

A Lightweight Automated Test Framework based on UML 2.0 Testing Profile for Component Testing

Mustafa DURSUN

**Workshop on Real-time, Embedded
and Enterprise-Scale Time-Critical Systems**
May 24-26, 2010, Washington, DC, USA

- Motivation
- Component-Based Software Development
- Component-Model and ASELSAN's Component-Based Development Practice
- UML Testing Profile
- Proposed Automated Test Framework for black-box testing of Embedded Components
- Summary

- Component-Based software development decreases software development time for many variants of products by using reusable components as building blocks.
- Model-based development of reusable components promotes cost-efficient development of component-based software systems.
- Validating reusable components with black-box testing much earlier in the development lifecycle increases reliability of reusable software components.
- Model-driven testing makes testing easier by use of a common language with application developers
- UML Testing Profile enables the use of a common language for model-driven testing throughout the development process.
- Automated test framework based on UML Testing Profile can significantly improves productivity by reducing manual model inspection.

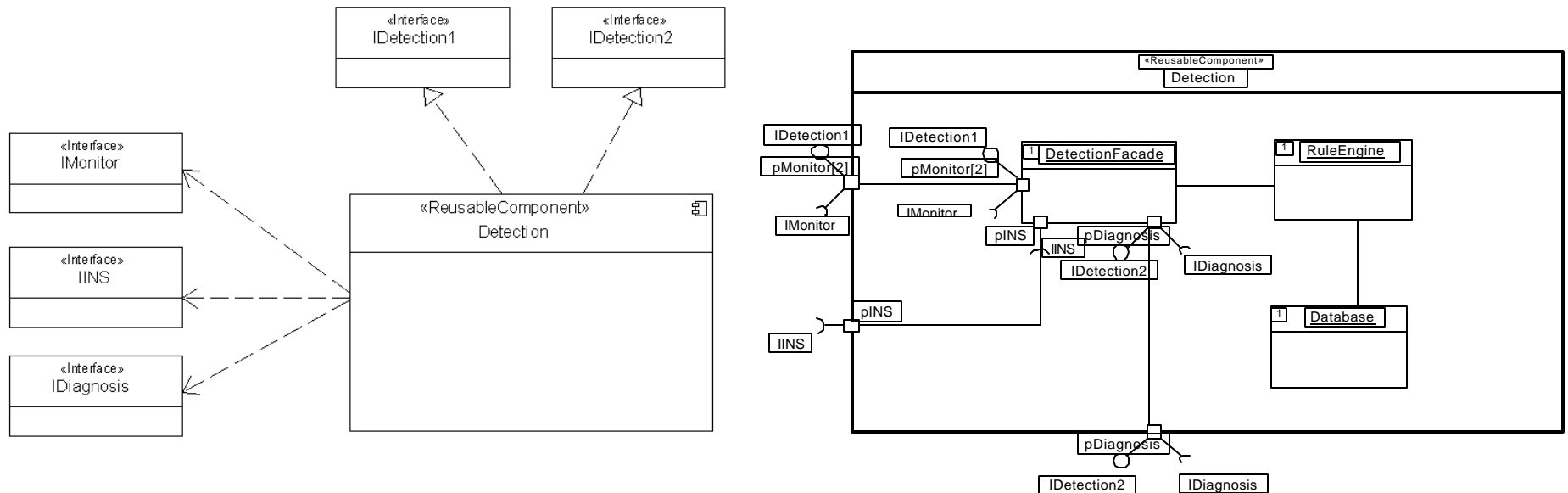
- Component-Based Development is an approach to software development in which all artifacts can be built by assembling, adapting, and binding - wiring together existing components into a variety of configurations.
- The main motivation of Component – Based Software Development (CBSD) is decreasing software development time for many variants of products

- A component should be independent from the other part of a system.
- A component should be used as a black-box.
- A component should provide and require interfaces.
 - Provided interfaces of component determines the external behavior and features of the component and allows the component to be used as a black box.
 - Required interfaces are necessary to specify contracts of components and independent development of components in object-oriented programming languages.
- A component should be composed with other components and replaceable in a composition.

