Model-based Testing and Analysis of Coordinated Components

Gabriel Ciobanu[1] Dorel Lucanu[2]

[1]Institute of Computer Science, Romanian Academy [2]Faculty of Computer Science, Alexandru Ioan Cuza University, Iași, Romania

OpenCert, 2009

Plan

1 Motivation

2 Coordinated Components

3 Constructing the Model

4 Testing and Analysis

5 Conclusion

Model-based testing

“A model program acts as an executable specification for another program or system, called the implementation. “
(http://www.codeplex.com/NModel)
A model program can be used for:

- checking disign errors in the implementation
- generate test cases for the implementation
- serving as oracle when testing the implementation
- etc

The main goal

A model for the coordinated components [Ciobanu & Lucanu, SAVCBS 2005]

- coordinated objects specified using preconditions, postconditions and invariants
- the interactions between components are modeled by means of a coordinator specified as a process algebra
- a wrapper bindings the abstract actions of the coordinator to concrete actions of the components

In this paper we present a method for constructing a model program for a system of coordinated components.

Plan

1 Motivation

2 Coordinated Components

3 Constructing the Model

4 Testing and Analysis

5 Conclusion

Coordinated Components: the intuition

The three components of the coordination:

Coordinated Components: the intuition

The three components of the coordination:

- coordinated objects

Coordinated Components: the intuition

The three components of the coordination:

- coordinated objects
- coordinator

Coordinated Components: the intuition

The three components of the coordination:

- coordinated objects
- coordinator
- a means to coordinate

The Components

```
class ATM
  ...
  cash()
    require isCardInserted = true
    require amount <= availAmount
    ensure resulting availAmount = availAmount - amount
```

```
class Bank
  ...
  grantMoney(aCardNumber as Integer, anAmount as Integer)
    require getAccount(aCardNumber).balance >= anAmount and
            getAccount(aCardNumber).isAccessible = true
    ensure resulting getAccount(aCardNumber).balance =
           old(getAccount(aCardNumber).balance) - anAmount
```

The Coordinator

$$
\begin{array}{ll}
A = ins.A_1 & B = \overline{na}.B + \overline{gb}.B + \overline{nem}.B + \overline{gm}.B \\
A_1 = rel.A + ep.A_2 & A_2 = (na + rel).A + ab.A_3 + ea.A_4 \\
A_3 = gb.A_2 & A_4 = nem.A_2 + gm.A
\end{array}
$$

The Wrapper

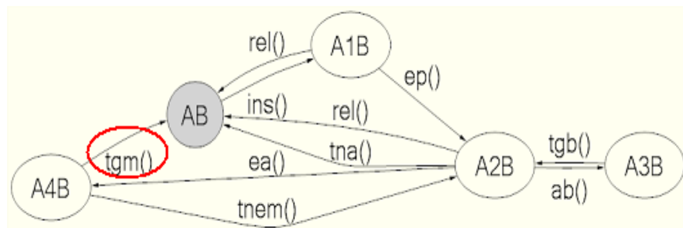
$$w[\tau(gm(atm), \overline{gm}(bank))] =$$
$$bank.grantMoney();$$
$$atm.cash();$$
$$atm.release()$$

Plan

1. Motivation
2. Coordinated Components
3. Constructing the Model
4. Testing and Analysis
5. Conclusion

Constructing the model 1/3

Start with the LTS of the coordinator:



and construct a simple model $M(P)$:

```
[Action]
tgm()        //τ(gm(atm), gm̄(bank))
  require (vp = A4B)
  vp := AB
```

Constructing the model 2/3

Then enrich the actions of $M(P)$ with the preconditions and updates deduced from the specification of the components and wrapper:

$$Pre_M(act) = Pre_{M(P)}(act) \land Pre(w[act])$$
$$Update_M(act) = Update_{M(P)}(act) \| Update(w[act])$$

Constructing the model 3/3

```
[Action]
tgm()
  require (vp = A4B)
  require atm.isCardInserted = true
  require amount <= availAmount
  require bank.getAccount(atm.cardNumber).balance >= atm.amou
  require bank.getAccount(atm.cardNumber).isAccessible = true
  vp := AB ||
  ( bank.grantMoney(atm.cardNumber, userAmount);
    atm.cash();
    atm.releaseCard() )
```

Plan

1 Motivation

2 Coordinated Components

3 Constructing the Model

4 Testing and Analysis

5 Conclusion

Conformance Testing

- define a subset of acceptance states in M
- a trace = a path connecting the initial state with an acceptance state
- check if the implementation under test (IUT) is able to follows all the traces.

Generating Test Case Suites 1/2

- a trace ⇒ a case test
- a challenging problem: how to find those instances able to follow a given trace?
- a possible solution: compute the path condition and use a constraint solver

Path	Condition
insertCard	
enterPin	enteredPin = cardPin ∧
enterAmount	enteredAmount ¡= balance ∧
grantMoney	account(card).isAccessibile = *true*

Generating Test Case Suites 2/2

- can we always find such instances?
- i.e., the constraints have always solution?
- this is refered as a consistency problem: if the components are able to perform all interactions specified by the coordinator

Plan

1. Motivation
2. Coordinated Components
3. Constructing the Model
4. Testing and Analysis
5. Conclusion

- a model for the coordinated components can be constructed
- the construction can be automated
- the model can be used for conformance testing and test cases generation
- the approach is suitable for both the closed systems and reactive systems