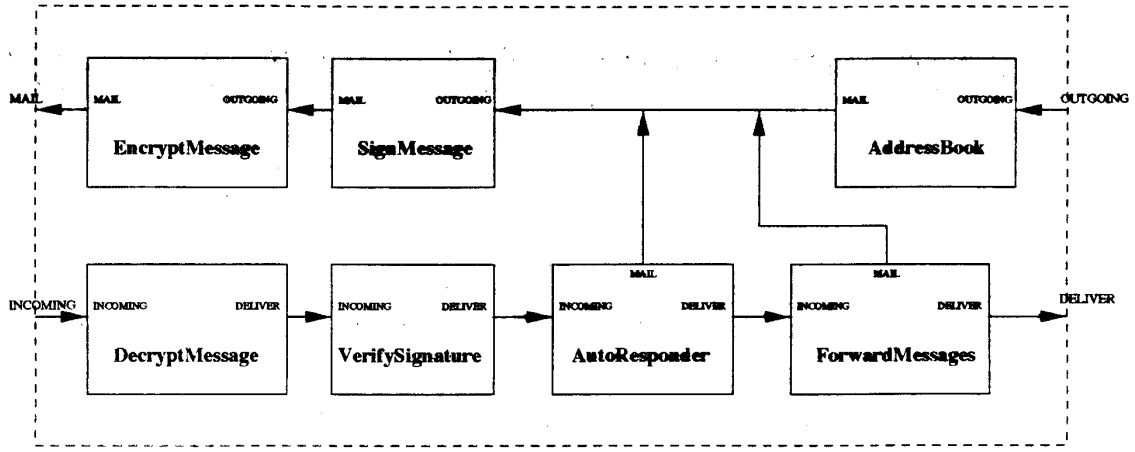
Feature Interactions in Electronic Mail



Robert J. Hall
AT&T Labs Research

Outline

- ☐ Motivation and Overview
- ☐ Ten Email Features
- ☐ Modeling and Combining Email Features
- ☐ Scenario Selection Methodology
- ☐ Case Study Results and Highlights
- ☐ Discussion

Motivation

Email has been in use for ≈ 20 years
...how well do we understand it?

Distributed service architecture:
– Internet
– RFCs 821, 822 (ca. 1982)
– MUAs, MTAs, hosts, filters, remailers, ...

Many features, many developers,
bounded knowledge

As new features are added, it becomes harder to *understand*, much less *predict* behavior
– feature interactions can occur
– program chair anecdote

□ MOTIVATION + OVERVIEW

Goals and Approach

Goals

- (Start on) *practical guide* to email feature interactions for users, administrators, and feature developers
- Methodology for detecting feature interactions in distributed feature architectures

Detection method:

0. *Select* set of primitive features of interest
1. *Model* one or more "typical" configurations
2. *Select* scenarios
3. *Simulate* scenarios
4. *Inspect* results for undesirable interactions

Ten Primitive Email Features

- AddressBook
- SignMessage
- EncryptMessage
- DecryptMessage
- VerifySignature
- AutoResponder
- ForwardMessages
- RemailMessage
- FilterMessages
- MailHost

□ MODELING + COMBINING

Email Feature Components

An *email feature component (EFC)* is a reactive system that operates on email messages.

- state machine (not necessarily finite-state)
- input events: init, configure, receive msg

```
INIT()
COMMAND(CMD-NAME:string, ARG-LIST:list)
INCOMING(MSG:message)
OUTGOING(MSG:message)
```

- output events: send, deliver messages

```
MAIL(MSG:message)
DELIVER(MSG:message, USER:string)
```

