An Aspect-Oriented Behavioral Interface Specification Language

FLACOS '08, Malta

Takuo Watanabe & Kiyoshi Yamada*

Department of Computer Science, Tokyo Institute of Technology

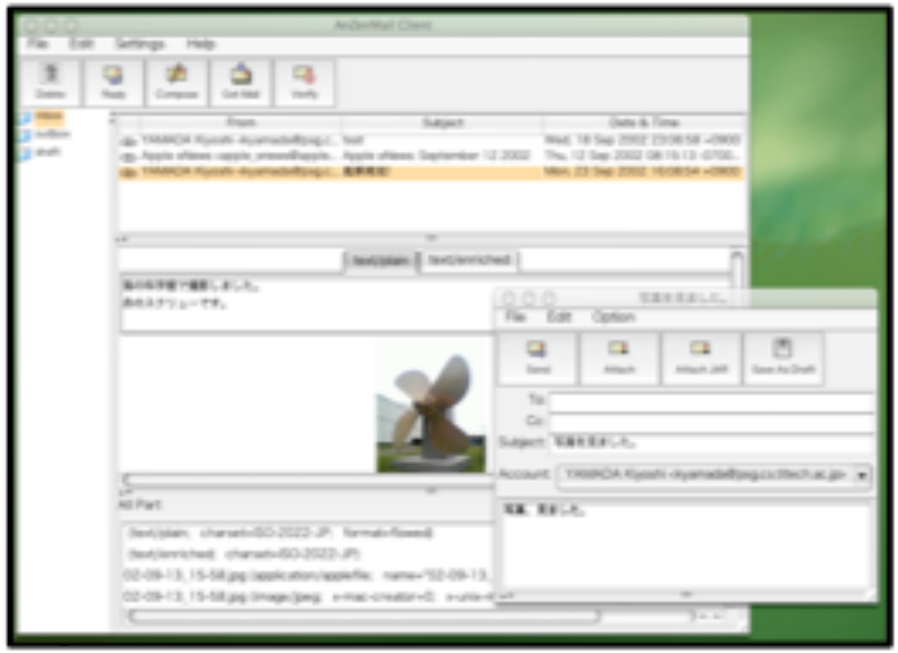
*Currently with: Research Center for Information Security, National Institute of Advanced Industrial Science and Technology

Talk Outline

- Background
  - Motivative Example: AnZenMail
  - Writing Specifications (Contracts) in JML
- Moxa: A BISL Supporting Assertion Aspects
- Result: JML vs. Moxa
- Extension
- Concluding Remarks & Future Work

Background: AnZenMail

- An E-mail System with Cutting-Edge Security Enhancement Technologies
  - Verified SMTP (w/Sender Auth.), Verified Server/Client Code, Behavior-Based Virus Detection, Security Plug-ins, etc.
  - joint research project funded by MEXT Japan.



AnZenMail Client (MUA)

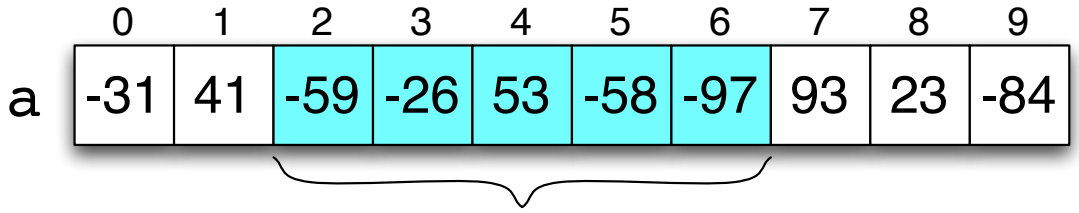
- We used JML (Java Modeling Language) to write the specification of some important components.
  - ex. Maildir Provider
    - the component that handles e-mail messages and folders in the file system
- Through the verification/validation process using JML tools (or by hand), we found lots of bugs and finally gained the solid code and firm specification.

