Automated Adaptor Generation for Behavioral Mismatching Software Components Using Pushdown Model Checking

Hsin-Hung Lin

Institute of Information Science
Academia Sinica
Taipei, Taiwan

13 Feb. 2013 @ Fukuoka
Self Introduction

- Ph.D. from JAIST (Sep. 2011)
  - Advisor: Prof. Katayama and Prof. Aoki
- Post-doctoral researcher
  - Institute of Information Science, Academy Sinica, Taipei
  - Supervisor: Prof. Bow-Yao Wang
- Research interests
  - Applications of model checking in software engineering
Outline

1 Introduction
   - Mismatches and Adaptation
   - Motivation: Non-regular Adaptation

2 Coordinator Guided Adaptor Generation
   - Models
   - Coordinator

3 Experiments
   - Fresh Market Update Service

4 Conclusions
1 Introduction
   - Mismatches and Adaptation
   - Motivation: Non-regular Adaptation

2 Coordinator Guided Adaptor Generation
   - Models
   - Coordinator

3 Experiments
   - Fresh Market Update Service

4 Conclusions
1 Introduction
   - Mismatches and Adaptation
   - Motivation: Non-regular Adaptation

2 Coordinator Guided Adaptor Generation
   - Models
   - Coordinator

3 Experiments
   - Fresh Market Update Service

4 Conclusions
Reuse of Components

- Reuse: developing new application by composing some existing components
- In general, components may refer to software components/web services under the context of reuse
- Existing components usually do not cooperate with each other since of inconsistency in development/design
- Errors in composition: Mismatches
Mismatches

Classes of Mismatches

- Technical mismatches: implementation related.
- **Signature** mismatches: different method names, etc.
- **Behavior/Protocol** mismatches: deadlocks after composition.
- QoS mismatches: security, efficiency, etc.
- Semantic mismatches: different interpretations of functionalities.

Behavior Mismatches are especially interested

- Signature/Behavioral mismatches are relating to **behavior interfaces** of components
- Extra information/models are required for solving higher levels of mismatches

Solution: Adaptation

- Adaptation solves **Signature** and **Behavioral** mismatches among behavioral interfaces of components
- Adaptor controls interactions of components
- All components are synchronized with adaptor
- Non-intrusive: No need to modify existing components
Adaptation: Existing Approach

Model-Based Adaptation for behavioral mismatches

- Input: (1) behavior interfaces of components; (2) adaptation contracts
- Output: adaptor
- Automata
  - special symbols: ! for sending, ? for receiving
- Adaptation Contracts
  - (1) mapping of labels; (2) ordering of interactions
- Adaptor generation (automated)
  - synchronous composition + deadlock elimination

1 Introduction
   ■ Mismatches and Adaptation
   ■ Motivation: Non-regular Adaptation

2 Coordinator Guided Adaptor Generation
   ■ Models
   ■ Coordinator

3 Experiments
   ■ Fresh Market Update Service

4 Conclusions
Problem: Non-regular Adaptor

Adaptor may be non-regular due to behavioral mismatches.

- Existing approach does not support non-regular behavior.
- Motivation Example: Fresh Market Update Service

![Diagram showing the interaction between Online Stock Broker, Research Department, and Investor.]
(Ordering) Behavior Mismatch:
- “Investor” expects \textit{Start} before receiving \textit{Data} while \textit{Data} is sent before \textit{Start}.

Adaptor with non-regular behavior is needed

One possible non-regular Adaptor:
Objective and Solution

Objective

- Non-regular adaptation
  - Generating adaptors with non-regular behavior

Solution

- Adaptor generation based on **pushdown model checking**
  - **Coordinator** Guided Adaptor Generation
Outline

1 Introduction
   - Mismatches and Adaptation
   - Motivation: Non-regular Adaptation

2 Coordinator Guided Adaptor Generation
   - Models
   - Coordinator

3 Experiments
   - Fresh Market Update Service

4 Conclusions
Outline

1 Introduction
   - Mismatches and Adaptation
   - Motivation: Non-regular Adaptation

2 Coordinator Guided Adaptor Generation
   - Models
   - Coordinator

3 Experiments
   - Fresh Market Update Service

4 Conclusions
Behavior Interfaces of Components

Interface Automata for Adaptation (IA4AD).