- It is possible to mention three types of binding time:
 - Compile-time
 - Link-time
 - Run-time
- In general, earlier binding times enable better static analysis and system optimization, whereas later binding times enable configuration by users and post deployment updates.
- Earlier binding is often appropriate for constrained systems such as embedded systems.

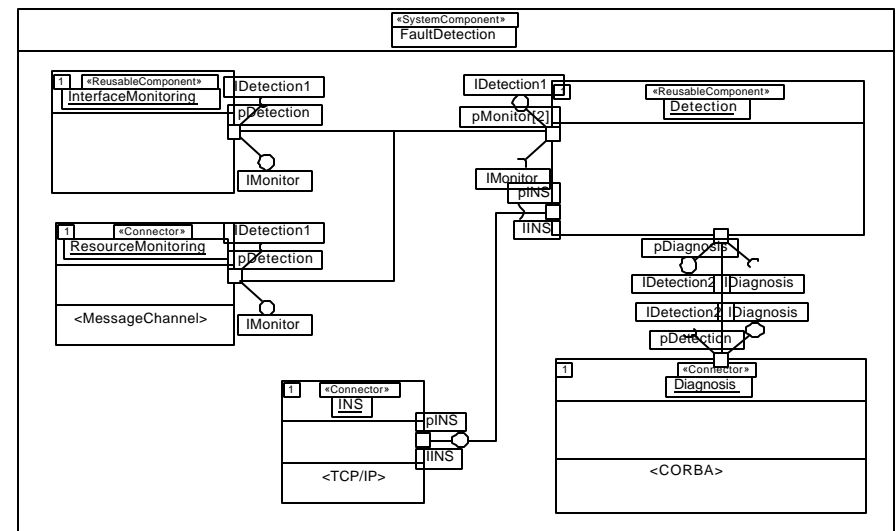
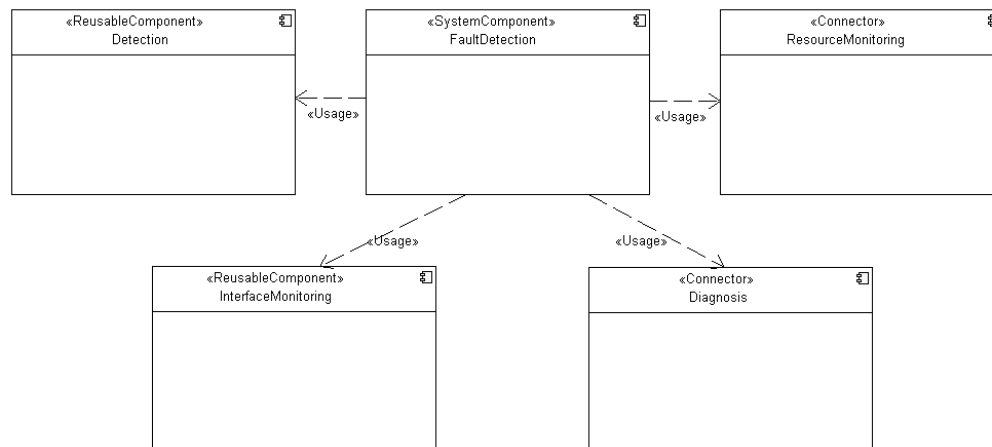
- Component model restrict composition to link time.
- Defines 3 type of components
 - Reusable Component
 - Connector
 - System Component

- Reusable components are hierarchical
- Reusable components are not distributed



- Connectors documented as reusable components
- Connectors implements proxy pattern for communication between distributed components

