Verification of Web Services

Jianwen Su
University of California, Santa Barbara
The Verification Problem

Given
- a web service/composition/choreography/workflow/ …
- a goal $\phi$

do all executions satisfy the goal?

Choices for $\text{and } \phi$
Outline

- Motivations
- Transitions systems
- BPEL services and compositions
- Choreographies (of BPEL services)
- Artifact-centric workflow
- Concluding remarks
Software Systems in the Real World

- Wide range of applications:
  - Web stores, e-tailors, …
  - Accounting, financial systems, …
  - Automated flight control, …
  - Patient profiles, cases, care records, …
  - Governments: local, federal, courts, prisons, …
  - …

- Challenges:
  - Interoperation & integration
  - Design and analysis
  - Improvements (evolution)
Web Services: Standardization

- The Web: Flexible human-software interaction
- Web services: Flexible software-software interaction
  - SAAS: Software As A Service
- A working definition: software services accessible via standardized protocols
- SOA: a potential basis for software system design, interoperation, integration, ...
  - Lots of interest in trade press, academic community, standards bodies, ...
  - Applications in e-commerce, telecom, science, cloud, government, education, ...
Fundamental Elements (WS Apps)

- **Process**: a collection of actions to be taken in a meaningful manner (sequential, parallel, conditional, …)

- Communication or **messages**: different software systems need to cooperate, collaborate

- **Data**: guide the actions to be taken and processes to follow

- **Actors** (human, external environment): their reasoning for making decisions may not be captured in the logic specification/running systems
Research Challenges (Biz Workflows)

- **Models:** process, data, messages, actors

- Analysis and verification

- Integration/interoperation

- Improvements
  - (biz intelligence, operation optimization, …)

- Management of workflows and executions
Goal of This Talk

- Focus on analysis & verification problem
  - Depending on models

- A sampler of verification problems, approaches and results
Outline

- Motivations
- Transitions systems
- BPEL services and compositions
- Choreographies (of BPEL services)
- Artifact-centric workflow
- Concluding remarks

? |= ¨φ
Transition Systems

- A finite transition system (Kripke structure) is a tuple $T = (S, I, R, L)$ where
  - a finite set of states $S$
  - a set of initial states $I \subseteq S$
  - a transition relation $R \subseteq S \times S$
  - a labeling function $L : S \rightarrow 2^P$

- $P$ : a set of atomic propositions
Example

- $P = \{q_1, q_2, q_3\}$

$L(s_3) = \{q_1, q_2\}$
Runs (Execution Paths)

- Given a finite transition system \( T = (S, I, R, L) \)
- A run is an infinite sequence of states
  \[ Z = s_0s_1s_2 \cdots \]
  where for each \( i \geq 0 \), \((s_i, s_{i+1}) \in R\)

\[ s_0s_1s_2s_3s_5s_1s_2 \cdots \]
Linear Temporal Logic (LTL)

- A set $P$ of atomic propositions: $q_1, q_2, q_3, \ldots$
- Logical connectives: $\land, \lor, \neg$
- Temporal operators:
  - $X \varphi$: $\varphi$ is true in the next state
  - $G \varphi$: $\varphi$ is true in every state
  - $\psi U \varphi$: $\psi$ is true in every state before the state $\varphi$ is true
  - $F \varphi$: $\varphi$ is true in some future state

$X$: next  \quad $G$: always  \quad $U$: until  \quad $F$: eventually

- Example: $G (money \rightarrow F food)$
Semantics of Temporal Operators

- Truth value of a formula is defined on runs
- Propositional connectives have the usual meaning
- Temporal operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X$</td>
<td>next</td>
</tr>
<tr>
<td>$G$</td>
<td>always</td>
</tr>
<tr>
<td>$U$</td>
<td>until</td>
</tr>
<tr>
<td>$F$</td>
<td>eventually</td>
</tr>
</tbody>
</table>

$F q_1 \equiv \text{true} \ U q_1$

$G q_1 \equiv \neg F \neg q_1$
LTL Semantics

- A state is a set of propositions
- A run $Z=s_0s_1s_2\cdots$ satisfies an LTL formula:
  - $Z \models q$ if $s_0 \models q$ or $q \in L(s_0)$
  - $Z \models \neg \varphi$ if $Z \not\models \varphi$
  - $Z \models \varphi \land \psi$ if $Z \models \varphi$ and $Z \models \psi$
  - $Z \models \varphi \lor \psi$ if $Z \models \varphi$ or $Z \models \psi$
  - $Z \models X \varphi$ if $s_1s_2\cdots \models \varphi$
  - $Z \models G \varphi$ if for each $i$, $s_is_{i+1}\cdots \models \varphi$
  - $Z \models F \varphi$ if for some $i$, $s_is_{i+1}\cdots \models \varphi$
  - $Z \models \psi U \varphi$ if for some $i$, $s_is_{i+1}\cdots \models \varphi$ and for each $j < i$, $s_js_{j+1}\cdots \models \psi$
Transition Systems and LTL