- Based on Interface Automata
  - Finite state machines - enough for representing protocols
  - Separate notations for input/output alphabets - convenient for dealing with communications

- Modifications are introduced
  - Interface Automata only define compatibility for two
  - In adaptation, we define compatibility for all given components

---

Definition (Interface Automata for Adaptation (IA4AD))

An interface automaton for adaptation is defined as tuples:
\[ P = (Q, q^0, A^I, A^O, A^H, \Delta, q^f), \]
where
- \( Q \): finite set of states.
- \( q^0 \in Q \): initial state.
- \( A^I \): finite set of input alphabets.
- \( A^O \): finite set of output alphabets.
- \( A^H \): finite set of internal alphabets.
- \( \Delta \subseteq Q \times A \times Q \): set of transition relations, where \( A = A^I \cup A^O \cup A^H \).
- \( q^f \in Q_i \): accepting state.

An IA4AD has to satisfy the following conditions:

- \( q^0 \neq q^f \)
- \( \forall t \in \Delta, \ t = (q, a, q'), \ q, q' \in Q, \ q = q^f \lor q' = q^0 \)
- \( \forall a \in A. \ \exists t \in \Delta, \ t = (q, a, q'), \ q, q' \in Q \)
Components form a closed system
- Input alphabet $a$ in one component = output alphabet $a$ in another component

**Definition (Composability of IA4AD)**

A set of interface automata for adaptation $P_i = (Q_i, q_0^i, A_i^l, A_i^o, A_i^h, \Delta_i, q_f^i), i \in [1, n]$, are composable if

- $A_i^l \cap A_j^o = \emptyset$
- $A_i^l \cap A_j^l = \emptyset$, $i \neq j$
- $A_i^o \cap A_j^o = \emptyset$, $i \neq j$
- $\bigcup_i A_i^l = \bigcup_i A_i^o$
- $\bigcup_i A_i^h \cap \bigcup_i A_i^l = \emptyset$
Behavior Interface of an Adaptor

Interface Pushdown System (IPS)

- Based on Pushdown System Model
  - An adaptor receives a message and sends it out later
  - Push a symbol into the stack and pop it later
  - First-In-Last-Out feature of stack is useful for ordering mismatch
  - Express non-regular behavior

- Modifications/Specializations are introduced
  - Stack symbols associate messages of components
  - Push/Pop a symbol associated to receiving/sending a message

Definition (Adaptor)

Given a set of composable IA4AD: \( P_i = (Q_i, q_i^0, A_i^I, A_i^O, A_i^H, \Delta_i, q_i^f) \), \( i \in [1, n] \). An adaptor for \( P_i, i \in [1, n] \) is an interface pushdown system: \( D = (Q_D, q_D^0, \Gamma, z, T, q_f^D) \), where

- **\( Q_D \)**: finite set of states.
- **\( q_D^0 \)**: initial state.
- **\( A_D = \bigcup_i A_i^I = \bigcup_i A_i^O \)**: finite set of alphabets.
- **\( \Gamma = A_D \cup \{z\} \)**: finite set of stack symbols.
- **\( z \)**: stack start symbol representing bottom of stack.
- **\( T \subseteq (Q_D \times \Gamma) \times (Q_D \times \Gamma^*) \)**: set of transition relations.
- **\( q_f^D \)**: accepting state.

