A test driven development of MAS

Juan Pablo Paz Grau – LG CNS Colombia

Andrés Castillo Sanz – Universidad Pontificia de Salamanca campus Madrid

Agenda

1. Introduction
2. Test Driven MAS Development
3. Reference Multi Agent System
4. The MAS Testing Toolkit
5. Remarks on Test Driven Development of MAS
6. Conclusions

Introduction

•Software testing is the most important task in software engineering

•The ability to introduce software testing in the implementation phase of a software lifecycle is a determinant CSF (Critical Success Factor)

- •To cut down costs
- •To cut down time
- •To guarantee quality assurance

•Nevertheless, the link from requirement analysis to software testing is recognized, but MAS methodologies partially address it

Introduction

- **Testing frameworks are usually tied to a specific MAS methodology**
  - They usually require the construction of complex XML documents depicting testing activities

- **Agents communicate by message passing instead of method invocation**
  - Current OO testing methodologies are not directly applicable to MAS
  - Agents cooperate but are autonomous, that is, they may run correctly when isolated, but incorrectly when deployed on an agent community

- **Some agents pose difficult challenges to make them "testable"**
  - *Smart agents* learn, producing different outputs between executions
  - *Scaled agents* (i.e. agents that do not interact with other agents) doesn´t give observable primitives to assert in a testing framework

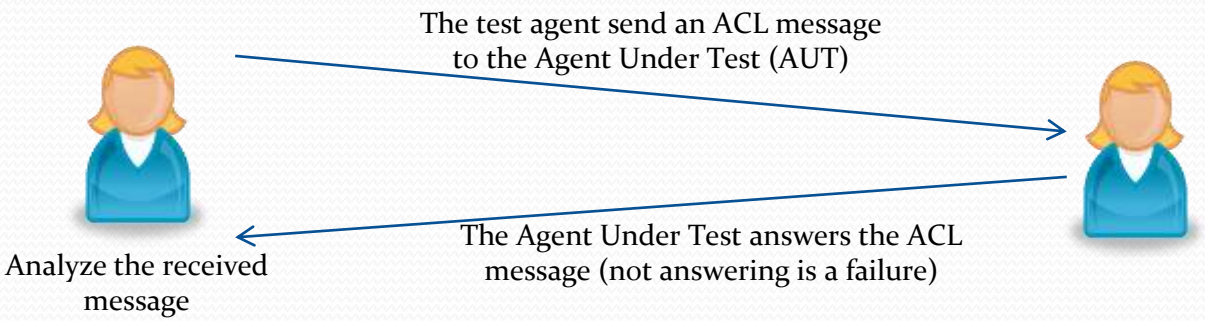
Test Driven MAS Development

The basics:

Agents communicate with its surrounding entities via message passing

- Development Constraint: Develop Polite Agents (agents that always respond to every message they receive)

- Then, to perform testing, we analyze the ACL messages received from an agent when a message is sent to it



Test Driven MAS Development

The basics

•Some agents can run correctly when isolated, but incorrectly when inserted into a community

- •It is needed to create a test scenario where mock agents mimic the agent´s community
- •Altough is a complex test scenario, with the “polite” constraint we ensure simpler test cases

Test Driven MAS Development

MAS Testing Approaches

Active Approach

- Based on test cases
- A set of test cases is extracted from analysis and design artifacts
- The agent environment is set to an initial state and then the agent response is evaluated after receiving an stimulus

Passive Approach

- Based on simulations
- Allows detection of abnormal behaviors while the simulation is running
- Need tools to visualize the agent platform interactions

Test Driven MAS Development

The testing approach should not be exclusive; both active and passive MAS testing should be used concurrently

Passive approach:

A set of test cases is derived from the analysis & design artifacts

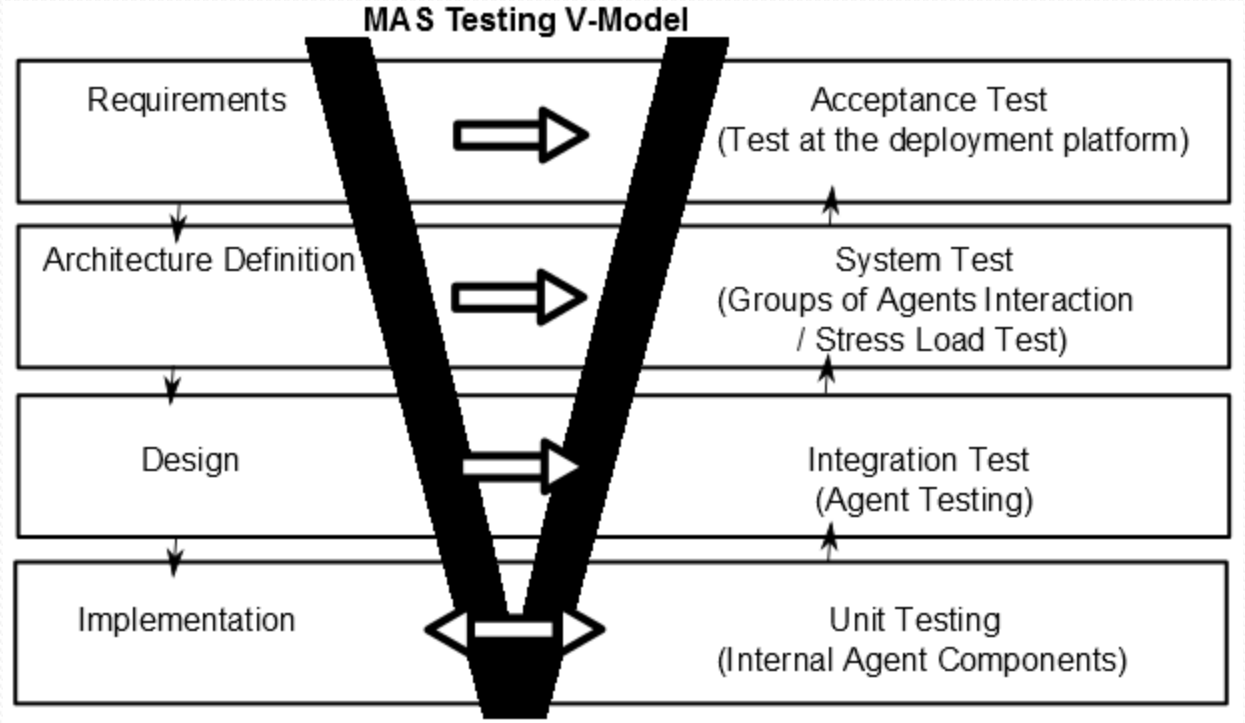
The agent´s environment is set to an initial state, and after stimulus, the agent response is evaluated