- System components are distributed

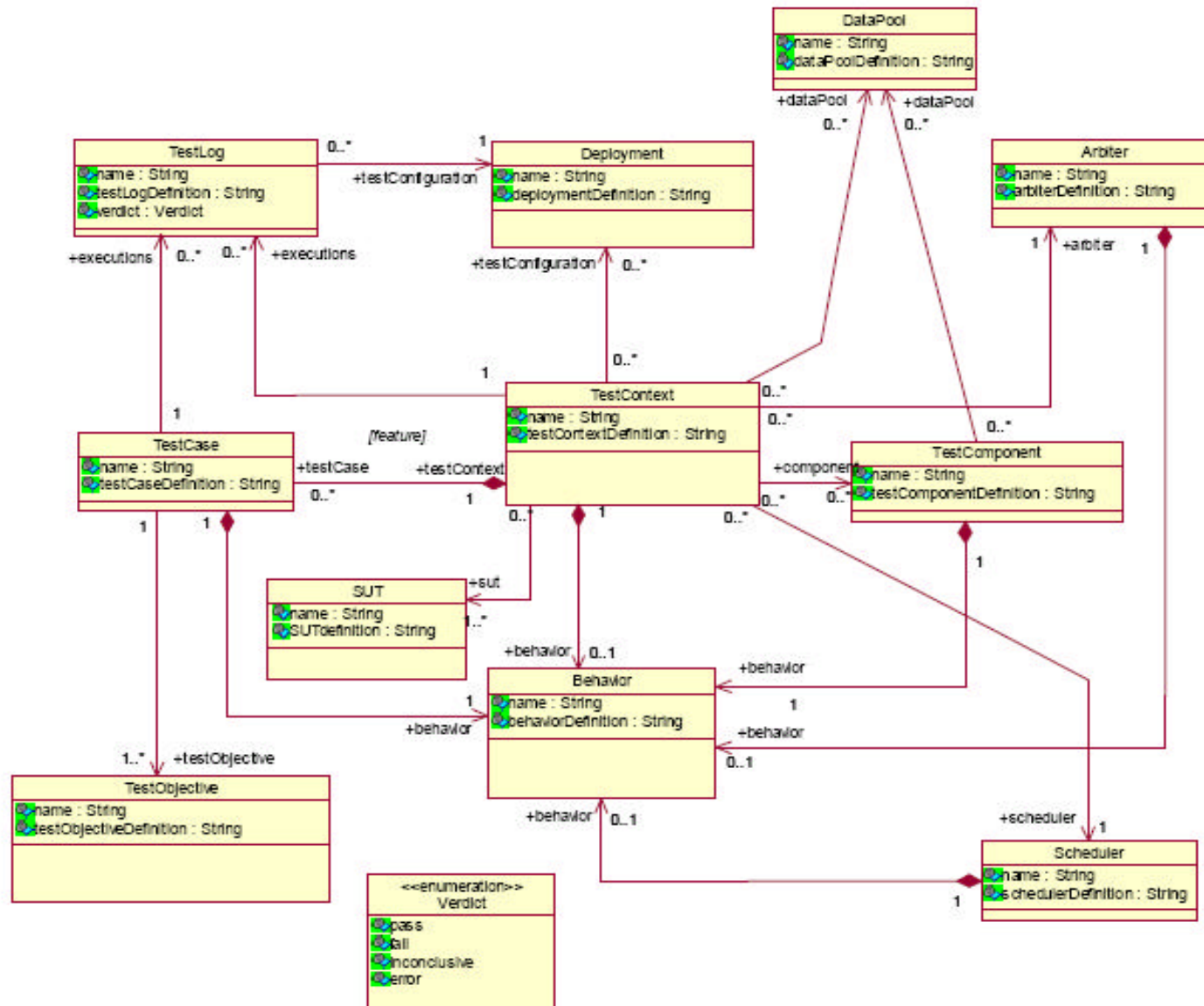


Agile development of components:

