Web Services Composition:
A Story of Models, Automata and Logics
<< Version of Sept 10 >>

Richard Hull
Bell Labs, Lucent Technologies

September 9-10, 2004
Presented at the EDBT 2004 Summer School
Sardinia, Italy

Based on the Hull-Su SIGMOD 2004 Tutorial, and some other things

Based in part on discussions and collaborations with several people, including Michael Benedikt, Daniela Berardi, Tefvik Bultan, Diego Calvanese, Vassilis Christophides, Giuseppe De Giacomo, Michael Gruninger, Grigoris Karvounarakis, Maurizio Lenzerini, Massimo Marcella, Sheila McIlraith, and Jianwen Su
Web Services (a.k.a. E-services)

- The Web: Flexible human-machine interaction
- Web Services: Flexible machine-machine interaction
- Working Definition: Network-resident software services accessible via standardized protocols
  - Simple Object Access Protocol (SOAP): very flexible remote procedure call

- Lots of interest in trade press, academic community, standards bodies, . . .

- Applications in e-commerce, telecom, science, GRID, government, education, . . .
E-commerce: Front-end

- Example of customer buying CDs

Abstract behavior of the Service:
Do until Client selects “End”

1. Give Client a choice of actions to be performed
2. Wait for Client choice
3. Perform action chosen by Client
E-Commerce: Back-end

- E.g., for inventory management
- Web services paradigm driven by e-commerce
- Add new services dynamically
• E.g., find best location for waste treatment plant
• Possibly 100s of nodes, and running for weeks
• Data size difference: calibrations (small) and experimental data (large)
• Provenance: need to access derivation history
Telecom Services

- Future: flexible, dynamic incorporation of features
- Emerging standards enable de-coupling (SIP, 3GPP IMS)
- Bearer traffic vs. signaling/control traffic
- Emphasis on asynchronous events “at any time”
- Convergence of telecom and web services
Web Services: The Big Questions

Simplify and/or automate web service

- Discovery
  - What properties should be described?
  - How to efficiently query against them?

- Composition
  - Specifying goals of a composition
  - Specifying constraints on a composition
  - Building a composition
  - Analysis of compositions

- Invocation
  - Keeping enactments separated
  - Providing transactional guarantees

- Monitoring
  - How to track enactments
  - Recovering from failed enactments

Primary focus of this talk
Why automated composition

• Composition “on-the-fly” will enable flexible use of vast numbers of services
  - E.g., pick the best services for your immediate need
  - E.g., get the job done even if your favorite component service is unavailable

• “Simple” forms of composition
  - Re-use of previous compositions
  - Customization of a generic composition

• Study of composition will give insight and aids for manual composition, e.g.,
  - Richer ways to describe, discover web services
  - Tools to better debug BPEL specs
Anatomy of Web Services Composition

- The Promise

**Goal(s)**

- Activities
  - Discovery/Self-description
  - Orchestration, Monitoring
  - Info sharing (Messaging)

- The reality: No unifying approach has emerged yet
  - BPEL, Mealy: Strong on orchestration, info sharing
  - OWL-S: Strong on goals, activities, discovery
  - “Roman” model: Strong on activities, orchestration
Key dimensions in web service composition

- OWL-S
- Commitment Protocols
- CTR-S
- SIP
- WSDL
- WSCL
- Roman
- CSP
- Mealy
- BPEL
- $\pi$-Calc
- BPML

Dimensions:
- Complexity of glue language
- Complexity of (interface to) component services
Key Disciplines for Web Services Composition

- **Science of Design, e.g.**,  
  - Workflow, Service-oriented computing, Reactive systems, Process algebras, Automata, Verification

- **AI, e.g.**,  
  - Knowledge Representation, Reasoning about actions, Agents, Planning

- **Logics and Finite Model Theory, e.g.**,  
  - Temporal Logics, Propositional Dynamic Logic, Description Logics, Situation Calculii, Verification

- **Databases, e.g.**,  
  - XML, Domain-specific query languages, Indexing, Transaction Management
Goals of this Talk:  
Tools for Composition

• Describe the playing field
  – Key standards that we’ll all build upon
  – Promising models and formalisms

• (Selected) Perspectives and Results on Composition
  – Automata-theoretic approaches
    • Digression into targeted logics
  – Incorporating “semantics”: The OWL-S community, PSL
  – Representative results from verification (if there’s time)
  – Towards a model for coordinating telecom services

• The key challenge going forward
  – Some foundations are laid, but how to unify them
Outline

- Introduction

- Standards

- Perspectives on Composition

- Analysis and Verification
  - A coordination model for telecom

- A Glimpse of the Future
Web Service Definition Language (WSDL)

