Implementing and Evaluating a Runtime Conformance Checker for Mobile Agent Systems

Ahmad Saifan, Juergen Dingel, Jeremy Bradbury, Ernesto Posse

Presented by:
Ahmad Saifan

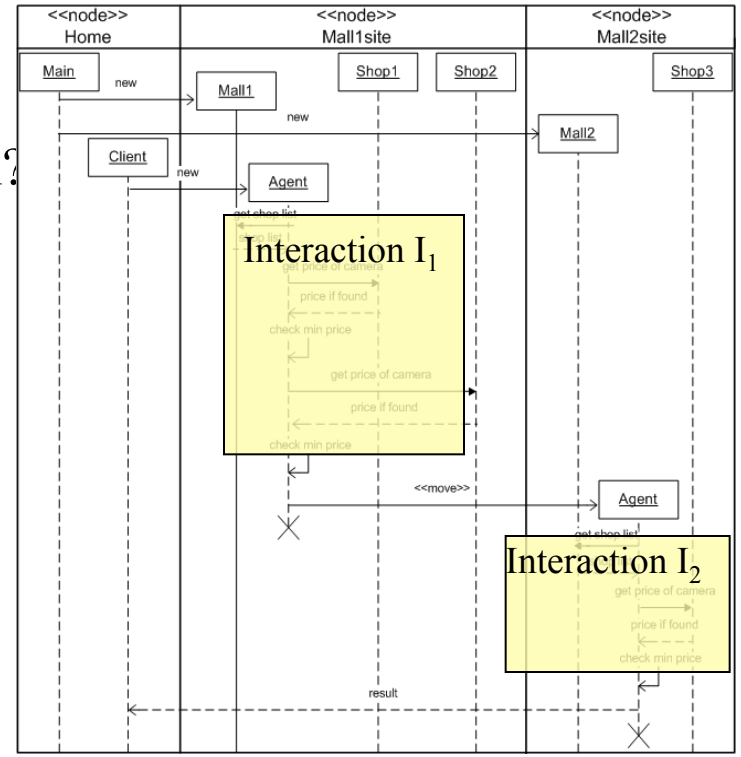
Tuesday, 5 June, 12

Faculty of Information Technology
Yarmouk University
Irbid, Jordan

Introduction

- Mobile Agent Systems (MASs) are special kind of distributed system (DS)
- In MAS, agent can move from one host to another
- Little existing work to support quality assurance techniques of MASs [Delamaro et al 03, Winikoff 08]

- The problem
  - Correctness of agent movement and interaction?
  - Do interactions yield expected results?
  - Constraints on: e.g.,
    - types of messages and data exchanged
    - order of messages
    - hosts visited
    - response times



Related Work

- **Model-based testing** [e.g., Utting 06, Brinksma et al 04]
  1. Build model
  2. Generate test cases
  3. Execute test cases
  4. Check conformance
- **Runtime monitoring**
  - Use run-time monitoring to check observed executions w.r.t. specification (e.g., Java-MaC [Kim et al 04], Java PathExplorer [Havelund et al 04])
  - Application domain (e.g., real-time systems, DS, sequential and concurrent systems, MAS)
- **Testing mobile code and agent systems**
  - E.g., JaBUTi/MA [Delamaro et al 03]
- **Our approach**
  - Assume high-level, executable specification (Kiltera)
  - Use run-time possibly distributed monitoring to check observed executions w.r.t. specification

Outline of Talk

1. Kiltera
2. Runtime conformance checking approach
3. Implementation and experimentation
   - Online shopping
4. Evaluation
   - Mobile agent mutation operators (IBM Aglets)
   - Mutation-based evaluation framework
   - Experiment & results
5. Conclusion