Active Approach:

During the test case execution, agent interactions are watched with monitoring tools

Test Driven MAS Development

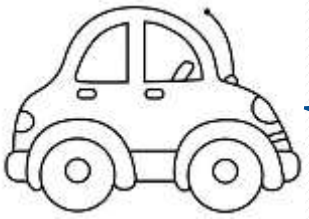
The reference model



Test Driven MAS Development

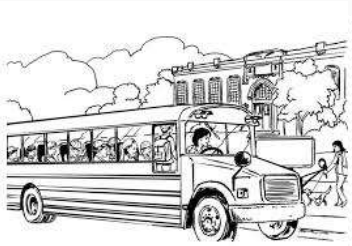
The V Model

- Simple and structured
- Contains all the testing phases
- Is easy to start from the "V" model and then extend to the targeted Agent framework / paradigm / platform



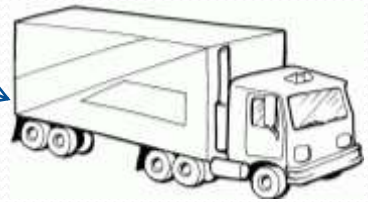
Simple Model

Easily Extensible!



Complex Model

Difficult to extend / customize



Test Driven MAS Development

Internal Agent Components Testing

- The agent is a complex set of objects integrated by a complex set of interactions
- The agent's integrating objects are unit tested before assembly
- The agent´s integrating objects have to meet their contracts

Test Driven MAS Development

Agent Testing

- Performed *in container* on a test agent platform
- The goal is to check that the agent fulfills its "Agent Contract"
- The agent is named "Agent Under Test" (AUT)
- Stub Agents and Mock Agents are used to established conversations with the AUT

Test Driven MAS Development

Stub/Test Agent

- Implement a set of test cases
- Analyzes the responses from the AUT to check if test cases are sucessful
- Can take adantage of mock agents to perform complex test cases
- Can be an incomplete implementation of an agent

Mock Agent

- Programatically sends/receives messages
- Do not analyze the messages received

Test Driven MAS Development

System Testing

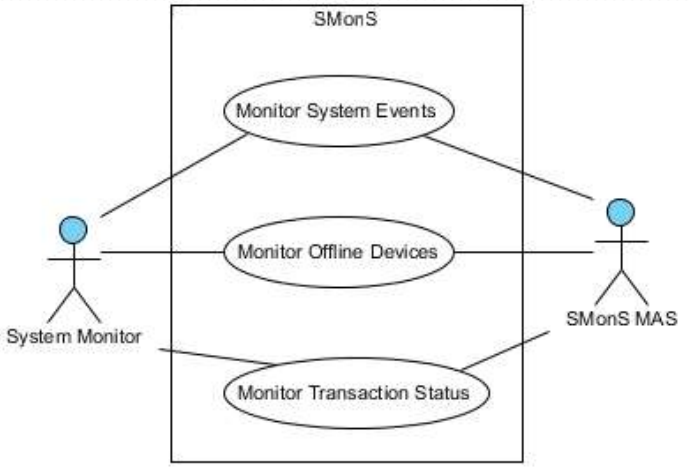
- A set of agents is deployed in the test agent platform
- The MAS is evaluated as a whole, not from a single agent point of view
- Quality properties can be evaluated at this stage