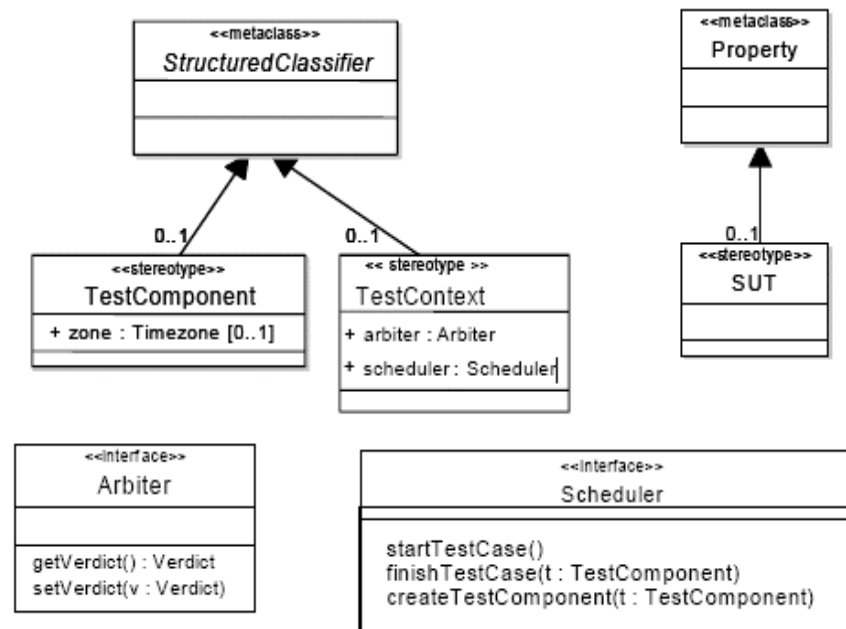
1. Stories for an iteration are identified by customer.
2. Component developers is organized to develop software components with pair programming.
3. The component software is continuously rebuilt and unit tested.
4. Customer makes functional black-box component testing.
5. At the end of each iteration, baseline is created for the component.

- The UML Testing Profile defines a language for designing, visualizing, specifying, analyzing, constructing, and documenting the artifacts of test systems.
- The UML Testing Profile is organized in four logical groups of concepts:
 - Test architecture defining concepts related to test structure and test configuration.
 - Test data defining concepts for test data used in test procedures.
 - Test behavior defining concepts related to the dynamic aspects of test procedures.
 - Test time defining concepts for a time quantified definition of test procedures.

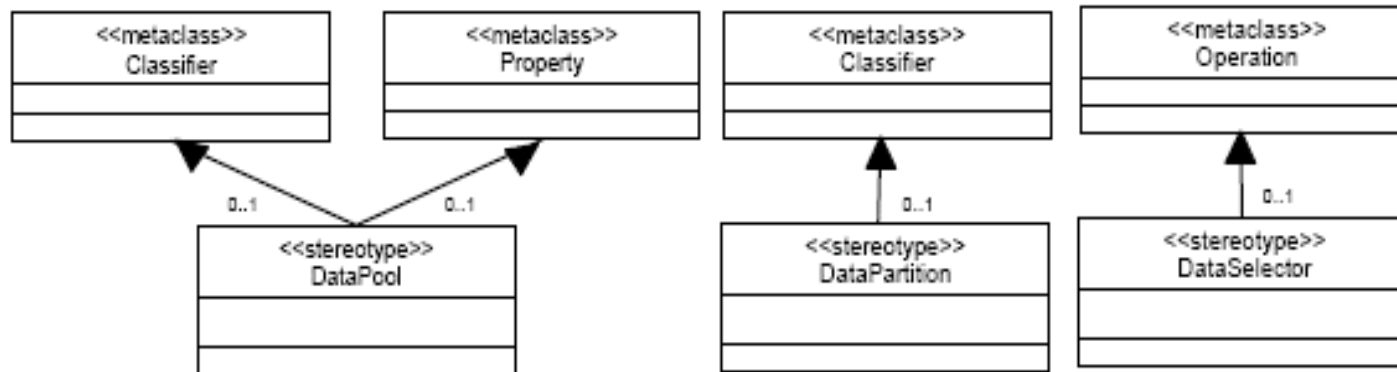
Test Architecture and Test Behavior



- *Test Context, Test Configuration*
- *System Under Test (SUT)*
- Test Component
- Arbiter
- Scheduler