Example Contract in JML

```
/*@ public normal_behavior  // Spec. A
  @   requires a != null && a.length > 0;
  @   ensures
  @     (\forall int i, j;
  @        0 <= i && i <= j && j < a.length;
  @        \result <= (\sum int k; i <= k && k <= j; a[k]))
  @ also
  @ public normal_behavior  // Spec. B
  @   requires a != null && a.length > 0;
  @   ensures
  @     (\exists int i, j;
  @        0 <= i && i <= j && j < a.length;
  @        \result == (\sum int k; i <= k && k <= j; a[k]))
  @*/
public /*@ pure @*/ int mss (int[] a) {
  // compute the minimum segment sum of a
}
```

Ex) Minimum Segment Sum

mss(a): minimum of segment sums



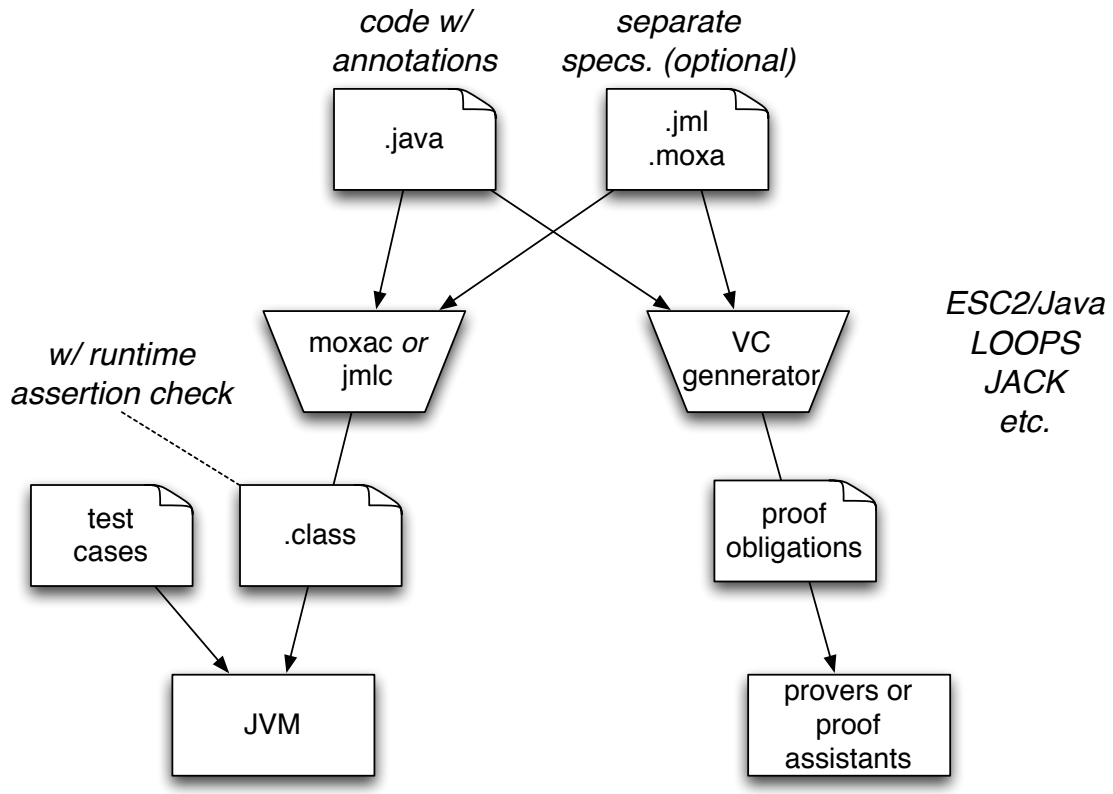
Specification Example:

$\{\texttt{a} \neq \texttt{null} \wedge \texttt{a.length} > 0\}$

```
r = mss(a);
```

$$\left\{ \begin{array}{l} \forall i, j \in \mathcal{Z}.(0 \leq i \leq j < \texttt{a.length} \Rightarrow \texttt{r} \leq \sum_{k=i}^{j} \texttt{a}[k]) \quad \wedge \\ \exists i, j \in \mathcal{Z}.(0 \leq i \leq j < \texttt{a.length} \wedge \texttt{r} = \sum_{k=i}^{j} \texttt{a}[k]) \end{array} \right\}$$

Verification/Varidation



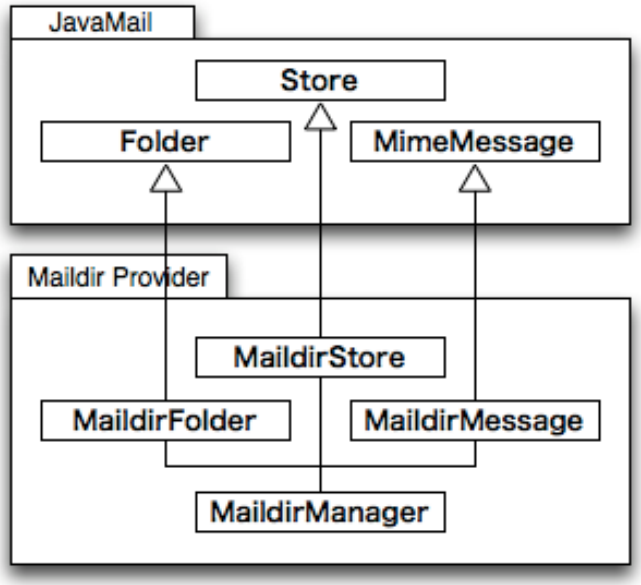
(1) runtime assertion checking

(2) static analysis and verification

Loop Invariant and Lemma

```
public /*@ pure @*/ int mss (int[] a) {
  int t = a[0], s = a[0], k = 1;
  //@ define INVs(x, m)
  //@   (\forall int i, j; 0 <= i && i <= j && j < m;
  //@       x <= (\sum int l; i <= l && l <= j; a[l]));
  //@ define INVt(x, m)
  //@   (\forall int i; 0 <= i && i < m;
  //@       x <= (\sum int l; i <= l && l < m; a[l]));
  //@ loop_invariant INVs(s, k) && INVt(t, k);
  while (k != a.length) {
    //@ assert INVs(s, k) && INVt(t, k) && k != a.length ==>
    //@   INVs(Math.min(s, Math.min(t + a[k], a[k])), k + 1) &&
    //@   INVt(Math.min(t + a[k], a[k]), k + 1);
    t = Math.min(t + a[k], a[k]);
    s = Math.min(s, t);
    k++;
  }
  return s;
}
```

Verification Process for AnZenMail



- Verified the behavioral subtype relations between JavaMail classes and Maildir classes.
- Verified that the classes correctly implement the Maildir functionality.
- Verified the consistency of Maildir Folder usages.

Scalability Problem

- Specification (based on assertions) becomes complex and bulky.
  - ex) in the Maildir Provider module:
    - 2.5k lines of Java code
    - 3.5k lines of JML annotations
- In an incremental development process, it is generally hard to keep the coherence of the specification and the consistency between the specification and the code.

```
public abstract class Folder {

/*@ public normal_behavior
  @ requires this.getStore().isConnected()
  @  && this.exists() && this.isOpen()
  @  && 1 <= msgnum && msgnum <= this.getMessageCount();
  @ ensures this.getStore() == \old(this.getStore())
  @  && this.getStore().isConnected() ==
  @     \old(this.getStore().isConnected())
  @  && this.exists() == \old(this.exists())
  @  && this.isOpen() == \old(this.isOpen())
  @  && this.getName() == null ? \old(this.getName()) == null :
  @     this.getName().equals(\old(this.getName()))
  @  && this.getFullName() == null ? \old(this.getFullName()) == null :
  @     this.getFullName().equals(\old(this.getFullName()))
  @  && this.getURLName() == null ? \old(this.getURLName()) == null :
  @     equals2URLName_model(this.getURLName(),
  @       \old(this.getURLName()))
  @  && \result != null && \result.getFolder() == this
  @  && \result.getMessageNumber() == msgnum;
  @ also public normal_behavior
  @  ...
  @*/
 public /*@ pure @*/ Message getMessage(int msgnum)
   throws MessagingException;

  ...
}
```

Observation: Crosscut in the Contract

- Many concerns in the contract are crosscutting over methods and classes.