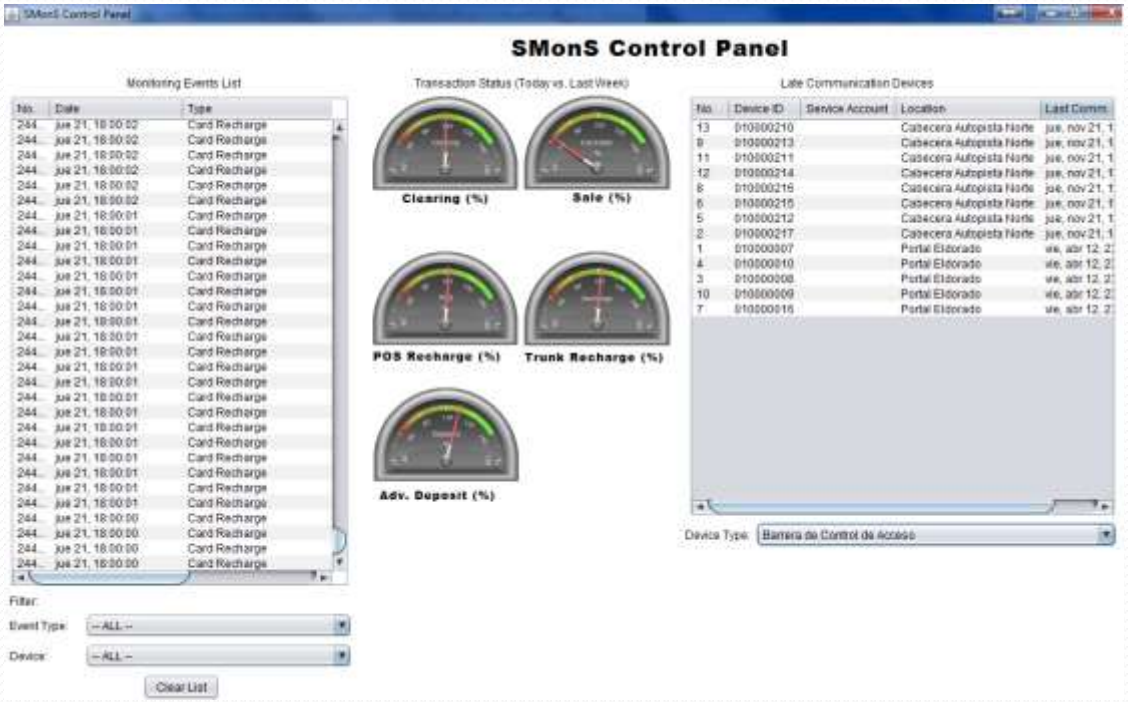
Test Driven MAS Development

Acceptance Test

- The MAS is deployed into an entrerprise architecture
- The enterprise architecture's components interact with the MAS
- It is asserted that the MAS comply with its use cases.