- events have typed parameters

EFCs are either *primitive* or *compound*

☐ MODELING + COMBINING

Primitive EFCs

Primitive EFCs are modeled using an *executable specification language*

- This study: ISAT's P-EBF
- Tools:
  - * simulator
  - * test coverage analyzer

Example: SignMessage

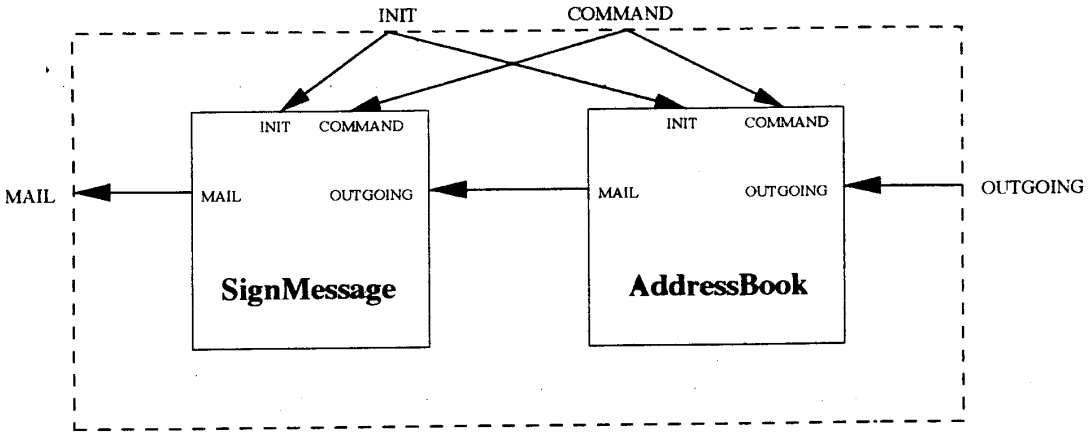
```
(spec SignMessage

  (include-theory Email)
  (include-theory Email-Feature-Acts)
  (include-theory Email-Own-Key)

  (handler (INIT)
    (set Own-Key ""))

  (handler (COMMAND (Cmd string) (Args list-of-string))
    (case Cmd
      (("SET_OWN_KEY")
       (set Own-Key (first Args)))
      (otherwise
        ;;;[Unhandled command type --> no action]
       )))

  (handler (OUTGOING (Msg message))
    (let ((key (lookup Own-Key)))
      (if (equal? key "")
          (act MAIL Msg)
          (act MAIL (sign-message Msg key)))))

  (handler (INCOMING (Msg message))
    ;;;[Incoming events are ignored --> no action]
    ))
```

□ MODELING + COMBINING

Compound EFCs

Compound EFCs are modeled as *interconnection diagrams* of EFCs.

- Each box is an EFC
- Events enter/exit via typed *ports*
- Ports connected by unidirectional data flow

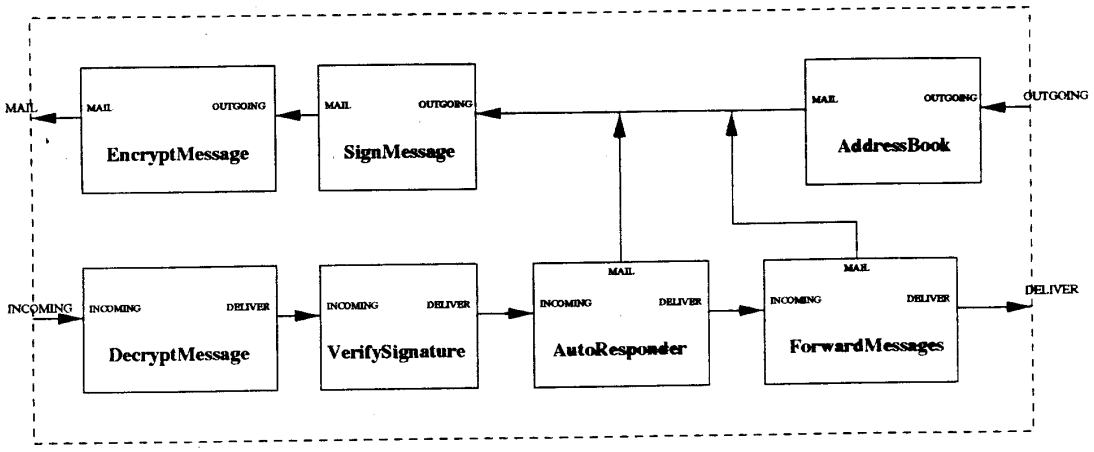


Reactions defined via *deterministic simulation*

- interleaving semantics
- ambiguous: may miss some event orderings
- EFC simulator picks a particular ordering
- f.i. detection is only heuristic anyway
- *orderable* EFCs guaranteed unambiguous