- A transition system $T$ satisfies an LTL formula $\varphi$ if every run of $T$ satisfies $\varphi$

\[
\begin{align*}
F q_3 \\
G(\neg q_3 \rightarrow X q_3)
\end{align*}
\]
Verifying LTL Properties

- Problem: given a transition system $T$, an LTL formula $\varphi$, determine if $\varphi$ is satisfied by $T$ (i.e. every run of $T$)

- A decision algorithm:
  1. Construct a Büchi automaton $B_{\neg \varphi}$ equivalent to $\neg \varphi$
  2. Explore (depth-first search) simultaneously $T$ and $B_{\neg \varphi}$,
     - if a repeat is found involving a final state of $B_{\neg \varphi}$, halt and output “no” (with the found path)
   Otherwise, output “Yes” ($T$ satisfies $\varphi$)
Büchi Automata

- $P$ is a (finite) set of propositions

- A Büchi automaton is a tuple $B = (Q, I, \delta, F)$ where
  
  - $Q$ is a finite set of states
  - $I \subseteq Q$ is a (nonempty) set of initial states
  - $F \subseteq Q$ is a set of final states
  - $\delta \subseteq Q \times 2^P \times Q$ is a transition relation

- Essentially nondeterministic finite state automata but accepting infinite words:
  
  - A word in $(2^P)^\omega$ is accepted if final states are entered infinitely often

The language of $B$, $L(B)$, is the set of words accepted
An Example
LTL to Büchi Automata

- A Büchi automaton $B$ is equivalent to an LTL formula $\varphi$: an infinite sequence $Z$ satisfies $\varphi$ iff $Z \in L(B)$

- For each LTL formula $\varphi$, one can construct a Büchi automaton $B_\varphi$ equivalent to $\varphi$
  - Number of states in $B_\varphi$ is $2^{O(|\varphi|)}$

- Emptiness of a Büchi automaton can be determined in $O(n)$ where $n$ is the number of states

[Merz MOVEP 2001]
Model Checking

$T$: a transition system, $\varphi$: an LTL formula

1. Construct a Büchi automaton $B_{\neg \varphi}$ equivalent to $\neg \varphi$
2. Explore (depth-first search) simultaneously $T$ and $B_{\neg \varphi}$,
   - if a repeat is found involving a final state of $B_{\neg \varphi}$,
     halt and output “no” (the trace is the counter example)
   Otherwise, output “Yes” ($T$ satisfies $\varphi$)

- **Complexity**: $O(2^{O(|\varphi|)|T|})$ time, PSPACE

[Merz MOVEP 2001]
Outline

■ Motivations
■ Transitions systems
■ BPEL services and compositions
■ Choreographies (of BPEL services)
■ Artifact-centric workflow
■ Concluding remarks
Business Process Execution Language

- Allow specification of compositions of Web services
  - business processes as coordinated interactions of Web services
- Allow abstract and executable processes
- Influences from
  - Traditional flow models
  - Structured programming
  - Successor of WSFL and XLANG
- Assumes WSDL ports
- OASIS standard
Illustrating a BPEL Service

\( \text{invoke} \ldots \)

\( \text{receive} \ldots \)

\( \text{assign} \ldots \)

\( \text{reply} \ldots \)

\( \text{invoke} \ldots \)
BPEL to Transition Systems

Translate each atomic activity to a transition system with single entry, single exit.

```
<receive ... operation = "approve" variable = "request" />
```

```
<invoke operation="approve", invar="request", outvar="aprvInfo">
    <catch faultname="loanfault">
        ... handler1 ...
    </catch>
</invoke>
```

Treat actions as propositions.
BPEL to Transition Systems

Control flow constructs: assemble pieces of transition systems

\[
\langle \text{sequence} \rangle \\
\langle \ldots \text{activity}_1 \ldots / \rangle \\
\langle \ldots \text{activity}_2 \ldots / \rangle \\
\langle /\text{sequence} \rangle
\]

\[
\langle \text{flow} \rangle \\
\langle \text{activity}_1 \ldots / \rangle \\
\langle \text{source} \text{\ linkname} = \text{"link1"} / \rangle \\
\langle /\text{activity}_1 \rangle \\
\langle \text{activity}_2 \ldots / \rangle \\
\langle \text{target} \text{\ linkname} = \text{"link1"} / \rangle \\
\langle /\text{activity}_2 \rangle \\
\langle /\text{flow} \rangle
\]

Disallow the orders prohibited by the link
Verifying BPEL Services

- $S$: a BPEL service, $P$: a set of propositions,
  $\varphi$: an LTL formula
- Determine if every execution of $S$ satisfies $\varphi$
- Algorithm:
  1. Construct a transition system $T_{S,P}$
  2. Determine if $T_{S,P}$ satisfies $\varphi$
- Complexity: $O(2^{O(|\varphi|)|S|})$ time

- Good news but
  - Control states (flow) only, no variables/data
  - Single service, no composition
Adding Data