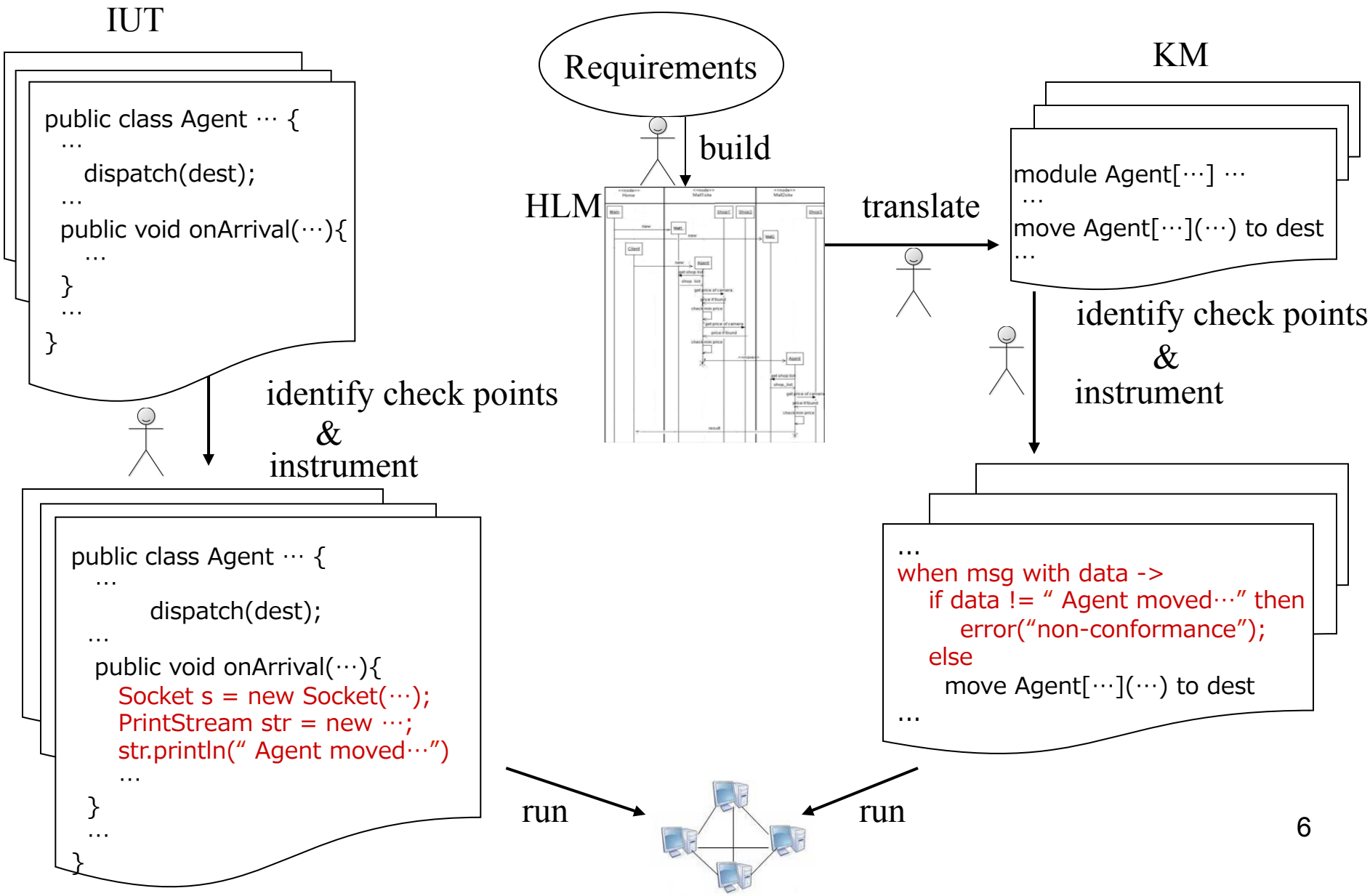
Kiltera

Kiltera: A language for concurrent, mobile, real-time, distributed computation, based on the pi-calculus. A kiltera system consists of concurrent processes which interact via message passing [Posse 08].