- WSDL provides a framework for defining
  - Interface: operations and input/output
  - Access specification: SOAP bindings (e.g., RPC)
  - Endpoint: the location of service

[from Leymann BTW 2003 talk]
WSDL 1.0 Operations

- Traditional I/O signatures (using XML Schema)
- Four operation types
  - Proactive: send request
    - send request, block till response
  - Reactive: receive request
    - receive request, send response

- Port: mechanism to cluster operations
  - Port as unit of interoperation between services

- WSDL 2.0: several more operation types
Business Process Execution Language (BPEL)

• 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

• Standardization through OASIS
BPEL in Action

Purchase Order service coordinates other services using ports in WSDL

[diagram showing processes such as Receive Purchase Order, Initiate Production Scheduling, Complete Production Scheduling, and others, along with operations like Price Calculation and Invoice Processing]

[from BPEL 1.1 standard]
BPEL Executable: Procedural Constructs

- Flowcharts “with parallelism”
- “Pick” construct to enable waiting for input (or time out)
- Synchronization within parallel threads
- Comparison of supported constructs: see [van der Aalst ’03]

```
Initialize

do until flag

pick

flag := true

end_case

end_date reached

receive order

begin parallel

send Order

Receive Bill1

Send Bill

receive Receipt1

Send Payment

Receive Payment

send Receipt

Receive Receipt1

end parallel
```

```
begin

parallel

send Order

Receive Bill1

Send Bill

receive Receipt1

Send Payment

send Receipt

Receive Receipt1

end parallel
```
BPEL Activities

- **Invokes** an operation on a partner service
  - Send to WSDL port, wait for a response
- **Receives** invocation from a partner
  - Wait for a message
- **Sends a reply** message in partner invocation
  - Send a message (corresponding to some earlier message)
- **Data assignment** between containers
  - Copy local data
- **Control structures**: sequence, flow (possibly with links), pick, loops, etc.
- **Scoping, exceptions, compensation**
Structure of a BPEL Process

```xml
<process ...
  <partners> ...
  </partners>
  <containers> ...
  </containers>
  <correlationSets> ...
  </correlationSets>
  <faultHandlers> ...
  </faultHandlers>
  <compensationHandlers> ...
  </compensationHandlers>
  (activities)*
</process>
```

- Web services the process interacts with
- Data used by the process
- Used to support asynchronous interactions
- Alternate execution path to deal with faulty conditions
- Code to execute when “undoing” an action
- What the process actually does
BPEL Structured Activities

<sequence>
execute activities sequentially

<flow>
execute activities in parallel

<while>
iterate execution of activities until condition is violated

<pick>
several event activities (receive message, timer event) scheduled for execution in parallel; first one is selected and corresponding code executed

<link ...>
defines a control dependency between a source activity and a target
Web Service Conversation Language (WSCL)

- A key to web service composition:
  - Interactions between services
- WSCL specifies a conversation (behavior signature) as a labeled graph:
  - Nodes: interactions, individual units of responses
  - Edges: transitions, sequencing of interaction
  - Edge labels: conditions on transitions

![Diagram of Web Service Conversation Language (WSCL)]
Conversation Specification

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Conversation name="simpleConversation"
    version="1.01"
    xmlns="http://www.w3.org/2002/02/wscl10"
    initialInteraction="Start"
    finalInteraction="End"
    description="Conversation for a Store Front Service">

<ConversationInteractions>
    ...list of all interactions
</ConversationInteractions>

<ConversationTransitions>
    ...list of all transitions
</ConversationTransitions>