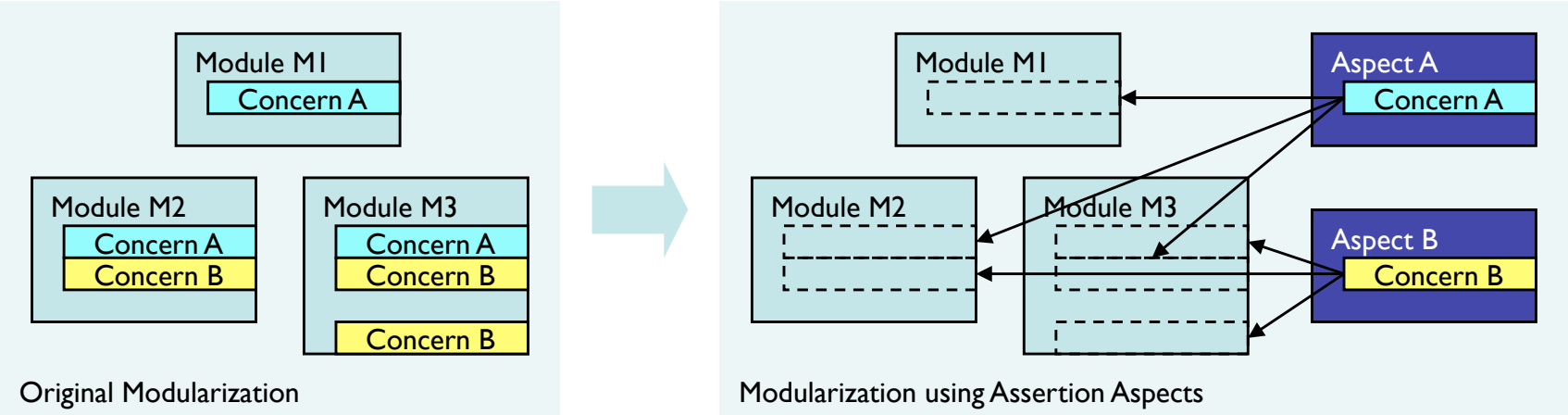
```
public class Folder {
  /*@ public behavior
    @ requires chkState_closed() && chkName()       && ..;
    @ ensures  chkState_open()   && chkName_eq(..) && ..;
    @*/
  public open();

  /*@ public behavior
    @ requires chkState_open()   && chkName()       && ..;
    @ ensures  chkState_open()   && chkName_eq(..) && ..;
    @*/
  public Message getMessage(int msgnum);
  ..
}
```

state concern

name concern

Assertion Aspects

- We introduce assertion aspects to modularize crosscutting concerns in assertions.
- Assertion aspects can increase the locality of description and decrease the size of descriptions.



Moxa

艾

- An Aspect-Oriented BISL tailored to Java
  - designed as an extension of JML
  - Moxa: MOdules for X-cutting Assrtions
    - The word "moxa" means the stuff used in moxibustion.
- Provies a language mechanism for describing assertion aspects
  - using a simple join-point model
- moxa2jml
  - a tool that weaves assertion aspects into assertions specified at classes

Assertion Aspects in Moxa

- Assertion Aspect
  - a collection of advice descriptions
- Advice
  - pointcut + logical expression
- Pointcut
  - method signature pattern
  - (logical expression pattern)

```
spec Folder_State { // assertion aspect
  /*@ public behavior  // advice
    @ requires chkState_closed();
    @*/
  public Folder.open(); // pointcut

  /*@ public behavior  // advice
    @ ensures chkState_open();
    @*/
  public void Folder.open(); // pointcut
  public Message Folder.getMessage(int msgnum);
  ..
}
```