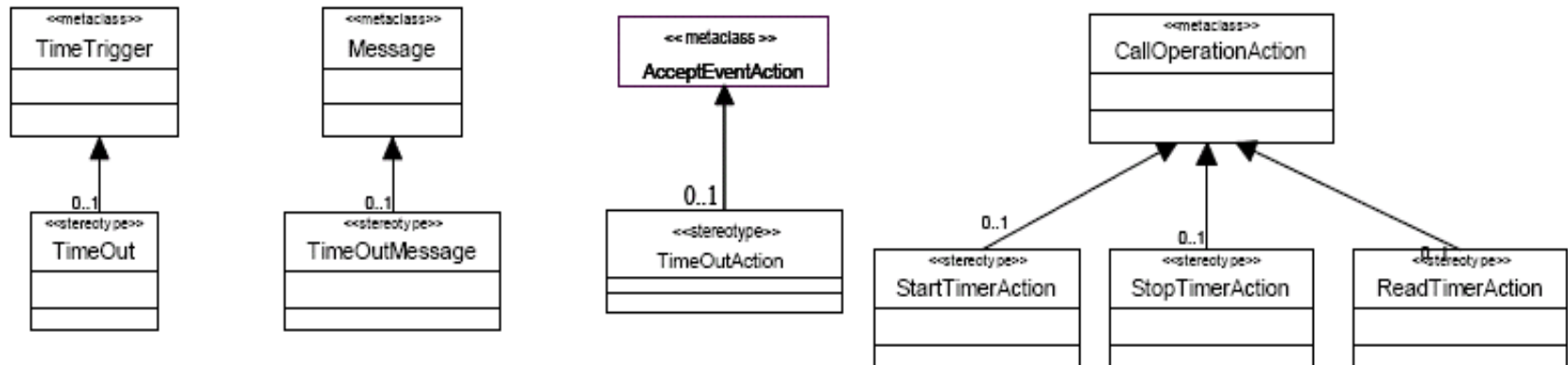


- Data Pool
- Data Partition
- Data Selector
- Coding Rule



- *Test objective*
- *Test case*
- *Defaults*
- *Verdict*
- *Validation action*
- *Finish Action*

- *Timers*
- *Time zones*



- Automated Test Framework provides systematic test execution control based on UML Testing Profile to run tests and validate test results
- Automated Test Framework provides solution for Message Sequence, and User-oriented component testing
- Test automation focus on a component model to support its related testing activities for different projects.

- SUT is the reusable component that is documented as a structured class with well defined interfaces.
- Test Components are defined as objects. Each test component has a port with complement interfaces of SUT.
- Test context is a system component whose parts are the SUT and test components.
- Scheduler and arbiter are implemented by Test context.

- Data Partitions are classes that are to store the test data defined in test context for the each messages defined in the SUT interfaces.
- Data Partitions keep
 - Stimuli Data: Stimuli message arguments data values and return data values of required interface messages.
 - Expected Data values: return data value of Stimuli message and arguments data values of required interface messages.
- Expected Data Partition defines:
 - Expected messages
 - Parameter values of the messages
 - The number of calls of a message
 - The order of the messages

- Data Pool contains data partitions that are associated with a test context and its test cases.
- Data Selector is a operation that is defined in Data Pool. Test components reaches the test data using data selectors.

- Test Cases define:
 - Stimuli message
 - Expected Message Sequence reaction to the stimuli message
 - Test Data Values; Stimuli, Expectation Data values
- Test Context creates data partitions for each test data values in the data pool.



- Test components responsible to stimulate SUT.
- Test components responsible for validation action by validating:
 - the value returned from the SUT stimuli message
 - SUT calls messages with expected parameter values
 - SUT calls messages in expected order
 - SUT calls messages in expected number
 - SUT calls messages in expected time
- Test components set verdict of the arbiter after validation actions.
- Test components responsible to perform finish action

- A test component performs validation action
 - At the instant of a message call from SUT
 - by comparing observed return value with the expected return value of the stimulus message call.
 - by comparing the observed order of message call with the expected order of message call.
 - by comparing the observed message information that is gained from the message call made by SUT in run-time, with the expectation message information that is defined in the test case.
 - At the end of time-out
 - by comparing observed number of calls of a message in the expected time with the expected number of calls of the message.