\( T \) is restricted to the following three patterns:

\[
\begin{align*}
\langle p, \gamma \rangle \leftrightarrow & \langle p', a \gamma \rangle: \text{push transition}, \\
\langle p, a \rangle \leftrightarrow & \langle p', \epsilon \rangle: \text{pop transition}, \\
\langle p, \gamma \rangle \leftrightarrow & \langle p', \gamma \rangle: \text{internal transition},
\end{align*}
\]

where \( a \in A_D, \gamma \in \Gamma, \epsilon: \text{empty word} \).
Adaptor makes synchronization of messages interleaving
Combinations of sending/receiving
Consideration of the stack head symbol
Coordinator is an Over-behavioral Adaptor

Idea of Coordinator

- Adaptor makes maximum interleaving of sending/receiving messages
- Over-Behavioral
- Behavior solves behavior mismatches should be included (if there is any)
Building coordinator

- One state with self transitions
- Set of transitions consists of pushing/popping transitions of all messages

Push(RawData),
Push(Data),
Push(EndOfData),
Push(Complete),
Push(Start),
Push(Ack)

Pop(RawData),
Pop(Data),
Pop(EndOfData),
Pop(Complete),
Pop(Start),
Pop(Ack)
Picking up Behavior of Adaptor from Coordinator

Composition of components with Coordinator - **System Behavior**

Pick up behavior without behavior mismatches

Build adaptor from the above behavior
Definition (Adapted Synchronous Composition)

Given a set of composable IA4WS: \( P_i = (Q_i, q_i^0, A_i^I, A_i^O, A_i^H, \Delta_i, q_i^f) \), \( i \in [1, n] \), with an adaptor \( D = (Q_D, q_D^0, \Gamma, z, T_D, F_D) \). The adapted synchronous composition is an interface pushdown system

\[
\Pi_i^D P_i = (Q, q^0, \Gamma, z, T', F)
\]

where

- \( Q = Q_1 \times \ldots \times Q_i \times \ldots \times Q_n \times Q_D \): finite set of states.
- \( q_0 = (q_1^0, q_2^0, \ldots, q_n^0, q_D^0) \): initial state.
- \( T' \subseteq (Q \times \Gamma) \times (Q \times \Gamma^*) \): set of transition relations
- \( F = \{(q_1^f, \ldots, q_i^f, \ldots, q_n^f)\} \times F_D \): finite set of final states.
\[ T' = \{ \]
\[ \{((q_1, \ldots, q_i, \ldots, q_n, q_D), \gamma) \mapsto ((q_1, \ldots, q_i', \ldots, q_n, q_D'), a\gamma) |\]
\[ (q_1, \ldots, q_i, \ldots, q_n, q_D), (q_1, \ldots, q_i', \ldots, q_n, q_D') \in Q \land \]
\[ (q_i, a, q_i') \in \Delta_i \land a \in A^O_i \land (q_D, \gamma) \mapsto (q_D', a\gamma) \in T_D \land \gamma \in \Gamma \} \]
\[ \cup \]
\[ \{((q_1, \ldots, q_i, \ldots, q_n, q_D), a) \mapsto ((q_1, \ldots, q_i', \ldots, q_n, q_D'), \epsilon) |\]
\[ (q_1, \ldots, q_i, \ldots, q_n, q_D), (q_1, \ldots, q_i', \ldots, q_n, q_D') \in Q \land \]
\[ (q_i, a, q_i') \in \Delta_i \land a \in A^I_i \land (q_D, a) \mapsto (q_D', \epsilon) \in T_D \} \]
\[ \cup \]
\[ \{((q_1, \ldots, q_i, \ldots, q_n, q_D), a, (q_1, \ldots, q_i', \ldots, q_n, q_D)) |\]
\[ (q_1, \ldots, q_i, \ldots, q_n, q_D), (q_1, \ldots, q_i', \ldots, q_n, q_D) \in Q \land \]
\[ (q_i, a, q_i') \in \Delta_i \land a \in A^H_i \} \]
\[ \} \]
The System Behavior is a Pushdown System and we want to pick up behavior to build adaptor.

