Verification of Communication Protocols in Web-Services

Anshuman Mukherjee
Prof. Zahir Tari and Assoc. Prof. Peter Bertok

School of Computer Science and IT
RMIT University
Melbourne, Australia

August 26, 2011

PhD Completion Seminar
Agenda

1. Introduction
2. Previous Work
3. Proposed Technique
4. Results
5. Conclusion
1 Introduction
- Motivation
- Service Composition
- The Problem
- Solution
- Formal Methods
- The Problem

2 Previous Work

3 Proposed Technique

4 Results

5 Conclusion

Verification of Communication Protocols in Web-Services
Companies Exposing their Functionality as Services

Why consume these service?

Anshuman Mukherjee et. al., RMIT University, Melbourne

Verification of Communication Protocols in Web-Services
Companies Exposing their Functionality as Services

Why consume these service?

⇒ Reduces time to market for consumers
Companies Exposing their Functionality as Services

Why consume these service?

⇒ Reduces time to market for consumers
⇒ Renders loosely coupled systems
Companies Exposing their Functionality as Services

Why consume these service?

⇒ Reduces time to market for consumers
⇒ Renders loosely coupled systems
⇒ Allows reusing a comprehensively tested functionality
Why consume these service?

- Reduces time to market for consumers
- Renders loosely coupled systems
- Allows reusing a comprehensively tested functionality
- No dependency on the underlying platform or technology
Companies Exposing their Functionality as Services

Why consume these service?

⇒ Reduces time to market for consumers
⇒ Renders loosely coupled systems
⇒ Allows reusing a comprehensively tested functionality
⇒ No dependency on the underlying platform or technology
⇒ Has opened new business opportunities for service providers.

Anshuman Mukherjee et. al., RMIT University, Melbourne
Why do we need to compose services?
Why do we need to compose services?

⇒ Calling individual services is cumbersome & exposes business logic
Why do we need to compose services?

⇒ Calling individual services is cumbersome & exposes business logic
⇒ Composing the services encapsulates business logic
The Problem

How reliable is the service composition?
The Problem

How reliable is the service composition?

⇒ There could be errors in service description
The Problem

How reliable is the service composition?

⇒ There could be errors in service description
⇒ Dynamic discovery and composition of services could lead to delinquency
The Problem

How reliable is the service composition?

⇒ There could be errors in service description
⇒ Dynamic discovery and composition of services could lead to delinquency
⇒ Ambiguities in BPEL language could undermine composition
How to verify a composition?

Traditional methods cannot be used to verify a service composition because:
How to verify a composition?

Traditional methods cannot be used to verify a service composition because:

⇒ Error is usually in the composition rather than source code
How to verify a composition?

Traditional methods cannot be used to verify a service composition because:

⇒ Error is usually in the composition rather than source code
⇒ The source code is not available for modification
How to verify a composition?

Traditional methods cannot be used to verify a service composition because:

⇒ Error is usually in the composition rather than source code
⇒ The source code is not available for modification
⇒ Modifying the source code could break other consuming applications
The Solution

How to verify a dynamic service composition?

- Uninvestigated Runtime States
- Design time States
The Solution

How to verify a dynamic service composition?

⇒ Verify the reachable states of the initial composition statically using a formal method.

[Diagram showing uninvestigated runtime states and design time states]
The Solution

How to verify a dynamic service composition?

⇒ Verify the reachable states of the initial composition statically using a formal method.
⇒ Thereupon monitor the states reached at runtime to detect any unverified state.
How to verify a dynamic service composition?

⇒ Verify the reachable states of the initial composition statically using a formal method.
⇒ Thereupon monitor the states reached at runtime to detect any unverified state.
⇒ Verify these detected states at runtime.
What are Formal Methods?
What are Formal Methods?

Formal Methods are languages, techniques and tools for specification and verification of the system behaviour.
Formal Methods

What are Formal Methods?
Formal Methods are languages, techniques and tools for specification and verification of the system behaviour.

Automated Theorem Proving
The property to be verified is stated as a theorem and proved using the system description, a set of logical axioms and inference rules.
Formal Methods

What are Formal Methods?

Formal Methods are languages, techniques and tools for specification and verification of the system behaviour.

Automated Theorem Proving

The property to be verified is stated as a theorem and proved using the system description, a set of logical axioms and inference rules.

Model Checking

A system verifies certain properties by means of an exhaustive search of all possible states that a system could enter during its execution.
What is Model Checking?

Model Checking is a formal method which verifies a concurrent system automatically. It has 3 basic steps.
What is Model Checking?

Model Checking is a formal method which verifies a concurrent system automatically. It has 3 basic steps.

Basic Steps of Model Checking

⇒ Modeling the system
Model Checking

What is Model Checking?

Model Checking is a formal method which verifies a concurrent system automatically. It has 3 basic steps.

Basic Steps of Model Checking

⇒ Modeling the system
⇒ Specifying the properties to be verified

\[ (\Box (\text{call} \land \Diamond \text{open}) \rightarrow (\neg \text{atfloor} \land \neg \text{open}) \cup (\text{open} \lor ((\neg \text{atfloor} \land \neg \text{open}) \cup (\text{open} \lor ((\neg \text{atfloor} \land \neg \text{open}) \cup (\text{open} \lor (\neg \text{atfloor} \land \text{not open})))))) \]
Model Checking

What is Model Checking?

Model Checking is a formal method which verifies a concurrent system automatically. It has 3 basic steps.

Basic Steps of Model Checking

⇒ Modeling the system
⇒ Specifying the properties to be verified
⇒ Verifying the properties using reachability graph
What is State-Space Explosion?

For a sufficiently complex system, the number of reachable states could be considerably large leading to a gigantic reachability graph.
The Problem and Solution

What is State-Space Explosion?

