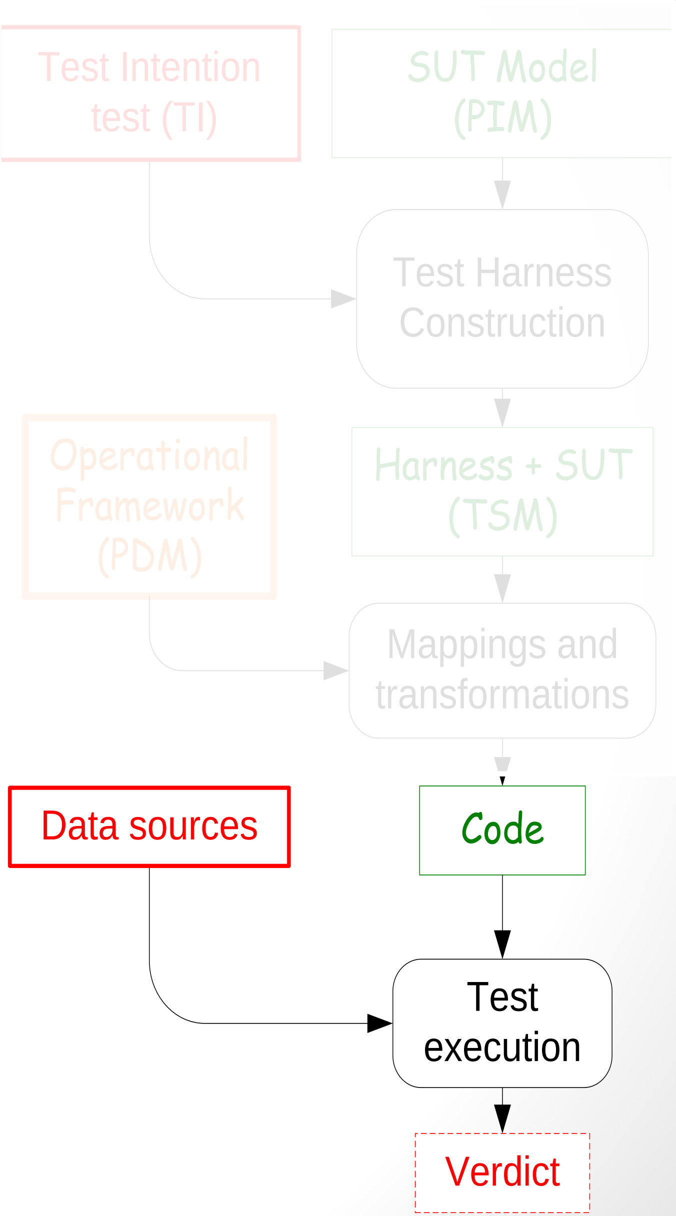
Assisting the test harness building
Running the tests

- Test harness construction
  - from SUT model
  - from a test intention
  - to Harness + SUT
- Mappings and transformations
  - from operational framework
  - Mappings
  - Transformations
    - generate Java code



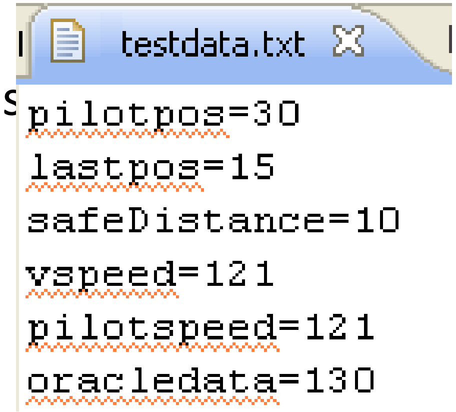
Running the tests

- Test harness construction
  - from SUT model
  - from a test intention
  - to Harness + SUT
- Mappings and transformations
  - from operational framework
  - Mappings
  - Transformations
    - generate Java code
- Test execution

