Automated Unit Testing For Agent Systems

(Agent-Oriented Software Engineering)

Zhiyong Zhang (zhzhang@cs.rmit.edu.au)
John Thangarajah (johthan@cs.rmit.edu.au)
Lin Padgham (linpa@cs.rmit.edu.au)
Overview & Background

A testing framework for agent based systems

- Agent-based systems
  - Agent: Autonomous, Interaction, Dynamic environment
  - Testing principles from traditional area: model based testing, unit testing
  - Traditional methods are not enough:
    - Different Units (agent, event, plan vs. class)
    - Different relationships and faults

- Model based testing
  - Test systems using their design models
    - e.g. OOP systems ← UML design model
    - e.g. Agent based systems ← design model of development methodology (Prometheus (RMIT Agent Group))
Example of Agent

Agent: attend_enase07

Goal: lookfor_transport

Look for a transport way to attend enase07

Expected: **one plan** achieves the goal
Eg: *I am from Mel. ⇒ Plan3 is executed.*

Fault1: **no plan** is available

Fault2: **Both** Plan 1 and Plan 2 are applicable.

Plan1: `book_airticket`
Context() {in Europe; }
Body() {order(air-ticket);}

Plan2: `book_trainticket`
Context() {in Europe; }
Body() {order(train-ticket);}

Plan3: `book_interticket`
Context() {outside Europe; }
Body() {order(inter-ticket);}

Book an intercontinental ticket
Components of an agent are tested based on the design specification, to verify if they work as expected from the design.

- **event (percept/action/message)**: the component to trigger the choice and execution of a plan
  - Test: the triggering relation
- **plan**: the component that achieve one goal
  - Test: the properties of plans
- **Belief-set**: agent’s knowledge about the environment
  - Test: future work
II. What does the structure look like?

--section.3.1 of paper

- Test driver
  - Test Agent: *sends the testing request, gather the results, generate the reports*
  - Test-driver plan: *initiate the testing process*
- Subsystem under test:
  - Portion of the system
  - Unit to be tested and the relevant components

*Structure for plan test:*

![Diagram showing the structure for plan test]

- Test-Agent
  - Sends Activation_Message
  - Sends Finished_Message
- Test-Driver plan
  - Sends Activation_Message
- Subsystem under test
  - Event_1
  - Plan-1
III. Automated Testing Process

- Appropriate testing order $\leftarrow$ the Prometheus design specification

- The test cases are automatically generated.

- The test cases for all units are performed automatically following the testing order determined.

- The testing reports are automatically generated.
Testing Order

--section.4 of paper

Necessity: dependency between units

• The success of a plan depends on the success of sub-task plans
  – Ex1: P0 after P11, P12, P2

• Event’s triggering is influenced by plans’ unexpected faults
  – Ex1: e1 after P11, P12

• Cyclic plans should be tested
  – As a single unit
  – Before their parent plans (Ex2: Cycle(P1, P2, P3) before P0)
  – After their children (Ex2: Cycle(P1, P2, P3) after P4)

Ex1:

Ex2:

treated as a single unit
Generation of Test Cases
--section.5 of paper

• The generation of test cases for one unit
  – Extract the definitions of the variables that participate in the unit
  – Generate value combinations
    • Reduce the size using combinatorial design (Cohen et al., 1997)
  – One test case is generated for each value combination

• Execution of test cases following the order
Implementation & Case study

--section.6 of paper

• Implementation
  – Input: An agent system
    • Designed by Prometheus
    • Implemented by JACK (agent development tool by Agent Oriented Software Group)
  – Output: test reports
    • For each unit test
    • For the complete process

• Case Study & Evaluation
  – One sample system: *Electronic Bookstore* system
  – All type of faults are introduced
  – Result: all faults can be detected
Contribution

• A distinct testing framework for agent based system
  – Unit testing: test the components of an agent to verify if they work properly.
  – Fault-direct testing that is implementation independent
  – the use of model based testing to agent systems
  – An addition to the AOSE (Agent Oriented Software Engineering) methodology

--section.7 of paper
Future Work

--section.7 of paper

• More work on evaluation

• Integration testing of agent systems

• Embedding the testing functionality into PDT (Prometheus Design Tool)