For a sufficiently complex system, the number of reachable states could be considerably large leading to a gigantic reachability graph.

Necessity of Storing States

⇒ If it is possible to generate a state using different sequence of events, it will be analysed more than once along with states reachable from it.
The Problem and Solution

What is State-Space Explosion?

For a sufficiently complex system, the number of reachable states could be considerably large leading to a gigantic reachability graph.

Necessity of Storing States

⇒ if there exists a non-empty sequence of events that cause no net change in state, the analysis might never terminate.
The Problem and Solution

What is State-Space Explosion?
For a sufficiently complex system, the number of reachable states could be considerably large leading to a gigantic reachability graph.

As a Result
⇒ Consequently, the memory costs increase tremendously
The Problem and Solution

What is State-Space Explosion?

For a sufficiently complex system, the number of reachable states could be considerably large leading to a gigantic reachability graph.

As a Result

⇒ Furthermore, there is a tremendous delay involved.
The Problem and Solution

What is State-Space Explosion?
For a sufficiently complex system, the number of reachable states could be considerably large leading to a gigantic reachability graph.

Solution Proposed
⇒ We propose to reduce the memory costs by storing the states as the difference from previous state.
What is State-Space Explosion?
For a sufficiently complex system, the number of reachable states could be considerably large leading to a gigantic reachability graph.

Solution Proposed
⇒ We propose to reduce the time requirements by exploring all modules of a hierarchical model in parallel.
What’s Next?

1 Introduction

2 Previous Work
   - BPEL Verification Techniques
   - Comparing Space Requirements
   - Comparing Delay

3 Proposed Technique

4 Results

5 Conclusion
## Existing techniques for formalising a BPEL specification

<table>
<thead>
<tr>
<th>Related Work</th>
<th>Tech</th>
<th>LN</th>
<th>ECF</th>
<th>TOOL</th>
<th>VIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Ferrara, 2004]</td>
<td>PA</td>
<td>☒</td>
<td>●</td>
<td>☒</td>
<td>☒</td>
</tr>
<tr>
<td>[Fahland, 2005]</td>
<td>ASM</td>
<td>●</td>
<td>●</td>
<td>☒</td>
<td>☒</td>
</tr>
<tr>
<td>[Fu et al., 2004]</td>
<td>AM</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>[Kang et al., 2007]</td>
<td>CPN</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>[Sloan and Khoshgoftaar, 2009]</td>
<td>CPN</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>[Stahl, 2005]</td>
<td>PN</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>☒</td>
</tr>
<tr>
<td>[Ouyang et al., 2007]</td>
<td>PN</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
</tbody>
</table>

**Tech:** Technique, **LN:** Support for Links, **ECF:** Support for handlers, **TOOL:** Tool support, **VIS:** Visualisation of model
What's the problem with existing solutions?
Problems with existing solutions

What’s the problem with existing solutions?

⇒ They target specific modeling language
What's the problem with existing solutions?

- They target specific modeling language
- Most of them do not formalise all BPEL activities
Problems with existing solutions

What’s the problem with existing solutions?

⇒ They target specific modeling language
⇒ Most of them do not formalise all BPEL activities
⇒ Most of them involve manual transformation
Problems with existing solutions

What’s the problem with existing solutions?

⇒ They target specific modeling language
⇒ Most of them do not formalise all BPEL activities
⇒ Most of them involve manual transformation
⇒ They are error prone and cumbersome
Problems with existing solutions

What’s the problem with existing solutions?

⇒ They target specific modeling language
⇒ Most of them do not formalise all BPEL activities
⇒ Most of them involve manual transformation
⇒ They are error prone and cumbersome
⇒ They are ad-hoc
The reduction provided at the expense of processing time is compared with related work.

<table>
<thead>
<tr>
<th>Method</th>
<th>Run-Time</th>
<th>Memory-Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Algorithm</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>[Schmidt, 2003]</td>
<td>130%</td>
<td>60%</td>
</tr>
<tr>
<td>[Evangelista and Pradat-Peyre, 2005]</td>
<td>300%</td>
<td>05%</td>
</tr>
<tr>
<td>[Holzmann, 1997]</td>
<td>280%</td>
<td>18.3%</td>
</tr>
<tr>
<td>Sequential Algorithm proposed</td>
<td>200%</td>
<td>05%</td>
</tr>
</tbody>
</table>
Comparing Delay

The processing time is compared with related work.

**Table:** A comparison of the proposed solutions

<table>
<thead>
<tr>
<th>Method</th>
<th>Run-Time</th>
<th>Number of Markings</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Algorithm</td>
<td>100%</td>
<td>Any</td>
</tr>
<tr>
<td>Evangelista and Pradat-Peyre (2006)</td>
<td>49.18%</td>
<td>45780</td>
</tr>
<tr>
<td>Kristensen and Valmari (1998)</td>
<td>89.3%</td>
<td>25</td>
</tr>
<tr>
<td>Proposed Technique</td>
<td>14%</td>
<td>25000</td>
</tr>
</tbody>
</table>

Anshuman Mukherjee et. al., RMIT University, Melbourne
What’s Next?

1. Introduction

2. Previous Work

3. Proposed Technique
   - Verifying a BPEL Specification
   - Memory Efficient State-Space Reduction
   - Time Efficient State-Space Reduction
   - Introducing Hierarchy into a Flat Model

4. Results

5. Conclusion

Anshuman Mukherjee et. al., RMIT University, Melbourne

Verification of Communication Protocols in Web-Services
Business Process Execution Language

Anshuman Mukherjee et. al., RMIT University, Melbourne
Business Process Execution Language

\[ \Rightarrow \text{BPEL is the de-facto industry standard for service composition} \]
Business Process Execution Language

⇒ BPEL is the de-facto industry standard for service composition
⇒ It offers a set of activities to create a business work-flow.
Business Process Execution Language

⇒ BPEL is the de-facto industry standard for service composition
⇒ It offers a set of activities to create a business work-flow.
⇒ It has both block and graph elements.