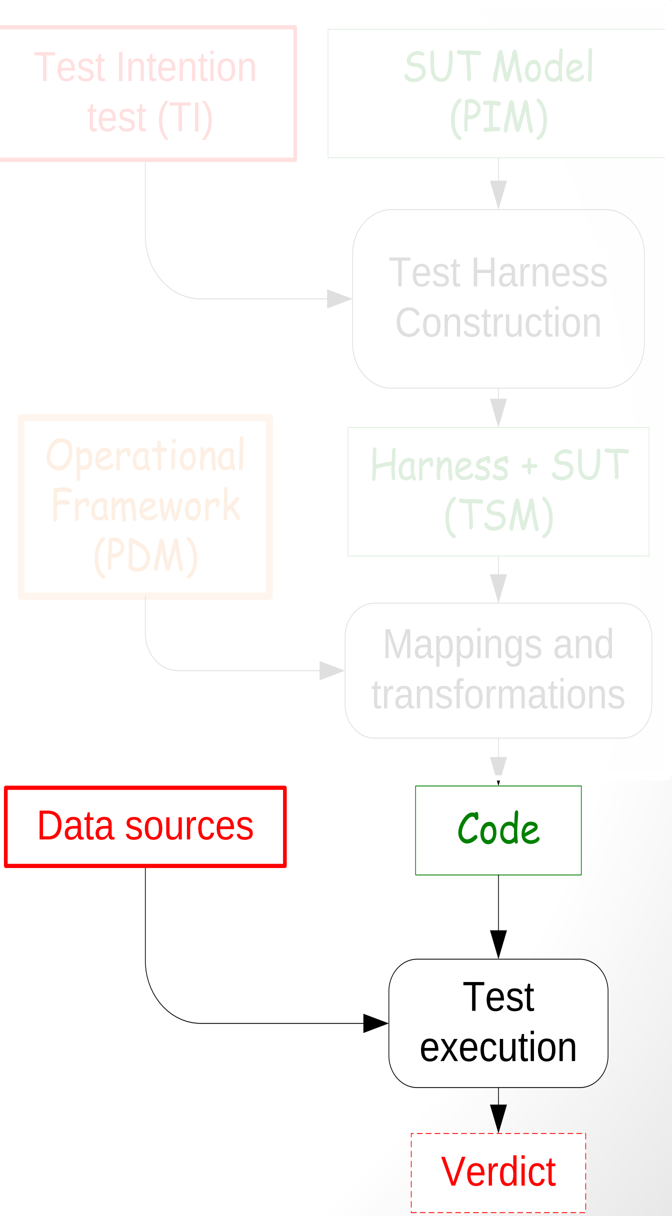


Running the tests

- Test harness construction
  - from SUT model
  - from a test intention
  - to Harness + SUT
- Mappings and transformations
  - from operational framework
  - Mappings
  - Transformations
    - generate Java code
- Test execution
  - from Data Sources

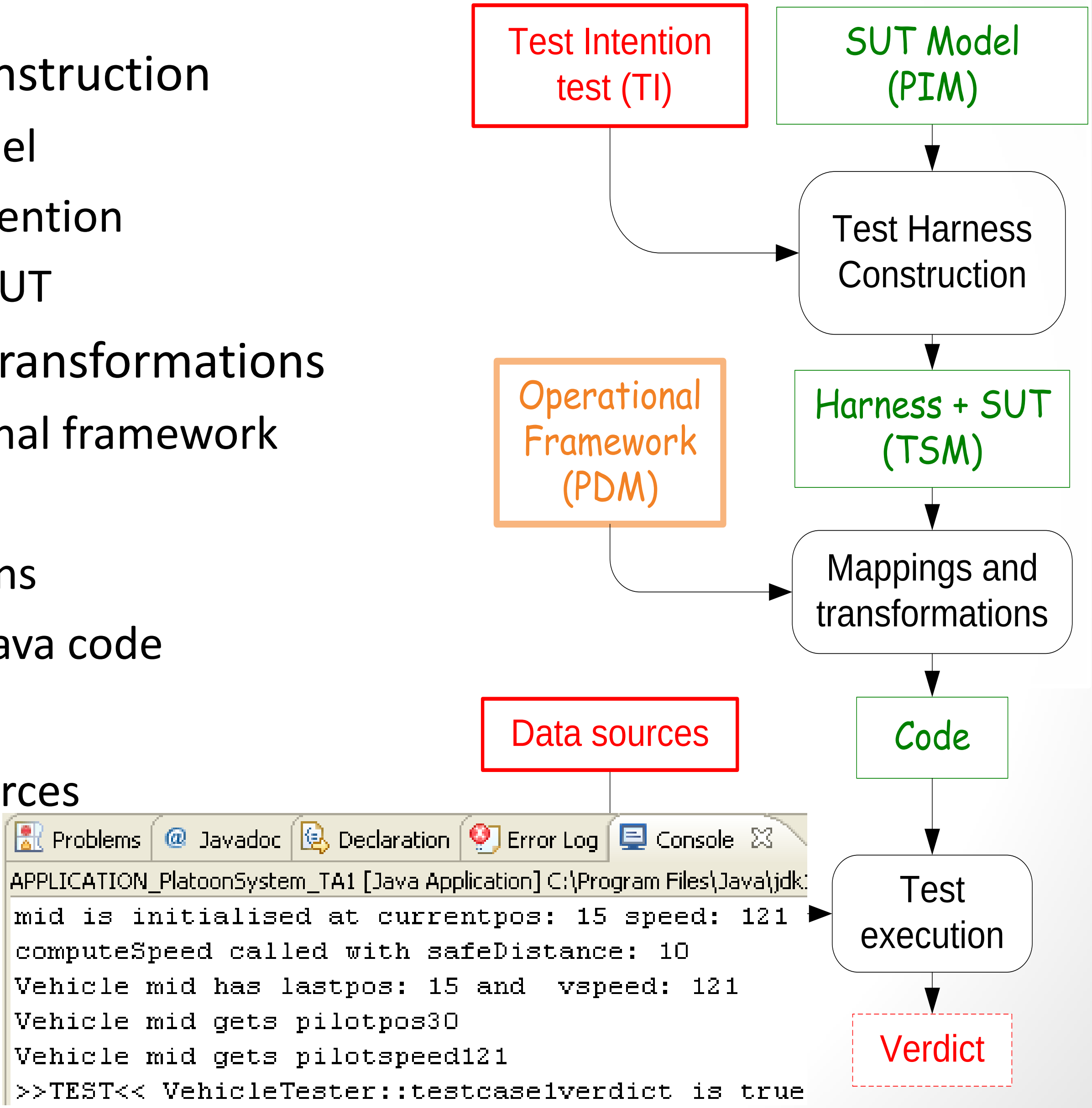


```
pilotpos=30
lastpos=15
safeDistance=10
vspeed=121
pilotspeed=121
oracledata=130
```