- BPEL allows variables to hold XML documents

- Bad news (folklore):
  BPEL is Turing (computationally) complete

- Immediate consequence:
  It is undecidable if a given BPEL service satisfies a given LTL formula

- One possible restriction: limit variables to
  - finite domains: the number of possible values is finite
Finite Domain Variables

- Represent variable contents explicitly through states

\[ \text{Transition states increased by } n^m \text{ times:} \]

\[ n : (\text{max}) \text{ domain size, } m : \text{number of variables} \]

- Complexity of verification: \( O(2^{O(|\varphi|}|S|n^m)) \) time

\( \varphi : \text{LTL formula, } S : \text{BPEL service} \)

[Fu-Bultan-S. ISSTA ’04]
Composition of BPEL Services

- Peer to peer
  - Investor
  - Stock Broker
  - Report
  - Accept, reject, bill
  - Register, ack, cancel

- Mediated or hub-and-spoke
  - Mediator
  - Investor
  - Research Dept.
  - Stock Broker
  - Request, terminate
Synchronous Messaging Model

- Two specific actions:
  - Send a message (!)
  - Receive a message (?)

```
<invoke>: request-response

store

authorize

ok

synchronization

bank

authorize

ok

<receive>: response

<invoke>: request
```
Product with Synchronous Messaging

- Two services

Their synchronous product as a transition system:
Product with Synchronous Messaging

- In general, the composition of \( k \) BPEL services with synchronous messaging can be modeled as a transition system with \( r^k \) states where
  - \( r \) is the (max) number of states in a single service

- Complexity of verification: \( O(2^{O(|\varphi|)}(|S|n^m)^k) \) time
  - \( \varphi \) : LTL formula
  - \(|S|\) : size of a BPEL service
  - \( n \) : domain size
  - \( m \) : number of variables in a BPEL service
  - \( k \) : number of BPEL services
Asynchronous Messaging

- Two specific actions:
  - Send a message (\(!\))
  - Receive a message (\(?\))

FIFO queues are used to buffer unconsumed messages
  - One queue per service for incoming messages

[Bultan-Fu-Hull-S. WWW ’03]
Verification is Undecidable

- Finite state automata with FIFO queues are Turing complete [Brand-Zafiroupolo JACM’83]

- Immediate consequence:
  Verification problem is undecidable

- One possible restriction: bound queue size
Observation: a bounded length queue has a finite number of states

Asynchronous + bounded queue can be simulated

Note: Only focus on message types not content
BPEL with Asynchronous Messaging

- Number of states for queues: $e^l$, where $e$ : number of message types, $l$ : queue length bound
- With message contents: $e^l n^l$, where $n$ is domain size

- Complexity of verification: $O(2^{O(|\varphi|)}(|S|^m e^l n^l)^k)$ time
  - $\varphi$ : LTL formula
  - $|S|$ : size of a BPEL service
  - $n$ : domain size
  - $m$ : number of variables in a BPEL service
  - $k$ : number of BPEL services
Summary of Verifying BPEL Services

- Focus on decidability boundary of LTL properties of BPEL + compositions (synchronous, bounded queue asynchronous messaging)

- Verification algorithms: map to exiting verifiers
  - ASM: [Farahbod-Classser-Vajihollahj 2004][Deutsch-Sui-Vianu 2004] [Fahland-Reisiq 2005]
Outline

- Motivations
- Transitions systems
- BPEL services and compositions
- Choreographies (of BPEL services)
- Artifact-centric workflow
- Concluding remarks
Composition: Common Topologies

- **Peer-to-peer**
  - Investor
  - Stock Broker
  - Research Dept.
  - Mediator
  - Investor
  - Research Dept.
  - Stock Broker

- **Mediated, or “hub and spoke”**
  - Investor
  - Stock Broker
  - Research Dept.
  - Mediator
  - Investor
  - Research Dept.
  - Stock Broker

Actions:
- Register, Ack, Cancel
- Accept, Reject, Bill
- Report
- Request, Terminate
Orchestration vs Choreography

- **Choreography**
  - Composition
    - WS-CDL
  - (Individual) Service Description
    - WSCL
  - XML Messaging
    - WSDL
  - OWL-S ServiceProfile

- **Composition**
  - BPEL
  - OWL-S ServiceModel

- **Messaging**
  - SOAP
WS Choreography Definition Language

- Specification of choreography
- Model complex business protocol (e.g. order management) to enable interoperability
- Generate computational logic of individual collaborating participants

Key concepts
- Collaborating participants: role, relationship, participants
- Information driven collaboration: channel, activities, workunit, choreography

Standardization through W3C (Version 1.0: December 2004)
Composition: BPEL and WS-CDL

- **Investor**
  - Focus on local actions
  - register, ack, cancel

- **Stock Broker**
  - Focus on global behaviors
  - accept, reject, bill
  - request, terminate

- **Research Dept.**
  - report

- **Mediator**

**BPEL**

**WS-CDL**
For “hub and spoke”, the difference is small
For “peer-to-peer”, the concept of choreography is interesting and not well understood
Automated Design: **Top-down vs Bottom-up**

**Top-down**
- Specification of individual services
- Orchestration
- E.g., BPEL
- Verification and analysis of choreography
- Focus on the conversation model

**Bottom-up**
- Specification of global behaviors
- Choreography
- E.g., WS-CDL

Investor → Register, Ack, Cancel → Stock Broker
Investor → Accept, Reject, Bill → Stock Broker
Investor → Report → Research Dept.
Stock Broker → Request, Terminate → Research Dept.
Research Dept. → Report

Verification of WS Choreography

- Verification of choreography of a WS (BPEL) composition

![Diagram of interactions between Investor, Stock Broker, and Research Dept.]