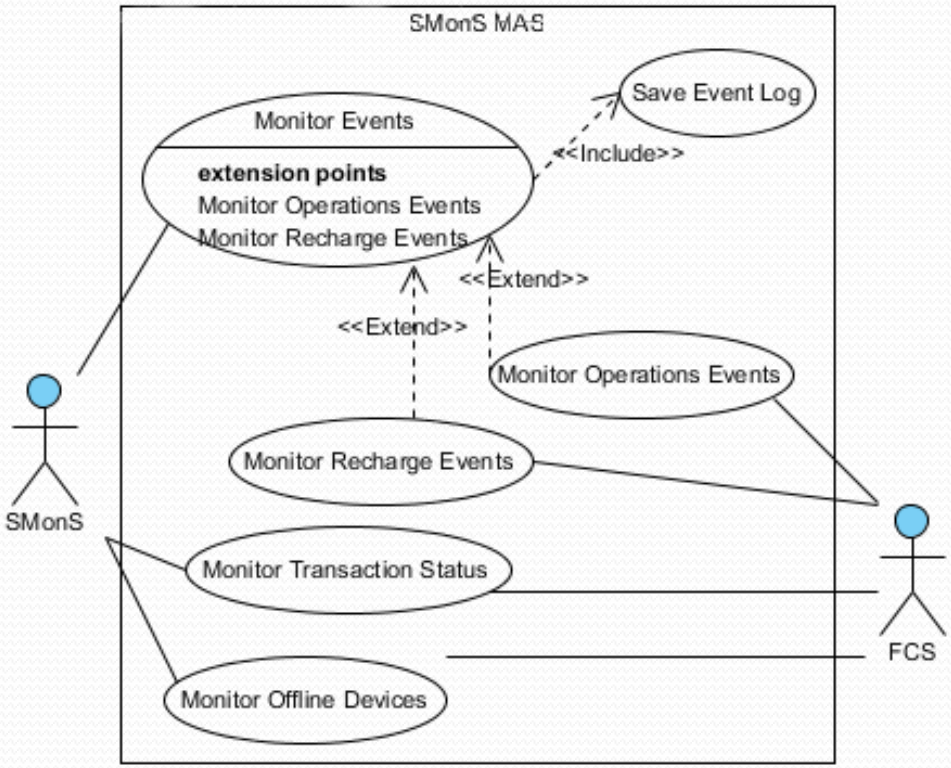
Reference MAS





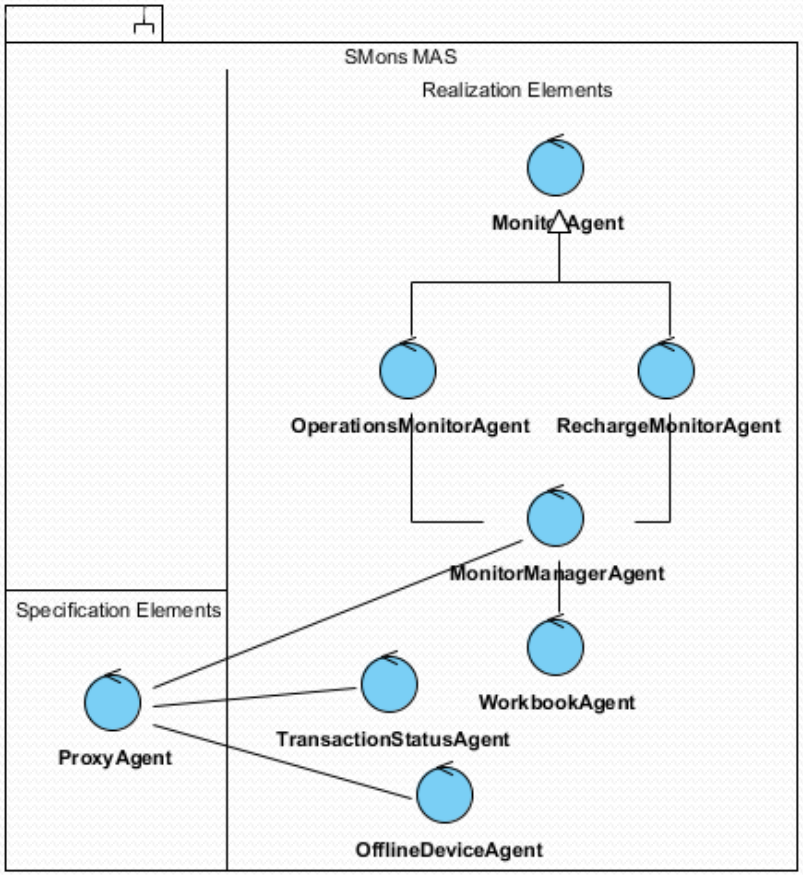
The SMonS System

Reference MAS



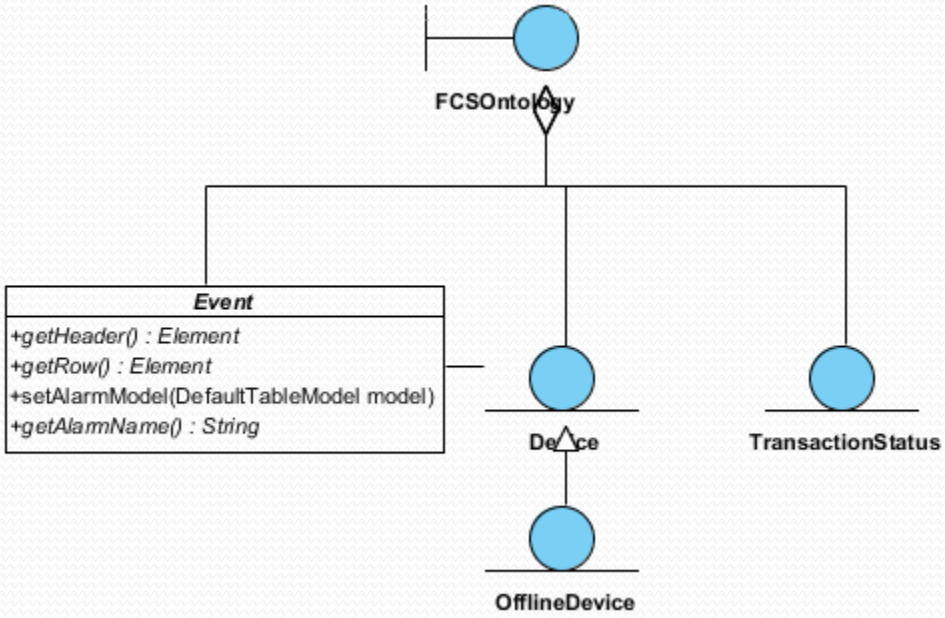
The MAS Use Case

Reference MAS



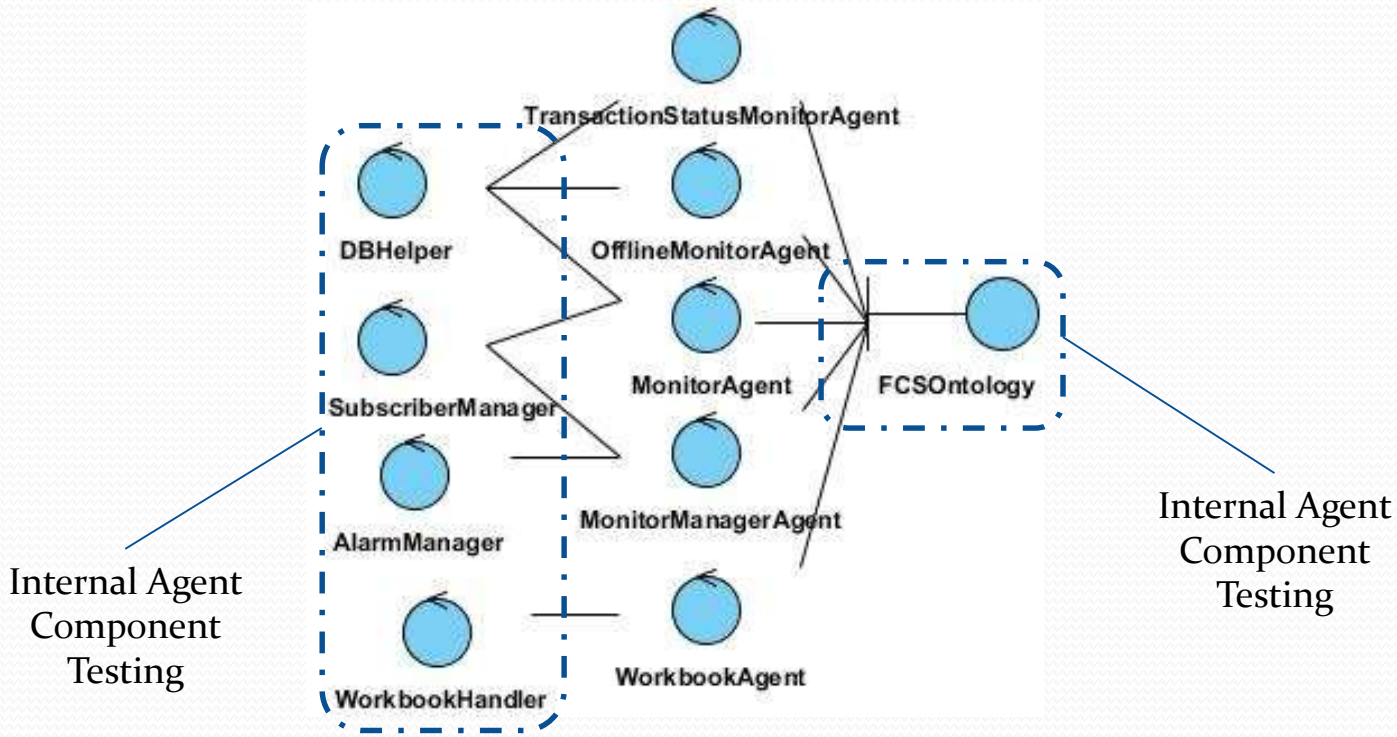
Agent Organization

Reference MAS



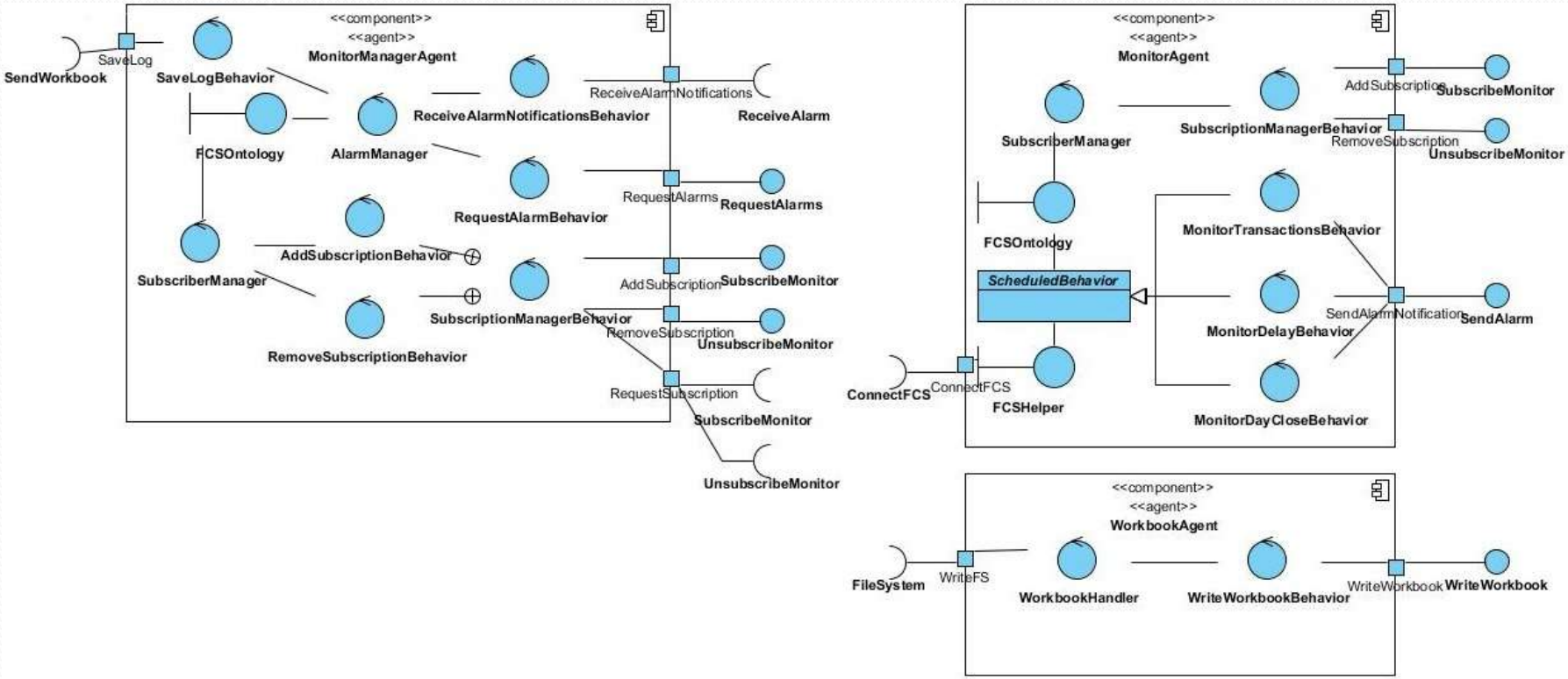
Agent Ontology

Reference MAS



Agent Model (Simplified)

Reference MAS



Agent Model

The MAS Testing Toolkit

We developed a toolkit to aid us in performing the TDD strategy

- The tools were extended from the Junit framework