Diagram:
- Activity
  - BasicActivity
    - Invoke
      - Wait
    - Receive
    - Reply
    - Assign
      - Throw
      - Empty
      - Terminate
  - StructuredActivity
    - Flow
    - Sequence
    - Switch
    - Pick
    - While
  - Scope
  - Compensate
Business Process Execution Language

⇒ BPEL is the de-facto industry standard for service composition
⇒ It offers a set of activities to create a business work-flow.
⇒ It has both block and graph elements.
⇒ It offers several advantages over any programming language in composing services.
Business Process Execution Language

⇒ BPEL is the de-facto industry standard for service composition
⇒ It offers a set of activities to create a business work-flow.
⇒ It has both block and graph elements.
⇒ It offers several advantages over any programming language in composing services.
⇒ However, it has textual specification and lacks mathematical semantics.
The solution

What is the solution?
What is the solution?

⇒ Transform BPEL into generic intermediate wherein:
The solution

What is the solution?

⇒ Transform BPEL into generic intermediate wherein:

⇒⇒ Transformation is automatic
The solution

What is the solution?

- Transform BPEL into generic intermediate wherein:
  - Transformation is automatic
  - Intermediate assimilate all the information from BPEL
What is the solution?

⇒ Transform BPEL into generic intermediate wherein:
⇒⇒ Transformation is automatic
⇒⇒ Intermediate assimilate all the information from BPEL
⇒⇒ This data can be accessed programmatically for actual formalisation
What is the solution?

⇒ Transform BPEL into generic intermediate wherein:
⇒ Transformation is automatic
⇒ Intermediate assimilate all the information from BPEL
⇒ This data can be accessed programmatically for actual formalisation
⇒ DTOs are design patterns for storing & transferring data between software sub-systems
The solution

What is the solution?

⇒ Transform BPEL into generic intermediate wherein:
⇒⇒ Transformation is automatic
⇒⇒ Intermediate assimilate all the information from BPEL
⇒⇒ This data can be accessed programmatically for actual formalisation
⇒ DTOs are design patterns for storing & transferring data between software sub-systems
⇒ It satisfies all the requirements necessitated by an intermediate
Converting BPEL to DTOs

How to convert BPEL to DTOs?
How to convert BPEL to DTOs?

⇒ BPEL has been converted into DTOs by Anshuman Mukherjee et. al., RMIT University, Melbourne.
Converting BPEL to DTOs

How to convert BPEL to DTOs?

⇒ BPEL has been converted into DTOs by
⇒⇒ Mapping each BPEL activity into a Java Bean
Converting BPEL to DTOs

How to convert BPEL to DTOs?

⇒ BPEL has been converted into DTOs by
⇒⇒ Mapping each BPEL activity into a Java Bean
⇒⇒ Extending the Spring framework to initialise these beans from BPEL

1. Instantiation

2. Dependency Injection

BPEL Process

Java Beans

Bean Factory

Component - A

4. Marshal

JAXB APIs

Component - B

XML based formal model

XML Schema or DTD

Binding Compiler

Schema derived Classes & Interfaces

Anshuman Mukherjee et al., RMIT University, Melbourne

Verification of Communication Protocols in Web-Services
How to convert DTOs to XML based formal model?

⇒ DTOs are converted into XML based model by:

1. Instantiation

2. Dependency Injection

- JAXB APIs
- Component - A

3. Bind

- Binding Compiler
- Component - B

4. Marshal

- JAXB APIs
- XML based formal model
- XML Schema or DTD

- Schema derived Classes & Interfaces
Converting BPEL to DTOs

How to convert DTOs to XML based formal model?

⇒ DTOs are converted into XML based model by:

⇒⇒ Compiling schema of an existing solution into meta-classes.

1. Instantiation
2. Dependency Injection
3. Bind
4. Marshal

BPEL Process
Java Beans
Bean Factory
Component - A

XML based formal model
JAXB APIs
Component - B

XML Schema or DTD
Binding Compiler
Schema derived Classes & Interfaces
Converting BPEL to DTOs

How to convert DTOs to XML based formal model?

⇒ DTOs are converted into XML based model by:
⇒ Compiling schema of an existing solution into meta-classes.
⇒ Initialise the meta-classes using DTOs and marshal them into an XML based formal model.
How to Map BPEL activities into Java Beans?

```java
class Invoke {
    private PartnerLink partnerLink;
    private PortType portType;
    private Operation operation;
    private InputVariable inputVariable;
    private OutputVariable outputVariable;
    // getters and setters
}
```
How to Map BPEL activities into Java Beans?

Anshuman Mukherjee et. al., RMIT University, Melbourne

Verification of Communication Protocols in Web-Services
What do we extend?

```xml
<partnerLink name="PL1"
  partnerRole="PR1"
  partnerLinkType="PLT1"/>

<variable name="V1"
  messageType="MT1"/>

<variable name="V2"
  messageType="MT2"/>

<invoke name="I1"
  partnerLink="PL1"
  portType="PT1"
  operation="OP1"
  inputVariable="V1"
  outputVariable="V2"/>
```

BPEL

Spring
What are Coloured Petri-Nets?

**Coloured Petri-Nets**

- CPNs are a graphically oriented modelling language for concurrent systems based on Petri Nets and the functional programming language Standard ML.
What are Coloured Petri-Nets?