Specify criteria of adaptor in LTL formula $\phi$:
- No behavior mismatch
- Unbounded Messages

Model checking negation of $\phi$ to get a counterexample:
- The counterexample satisfies the criteria for adaptor
- Build an adaptor from the counterexample
- MOPED pushdown model checker (S. Schwoon, 2002)
Criteria: Behavior Mismatch Free

- Basic and essential criteria of adaptor: Make the system **deadlock free**
- Each component can reach its final state
- All traces of Pushdown System (the System Behavior) must:
  - Reach accepting (final) state.
  - Stack must be empty when accepting state is reached.
- We call the property: **Behavior Mismatch Free**
Define proposition for the final state with empty stack: $p_{accept}$

Use $\Diamond p_{accept}$ to indicate the final state is eventually reached

**Definition (Behavior Mismatch Free in LTL)**

Given an IPS $S = (Q, q^0, \Gamma, z, T, q^f)$ which is the composition of a set of composable IA4ADs and their adaptor, the property of behavior mismatch free is written in a LTL formula $\Diamond p_{accept}$. $p_{accept}$ is an atomic proposition and a labeling function for state $s$ and stack head $\gamma$ is defined as $L((s, \gamma)) : \{ p_{accept} \mid s = q^f \land \gamma = z \}$, where $z$ is the start symbol of stack.
Criteria: Unbounded Messages

- Behavior Mismatch Free is not enough
- *RawData* and *Data* should be multiply sent arbitrary times
  - Only one time is not acceptable (though no behavior mismatch)
  - *RawData* and *Data* are called **Unbounded Messages**
- Unbounded Messages are resulted by **Loop Transitions**
  - Fairness property of visiting loop transitions
Abstracted Algorithm of Finding Looped Transitions

1. For each component (IA4AD), search for Strongly Connected Components (SCCs) by Tarjan’s algorithm.
2. For each found SCC, locate and gather Looped Transitions.

Counterexample

A counterexample is a finite sequence of configurations:

$$\sigma = c_0 c_1 c_2 \ldots c_k.$$  

A configuration is a pair of state and stack content: $c = (q, w)$.

Demonstrated Idea of Algorithm

\[
\begin{align*}
( s_i, \langle a \ b \ z \rangle ) & \\
( s_{i+1}, \langle c \ a \ b \ z \rangle ) & \quad ?c \text{ (push transition)} \\
( s_{i+2}, \langle a \ b \ z \rangle ) & \quad !c \text{ (pop transition)}
\end{align*}
\]
Algorithm 1: Counterexample to adaptor

Input: A set of composable IA4WS: \(P_i = (Q_i, q_0^i, A_i^l, A_i^o, A_i^h, \Delta_i, q_f^i), i \in [1, n]\);
Configurations \(c_i = (s_i, w_i), i \in [0, m]\); Loop start index \(k\).

Output: Adaptor \(D = (Q_D, q_0^D, \Gamma_D, z, T_D, F_D)\)

\[Q_D \leftarrow \{s_k\}; \quad q_0^D \leftarrow s_0; \quad T \leftarrow \emptyset;\]
\[\Gamma_D := \bigcup_i A_i^o \cup \{z\} \cup \{\epsilon\};\]
foreach \(c_i = (s_i, w_i), i \in [0, m]\) do
  if \(i = m\) then
    \((s_i', w_i') = (s_k, w_k)\)
  else
    \((s_i', w_i') = (s_{i+1}, w_{i+1})\)
  \(Q_D \leftarrow Q_D \cup \{s_i\};\)
  if \(|w_i| - |w_i'| = 1\) then
    \(T_D \leftarrow T_D \cup \{(s_i, w_i(0)) \leftrightarrow (s_i', \epsilon)\};\)
  if \(|w_i| - |w_i'| = -1\) then
    \(T_D \leftarrow T_D \cup \{(s_i, w_i(0)) \leftrightarrow (s_i', w_i'(0)w_i(0))\};\)
  if \(|w_i| - |w_i'| = 0 \land s_i' \neq q_0^D\) then
    \(T_D \leftarrow T_D \cup \{(s_i, w_i(0)) \leftrightarrow (s_i', w_i'(0)), (s_i', w_i'(0)) \leftrightarrow (s_i, w_i(0))\};\)
\(F_D \leftarrow Q_D;\)
return \(D = (Q_D, q_0^D, \Gamma_D, z, T_D, F_D)\)
Tool Implementation