- The aim was to facilitate coding test cases

Components

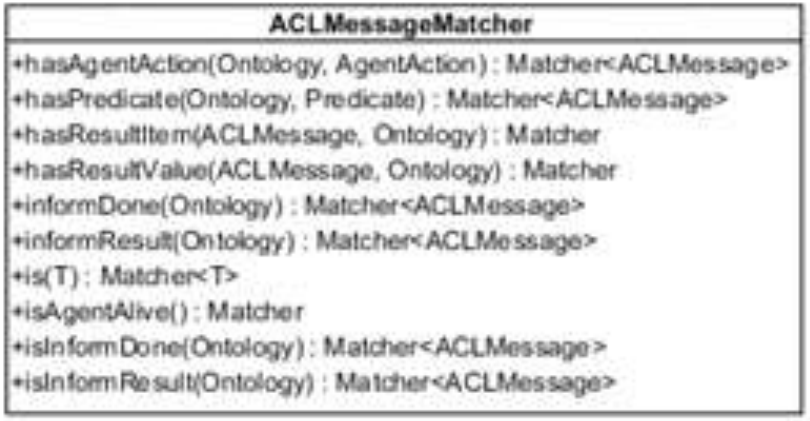
1. A set of ACL Message Matchers
2. The Mock Agent

The MAS Testing Toolkit

The ACL Message matchers

•Junit already implement a vast set of matchers on its libraries

•But, although, it facilitates extending the matchers library



•Basic matchers (default matchers in Junit) can be seamlessly combined with the custom matchers to structure complex assertions

•We rely on default matchers to assert basic ACL features (ACL slot values).