- Services: finite transition systems on messaging actions
- Unbounded FIFO queues for messages
- Choreography: message sequences (send only)
  - How to model?
- LTL on choreography

[ Fu-Bultan-S. WWW’04, ISSTA’04 ]
An Example: **Stock Analysis Service (SAS)**

Three peers: Investor, Stock Broker, and Research Dept

- **Inv** initiates the stock analysis service by sending a *register* message to **SB**
- **SB** may *accept* or *reject* the registration
- If the registration is accepted, **SB** sends an analysis *request* to the **RD**
- **RD** sends the results of the analysis directly to the **Inv** as a *report*
- After receiving a *report* **Inv** can either send an *ack* to **SB** or *cancel* the service
- Then, **SB** either sends the *bill* for the services to **Inv**, or continues the service with another analysis *request*
**SAS Composition**

- SAS is a web service composition
  - a finite set of peers: Inv, SB, RD, and
  - a finite set of message classes: register, ack, cancel, accept, ...

![Diagram of SAS Composition]

- Investor (Inv)
- Stock Broker (SB)
- Research Dept. (RD)
Asynchronous Messaging

- We assume that the messages among the peers are exchanged through reliable and asynchronous messaging
  - FIFO and unbounded message queues

- This model is similar to industry efforts such as
  - JMS (Java Message Service)
  - MSMQ (Microsoft Message Queuing Service)
Mealy Service Model

- Finite state control
- Acts on a finite set of message classes
- Transitions are based on receiving a message \(?m\) or sending a message \(!m\)

[Bultan-Fu-Hull-S. WWW’03]
Composite Mealy Service Execution

**Investor**

**Stock Broker Firm**
- !request → !reject → ?accept → !ack → !bill → !cancel → !bill → !terminate

**Research Dept.**
- ?request → !report → ?terminate
- !request → !report → ?terminate

- **Execution halts if**
  - All Mealy services are in final states, and
  - All queues are empty
Conversations and Conversation Protocols

- **Conversation**: a message sequence
- A conversation protocol specifies the set of desired conversations
Conversations of Composite Services

- A virtual watcher records the messages as they are sent.

- A conversation is a sequence of messages the watcher sees in a successful run (or enactment).

- Conversation language: the set of all possible conversations.

- What properties do conversation languages have?

---

**Diagram:**

- Investor (Inv)
- Stock Broker (SB)
- Research Dept. (RD)
- Watcher

Messages:
- register
- accept
- ack
- bill
- request
- terminate
- report

Roles:
- Investor
- Stock Broker
- Research Dept.
- Watcher
Conversation Languages Are Not Regular

-The set of conversations $\mathcal{CL} \cap a^*b^* = a^n b^n$

-Conversation languages are not always regular
  -Some may not even be context free

-Causes: asynchronous communication & unbounded queue

-Bounded queues or synchronous: CL always regular

-CLs are always context sensitive
Remarks

- Communicating finite state machines with queues are computationally Turing complete
  - Conversation languages ≠ tracing execution states

- Why regular languages?
  - They would allow static analysis, e.g. model checking
    - Testing and debugging in SOA are harder

- Queue v.s. no queue: design time decision!
Two Key Questions

- Is the composition of (BPEL) services “correct”?
  - Verify conversations
- Automated design of services from the desired conversation protocol?
Temporal Properties of Conversations

- The notion of conversation enables reasoning about temporal properties of the composite web services
- Extend LTL extends naturally to conversations
  - LTL temporal operators
    - $X$ (next), $U$ (Until), $G$ (Globally), $F$ (Future)
  - Atomic properties
    - Predicates on message classes (or contents)
  - Example: $G(accept \rightarrow F bill)$
- Verification problem: Given an LTL property, does the conversation language (i.e. every conversation) satisfy the property?
Given a composition of services, does its CL satisfy the LTL properties?