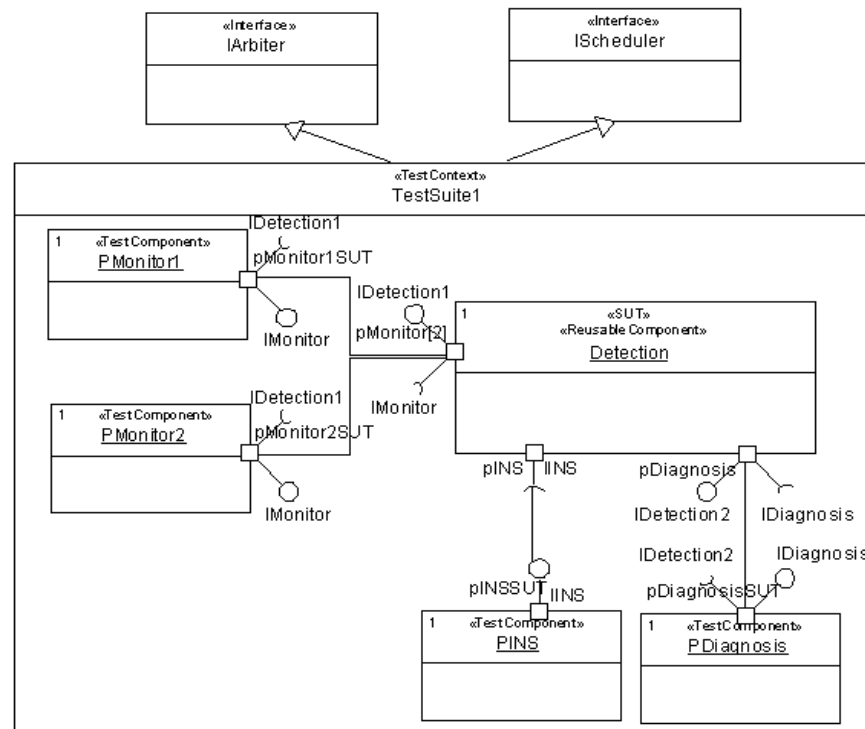
- Observability of a message: It is not always possible to know the correct order/number of message call made by SUT. In these cases the message should not be observable. By this way a test component does not compare the order or number of message calls.

- A test component performs finish action after
 - Instantaneous validation action if validation action fails.
 - Time-out validation action regardless of the validation action value.

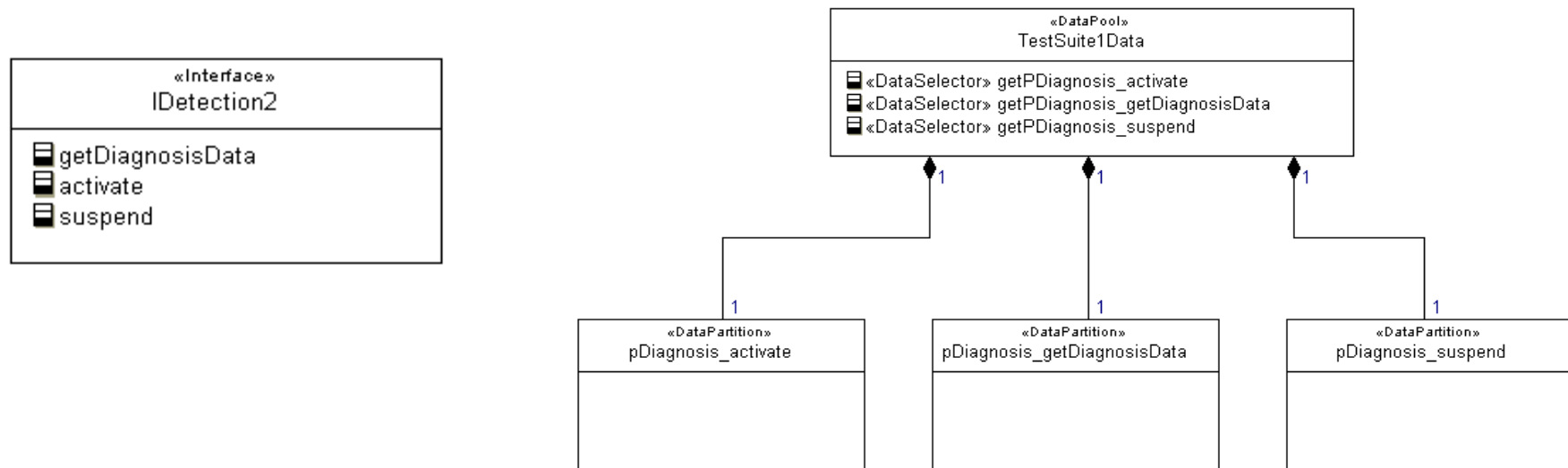
| Test Architecture | Test Data | Test Behavior | Test Time |
|-------------------|----------------|-------------------|-----------|
| SUT | Data Pool | Test Case | Timer |
| Test Context | Data Partition | Test Objective | |
| Test Component | Data Selector | Validation Action | |
| Arbiter | | Finish Action | |
| Scheduler | | Verdict | |