Parse input & Build Pushdown System

- Read Input (Behavior Interfaces)
  - Compute Coordinator
  - Compute Synchronous Composition with Coordinator
    - Compute Criteria of Adaptor
      - Pushdown System & LTL formula $\phi$
      - Pushdown Model Checking (MOPED)

Parse Counterexample & Build Adaptor

- Output Behavior Interface of Adaptor
  - Adaptor Generation
  - Counterexample
Tool Usage Demonstration

input file

```plaintext
# Fresh Market Update Example

service:: OnlineStockBroker
init: s0
final: s0
(s0, RawData, s1)
(s1, EndOfData, s2)
(s2, Start, s3)
(s3, Ack, s0)

service:: ResearchDepartment
init: s0
final: s0
(s0, RawData, s1)
(s1, Data, s2)
(s2, EndOfData, s4)
(s4, Complete, s0)

service:: Investor
init: s0
final: s0
(s0, Start, s1)
(s1, Data, s2)
(s2, Complete, s3)
(s3, Ack, s0)
```

adaptor

```
qu_0.0.0 -> q1_0.0.0 [label="?RawData<z>"];
qu_1.1.0 -> q2_1.0.0 [label="?EndOfData<z>"];
qu_2.1.0 -> q3_1.0.0 [label="?Start<EndOfData>"];
qu_3.1.1 -> q4_3.2.1 [label="?Data<EndOfData>"];
qu_4.3.2 -> q5_3.5.2 [label="?Complete<z>"];
qu_5.0.3 -> q6_5.4.0 [label="?Ack<z>"];
qu_6.1.0 -> q7_1.2.0 [label="?Data<z>"];
qu_7.2.0 -> q1_7.2.0 [label="?RawData<Data>"];
qu_1.3.0 -> q2_1.2.0 [label="?Data<Data>"];
qu_2.3.0 -> q3_2.3.0 [label="?Start<EndOfData>"];
qu_3.3.1 -> q4_3.2.1 [label="?Data<EndOfData>"];
qu_4.0.0 -> q1_4.1.0 [label="?RawData"];
qu_1.3.0 -> q2_1.1.1 [label="?Start"];
qu_2.2.1 -> q3_2.2.2 [label="?Data"];
qu_3.2.2 -> q4.3.4.2 [label="?EndOfData"];
qu_4.5.2 -> q5_4.5.3 [label="?Complete"];
qu_5.4.0 -> q6.5_4.4 [label="?Ack"];
qu_6.2.0 -> q1_6.3.0 [label="?RawData"];
qu_1.3.0 -> q2_1.3.1 [label="?Start"];
qu_2.4.2 -> q3.4_4.2 [label="?Data"];
```

Model checking by MOPED

Counterexample
1 Introduction
   - Mismatches and Adaptation
   - Motivation: Non-regular Adaptation

2 Coordinator Guided Adaptor Generation
   - Models
   - Coordinator

3 Experiments
   - Fresh Market Update Service

4 Conclusions
1 Introduction
   - Mismatches and Adaptation
   - Motivation: Non-regular Adaptation

2 Coordinator Guided Adaptor Generation
   - Models
   - Coordinator

3 Experiments
   - Fresh Market Update Service

4 Conclusions
Fresh Market Update Service

Online Stock Broker

- !RawData
  - !RawData
  - !EndOfData
  - !Start
  - ?Ack

Research Department

- ?RawData
  - !Data
  - ?RawData
  - !Complete

Investor

- ?Start
  - ?Data
  - ?Complete
  - !Ack
# Fresh Market Update Service

service:: OnlineStockBroker
init::S0
final::S4
(S0, !RawData, S1) (S1, !RawData, S1)
(S1, !EndOfData, S2)
(S2, !Start, S3) (S3, ?Ack, S4)

service:: ResearchDepartment
init::S0
final::S5
(S0, ?RawData, S1) (S1, !Data, S2)
(S2, ?RawData, S3)
(S3, !Data, S2) (S2, ?EndOfData, S4)
(S4, !Complete, S5)

service:: Investor
init::S0
final::S4
(S0, ?Start, S1) (S1, ?Data, S2) (S2, ?Data, S2)
(S2, ?Complete, S3) (S3, !Ack, S4)
# initial state
(q0_0_0_c0 <z>)