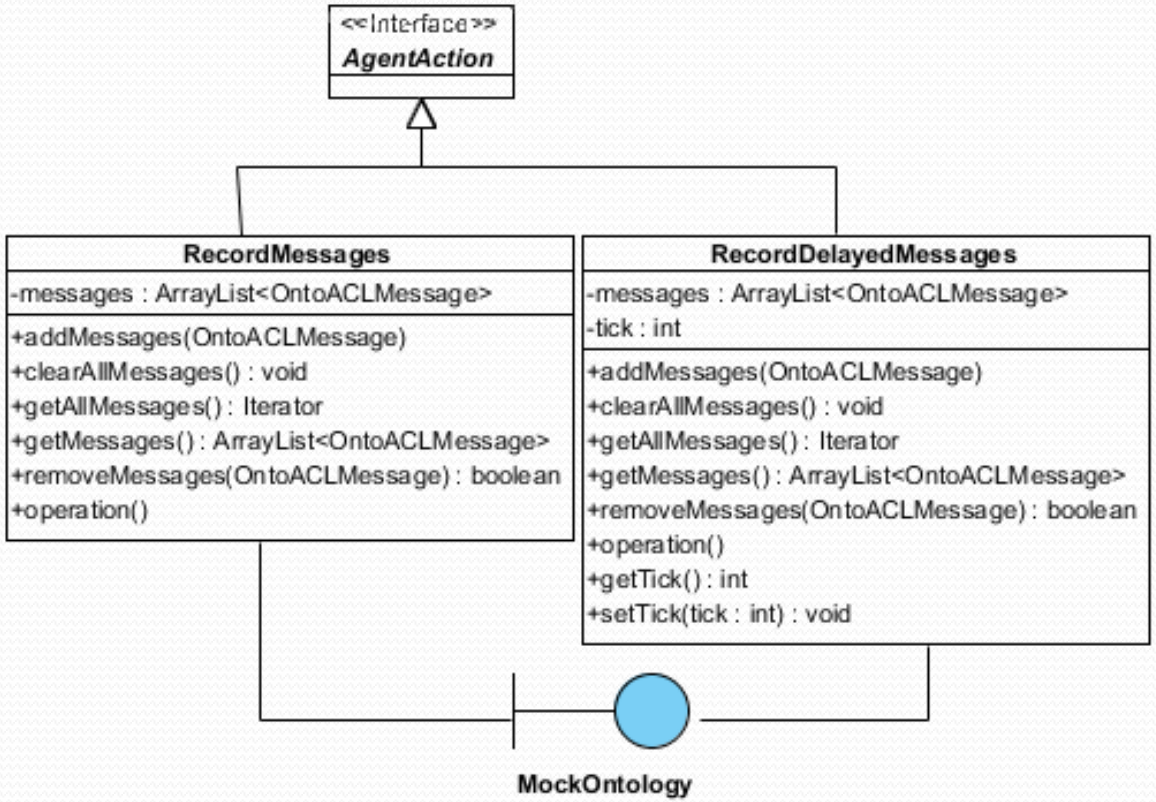
The MAS Testing Toolkit

The Mock Agent infrastructure

1. The Mock Ontology
2. The Mock Agent Gateway
3. The Mock Agent

The MAS Testing Toolkit

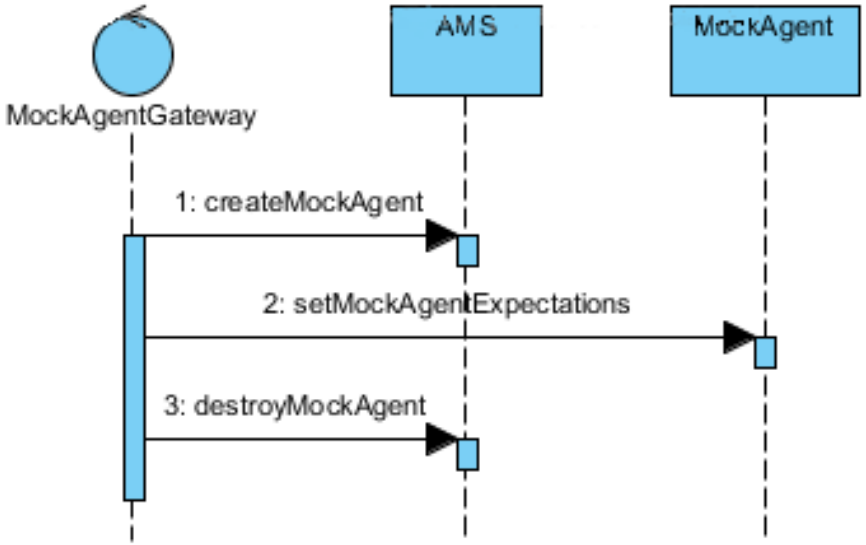
The Mock Ontology



The MAS Testing Toolkit

The Mock Agent Gateway

MockAgentGateway
-DELAYED_MESSAGES : int -MESSAGES : int
+createMockAgent(String, parameter) +destroyMockAgent(String) +getInstance(DynamicGateway) : MockAgentGateway +setMockAgentExpectations(String, ArrayList<ACLMessage>, int)

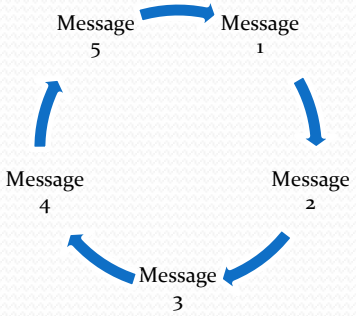


The MAS Testing Toolkit

The Mock Agent

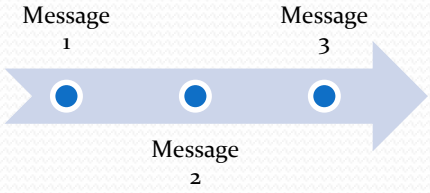
Circular Mode

- The agent is in listening mode
- When a message is received, it replies with the next message in its circular message queue



Delayed Message Mode

- The agent sends messages in defined time intervals
- When the last message in the list has been sent, it stops



Remarks on TDD of MAS

Test Plan Prerrequisites

1. Complete the design models of the MAS
2. Approve the MAS design diagrams
3. Design test cases for each agent or group of agents

Test Case Design

1. List all the roles played by the agent
2. For each role, list the agents the AUT interacts with
3. For each AUT interaction
    1. Design a positive test case
    2. List possible negative test cases
    3. Design negative test cases for each scenario

Remarks on TDD of MAS

Positive test cases

- Test goal fullfilment
- Conformance based
- Test whether the agent meets its “Agent Contract”

Negative test cases

- Ensures the agent informs of failure under certain situation
- Test the agent’s error management
- Fault directed

Remarks on TDD of MAS

Some negative test cases:

•Send the agent an ACL Message with FAILURE performative

•Send the agent an unexpected message according to the protocol, and expect as the response an ACL message with NOT UNDERSTOOD performative