Running the tests

- Test harness construction
  - from SUT model
  - from a test intention
  - to Harness + SUT
- Mappings and transformations
  - from operational framework
  - Mappings
  - Transformations
    - generate Java code
- Test execution
  - from Data Sources
  - to Verdict

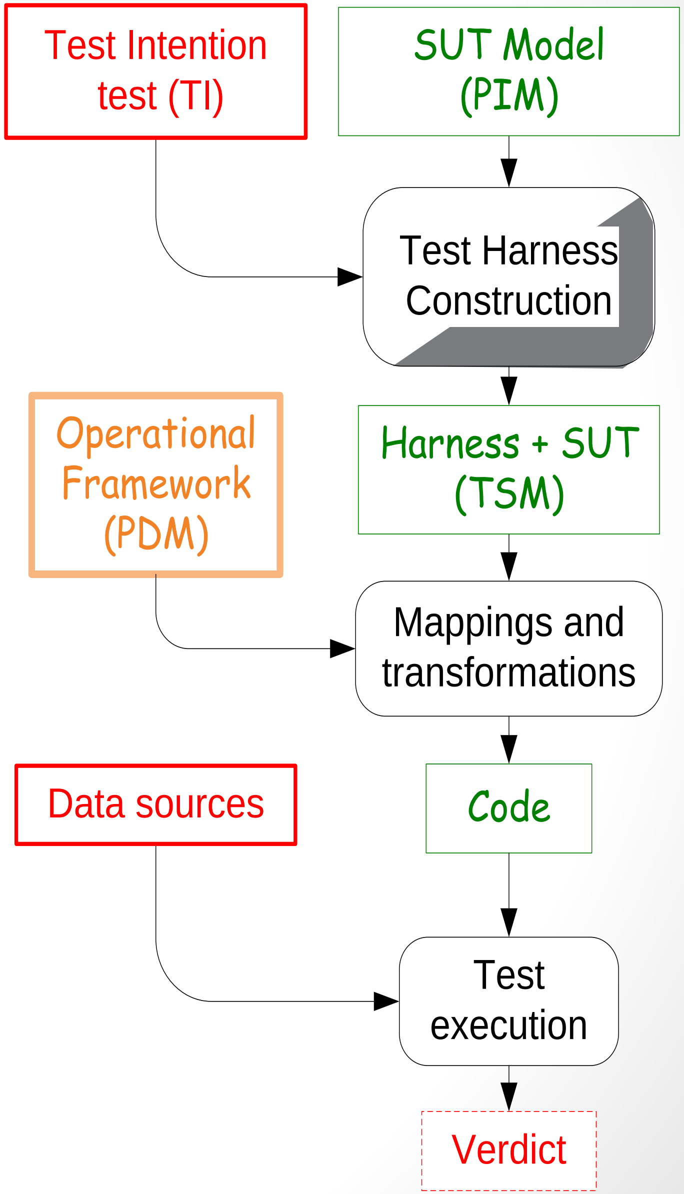


```
APPLICATION_PlatoonSystem_TA1 [Java Application] C:\Program Files\Java\jdk
mid is initialised at currentpos: 15 speed: 121
computeSpeed called with safeDistance: 10
Vehicle mid has lastpos: 15 and  vspeed: 121
Vehicle mid gets pilotpos30
Vehicle mid gets pilotspeed121
>>TEST<< VehicleTester::testcase1verdict is true
```