JML vs. Moxa

- We wrote Moxa specifications for the same target (AnZenMail) we had done in JML.
  - `Service` and `Store` classes in the Maildir Provider
- Then we compared the both specifications in the following points:
  - Size of descriptions
    - # of modules and # of lines in a module
  - Ease of changes
    - # of lines affected by changes

Result: Size of Specifications

	JML		Moxa	
	Service	Store	Service	Store
# of modules	(1)	(1)	3	5
# of assertions	42	53	13	18
# of spec lines	190	149	152	286*
# of lines per module	190	149	51	57

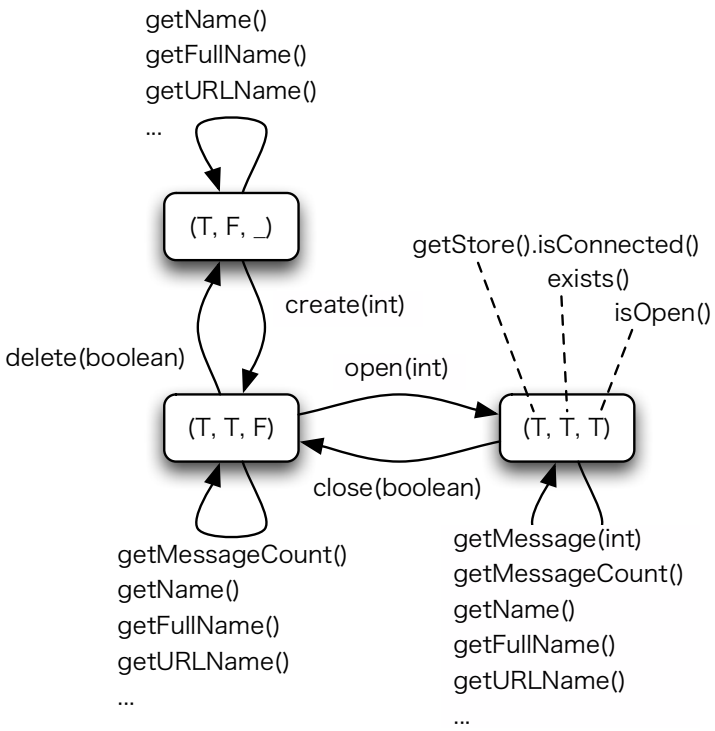
Result: Ease of Changes

	JML		Moxa	
	Service	Store	Service	Store
# of changes	42	53	6	4
# of lines changed	190	149	54	40

Other Examples

- A Small Web Application (5.5 kloc)
  - # of assertion lines in JML: 4250
  - # of assertion lines in Moxa: 1980
  - # of assertion aspects: 21
- AST2J (2.5 kloc)
  - a tool for generating Java code from AST descriptions
  - # of assertion lines in JML: 720
  - # of assertion lines in Moxa: 560
  - # of assertion aspects: 8

Extension: Transitional Assertion Aspects



- Transition of states in a class (or in a component) can be extracted as assertion aspects.
  - protocol aspects
    - graphical representation
    - model checking
- assertion aspect as a unit of extended contract definition

Concluding Remarks

- Thanks to assertion aspects, Moxa enables us to write handy and scalable behavioral interface specification for Java programs.
- Assertion aspects can modularize some properties commonly appeared in a complex system.
  - ex. protocols

Future Work

- Language Design
  - join-point model or other modularization mechanism
  - dealing with threads, aliasing, etc.
  - contract languages
    - AA as a Pluggable Contract Description Mechanism
      - Protocol Aspects
      - Aspects for Execution Monitoring using PT-LTL (à la MOP)
        - ex) `access -> <*>authenticate`
- Tools
  - moxac, moxarac, moxadoc, Eclipse plug-ins, etc.
- More Experiments