•Subscribe / Unsubscribe from a previously subscribed / unsubscribed agent

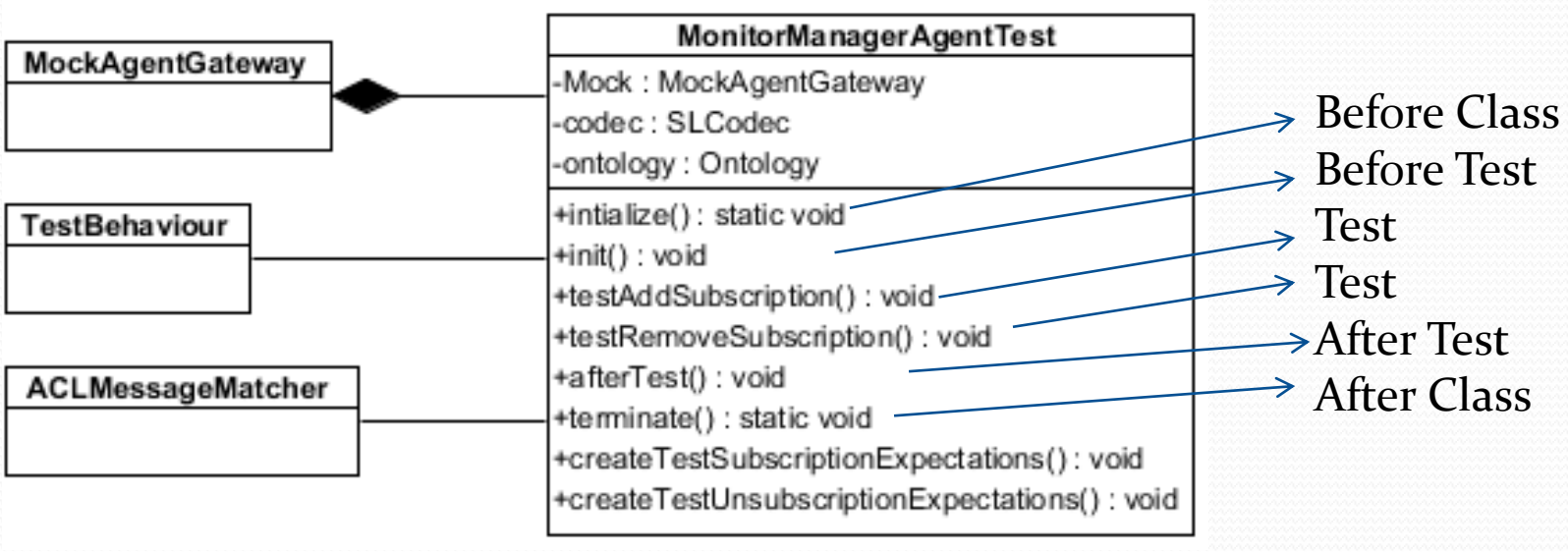
Remarks on TDD of MAS

The testing plan

Agent	Test Stage	Test Name
Transaction Status Monitor Agent	Agent Testing	testRequest testInvalidRequest testInvalidPerformativeRequest
Offline Device Monitor Agent	Agent Testing	testRequest testInvalidRequest testInvalidPerformativeRequest
Workbook Agent	Internal Agent Components Agent Testing	testWorkbookHandler testRequest testInvalidRequest testInvalidPerformativeRequest
Monitor Manager Agent / Monitor Agent	System testing	testAddSubscription testAddInvalidSubscription testRemoveSubscription testRemoveInvalidSubscription testExportAlarms testReceiveAlarm