Problem: the general case is undecidable

[Brand-Zafiropulo J ACM’83]
Design Scenario 2: Top Down

- Specify the global messaging behavior explicitly as a conversation protocol
- Determine if the conversations allowed by the protocol satisfy LTL properties

**Conversation Protocol**

\[
\begin{align*}
A \rightarrow B: & \text{ msg1} \\
B \rightarrow A: & \text{ msg2} \\
B \rightarrow C: & \text{ msg3} \\
C \rightarrow B: & \text{ msg4} \\
B \rightarrow A: & \text{ msg5} \\
B \rightarrow C: & \text{ msg6}
\end{align*}
\]

\[\models G(\text{msg1} \rightarrow F(\text{msg3} \lor \text{msg5}))\]

**Problem**: the conversation protocol may not be realizable
Approaches

- *(Bottom up)* verification is undecidable
  - Approach 1: check if the conversations using *bounded* queue satisfy LTL property
    — partial verification
  - Approach 2: sufficient condition for bounded queue
    \( CL = \) unbounded queue \( CL \)
    — synchronizability

- *(Top down)* protocol may be unrealizable
  - Approach 3: sufficient condition for realizability
Realizability Problem

- Not all conversation protocols are realizable!

Conversation protocol
A → B: m1
C → D: m2

Projection of the conversation protocol to the peers
Peer A
Peer B
Peer C
Peer D

Conversation “m2 m1” will be generated by any legal peer implementation which follows the protocol.
Another Non-Realizable Protocol

Generated conversation: m2 m1 m3
A Sufficient Condition for Realizability

Three parts for realizability (contentless messages)

- **Lossless join**
  Conversation protocol should be equal to the “join” of its projections to each peer

- **Synchronous compatible**
  When the projections are composed synchronously, there should not be a state where a peer is ready to send a message while the corresponding receiver is not ready to receive

- **Autonomous**
  Each peer should be able to make a deterministic decision on whether to send or to receive or to terminate

[Fu-Bultan-S. CIAA ’03]
Bottom-Up Approach

- Given a composition of web services, check if its conversations satisfy some LTL properties.

- General problem is undecidable due to asynchronous communication (with unbounded queues).

- Naïve idea: limit the queue length
  - Problem 1: only partial verification, unless we are lucky
  - Problem 2: state size explosion
Example 1: Regular CL, Bounded Queues

- Conversation language is regular: \((r_1a_1 \mid r_2a_2)^* e\)
- During every halting run two queues are bounded
Example 2: Not Regular, Unbounded

- Conversation language is not regular
- Queues are not bounded
Example 3: Regular, Unbounded

- Conversation language is regular: \((r_1 \mid r_2 \mid ra)^* e\)
- Queues are not bounded
Verification of Examples 2 and 3 are difficult even if we bound the queue length.

How can we distinguish Examples 1 and 3 (with regular conversation languages) from 2?

→ Synchronizability Analysis
Synchronizability Analysis

- A composite web service is synchronizable, if its conversation language does not change
  - when asynchronous communication with unbounded queues is replaced with synchronous communication or bounded queues

- A composite web service is synchronizable, if it satisfies the synchronous compatible and autonomous conditions

[Fu-Bultan-S. WWW’04]
## Are These Conditions Too Restrictive?

<table>
<thead>
<tr>
<th>Problem Set</th>
<th>Name</th>
<th>#msg</th>
<th>#states</th>
<th>#trans.</th>
<th>Synchronizable?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Source</strong></td>
<td><strong>Name</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISSTA’04</td>
<td>SAS</td>
<td>9</td>
<td>12</td>
<td>15</td>
<td>yes</td>
</tr>
<tr>
<td>IBM Conv.</td>
<td>CvSetup</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>yes</td>
</tr>
<tr>
<td>IBM Conv.</td>
<td>MetaConv</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>no</td>
</tr>
<tr>
<td>IBM Support</td>
<td>Chat</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>yes</td>
</tr>
<tr>
<td>IBM Support</td>
<td>Buy</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>yes</td>
</tr>
<tr>
<td>IBM Support</td>
<td>Haggle</td>
<td>8</td>
<td>5</td>
<td>8</td>
<td>no</td>
</tr>
<tr>
<td>IBM Support</td>
<td>AMAB</td>
<td>8</td>
<td>10</td>
<td>15</td>
<td>yes</td>
</tr>
<tr>
<td><strong>BPEL</strong></td>
<td>shipping</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>yes</td>
</tr>
<tr>
<td><strong>BPEL</strong></td>
<td>Loan</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>yes</td>
</tr>
<tr>
<td><strong>BPEL</strong></td>
<td>Auction</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>yes</td>
</tr>
<tr>
<td>collaxa.com (Oracle)</td>
<td>StarLoan</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>yes</td>
</tr>
<tr>
<td>collaxa.com (Oracle)</td>
<td>Caution</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>yes</td>
</tr>
</tbody>
</table>
Summary

- Verification of choreography is intricate
  - Choreography of composition may not be regular and does not fall into natural formal language classes
  - Must be concerned with the realizability problem

- Realizability and verification on conversations with Mealy machines [Fu-Bultan-S. 2003-6]

- Realizability on process algebras, choreography languages [many, 2005-]
Outline

- Motivations
- Transitions systems
- BPEL services and compositions
- Choreographies (of BPEL services)
- Artifact-centric workflow
- Concluding remarks

? \models \varphi
Workflow (Business Process)

- A bookseller example: Traditional control-centric model
Workflow (Business Process)

- A bookseller example: Traditional control oriented model
- Multiple steps needed for each activity

In practice, 100s to 1000s of nodes

Hard to reason, find useful views: missing data
Business Intelligence: Data View

- Extract-Transform-Load

Transactions → catalog → Data Warehouse → Analysis

Data Warehouse

transactions

workflow activities

workflow is missing!

cust_db

inventory

transactions
Business Artifacts!