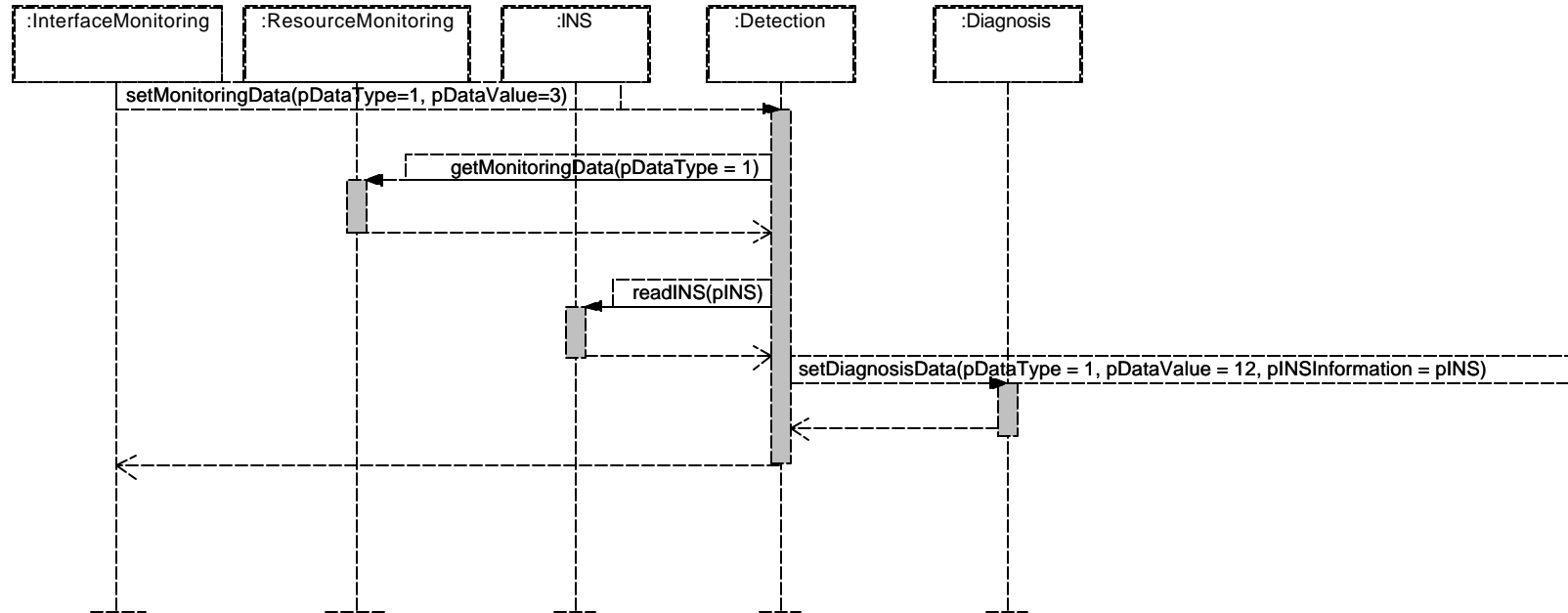
 Automated Elements
 Manuel Elements

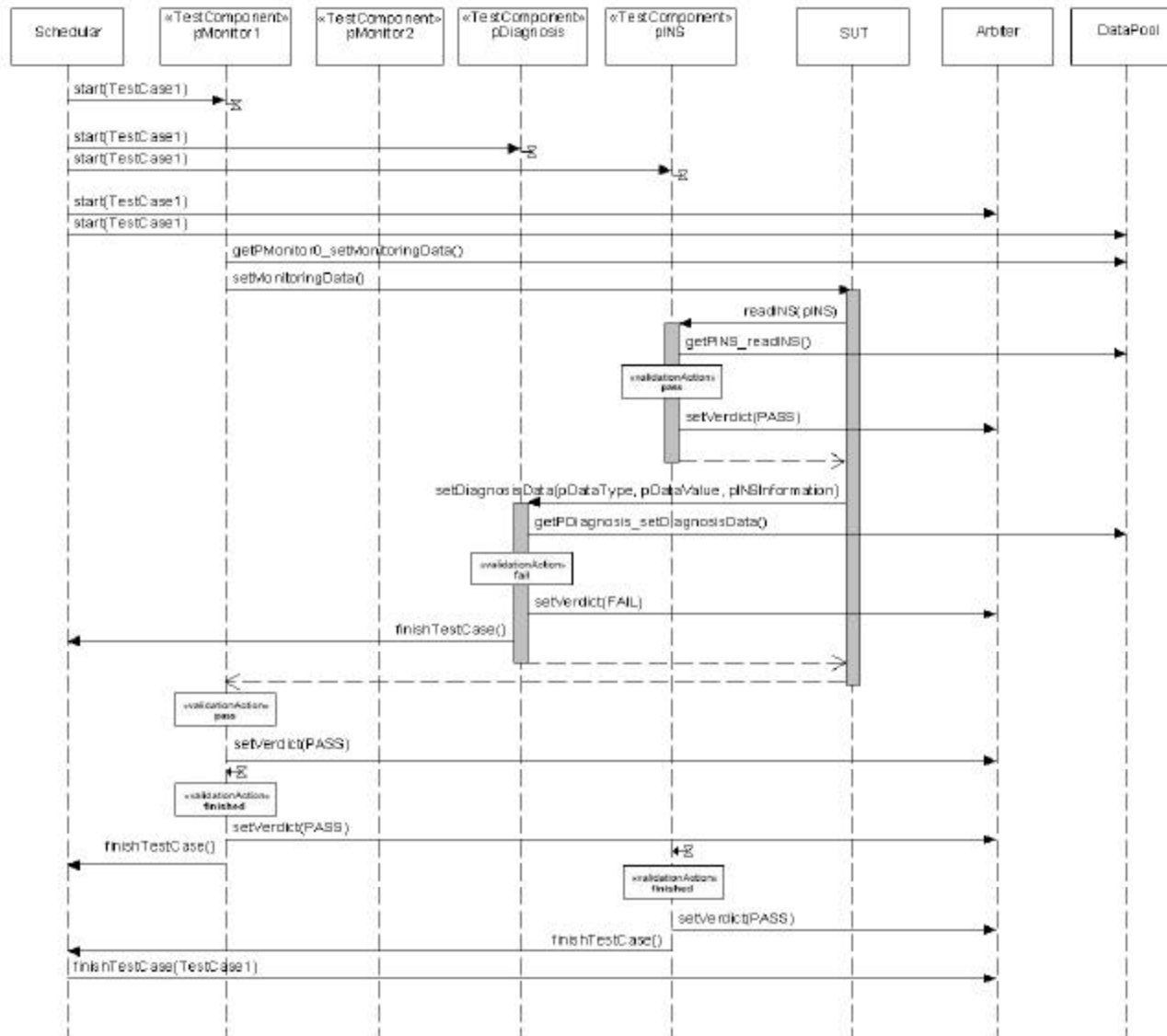
- For each port of SUT a test component is created.
- Test context implements IScheduler, and IArbiter.
- Connections of SUT, test components, IScheduler, and IArbiter are created in test context



- A data pool is created for each test context.
- A data partition is created for each message of an interface.
- A data selector is created as an operation of data pool for a data partition.
- Test Context creates data partition objects for each test cases.







- ATF allows model-driven functional black-box testing compatible with UML Testing Profile for reusable embedded components.
- ATF allows validating reusable components for application developers by using common modeling language.
- ATF increases testability of reusable components by abstracting testers from implementation details of ports and structure classes.
- ATF improves productivity by reusing same framework for different projects developed with the same component-model.

Thanks for your attention!

Sorular?
(Questions?)