Coloured Petri-Nets

- CPNs are a graphically oriented modelling language for concurrent systems based on Petri Nets and the functional programming language Standard ML.
- The circles are called Places.
What are Coloured Petri-Nets?

Coloured Petri-Nets

- CPNs are a graphically oriented modelling language for concurrent systems based on Petri Nets and the functional programming language Standard ML.
- The number of tokens in a place and their names are inscribed next to it.
What are Coloured Petri-Nets?

Coloured Petri-Nets

- CPNs are a graphically oriented modelling language for concurrent systems based on Petri Nets and the functional programming language Standard ML.
- A place also has its colour inscribed next to it.
What are Coloured Petri-Nets?

Coloured Petri-Nets

- CPNs are a graphically oriented modelling language for concurrent systems based on Petri Nets and the functional programming language Standard ML.
- The rectangles are known as Transitions.

Anshuman Mukherjee et. al., RMIT University, Melbourne
Verification of Communication Protocols in Web-Services
What are Coloured Petri-Nets?

Coloured Petri-Nets

- CPNs are a graphically oriented modelling language for concurrent systems based on Petri Nets and the functional programming language Standard ML.
- A place and a transition are connected by one or more Arcs.
What are Coloured Petri-Nets?

Coloured Petri-Nets

- CPNs are a graphically oriented modelling language for concurrent systems based on Petri Nets and the functional programming language Standard ML.
- A transition (T1) fires for a particular binding (x=7), often displacing one or more tokens (1\'7).
States in CPN

Representing a CPN State
States in CPN

Representing a CPN State

⇒ Each token in a CPN model is assigned a unique name.
States in CPN

Representing a CPN State

⇒ Each token in a CPN model is assigned a unique name.
⇒ The containing place for a token is assigned using an arrow ‘→’.
States in CPN

Representing a CPN State

⇒ Each token in a CPN model is assigned a unique name.
⇒ The containing place for a token is assigned using an arrow ‘→’.
⇒ The value of token is assigned using an equal ‘=’.
States in CPN

Change in State of CPN Model
Change in State of CPN Model

⇒ When a token moves to a different place or has its value altered, there is a corresponding change in state.
States in CPN

Change in State of CPN Model

⇒ When a token moves to a different place or has its value altered, there is a corresponding change in state.
⇒ Creation or deletion of a token also causes a change in state.
Storing States in Difference Form

Sequential Algorithm
Storing States in Difference Form

Sequential Algorithm

⇒ Sequential algorithm stores states in difference form
Storing States in Difference Form

Sequential Algorithm

⇒ Sequential algorithm stores states in difference form
⇒ The difference form of a state $S_{st}$, with previous state $S_{pv}$, is defined as the changes necessary in $S_{pv}$ to generate $S_{st}$. 

Anshuman Mukherjee et. al., RMIT University, Melbourne
Storing States in Difference Form

Sequential Algorithm

⇒ Sequential algorithm stores states in difference form
⇒ The difference form of a state $S_{st}$, with previous state $S_{pv}$, is defined as the changes necessary in $S_{pv}$ to generate $S_{st}$.
Storing States in Difference Form

Sequential Algorithm

⇒ Sequential algorithm stores states in difference form
⇒ The difference form of a state $S_{st}$, with previous state $S_{pv}$, is defined as the changes necessary in $S_{pv}$ to generate $S_{st}$.
Backtracking: Expanding a Difference State

Need for Backtracking
Need for Backtracking

⇒ During state-space analysis, a state should be analysed only once when it is generated for the first time.
Need for Backtracking

⇒ During state-space analysis, a state should be analysed only once when it is generated for the first time.

⇒ To ensure this, a generated state is compared with stored states to determine if it is new.
Need for Backtracking

⇒ During state-space analysis, a state should be analysed only once when it is generated for the first time.

⇒ To ensure this, a generated state is compared with stored states to determine if it is new.

⇒ However, the stored states are in difference form which could be same for dissimilar states.
Need for Backtracking

⇒ During state-space analysis, a state should be analysed only once when it is generated for the first time.

⇒ To ensure this, a generated state is compared with stored states to determine if it is new.

⇒ However, the stored states are in difference form which could be same for dissimilar states.

⇒ Consequently, the states must be expanded before comparison.
Backtracking: Expanding a Difference State

What is Backtracking?
Backtracking: Expanding a Difference State

What is Backtracking?

Backtracking is the process of regenerating a state by recursively adding the most recent changes for each token to its previous state until a state stored in explicit form is reached.
Backtracking: Expanding a Difference State

What is Backtracking?

Backtracking is the process of regenerating a state by recursively adding the most recent changes for each token to its previous state until a state stored in explicit form is reached.
Reduce Cost of Backtracking

The Value of $\delta$

$\Rightarrow$ Backtracking is necessary to expand a state
Reduce Cost of Backtracking

The Value of $\delta$

⇒ Backtracking is necessary to expand a state
⇒ However, Backtracking is costly
Reduce Cost of Backtracking

The Value of $\delta$

$\Rightarrow$ Backtracking is necessary to expand a state
$\Rightarrow$ However, Backtracking is costly
$\Rightarrow$ To reduce the costs, each state at depth $\delta$ is stored in explicit form

Diagram showing states and transitions in a tree structure.
Hierarchical CPN

Representing a HCPN model
Representing a HCPN model

⇒ Consists of a set of non-hierarchical CPN, called *modules*
Hierarchical CPN

Representing a HCPN model

⇒ Consists of a set of non-hierarchical CPN, called *modules*
⇒ Substitution transition acts as a substitute for a module.
Hierarchical CPN

Representing a HCPN model

⇒ Consists of a set of non-hierarchical CPN, called **modules**
⇒ Substitution transition acts as a substitute for a module.
⇒ The module executes when its substitution transition fires.
Hierarchical CPN

Representing a HCPN model
Representing a HCPN model

⇒ HCPN binds the modules together using port-socket mapping.
Rep resenting a HCPN model

⇒ HCPN binds the modules together using port-socket mapping.
⇒ During state-space analysis, the modules are combined to produce a flat model.
Parallel State-Space Exploration

Anshuman Mukherjee et. al., RMIT University, Melbourne

Verification of Communication Protocols in Web-Services
Parallel State-Space Exploration

⇒ The modules of a HCPN model are explored in parallel.
Parallel State-Space Exploration

⇒ The modules of a HCPN model are explored in parallel.
⇒ This helps in reducing the delay.
Access-table & Parametrised Reachability Graph

Access-table
**Access-table & Parametrised Reachability Graph**

**Access-table**

⇒ Emulates the behaviour of a module.