# transition rules
q0_0_0_c0 <RawData> --> q1_0_0_c0 <RawData RawData>
q0_0_0_c0 <EndOfData> --> q1_0_0_c0 <RawData EndOfData>
q0_0_0_c0 <Start> --> q1_0_0_c0 <RawData Start>
q0_0_0_c0 <Data> --> q1_0_0_c0 <RawData Data>
q0_0_0_c0 <Complete> --> q1_0_0_c0 <RawData Complete>
q0_0_0_c0 <Ack> --> q1_0_0_c0 <RawData Ack>
q0_0_0_c0 <z> --> q1_0_0_c0 <RawData z>
q0_0_0_c0 <RawData> --> q0_1_0_c0 <>
q0_0_0_c0 <Start> --> q0_0_1_c0 <>
q0_0_1_c0 <RawData> --> q1_0_1_c0 <RawData RawData>
q0_0_1_c0 <EndOfData> --> q1_0_1_c0 <RawData EndOfData>
q0_0_1_c0 <Start> --> q1_0_1_c0 <RawData Start>
q0_0_1_c0 <Data> --> q1_0_1_c0 <RawData Data>
q0_0_1_c0 <Complete> --> q1_0_1_c0 <RawData Complete>
q0_0_1_c0 <Ack> --> q1_0_1_c0 <RawData Ack>
q0_0_1_c0 <z> --> q1_0_1_c0 <RawData z>
q0_0_1_c0 <RawData> --> q0_1_1_c0 <>

......
......
Fresh Market Update Service: Adaptor

q0_0_0 -> q1_0_0 [label="?RawData<z>"];
q1_1_0 -> q2_1_0 [label="?EndOfData<z>"];
q2_1_0 -> q3_1_0 [label="?Start<EndOfData>"];
q3_1_1 -> q3_2_1 [label="?Data<EndOfData>"];
q3_4_2 -> q3_5_2 [label="?Complete<z>"];
q3_5_3 -> q3_5_4 [label="?Ack<z>"];
q1_1_0 -> q1_2_0 [label="?Data<z>"];
q1_2_0 -> q1_2_0 [label="?RawData<Data>"];
q1_3_0 -> q1_2_0 [label="?Data<Data>"];
q1_3_0 -> q2_3_0 [label="?EndOfData<Data>"];
q2_3_0 -> q3_3_0 [label="?Start<EndOfData>"];
q3_3_1 -> q3_2_1 [label="?Data<EndOfData>"];
q1_0_0 -> q1_1_0 [label="!RawData"];
q3_1_0 -> q3_1_1 [label="!Start"]; q3_2_1 -> q3_2_2 [label="!Data"]; q3_2_2 -> q3_4_2 [label="!EndOfData"];
q3_5_2 -> q3_5_3 [label="!Complete"];
q3_5_4 -> q4_5_4 [label="!Ack"]; q1_2_0 -> q1_3_0 [label="!RawData"]; q3_3_0 -> q3_3_1 [label="!Start"]; q3_4_2 -> q3_4_2 [label="!Data"];
Fresh Market Update Service: Adaptor (Graphviz)
Fresh Market Update Service: Adaptor
BPEL adaptor

- Adaptor can be implemented by two BPEL processes: BPEL-adaptor and BPEL-stack
- Simple directions of implementation are given
Assembling and Testing
Outline

1 Introduction
   - Mismatches and Adaptation
   - Motivation: Non-regular Adaptation

2 Coordinator Guided Adaptor Generation
   - Models
   - Coordinator

3 Experiments
   - Fresh Market Update Service

4 Conclusions
Contributions of the Approach

Dealing with non-regular behavior in adaptors due to behavioral mismatches

- The approach introduces pushdown systems model to represent adaptors with non-regular behavior
- A first attempt in adaptation