- A **business artifact** is a key conceptual business entity that is used in guiding the operation of the business
  - *fedex package delivery, patient visit, application form, insurance claim, order, financial deal, registration, …*
  - both “information carrier” and “road-maps”

- Very natural to business managers and BP modelers
- Includes two parts:
  - **Information model:**
    - data needed to move through workflow
  - **Lifecycle:**
    - possible ways to evolve
Example: Restaurant

Artifacts

- Guest Check
  - Create Guest Check
  - Open GCs
  - Add Item

- Kitchen Order
  - Pending KOs
  - Prepare & Test Quality

- Receipt
  - Pending Receipts
  - Closed GCs
  - Prepare Receipt

- Cash Balance
  - Pending Receipts
  - Closed GCs
  - Paid Receipts
  - Update Cash Balance
  - Recalculate Receipt
  - Delivered KOs
  - Ready KOs
  - Archived Receipts
  - Archived GCs
  - Archived KOs
  - Cash Balance

Add Item

Remove Item

Create Guest Check

Open GCs

Close GCs

Add Item

Pending KOs

Prepare & Test Quality

Pending Receipts

Closed GCs

Paid Receipts

Update Cash Balance

Recalculate Receipt

Delivered KOs

Ready KOs

Archived Receipts

Archived GCs

Archived KOs

Cash Balance
Example: Restaurant

Artifacts

<table>
<thead>
<tr>
<th>Guest Check</th>
<th>Kitchen Order</th>
<th>Receipt</th>
<th>Cash Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create GC</td>
<td>Open GCs</td>
<td>Add KO</td>
<td>Archived GCs</td>
</tr>
<tr>
<td>Pending RCs</td>
<td>Closed GCs</td>
<td>Pending KOs</td>
<td>Prepare &amp; Test Quality</td>
</tr>
<tr>
<td>Pending Receipts</td>
<td>Closed GCs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paid Receipts</td>
<td></td>
<td>Ready KOs</td>
<td></td>
</tr>
<tr>
<td>Recalculate Receipt</td>
<td></td>
<td></td>
<td>Delivered KOs</td>
</tr>
<tr>
<td>Disagreed Receipts</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Emerging Artifact-Centric BPs

Artifacts (Info models)

- Informal model [Nigam-Caswell IBM Sys J 03]
- Systems: BELA (IBM 2005), Siena (IBM 2007)
- Formal models
  - State machines [Bhattacharya-Gerede-S. SOCA 07] [Gerede-S. ICSOC 07]

Specification of artifact lifecycles
A Logical Artifact Model for BPs

- A variation of [Bhattacharya-Gerede-Hull-Liu-S. BPM 07]
- [Hull-S. 09] (in preparation)

Artifacts (info models) + Semantic services (IOPEs) + Condition-action

if C enable ...
Given a workflow and a goal, do all executions of the workflow satisfy the goal?

Verification Problem

Artifacts (Info models) + Semantic services (IOPEs) + Condition-action

\[ ? \models \phi \]

[Bhattacharya-Gerede-S. SOCA 07] [Gerede-S. ICSOC 07]
[Bhattacharya-Gerede-Hull-Liu-S. BPM 07]
[Deutsch-Hull-Patrizi-Vianu ICDT 09]
[Vianu ICDT 09]
Synthesis Problem

- Given a goal and a set of services, construct a set of rules so that every execution satisfies the goal

\[ \text{Artifact (Info model)} + \text{Semantic services (IOPEs)} + \varphi \rightarrow \text{if } C \text{ enable...} \]

[\text{Fritz-Hull-S. ICDT 09}]

(restricted to single artifact, first-order goals)
A workflow schema is a triple

\[ W = ( \Gamma, S, R ) \]

- \( \Gamma \) : a set of artifacts classes (artifact schema)
- \( S \) : a set of (semantic) services
- \( R \) : a set of condition-action rules
A First-Order Logic + Structure

- Assuming some first order logic $L$ with a fixed structure
  - $U$ is the universe

- Existence of an infinite set of artifact IDs

- Existence of an infinite set of attributes
Artifact Classes

- An artifact class consists of
  - a finite set of attributes, of type $U$ or artifacts IDs
  - a finite set of states, initial and final states (transitions not defined)

- An artifact is a pair:
  - a mapping from attributes to $U \cup \text{IDs} \cup \{\bot\}$
  - a state

GuestCheck Artifact

<table>
<thead>
<tr>
<th>GCID</th>
<th>date</th>
<th>time</th>
<th>Name</th>
<th>KOID</th>
<th>table#</th>
<th>TOTAL</th>
<th>Payment</th>
<th>ptime</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Waiting for table → Seated → Ordered → Delivered → Completed
Artifacts in a Workflow