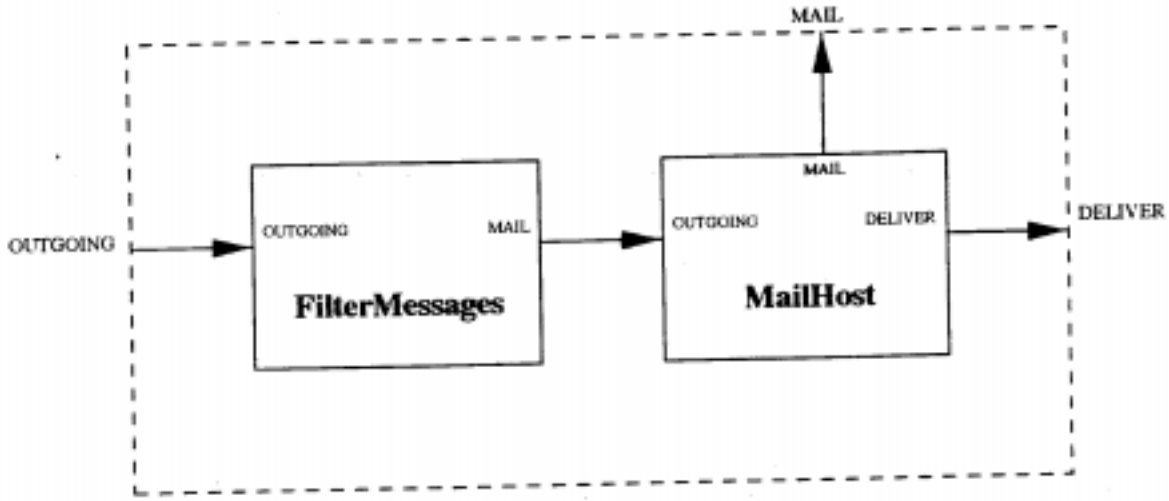
□ Modeling & Combining

The Client EFC



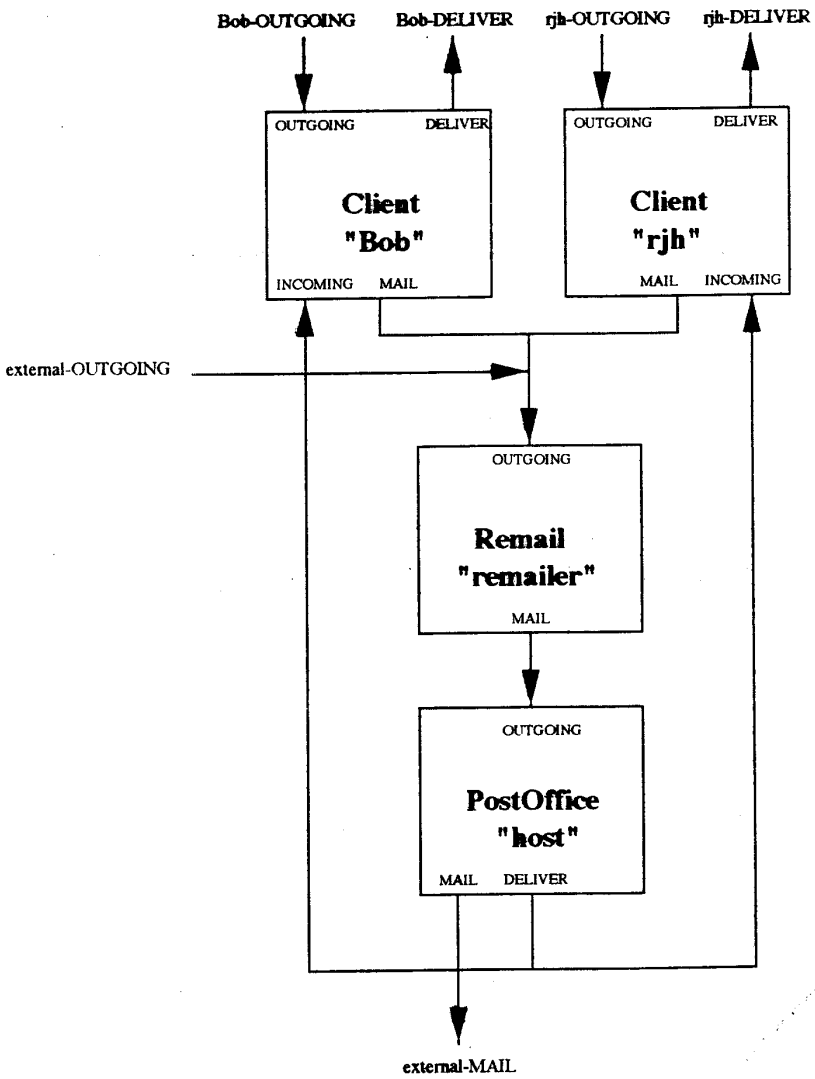
□ Modeling + Combining

The PostOffice EFC



☐ Modeling + Combining

The Network EFC



☐ Scenario Selection

Scenario Selection

Problem: Too many scenarios!
- infinitely many
- $\approx$ 34560 ignoring cycles

Even if tool can generate, human can't inspect

Methodology:

1. Construct and validate scenarios for each primitive EFC. (ISAT tool suite)
2. For each pair of primitive features f_1, f_2:
   - Human selects subset of f_1 scenarios as "of interest" to f_2
   - For each such *seed* scenario, construct set of scenarios such that
     * message is sent from f_1 to f_2
     * executes same path through f_1 as seed
     * set *covers* responses of f_2

     $\longrightarrow$ formal coverage metric and tool

Note *asymmetry* requires **ordered** feature pairs
- e.g. remail-then-sign vs sign-then-remail

□ SCENARIO SELECTION

Simulation Example

```
((INIT)
 (HOST-COMMAND "SET_HOSTNAME" ("PostOffice"))
 (HOST-COMMAND "INIT_USER" ("bob"))
 (HOST-COMMAND "INIT_USER" ("rjh"))
 (BOB-COMMAND "SET_OWN_KEY" ("bob.key"))
 (REMAILER-COMMAND "SET_HOSTNAME" ("remailer"))
 (REMAILER-COMMAND "CREATE_USER_PSEUDONYM" ("bob@PostOffice"))
 (BOB-OUTGOING (simple-message "bob@PostOffice"
                               "remail@remailer"
                               ("rjh@PostOffice"
                                "The toxic waste was dumped by..."))))

== Network simulator ==> ...event trace...

(DELIVER (simple-message "pn0@remailer"
                         "rjh@PostOffice"
                         ("rjh@PostOffice"
                          "The toxic waste was dumped by..."
                          "Signature Block: <bob.key signature>"))
         "rjh@PostOffice")
```

□ RESULTS + HIGHLIGHTS

Case Study Results

100 ($= 10 \times 10$) ordered feature pairs

26 distinct feature interactions found

All ten basic features had some interactions

Considered 155 scenarios
  $\approx$ 1 in 6 scenarios had unexpected behavior
  (!)

Time cost: 27 hours
– 10 minutes per scenario
– 1 hour per interaction

□ RESULTS + HIGHLIGHTS

Case Study Highlights

AddressBook vs EncryptMessage
– sent encrypted *and* clear

SignMessage vs RemailMessage
– Oops. Don't sign anonymous message

AutoResponder vs RemailMessage(1)
– autoresponse leaks identity

ForwardMessages vs MailHost
– accidentally forward to nonexistent user

EncryptMessage vs AutoResponder
– unencrypted autoresponse leaks subject line

□ Discussion

Related Work

Distributed/Modular Approaches

- EFCs, Jackson/Zave(98), Zibman et al (95)
- Features are modular
- Combined by *interconnection*
- Asymmetric, coverage-based f.i. detection methodology applicable

Conjunctive Approaches

- FG/BG Models Hall(98), Bergstra/Bouma (96), Blom/Bol/Kempe(95)
- Create logical models of base + features
- Shared state (typically non-modular)
- Combined by (form of) *logical conjunction*

Comparison

- Feature Interactions present in both
- Distributed/Modular avoids shared-state interactions
- *Tradeoff:* fewer interactions in spec for difficulty in implementation
- Conjunctive often closer to efficient impl. ...but not in email domain

□ Discussion

Limitations and Future Work

Somewhat simplistic models
- while these results generalize well...
- models more faithful to implementations will find more interactions
- both primitive features and compound EFCs

Heuristic f.i. detection
- recall ∞ / 34560 numbers
- tries to combine human intuition and machine-enforced systematicity
- Other tools could detect other types assertion checking, cycle detection
- Finer/Coarser grained coverage tool varies sensitivity and time/tediousness

◘ Discussion

Summary

Email getting hard to *understand* and *predict* due to interactions among features

EFCs constitute distributed, modular modeling formalism that maps naturally to implementation

Asymmetric, coverage-based feature interaction detection methodology combines human intuition and machine systematicity

Results:

- Beginning of practical guide to email f.i.s
- Practical methodology for users, admins, and developers