- Distribution:
  - Processes execute in *sites*
  - Behavior can be site-dependent
- Mobility:
  - Channel mobility: changing network topology
  - Process mobility: migrating to other sites
- Real-time:
  - Constructs to, e.g., measure wait-time, timeouts
  - Behavior can be timing-dependent

⇒ kiltera models are executable and analyzable

Our Approach: Overview



Implementation and Experimentation

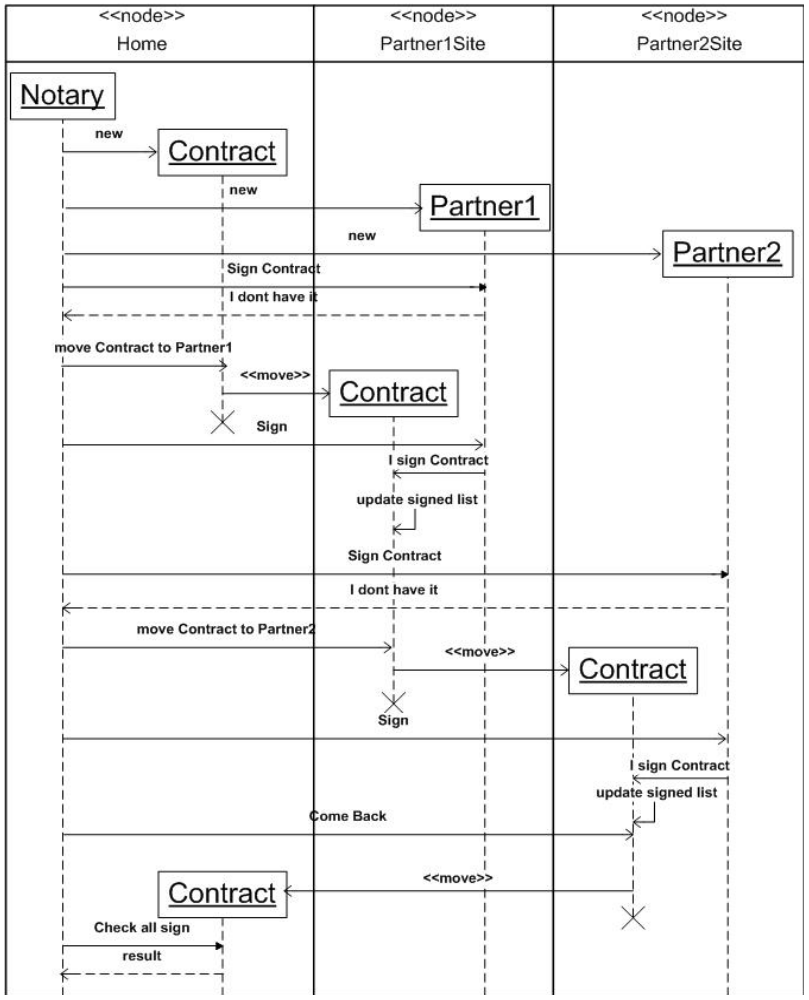
- Operational prototype
  - Use Java Aglets [IBM Tokyo Lab] for IUT
  - Use UML Agent profile for HLM [Kusek and Jezic 06]
  - Use Python connector to connect IUT with KM

- Case studies:
  - Online shopping
  - Contract signing
  - Token ring algorithm
  - Lamport's distributed mutex algorithm

Signing Contract

- Sample properties:
  - *"Agent moved to correct site"*
  - *"At the end all partners have signed the contract"*
- Observation:
  - Able to detect seeded faults
  - KM is considerably smaller than IUT

File Name	IUT	KM
Notary	87	30
Contract	132	28
Partner1	71	16
Partner2	71	16



Evaluation

- Designed and implemented mutation-based evaluation framework
  - Two phases:
    1. Mutant generation phase
    2. Execution phase
- Mutation Testing
  - A mutant is a slightly changed version of original program that arises through application of a mutation operator
  - Developed 29 mutation operators for Aglets in 6 different categories:
    1. Mobility
    2. Communication
    3. Run method
    4. Agent creation
    5. Event listeners
    6. Agent proxy

Mobile Agent Mutation Operators (Aglets)