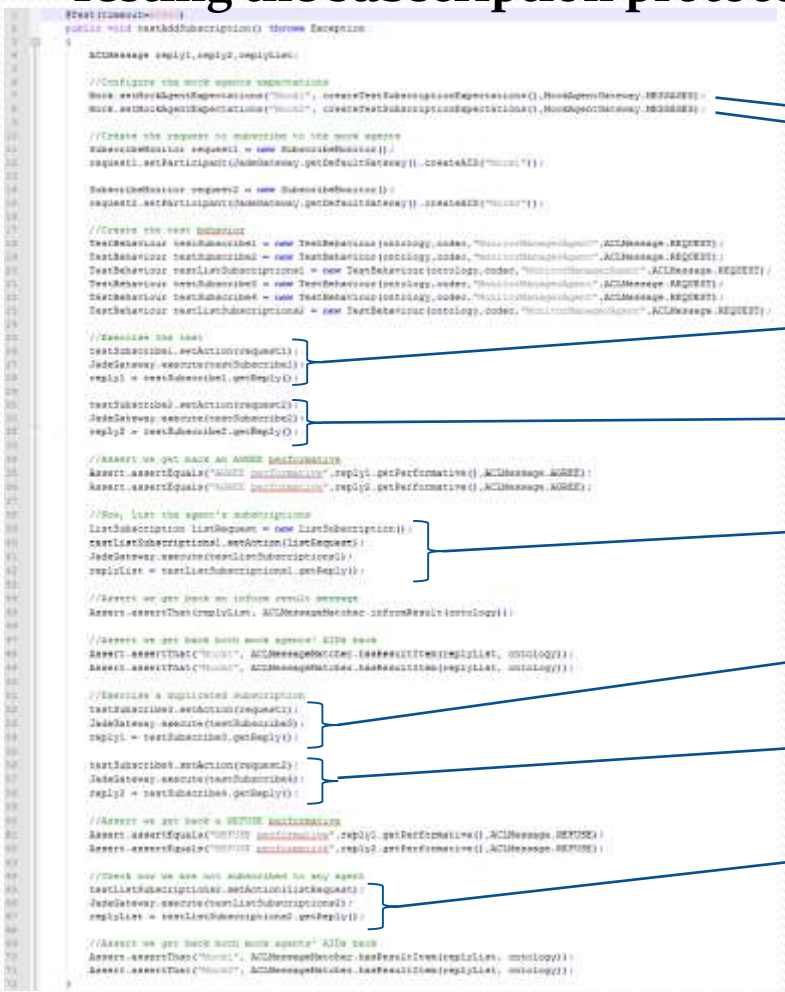
Remarks on TDD of MAS

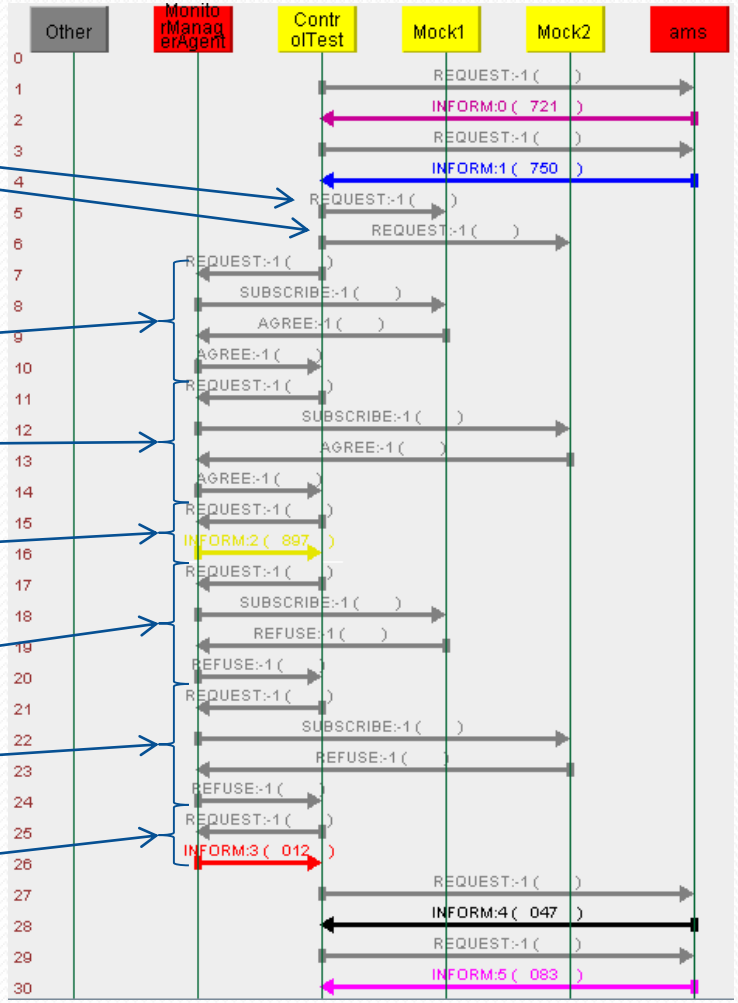
Testing the subscription protocol (Test Configuration)



Remarks on TDD of MAS

Testing the subscription protocol





Remarks on TDD of MAS

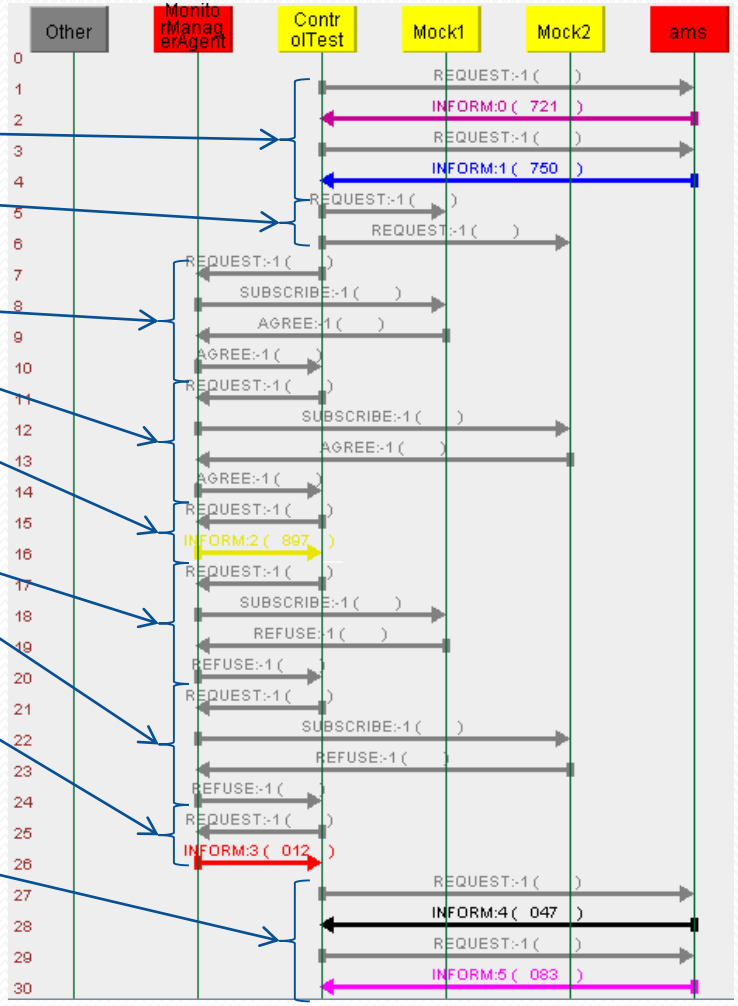
Testing the subscription protocol

1. Create Mock Agents
2. Setup the mock agents

3. Subscription to the first mock agent
4. Subscription to the second mock agent
5. Inquiry the list of subscribed agents

6. Subscription to the first mock agent
7. Subscription to the second mock agent
8. Inquiry the list of subscribed agents

9. Destroy Mock Agents



Remarks on TDD of MAS

Testing the subscription protocol

- Both passive and active testing are used in conjunction
- Positive and negative test cases
- System improvement from TDD
  - To request a subscription to another agent was not in the initial design (Initial design was to automatically search the DF on agent startup)
  - To inquiry the list of subscribed agents was not in the initial design

Conclusions

•We presented a testing approach for MAS avoiding making reference to any development paragidm or MAS framework

•Our goal is that the testing method can be applied to any type of MAS system without regard of the design methodology of platform

•We presented a set of testing to follow while testing MAS via the “V” model

•Also, the blueprints for the testing toolkit we developed were presented to allow other researchers to implement and extend their functionality

Thanks for your attendance! Any questions?