Model Extraction from Context-Aware Code
Paradigm:

- Interactions with networked data-processing systems available everywhere.
- ...which autonomously monitor and adapt themselves and the environment.
Concept: Context Awareness

Context awareness:

- sense / discover changes in computational surrounding (local or networked)
- dynamically adapt behaviour to change.

Context:

- any environmental information relevant to application behaviour; unpredictable
...or in fewer metaphors:

- Application’s execution environment:
  - driven by external input
  - unpredictable

- Programming behaviour:
  - asynchronous languages
  - middleware-driven
  - flavours of multithreaded code
And effort goes towards...

Verifying application behaviour

network-wide at node
Automated Generation of Context-Aware Tests
Zhimin Wang, Sebastian Elbaum, David S. Rosenblum
ICSE '07

Related:

Keywords: middleware, Java, validation.

TourApp application

Application

main

showInterestLevelWin
recordInterestLevel
initPMConn
refreshPM
disconnectPM

updateVisitedDemo
updatePowerMeter

contextHandler
contextHandler
...

loadConfig
startCommunication
subscribeTo

Communication
Middleware

startListenSocket
addHandler
sendRequest

acceptData
startThread
run
run

notifySubscriber
notifySubscriber

findHandler

Get handler from vector

Context Information for a specific subscriber

Widget

Sensor

Context Information

Function Call
Remote Data Connection

3 Incorporating Context-Awareness

Given a program \( P \) and its test suite \( T \), our approach enhances \( T \) by manipulating \( P \) during the execution of each test case \( t \in T \) with the objective of forcing the exploration of potentially interesting contextual scenarios. The underlying assumption is that, given the proper manipulation mechanisms, the potential of \( T \) to explore much more of \( P \)'s behavior can be increased.

The approach's novelty consists in the integrated application of existing analysis techniques to identify and control what contextual scenarios to explore. We now present the approach in terms of its supporting infrastructure depicted in Figure 3. The infrastructure consists of the following components:

- **Context-aware program points (capps) Identifier.** This component aims to identify program points where context changes may affect the application's behavior.
- **Context Driver Generator.** This component forms potential context interleavings that may be of value to fulfill a context-coverage criterion.
- **Program Instrumentor.** This component incorporates a scheduler and capp controllers into the application to enable direct context manipulation.
- **Context Manipulator.** This component attempts to expose the application to the enumerated context interleavings through the manipulation of the scheduler.
Related:

Keywords: middleware, Java, validation.

context-aware program points (capps): read/write
Finally we introduce example presented in the previous section there are such value of the Boolean propositional context variables $z$ in the state in a transition by means of a vector.

As shown in the table, the predicates and priorities of some cases a rule name is used in place of a full predicate.

Also, the table does not show the actions of PhoneAdapter.

This represents the set of adaptation rules we defined for some variables to encode "destination" names for simplicity.

For example, one such relational expression is represented throughout the rules by the Boolean expression $\neg a$.

The set of states reachable from the initial state by some context variables as they are used in PhoneAdapter.

This verification algorithm computes the set of states reachable from the initial state by particular, given the Boolean formula.

This presentation and enabling symbolic computations. In particular, given the Boolean formula.

The representation above allows for the encoding of a CAAA as shown in the table.

As shown in the table, the predicates and priorities of some context variables as they are used in PhoneAdapter.

Related:

Keywords: rule-based adaptation model, symbolic verification

Faults:

- reachability
- determinism
- state/rule liveness
- stability
Related:

Keywords: high-level model, validation

| **Situation** visitornearby = (C_in, p_in, computIllum) |
|---|---|
| **Contexts** | C_in = {d, V_oc, R, I, I_max, E_v} |
| **Triggering condition** | p_in = (d ≤ 5) |
| **Adaptive action** | computIllum { |
| | $V_{oc} = V_{oc}$; |
| | $R = R$; |
| | $I = V_{oc} / (r + R)$; |
| | $I = I$ |
| | $P = k * I *$I * r; |
| | $d = d$; |
| | $E_v = P / (d * d)$; |
| | }
Related:

Keywords: asynchronous, nesC, model extraction.
These being said,

Model Extraction from Context-Aware Code
Capps: a generalization

... of side-effect-, escape-like analyses

- **read capps**
  
  ```
  if(x > MAX)
    y = x;
  ```

- **write capps**
  
  ```
  x = ctx;
  ```

- **termination capps**
  
  ```
  if(x > MAX)
    while 1;
  y = 1;
  ```

- **user capps**
  
  ```
  print*(ctx);
  or if(ctx)
    set_register(r);
  ```
... or in pictures:

unsigned power, demo;

void update_power(ctxt *power_ctx)
{
    power = power_ctx->value;
}

void update_demo(ctxt *demo_ctx)
{
    if(demo_ctx->value == 1 && power>30)
        demo = 1;
    else if(demo_ctx->value == 2 && power>30)
        demo = 2;

    if(demo == 1)
        printf("demo 1");
    if(demo == 2)
        printf("demo 2");
}

void* context_handler(void *arg)
{
    if(((ctxt *)arg)->type == TYPE_POWER)
        update_power(arg);
    else if(((ctxt *)arg)->type == TYPE_DEMO)
        update_demo(arg);

    return NULL;
}

void middleware()
{
    ctxt ctx1 = [..];
    pthread_create(id, NULL, context_handler, &ctx1);
}
To FSM:

- FSM:
  - situations
    - <contexts, predicate, action>
  - ... to state machine: transitions, states

- Take e.g.
  - transitions: all context-controlled control flow
  - states: all other side effects
... or in pictures:

```
18: Start of Thread

20: arg = (void *&)ctx1;

21: !(int)ctx1.type == 0?
   true
   false

22: !(int)ctx1.type == 1?
   false
   23: power_ctx = (&arg);

29: demo_ctx = (&arg);

30: !(int)ctx1.value == 1 && (int)power > 30?
   true
   false

31: demo = 1;

32: !(int)demo == 1?
   false
   true

33: !(int)ctx1.value == 2 && (int)power > 30?
   true
   false

34: demo = 2;

35: !(int)demo == 2?
   true
   false

36: PRINTF((void *&)"demo 1\[0\]");

37: !(int)demo == 2?
   true
   false

38: PRINTF((void *&)"demo 2\[0\]");

40: NULL;

42: End of Thread
```

state ids

End of Thread
... or in pictures:

18: Start of Thread
20: arg = (void *)&ctx1;
21: !((int)ctx1.type == 0)?
22: !((int)ctx1.type == 1)?
23: power_ctx = (int)arg;
24: power = ctx1.value;
29: demo_ctx = (int)arg;
30: !((int)ctx1.value == 1 && (int)power > 30)?
31: demo = 1;
32: !((int)ctx1.value == 2 && (int)power > 30)?
33: !((int)ctx1.value == 1 && (int)power > 30)?
34: demo = 2;
35: !((int)demo == 1)?
36: PRINTF((void *)&"demo 1\[0\]":)
37: !((int)demo == 2)?
38: PRINTF((void *)&"demo 2\[0\]":)
40: NULL;
42: End of Thread

state ids, transitions
... or in pictures:

state ids, transitions, states

// state composition
(18, 20, 21)
(23, 24, 40, 42)
(27)
(29, 30)
(31, 35)
(33)
(34, 35)
(35)
(36, 37)
(37)
(38, 40, 42)
(40, 42)