### Access-table

The access-table for Sender

<table>
<thead>
<tr>
<th>Input Ports</th>
<th>Condition</th>
<th>Output Ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send Data</td>
<td>D</td>
<td>n=v?</td>
</tr>
<tr>
<td>Data</td>
<td>empty</td>
<td>A</td>
</tr>
<tr>
<td>Ack(n)</td>
<td>empty</td>
<td>Send Data</td>
</tr>
<tr>
<td>Data(n,d)</td>
<td>YES</td>
<td>Data(n,d)^+</td>
</tr>
<tr>
<td>empty</td>
<td>NO</td>
<td>empty</td>
</tr>
<tr>
<td>Ack(m)</td>
<td>-</td>
<td>empty</td>
</tr>
<tr>
<td>empty</td>
<td>empty</td>
<td>empty</td>
</tr>
</tbody>
</table>

---

Anshuman Mukherjee et. al., RMIT University, Melbourne

Verification of Communication Protocols in Web-Services
Access-table & Parametrised Reachability Graph

**Access-table**

⇒ Emulates the behaviour of a module.
⇒ Used to determine the outcome of executing a module.

Table: The access-table for Sender

<table>
<thead>
<tr>
<th>Input Ports</th>
<th>Condition</th>
<th>Output Ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send Data</td>
<td>D</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>n=v?</td>
<td>Send Data</td>
</tr>
<tr>
<td>Data(n,d)−</td>
<td>empty</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data(n,d)+</td>
</tr>
<tr>
<td>Data(n,d)+</td>
<td>empty</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>empty</td>
</tr>
<tr>
<td>empty</td>
<td>Ack(m)−</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>empty</td>
</tr>
</tbody>
</table>

Anshuman Mukherjee et. al., RMIT University, Melbourne
Access-table & Parametrised Reachability Graph

Access-table

⇒ Emulates the behaviour of a module.
⇒ Used to determine the outcome of executing a module.
⇒ Contains all possible set of inputs for which a module is enabled.

Table: The access-table for Sender

<table>
<thead>
<tr>
<th>Input Ports</th>
<th>Condition</th>
<th>Output Ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send Data</td>
<td>n=v?</td>
<td>A</td>
</tr>
<tr>
<td>Data</td>
<td></td>
<td>Send Data</td>
</tr>
<tr>
<td>Data(n,d)−</td>
<td>empty</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data(n,d)⁺</td>
</tr>
<tr>
<td>Ack(m)⁻</td>
<td>empty</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>empty</td>
</tr>
<tr>
<td>empty</td>
<td>Ack(m)⁻</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>empty</td>
</tr>
</tbody>
</table>

Anshuman Mukherjee et. al., RMIT University, Melbourne
Access-table & Parametrised Reachability Graph

Access-table

⇒ Emulates the behaviour of a module.
⇒ Used to determine the outcome of executing a module.
⇒ Contains all possible set of inputs for which a module is enabled.
⇒ Maps each input to the set of tokens produced by module after execution.

Table: The access-table for Sender

<table>
<thead>
<tr>
<th>Input Ports</th>
<th>Condition</th>
<th>Output Ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send Data</td>
<td>D</td>
<td>n=v?</td>
</tr>
<tr>
<td>Send Data</td>
<td>empty</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>empty</td>
<td>NO</td>
</tr>
<tr>
<td>Ack(n)</td>
<td></td>
<td>empty</td>
</tr>
<tr>
<td>empty</td>
<td>Ack(m)</td>
<td>-</td>
</tr>
<tr>
<td>empty</td>
<td>empty</td>
<td>empty</td>
</tr>
<tr>
<td></td>
<td>empty</td>
<td>empty</td>
</tr>
<tr>
<td>PACKET</td>
<td></td>
<td>empty</td>
</tr>
<tr>
<td>SendPacket</td>
<td></td>
<td>empty</td>
</tr>
<tr>
<td>Data(n,d) −</td>
<td>empty</td>
<td>empty</td>
</tr>
<tr>
<td>Data(n,d) +</td>
<td></td>
<td>empty</td>
</tr>
<tr>
<td></td>
<td></td>
<td>empty</td>
</tr>
<tr>
<td>Data(n,d) +</td>
<td>empty</td>
<td>empty</td>
</tr>
<tr>
<td></td>
<td>empty</td>
<td>empty</td>
</tr>
<tr>
<td></td>
<td>empty</td>
<td>empty</td>
</tr>
</tbody>
</table>
Access-table & Parametrised Reachability Graph

Parametrised Reachability Graph

Anshuman Mukherjee et. al., RMIT University, Melbourne
Access-table & Parametrised Reachability Graph

⇒ Access table contains information of port places.

Anshuman Mukherjee et. al., RMIT University, Melbourne

Verification of Communication Protocols in Web-Services
Access-table & Parametrised Reachability Graph

⇒ Access table contains information of port places.
⇒ Consequently the information for non-port places might be lost.