Operator Category	Mutation Operator
Mobility	CDD: Change Dispatch Destination
	IDS: Insert Deactivate Statement
	RAPD: Remove Aglet Proxy from Dispatch statement
	RDS: Remove Dispatch Statement
	RDD: Replace Dispatch with Dispose
	RDR: Replace Dispatch to Retract
	SICD: Shrink ifelse Containing Dispatch
	SDS: Switch Dispatch Statement
Communication	CMP: Change Message Parameter
	CMOW: Change Message to One Way message
	RSMM: Remove Send Message Method
	MSKP: Modify SameKind Parameter
	MSR: Modify SendReply Parameter
	RPSM: Remove a Parameter from a Set of Methods
	RSRM: Remove SendReply Method
	MCMC: Move the Communication Method Calls in ifElse
	NAN: Notify All message to Notify message
Agent's Run Method	RMRM: ReMove Run Method
	RPRM: RePlace Run Method
Agent Creation	MICA: Modify Create Aglet Parameter
	MFCA: Modify the File name in Create Aglet
	ROCM: Replace onCreation with other Method
	ACON: Add clone method in onCreation
Event Listeners	RCBMN: Replace CallBack Method Name
	RARL: Replace Add listener with Remove Listener
Agent Proxy	RAID: Remove AgletProxy from getAgletID
	CPCN: Change Proxy name in getAgletClassName
	CSAP: Change the State in getAgletProxies
	CNP: Change Number of Proxies

Mobile Agent Mutation Operators (Aglets)

RDS: Remove Dispatch Statement:

Original Code

```
try{
    Contract.dispatch(dest);
} catch ...
```

Apply RDS Operator

```
try{
    /*removed dispatch*/
} catch ...
```

Operator Category	Mutation Operator
Mobility	CDD: Change Dispatch Destination
	IDS: Insert Deactivate Statement
	RAPD: Remove Aglet Proxy from Dispatch statement
	RDS: Remove Dispatch Statement
	RDD: Replace Dispatch with Dispose
	RDR: Replace Dispatch to Retract
	SICD: Shrink ifelse Containing Dispatch
	SDS: Switch Dispatch Statement
Communication	CMP: Change Message Parameter
	CMOW: Change Message to One Way message
	RSMM: Remove Send Message Method
	MSKP: Modify SameKind Parameter
	MSR: Modify SendReply Parameter
	RPSM: Remove a Parameter from a Set of Methods
	RSRM: Remove SendReply Method
	MCMC: Move the Communication Method Calls in ifElse
	NAN: Notify All message to Notify message
Agent's Run Method	RMRM: ReMove Run Method
	RPRM: RePlace Run Method
Agent Creation	MICA: Modify Create Aglet Parameter
	MFCA: Modify the File name in Create Aglet
	ROCM: Replace onCreation with other Method
	ACON: Add clone method in onCreation
Event Listeners	RCBMN: Replace CallBack Method Name
	RARL: Replace Add listener with Remove Listener
Agent Proxy	RAID: Remove AgletProxy from getAgletID
	CPCN: Change Proxy name in getAgletClassName
	CSAP: Change the State in getAgletProxies
	CNP: Change Number of Proxies