Automated adaptor generation using pushdown model checking

- The approach uses pushdown model checking for adaptor generation
- Fully automated adaptor generation is achieved in the approach
Future Work

Dealing with scalability

- Size of pushdown system model for MOPED increases with
  - Number of components - introducing mapping components magnify the problem
- Improve encoding of input for pushdown model checker
- Reduce fairness conditions - Some transitions is not really necessary

Automatic or systematic way of specifying mapping components

- Mapping components are manually specified (from mappings of labels) and may influence result of adaptor generation
- Techniques for systematic generation of mappings of labels are needed
Thank You.

Q & A
Supplementaries
Considering General Cases of Adaptation

- Our approach focuses on generating non-regular adaptor for behavioral mismatches
- Applicability on general cases of adaptation is not confirmed
- From investigations of work of adaptation, we should consider
  - General case 1: signature mismatches
  - General case 2: branchings
  - Example: File Download Service Fresh Market Update Service ver.2
Signature mismatches are not mentioned

- Our approach generates adaptors for behavioral mismatches
- Signature mismatches are either solved or not exist in behavior interfaces
  - Unrealistic

Mapping of labels can be represented by mapping components

- Mapping components are special components representing mappings of labels
- Still form a closed system after introducing mapping components
- Our approach is also applicable when signature mismatches exist
Examples of Mapping Components

- ! ok → ? ok
- ? ack → ! ack
- ? title → ? author
- ! title → ! author
- ? title → ? author
- ! author → ! title
Branchings - Problems

Branchings are common in behavior interfaces

- Perform decision making based on received message or computation results
- Basic scenario: returning results of `succ` or `fail`
- Branchings are usually exclusive: only one can be chose in one execution

Our approach needs improvements for exclusive branchings

- Pushdown model checking returns a counterexample for building an adaptor
  - A counterexample is a trace
  - A trace can only execute through a branching
  - Generally, an adaptor is expected to be able to execute for all branchings
Branchings - Solution

- Before pushdown model checking, add a epsilon move connecting final and initial state of the coordinator adapted system.
- Model checking algorithm can search through the behavior multiple times.
- Marking Branching Transitions for specifying fairness property for branchings.
- MOPED checks all marked branchings now.
Extended FMUS Service
File Download Service - Mapping Components

Mapping 1
- ?UserName
- ?Password
- !Login

Mapping 2
- ?Connected

Mapping 3
- ?Download
- !GetFile

Mapping 4
- ?Result
- !Data

Mapping 5
- ?NoSuchFile
- !Data

Mapping 6
- !Quit
File Download Service - Adaptor
## Performance

<table>
<thead>
<tr>
<th></th>
<th>FMUS service</th>
<th>FD service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num. of Components/(Mapping Components)</td>
<td>3(0)</td>
<td>2(6)</td>
</tr>
<tr>
<td>Num. of Composite States</td>
<td>150</td>
<td>840</td>
</tr>
<tr>
<td>Num. of Lines of PDS</td>
<td>1923</td>
<td>42248</td>
</tr>
<tr>
<td>Num. of Transitions in Fairness Property</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Num. of Configurations in Counterexample</td>
<td>23</td>
<td>61</td>
</tr>
<tr>
<td>Exec. Time (Tool part1)</td>
<td>0.169 sec</td>
<td>1.091 sec</td>
</tr>
<tr>
<td>Exec. Time (Tool part2)</td>
<td>0.092 sec</td>
<td>0.154 sec</td>
</tr>
<tr>
<td>Exec. Time (Never Claim)</td>
<td>0.077 sec</td>
<td>0.092 sec</td>
</tr>
<tr>
<td>Exec. Time (MOPED)</td>
<td>0.62 sec</td>
<td>61.25 sec</td>
</tr>
</tbody>
</table>

**Table**: Some Data from Experiments (P4-1.83G, 2G RAM)