Anshuman Mukherjee et. al., RMIT University, Melbourne
Access-table & Parametrised Reachability Graph

⇒ Access table contains information of port places.
⇒ Consequently the information for non-port places might be lost.
⇒ This is captured using Parametrised Reachability Graph.

$\text{Send Data} \rightarrow \emptyset$
$\text{NextSend} \rightarrow 1^{\prime}v$
$D \Rightarrow 1^{\prime}\text{Ack}(m)$
$A \Rightarrow \emptyset$

$\text{Send Data} \rightarrow \emptyset$
$\text{NextSend} \rightarrow 1^{\prime}m$
$D \Rightarrow \emptyset$
$A \Rightarrow \emptyset$

$\text{Send Data} \rightarrow 1^{\prime}\text{(Data}(n,d))$
$\text{NextSend} \rightarrow 1^{\prime}v$
$D \Rightarrow \emptyset$
$A \Rightarrow \emptyset$

$\text{Send Data} \rightarrow 1^{\prime}\text{(Data}(n,d))$
$\text{NextSend} \rightarrow 1^{\prime}v$
$D \Rightarrow \emptyset$
$A \Rightarrow 1^{\prime}\text{(Data}(n,d))$

Anshuman Mukherjee et. al., RMIT University, Melbourne
Verification of Communication Protocols in Web-Services
How to Introduce Hierarchy into a Flat model?

⇒ Identify structurally similar components in a model.

Need to:
1) name each vertex uniquely
2) find their indegree and outdegree

CreateHashTables (First Pass)

Need to Store vertices in hash-tables based on their indegree and outdegree
CreateHashTables (Second Pass)

Need to fetch identical vertices at each index to form elementary groups
findElementaryGroups

Need to recursively attach the adjoining nodes of the vertices in a group to form larger groups
Group
How to Introduce Hierarchy into a Flat model?

⇒ Identify structurally similar components in a model.
⇒ Recursively attach the adjoining vertices of these components.

Need to:
1) name each vertex uniquely
2) find their indegree and outdegree

CreateHashTables
(First Pass)

Need to Store vertices in hash-tables
based on their indegree and outdegree

CreateHashTables
(Second Pass)

Need to fetch identical vertices at
each index to form elementary groups

findElementaryGroups

Need to recursively attach the adjoining nodes
of the vertices in a group to form larger groups

Group
How to Introduce Hierarchy into a Flat model?

⇒ Identify structurally similar components in a model.
⇒ Recursively attach the adjoining vertices of these components.
⇒ At each step compare the new components to determine larger identical components.

Need to:
1) name each vertex uniquely
2) find their indegree and outdegree

CreateHashTables (First Pass)

Need to Store vertices in hash-tables based on their indegree and outdegree
CreateHashTables (Second Pass)

Need to fetch identical vertices at each index to form elementary groups
findElementaryGroups

Need to recursively attach the adjoining nodes of the vertices in a group to form larger groups
Group
Example Net

Anshuman Mukherjee et. al., RMIT University, Melbourne

Verification of Communication Protocols in Web-Services
Example Net

**Table:** Hash-table for places created in 2\textsuperscript{nd} pass

<table>
<thead>
<tr>
<th>Index</th>
<th>H(in,out)</th>
<th>Colour</th>
<th>List of places(in:out Transition)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>H(0,1)</td>
<td>Yellow</td>
<td>P1(T1)</td>
</tr>
<tr>
<td>10</td>
<td>H(1,0)</td>
<td>Red</td>
<td>P11(T8), P12(T9), P10(T9), P3(T4)</td>
</tr>
<tr>
<td>11</td>
<td>H(1,1)</td>
<td>Blue</td>
<td>P2(T1:T3), P5(T3:T6), P9(T8:T7)</td>
</tr>
<tr>
<td>12</td>
<td>H(1,2)</td>
<td>Purple</td>
<td>P6(T5:T4,T2)</td>
</tr>
<tr>
<td>21</td>
<td>H(2,1)</td>
<td>Green</td>
<td>P7(T5,T6:T8), P8(T4,T7:T9)</td>
</tr>
<tr>
<td>22</td>
<td>H(2,2)</td>
<td>Brown</td>
<td>P4(T2,T3:T1,T5)</td>
</tr>
</tbody>
</table>
Example Net

Table: Hash-table for transitions created in 2\textsuperscript{nd} pass

<table>
<thead>
<tr>
<th>Index</th>
<th>H(in,out)</th>
<th>Colour</th>
<th>List of transitions(in:out Places)</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>H(1,1)</td>
<td>Cyan</td>
<td>T2(P5:P3), T6(P4:P6), T7(P8:P7)</td>
</tr>
<tr>
<td>15</td>
<td>H(1,2)</td>
<td>Black</td>
<td>T3(P1:P3,P4), T4(P5:P2,P7), T5(P3:P5,P6), T8(P6:P8,P10), T9(P7:P9,P11)</td>
</tr>
<tr>
<td>27</td>
<td>H(2,1)</td>
<td>Orange</td>
<td>T1(P3,P12:P1)</td>
</tr>
</tbody>
</table>
Example Net

Anshuman Mukherjee et. al., RMIT University, Melbourne

Verification of Communication Protocols in Web-Services
Example Net
Example Net
Example Net
What’s Next?