- During runtime, each artifact class in $\Gamma$ may have a finite set of artifacts

- The union $\mathcal{I}$ of sets of artifacts must be closed under “cross-referencing”
(Semantic) Services

- A service has a precondition and effects, conditions on
  - Attribute values
  - Defined-ness of attribute values
  - Equality of artifact IDs
  - An attribute holds the ID of a newly created artifact

SERVICE SeatingGuests

WRITE: \( \{ x: \text{GuestCheck} \} \)

READ: \( \{ x: \text{GuestCheck}, y: \text{Table} \} \)

PRE-CONDITION: \( \neg \text{Defined}(x.\text{table#}) \land \neg \text{Defined}(y.\text{GCID}) \)

EFFECTS:
- \( \text{Defined}(x.\text{table#}) \land \text{Seated}(x) \)
- \( \neg \text{Defined}(x.\text{table#}) \land \text{Waiting4table}(x) \)
Another Example

\[ 0 \leq A \leq 2 \]

\[ 0 \leq A < 1 \land 0 \leq B \land 1 \leq A \leq 2 \land 1 \leq B \]
A (semantic) service is a tuple \((\sigma, R, W, \pi, \rho)\), where

- \(\sigma\) is a task name
- \(R, W\) are finite sets of (resp., read, write) artifacts
- \(\pi, \rho\) are quantifier-free formulas (pre- and post-condition, resp.) over attributes of artifacts in \(R, R \cup W\), resp.

allow Defined\((A)\) for an attribute \(A\)

\(I'\) is the result of executing \(\sigma\) on \(I\), \(I \xrightarrow{\sigma} I'\), if

- \((I, I') \models \pi \land \rho\), and
- frame conditions are satisfied
Condition-Action Rules

- Rules that define business logic
  - Invoke a service
  - Change artifact states
    states are used to organize the processing

\[
\text{if } \text{Waiting4Table}(x) \text{ enable } \text{SeaingGuest}(x) \\
\text{if } \text{Defined}(x.\text{GCID}) \land \text{Defined}(x.\text{GCID}.\text{table#}) \\
\quad \text{change state to } \text{Taken}(x) \land \text{Seated}(x.\text{GCID})
\]
A condition-action rule is an expression of form “if $\varphi$ enable $\sigma$” or “if $\varphi$ change state to $\phi$” or where

- $\varphi$ is a (quantifier-free) formula
- $\sigma$ is a semantic service
- $\phi$ is a state changing formula

$I'$ is the result of executing a rule $r : \text{if } \varphi \ldots \text{ on } I, I \overset{r}{\rightarrow} I'$, if

- $I \vDash \varphi$, and
- $I \overset{\sigma}{\rightarrow} I'$ or $I, I'$ only differ on states as specified
Workflow Schema

- A workflow schema is a triple $W = (\Gamma, S, R)$
  - $\Gamma$: artifact schema
  - $S$: a finite set of semantic services
  - $R$: a finite set of condition-action rules

- Denote $\rightarrow^*$ the closure of $\bigcup_{r \in R} r$
Verification Problem

- Given a workflow and a goal, do all executions of the workflow satisfy the goal?

Artifacts
(Info models) + Semantic services
(IOPEs) + Condition-action

if C enable ...

\[ ? \models \varphi \]

[Bhattacharya-Gerede-Hull-Liu-S. BPM 07]
[Deutsch-Hull-Patrizi-Vianu ICDT 09]
Analysis Problems

- An artifact system $W = (\Gamma, S, R)$
  artifacts, services, rules

- Completion:
  Does $W$ allow a complete run of some artifact?

- Dead-end:
  Does $W$ have a dead-end path?

- Attribute redundancy:
  Does $W$ have a redundant attribute?

No attribute value comparisons

[Bhattacharya-Gerede-Hull-Liu-S. BPM 07]
## Results

- The problems are undecidable
  
  Primary reason: workflow language is Turing complete

- If we disallow creation of new artifacts
  
  - Initial: if each artifact has only initial attributes defined
  
  The analysis problems are PSPACE-complete

  - even for a single artifact

  [Bhattacharya-Gerede-Hull-Liu-S. BPM 07]

- Consider only a single artifact
Monotonic Workflow

- Once an attribute is assigned a value, it cannot be changed.

- For monotonic services:
  Complexity ranging from linear to intractable under various conditions.

[Bhattacharya-Gerede-Hull-Liu-S. BPM 07]
Completion (Monotonic Workflow)

- Linear time if
  - Services are deterministic (single effect)
  - Preconditions has no negation
  - Rule conditions are positive and does not check state information
- NP-complete if the above conditions are slightly relaxed

(single artifact)

[Bhattacharya-Gerede-Hull-Liu-S. BPM 07]
Dead-End & Redundancy (Monotonic Workflow)

- Checking if there is a dead end path is $\Sigma^p_2$-complete, even with various restrictions.

- Checking redundant attributes is co-NP-complete, even with various restrictions.