CSAP: Change the State in `getAgletProxies`

Original Code

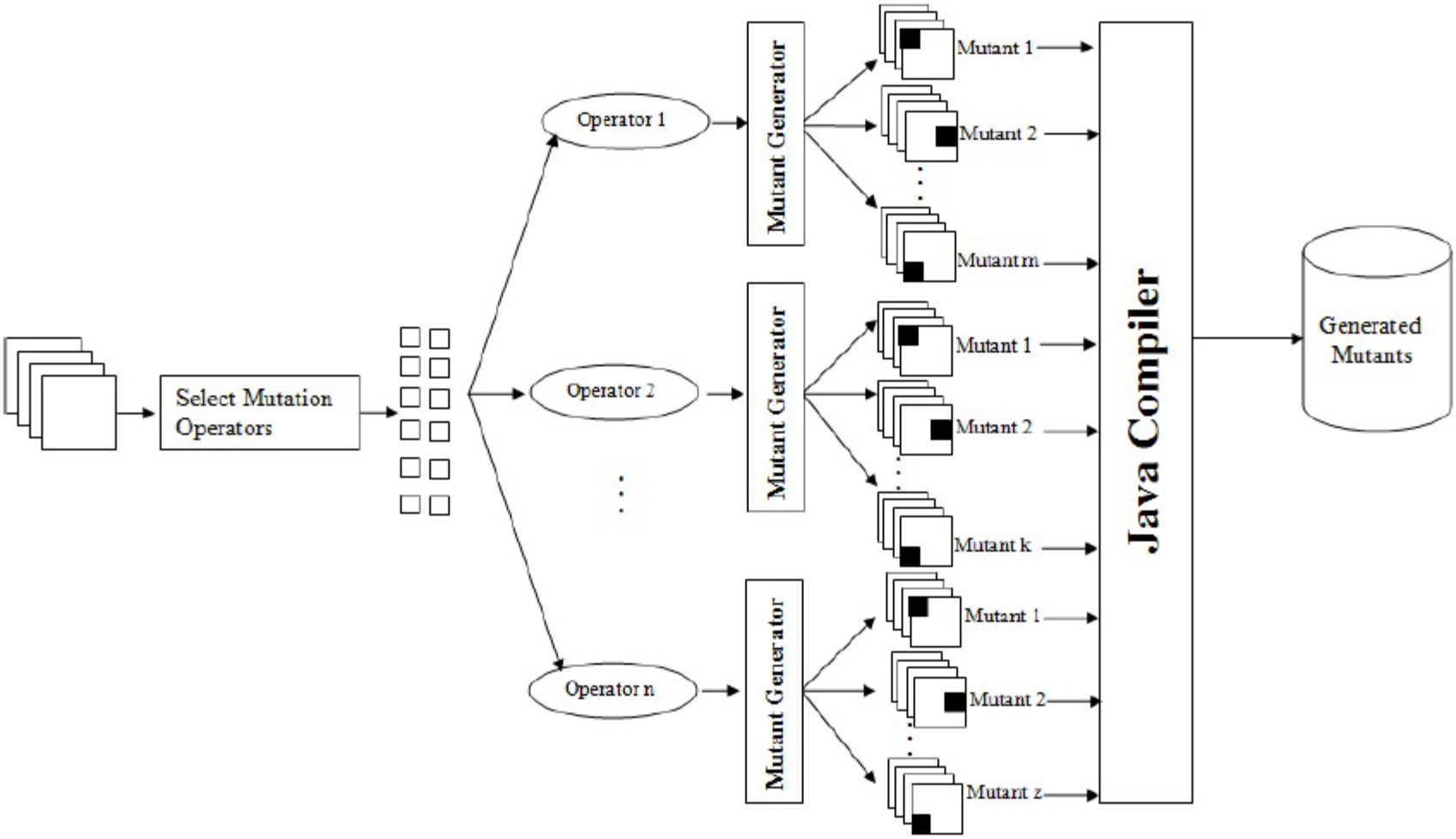
```
public Enumeration e;
...
e=AgletContext().getAgletProxies(ACTIVE);
```

Apply CSAP Operator

```
public Enumeration e;
...
e=AgletContext().getAgletProxies(INACTIVE);
```

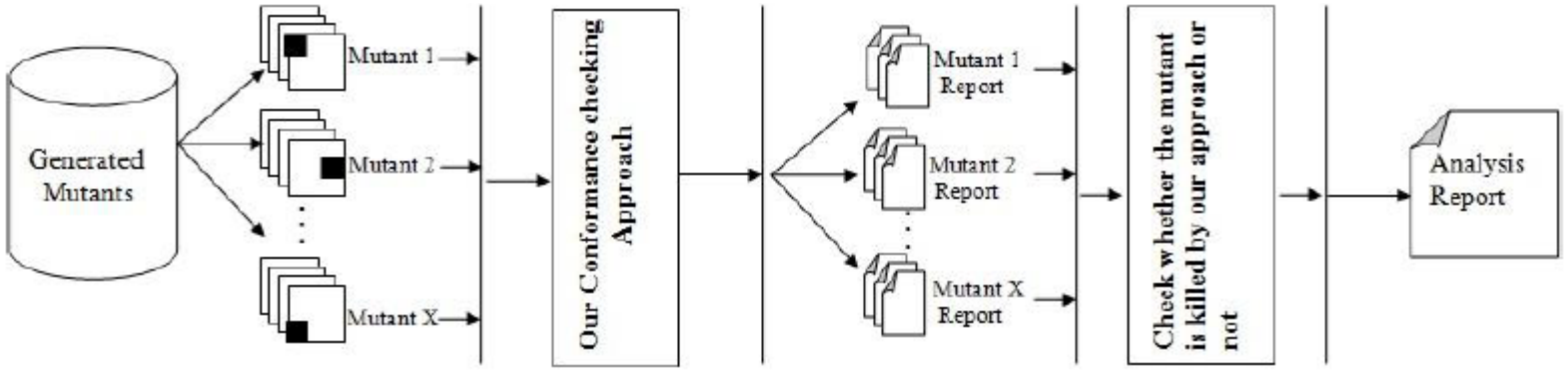
Automatic Mutation-Based Evaluation Framework