1. Introduction

2. Previous Work

3. Proposed Technique

4. Results
   - BPEL Verification
   - Memory Efficiency
   - Time Efficiency
   - Introduce Hierarchy

5. Conclusion
## Data Set Used

<table>
<thead>
<tr>
<th>M1</th>
<th>M2 &amp; M3</th>
<th>M4</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;receive&gt;</code></td>
<td><code>&lt;receive&gt;</code></td>
<td><code>&lt;receive&gt;</code></td>
</tr>
<tr>
<td><code>&lt;invoke&gt;</code></td>
<td><code>&lt;scope&gt;</code></td>
<td><code>&lt;flow&gt;</code></td>
</tr>
<tr>
<td><code>&lt;flow&gt;</code></td>
<td><code>&lt;scope&gt;</code></td>
<td><code>&lt;links&gt;</code></td>
</tr>
<tr>
<td><code>▷ &lt;invoke&gt;</code></td>
<td><code>▷ &lt;flow&gt;</code></td>
<td><code>▷ &lt;invoke&gt;</code></td>
</tr>
<tr>
<td><code>▷ &lt;invoke&gt;</code></td>
<td><code>▷ &lt;invoke&gt;</code></td>
<td><code>▷ &lt;invoke&gt;</code></td>
</tr>
<tr>
<td><code>&lt;/flow&gt;</code></td>
<td><code>▷ &lt;flow&gt;</code></td>
<td><code>&lt;switch&gt;</code></td>
</tr>
<tr>
<td><code>&lt;switch&gt;</code></td>
<td><code>▷ &lt;switch&gt;</code></td>
<td><code>&lt;/flow&gt;</code></td>
</tr>
<tr>
<td><code>◁ &lt;flow&gt;</code></td>
<td><code>&lt;/scope&gt;</code></td>
<td><code>&lt;flow&gt;</code></td>
</tr>
<tr>
<td><code>&lt;/scope&gt;</code></td>
<td><code>&lt;/scope&gt;</code></td>
<td><code>&lt;invoke&gt;</code></td>
</tr>
</tbody>
</table>
Number of activities in BPEL specification

Number of activities in BPEL specification

M_4

M_3

M_2

M_1

40 50 60 70 80
The time taken for BPEL to CPN transformation

Time (in Nanosec)

M₄
M₃
M₂
M₁

6 6.5 7 7.5 8 8.5

$\cdot 10^8$
Number of places and transition in the model rendered

Number of places and transition

- M4: Places: 40 | Transitions: 60
- M3: Places: 80 | Transitions: 140
- M2: Places: 80 | Transitions: 120
- M1: Places: 40 | Transitions: 80

Anshuman Mukherjee et al., RMIT University, Melbourne
Number of states and edges in the state space of transformed models

<table>
<thead>
<tr>
<th>Model</th>
<th>States</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>M2</td>
<td>1,000</td>
<td>2,000</td>
</tr>
<tr>
<td>M3</td>
<td>3,000</td>
<td>4,000</td>
</tr>
<tr>
<td>M4</td>
<td>5,000</td>
<td>6,000</td>
</tr>
</tbody>
</table>

Verification of Communication Protocols in Web-Services

Anshuman Mukherjee et. al., RMIT University, Melbourne
The time taken for generating first 50,000 states

- bernoulli
- binomial(n=1000)
- binomial(n=10000)
Space occupied decrease with increase in $\delta$.

The graph shows the relationship between space occupied (in bytes) and the value of $\delta$, with different lines representing different values of $n$ and $m$. The data points for $n=4, m=5$, $n=60, m=90$, $n=200, m=400$, $n=400, m=700$, $n=800, m=1000$, and $n=1500, m=2000$ are plotted, demonstrating the decrease in space occupied as $\delta$ increases.
Time for state-space analysis depends on $\delta$ as well as the size of model.
Time vs Space

With an increase in value of $\delta$, time increase while space requirement decrease.
Test Case

Anshuman Mukherjee et. al., RMIT University, Melbourne

Verification of Communication Protocols in Web-Services
Exploration time

\[ \frac{dy}{dx} = 75 \]

\[ \frac{dy}{dx} = 7.8 \]

Number of markings explored (in Thousands)

Time Taken (in Sec)

- CPN Tools
- Implemented Model-Checker

Anshuman Mukherjee et. al., RMIT University, Melbourne
Exploration time

Anshuman Mukherjee et. al., RMIT University, Melbourne
Verification of Communication Protocols in Web-Services
Exploration time

![Exploration time graph](image_url)

- **Original Net**
- **Modified Net**

**Anshuman Mukherjee et. al., RMIT University, Melbourne**

Verification of Communication Protocols in Web-Services
What’s Next?

1. Introduction
2. Previous Work
3. Proposed Technique
4. Results
5. Conclusion
Conclusion

Summary
We have identified the hierarchical relationship among BPEL activities.
Summary

- We have identified the hierarchical relationship among BPEL activities.
- Based on this relationship, we have mapped BPEL activities to Java beans.
We have identified the hierarchical relationship among BPEL activities. Based on this relationship, we have mapped BPEL activities to Java beans. We have extended Spring framework to render the bean-factory of a BPEL specification.
Summary

- We have identified the hierarchical relationship among BPEL activities.
- Based on this relationship, we have mapped BPEL activities to Java beans.
- We have extended Spring framework to render the bean-factory of a BPEL specification.
- We have proposed a formalisation technique for XML based models.
We have identified the hierarchical relationship among BPEL activities. Based on this relationship, we have mapped BPEL activities to Java beans. We have extended Spring framework to render the bean-factory of a BPEL specification. We have proposed a formalisation technique for XML based models. It is possible to use other non-XML formalisation techniques.
Conclusion

Summary

- We have identified the hierarchical relationship among BPEL activities.
- Based on this relationship, we have mapped BPEL activities to Java beans.
- We have extended Spring framework to render the bean-factory of a BPEL specification.
- We have proposed a formalisation technique for XML based models.
- It is possible to use other non-XML formalisation techniques.
- We have reduced the memory requirement for model checking by storing states in difference form.
We have identified the hierarchical relationship among BPEL activities.