(single artifact)

[Bhattacharya-Gerede-Hull-Liu-S. BPM 07]
Three Analysis Problems: Review

- An artifact system $W = (\Gamma, S, R)$
  - artifacts, services, rules

- Completion: Does $W$ allow a complete run of an artifact?
- Dead-end: Does $W$ have a dead-end path?
- Attribute redundancy: Does $W$ have a redundant attribute?

- Undecidable in general, PSPACE if no artifact creation, intractable for monotonic workflows
  [Bhattacharya-Gerede-Hull-Liu-S. BPM 07]

- Ad hoc properties, restricted to defined-ness

- How to verify LTL properties?
  [Deutsch-Hull-Patrizi-Vianu ICDT 09]
Adding Infinite States to Artifacts

- An artifact is a pair:
  - a mapping from attributes to $U \cup IDs \cup \{\bot\}$
  - a state relation

GuestCheck Artifact

<table>
<thead>
<tr>
<th>GCID</th>
<th>date</th>
<th>time</th>
<th>Name</th>
<th>KOID</th>
<th>table#</th>
<th>TOTAL</th>
<th>Payment</th>
<th>ptime</th>
</tr>
</thead>
</table>

- Waiting for table
- Seated
- Ordered
- Delivered
- Completed

Items

<table>
<thead>
<tr>
<th>ItemNo</th>
<th>Qty</th>
<th>cookingReq</th>
<th>Table#</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Services Can Update State Relations

- Model operations on artifacts
  - updates of the artifact attributes
  - insertions/deletions in artifact states

- Insertions & updates can draw values from …
  - current artifacts, state relations
  - external inputs (by programs or humans), computation that returns new values
Service Specification

Consists of

- **pre-condition**: a Boolean query on current snapshot of artifact system
- **post-condition**: constraints on the updated artifacts
- for each state relation, **state insertion/deletion rules**
  - specify tuples to add to (remove from) state relations
  - Defined as queries (over current snapshot)

queries, constraints: FO logic formulas
LTL(FO) to Express Properties

- LTL with propositions replaced by FO formulas (statements on individual snapshots)

- Classic LTL temporal operators
  - $Xp$ $p$ holds in next snapshot
  - $p U q$ $p$ is true in every snapshot until $q$ is
  - $Fp$ $p$ is eventually true
  - $Gp$ $p$ is always true

- Example (with slight abuse of notation) :
  $$G \neg (\neg \text{Defined}(table\#) \land \exists z \text{Items}(z))$$

- The domain is dense order without endpoints
In general, it is undecidable [Deutsch-Hull-Patrizi-Vianu ICDT 09]

Need restrictions to turn it into decidable
Guarded FO

Guarded FO formulas restrict quantifications:

\[ \exists x \varphi(x) \Rightarrow \exists x (A(...,x,...) \land \varphi(x)) \]
\[ \forall x \varphi(x) \Rightarrow \forall x (A(...,x,...) \rightarrow \varphi(x)) \]

\( A(...,x,...) \): \( x \) is an attribute value and \( x \) cannot appear in any state atoms in \( \varphi \)

- All formulas used to update states are guarded FO
- Guarded LTL(FO): only allow guarded FO formulas
- Originated from input boundedness of [Spielmann 2003]
Guardedness is a Serious Limitation

- Not guarded:
  \[ \mathbf{G} \rightarrow (\neg \text{Defined}(\text{table}\#) \land \exists z \text{ Items}(z)) \]

- Guarded:
  \[ \mathbf{G} \rightarrow (\neg \text{Defined}(\text{table}\#) \land \text{Items}(\text{fish}, 1, x, 12)) \]
Decidability Result

- It can be decided in PSPACE if a guarded artifact schema satisfies a (guarded) LTL(FO)

- Actually complete in PSPACE

[Deutsch-Hull-Patrizi-Vianu ICDT 09]
Summary

- Biz workflow a very promising application area for WS—tremendous impact (potentially)
- Analysis is hard but could be helped with modeling choices
- Artifact-centric workflow models: right intuition and positive experiences in practice (IBM)
  - More than 20 contributors, experts from CS, MIS, digital government, healthcare, scientific workflow
Concluding Remarks

- WS analysis and verification is important & interesting
  - Modeling
  - Design
- Current results: a good starting point
- SOA themes are yet to emerge, many open issues related to analysis
- Dynamic analysis
**Acknowledgements**

- **Collaborators:**
  - Tevfik Bultan (U C Santa Barbara)
  - Xiang Fu (Hofstra University)
  - Richard Hull, Kamal Bhattacharya, Rong Liu (IBM TJ Watson)

- **Others:**
  - Victor Vianu, Alin Deutsch (UCSD)
  - Fabio Patrizi (U of Rome)
References


X. Fu, T. Bultan, and J. Su: Analysis of interacting BPEL web services. WWW 2004: 621-630

X. Fu, T. Bultan, and J. Su: Model checking XML manipulating software. ISSTA 2004: 252-262


S. Merz: Model Checking: A Tutorial Overview. MOVEP 2000: 3-38