- 1) Mutant generation phase



Automatic Mutation-Based Evaluation Framework (Cont'd)

- 2) Execution phase



Experiment

- Examples used

Aglets Program	loc
OnlineSearching	1157
Main.java	30
Client.java	61
Agent.java	212
Mall1 with 3 shops	186
Mall2 with 2 shops	131
Mall3 with 4 shops	241
Mall4 with 5 shops	296
SigningContract	549
Notary.java	121
Contract.java	144
Partner1.java	71
Partner2.java	71
Partner3.java	71
Partner4.java	71

- Mutation operators used
  - All mutation operators are applied

Results

Number of equivalent and killed mutants

- All 587 mutants evaluated in less than 15 hours
- Maximal execution time for each mutant: 1.5 minutes
- Single user, single processor machine (Pentium 4 3.06GHZ) with 3GB of memory running the Redhat Linux operating system

Operator	Mutant Generated	Equivalent	Non-Equivalent	killed	Killed Rate
RDR	15	0 (0%)	15	15	(100%)
CDD	15	0 (0%)	15	15	(100%)
IDS	15	0 (0%)	15	15	(100%)
RAND	13	0 (0%)	13	13	(100%)
RDS	15	0 (0%)	15	15	(100%)
RDD	7	0 (0%)	7	7	(100%)
SIHD	2	0 (0%)	2	2	(100%)
SDS	7	0 (0%)	7	7	(100%)
CMP	11	0 (0%)	11	11	(100%)
CMOW	8	0 (0%)	8	8	(100%)
RSMM	11	0 (0%)	11	11	(100%)
RPSM	50	11 (22%)	39	33	(84.62%)
MSKP	27	0 (0%)	27	27	(100%)
MSR	41	13 (31.71%)	38	38	(100%)
RSRM	44	13 (29.55%)	31	31	(100%)
MCMC	46	0 (0%)	46	46	(100%)
NAN	0	0	(-)	0	(-)
RMRM	2	0 (0%)	2	2	(100%)
RPRM	18	0 (0%)	18	18	(100%)
MICA	2	0 (0%)	2	2	(100%)
MFCA	25	0 (0%)	25	25	(100%)
ROCM	87	0 (0%)	87	87	(100%)
ACON	11	6 (54.55%)	5	5	(100%)
REN	80	0 (0%)	80	80	(100%)
RARL	10	0 (0%)	10	10	(100%)
RAID	8	3 (37.50%)	5	5	(100%)
CPCN	9	0 (0%)	9	9	(100%)
CNP	5	0 (0%)	5	5	(100%)
CSAP	3	0 (0%)	3	3	(100%)
Total	587	46 (7.84%)	541	535	(98.89%)

Conclusion

- New approach for checking conformance of mobile, distributed applications with respect to an executable model at runtime
  - Empirically validated approach using four case studies
  - Allowed us to find seeded faults in all implementations
  - Support for distributed monitoring
  - KM can be analyzed
  - Approach language independent
  - Should scale to much larger systems

- Designed and implemented a mutation-based evaluation framework
  - New mutation operators for mobile agent systems (Aglets)
  - Our approach is very effective in killing the non-equivalent mutants with respect to the programs used
  - Most of the mutation operators did not generate any equivalent mutants

Implementing and Evaluating a Runtime Conformance Checker for Mobile Agent Systems

Ahmad Saifan, Juergen Dingel, Jeremy Bradbury, Ernesto Posse

Presented by:
Ahmad Saifan

Tuesday, 5 June, 12

Faculty of Information Technology
Yarmouk University
Irbid, Jordan