Based on this relationship, we have mapped BPEL activities to Java beans.

We have extended Spring framework to render the bean-factory of a BPEL specification.

We have proposed a formalisation technique for XML based models.

It is possible to use other non-XML formalisation techniques.

We have reduced the memory requirement for model checking by storing states in difference form.

Consequently, model checking would acquire a bigger role in verification of a wide range of software.
Conclusion

Summary

- We have identified the hierarchical relationship among BPEL activities.
- Based on this relationship, we have mapped BPEL activities to Java beans.
- We have extended Spring framework to render the bean-factory of a BPEL specification.
- We have proposed a formalisation technique for XML based models.
- It is possible to use other non-XML formalisation techniques.
- We have reduced the memory requirement for model checking by storing states in difference form.
- Consequently, model checking would acquire a bigger role in verification of a wide range of software.
- Experimental results indicate that our algorithm performs remarkably better for large models.
We have identified the hierarchical relationship among BPEL activities. Based on this relationship, we have mapped BPEL activities to Java beans. We have extended Spring framework to render the bean-factory of a BPEL specification. We have proposed a formalisation technique for XML based models. It is possible to use other non-XML formalisation techniques. We have reduced the memory requirement for model checking by storing states in difference form. Consequently, model checking would acquire a bigger role in verification of a wide range of software. Experimental results indicate that our algorithm performs remarkably better for large models. The proposed algorithm is addressing a niche for contemporary systems which have high level of complexity.
We have identified the hierarchical relationship among BPEL activities. Based on this relationship, we have mapped BPEL activities to Java beans. We have extended Spring framework to render the bean-factory of a BPEL specification. We have proposed a formalisation technique for XML based models. It is possible to use other non-XML formalisation techniques. We have reduced the memory requirement for model checking by storing states in difference form. Consequently, model checking would acquire a bigger role in verification of a wide range of software. Experimental results indicate that our algorithm performs remarkably better for large models. The proposed algorithm is addressing a niche for contemporary systems which have high level of complexity. In future, we aim propose a distributed model checking algorithm by extending the sequential algorithm.
Summary

- We have identified the hierarchical relationship among BPEL activities.
- Based on this relationship, we have mapped BPEL activities to Java beans.
- We have extended Spring framework to render the bean-factory of a BPEL specification.
- We have proposed a formalisation technique for XML based models.
- It is possible to use other non-XML formalisation techniques.
- We have reduced the memory requirement for model checking by storing states in difference form.
- Consequently, model checking would acquire a bigger role in verification of a wide range of software.
- Experimental results indicate that our algorithm performs remarkably better for large models.
- The proposed algorithm is addressing a niche for contemporary systems which have high level of complexity.
- In future, we aim propose a distributed model checking algorithm by extending the sequential algorithm.
- We have reduced the time requirement for model checking by generating the state-space in parallel.
We have identified the hierarchical relationship among BPEL activities.

Based on this relationship, we have mapped BPEL activities to Java beans.

We have extended Spring framework to render the bean-factory of a BPEL specification.

We have proposed a formalisation technique for XML based models.

It is possible to use other non-XML formalisation techniques.

We have reduced the memory requirement for model checking by storing states in difference form.

Consequently, model checking would acquire a bigger role in verification of a wide range of software.

Experimental results indicate that our algorithm performs remarkably better for large models.

The proposed algorithm is addressing a niche for contemporary systems which have high level of complexity.

In future, we aim to propose a distributed model checking algorithm by extending the sequential algorithm.

We have reduced the time requirement for model checking by generating the state-space in parallel.
Conclusion

Summary

- We have identified the hierarchical relationship among BPEL activities.
- Based on this relationship, we have mapped BPEL activities to Java beans.
- We have extended Spring framework to render the bean-factory of a BPEL specification.
- We have proposed a formalisation technique for XML based models.
- It is possible to use other non-XML formalisation techniques.
- We have reduced the memory requirement for model checking by storing states in difference form.
- Consequently, model checking would acquire a bigger role in verification of a wide range of software.
- Experimental results indicate that our algorithm performs remarkably better for large models.
- The proposed algorithm is addressing a niche for contemporary systems which have high level of complexity.
- In future, we aim to propose a distributed model checking algorithm by extending the sequential algorithm.
- We have reduced the time requirement for model checking by generating the state-space in parallel.
We have identified the hierarchical relationship among BPEL activities. Based on this relationship, we have mapped BPEL activities to Java beans. We have extended Spring framework to render the bean-factory of a BPEL specification. We have proposed a formalisation technique for XML based models. It is possible to use other non-XML formalisation techniques. We have reduced the memory requirement for model checking by storing states in difference form. Consequently, model checking would acquire a bigger role in verification of a wide range of software. Experimental results indicate that our algorithm performs remarkably better for large models. The proposed algorithm is addressing a niche for contemporary systems which have high level of complexity. In future, we aim propose a distributed model checking algorithm by extending the sequential algorithm. We have reduced the time requirement for model checking by generating the state-space in parallel.
We have identified the hierarchical relationship among BPEL activities.
Based on this relationship, we have mapped BPEL activities to Java beans.
We have extended Spring framework to render the bean-factory of a BPEL specification.
We have proposed a formalisation technique for XML based models.
It is possible to use other non-XML formalisation techniques.
We have reduced the memory requirement for model checking by storing states in difference form.
Consequently, model checking would acquire a bigger role in verification of a wide range of software.
Experimental results indicate that our algorithm performs remarkably better for large models.
The proposed algorithm is addressing a niche for contemporary systems which have high level of complexity.
In future, we aim propose a distributed model checking algorithm by extending the sequential algorithm.
We have reduced the time requirement for model checking by generating the state-space in parallel.
Thank You!

Questions?