Assisting the test harness building

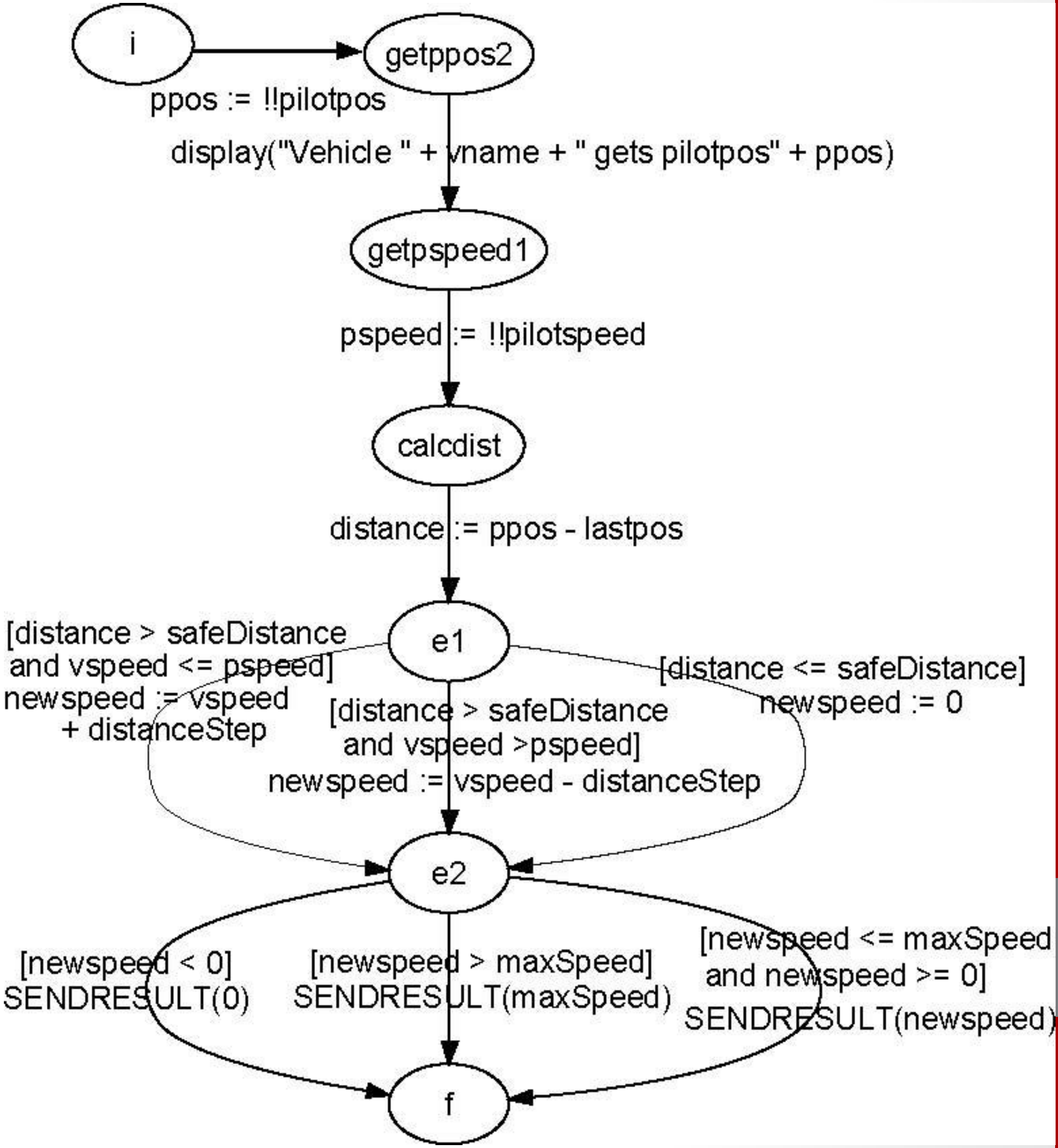
Running the tests

- Most tool and its transformations have been developed
- Applicability dependent on the SUT Model
  - Modeling the behavior in the model.
- Developments are dedicated to Kmelia
  - Could be ported to rCOS, SOFA1



Running the tests

- In Kmelia, behavior is modelized with automata



Conclusion

- We describe a method to integrate testing early in an MDD process
  - by designing test artefacts as models
- We assist the tester in building component test harnesses
- We are developing a prototype with Kmelia
  - A language with behavioral description at the model level can support the approach.
- We plan
  - to finish the developments and
  - to experiment the approach
- We are interested in considering techniques to build and qualify tests at the model level
  - e.g. Mutation Analysis

UNIVERSITÉ DE NANTES ECOLE DES MINES DE NANTES

Building Test Harness from Service-based Component Models

Pascal André, Jean-Marie Mottu, and Gilles Ardourel

AeLoS Team, University of Nantes, France

LABORATOIRE D'INFORMATIQUE DE NANTES ATLANTIQUE