</Conversation>
```
WS Choreography

• An emerging standard from W3C
  – Drawing inspiration from the $\pi$-calculus
• Global view of composite service interactions
  – Global model: interactions and choreography
  – Choreography Definition Language (WS-CDL)
• Key technical elements
  – Participants and roles: what services are involved
  – Channels: where and how the messages are sent
  – Interactions: message exchange patterns
  – Activities and control structures: e.g., sequencing
  – Choreography: a global description of a composition
    • Interactions, exceptions, finalizer
    • Composable
<role name="Consumer">
    <behavior name="consumerForRetailer" interface="cns:ConsumerRetailerPT"/>
    <behavior name="consumerForWarehouse" interface="cns:ConsumerWarehousePT"/>
</role> …

<relationship name="ConsumerRetailerRelationship">
    <role type="tns:Consumer" behavior="consumerForRetailer"/>
    <role type="tns:Retailer" behavior="retailerForConsumer"/>
</relationship> …

<choreography name="ConsumerRetailerChoreo" root="true">
    …
    <interaction channelVariable="tns:retailer-channel" operation="handlePurchaseOrder" align="true" initiateChoreography="true">
        <participate relationship="tns:ConsumerRetailerRelationship" fromRole="tns:Consumer" toRole="tns:Retailer"/>
        <exchange messageContentType="tns:purchaseOrderType" action="request">
            <use variable="cdl:getVariable(tns:purchaseOrder, tns:Consumer)"/>
            <populate variable="cdl:getVariable(tns:purchaseOrder, tns:Retailer)"/>
        </exchange>
        <exchange messageContentType="purchaseOrderAckType" action="respond">
            <use variable="cdl:getVariable(tns:purchaseOrderAck, tns:Retailer)"/>
            <populate variable="cdl:getVariable(tns:purchaseOrderAck, tns:Consumer)"/>
        </exchange>
    </interaction>
    …
</choreography>
**BPEL** meets **WS-Choreography**

- A scripted composition using
  - WSDL messages
  - Control structures with constrained parallelism
- More procedural
- Executable or abstract
- Favor centralized composition

- A global description of what and how WDSL messages are exchanged
- Declarative flavor
- Abstract and not executable (yet)
- Neutral re composition topology
- Channels can be passed around (c.f., π-calculus)
  - Supports a form of delegation
OWL-S (Formerly DAML-S)

- An emerging standard to add semantics:
  - An upper ontology for describing properties & capabilities of web services using OWL
- Enable automation: service discovery & selection, invocation, composition & interoperation, execution monitoring

```
Resource provides presents
Service providers what it does
Service Profile

Service Grounding supports describedby
Service (how to access) (how it works)

- input types
- output types
- preconditions
- effects

- process flow
- composition hierarchy
- process definitions
```

- communication protocol (RPC, HTTP, …)
- port number
- marshalling/serialization
• Service profile defines what the service provides:
  – Functional descriptions: In/Output, Preconditions, Effects
  – Non functional descriptions: name, category, QoS, ...

• Can use situation calculi (or PSL) as formal basis for pre-conditions, effects
  – See below

• Reasoning with pre-conditions and effects

However: OWL-S assumes pre-conditions/effects are in OWL-DL – cannot work with variables in rich ways
OWL-S Process Model

• Constructs for composite processes
  - Sequence
  - Concurrency: Split; Split+Join; Unordered
  - Choice
  - If-Then-Else
  - Looping: Repeat-Until; Iterate (non-deterministic)
  - Note: In spirit of Golog, these can be viewed as constraints

• Data Flow
  - No representation of explicit variables, no internal data store
  - Predicate “sameValues” to match input of composite service and input of subordinate service

• Less refined than, e.g., BPEL

• Message behavior of composed OWL-S services not well-understood
OWL-S Service Model

[Diagram showing the OWL-S service model with nodes and arrows indicating relationships and properties such as `precondition`, `hasProfile`, `output`, `computedPrecondition`, `computedInput`, `computedOutput`, `computedEffect`, `invocable`, `hasGrounding`, and `hasProcess`.]
Functionality Description

• Preconditions
  – Set of conditions that should hold prior to service invocation

• Inputs
  – Set of necessary inputs that the requester should provide to invoke the service

• Outputs
  – Results that the requester should expect after interaction with the service provider is completed

• Effects
  – Set of statements that should hold true if the service is invoked successfully
  – Often refer to real-world effects, e.g., Package being delivered, or Credit card being debited
Non Functional Properties

- Provides supporting information about the service
Profile Hierarchy

- Sub-classing the Profile model facilitates the creation and specialisation of service categories
- Each subclass can:
  - Introduce new properties
  - Place restrictions on existing properties
- Sub-classing can also be used to specialise requests for service
- An example Profile Hierarchy is provided, but others could just as easily be defined
Universal Description, Discovery and Integration (UDDI)

• Directory for web services
  – Communicate via SOAP
  – Includes descriptions of services, in terms of:
    • Business, services, binding, “technical fingerprints”

• tModels
  – “Schemas” for describing service templates (PortTypes)
    • There are tModel’s for WSDL descriptions of a service, for ebXML, ...
    • When a service registers with UDDI, the technical fingerprint includes listing of tModels that it uses
  – tModel’s can be registered, and incorporated into taxonomies

• Allows queries over services, tModels, implementations, and other information

• UDDI expected to expand over time, enabling richer service descriptions
OWL-S Profile Ontology is Analogous to the Concept of UDDI Taxonomy
Outline

• Introduction

• Standards

• Perspectives on “Composition”

• Analysis and Verification
  – A coordination model for telecom

• A Glimpse of the Future
Models of Interoperation

Different models focus on different aspects

• Automata-based
  – Intricate structure for atomic services
  – Rich interleaving between atomic components
  – Activity-based or message-based

• Logic-based perspectives
  – Frameworks supporting proof and model theories
  – Possible to model “effect on the world”
  – Different ways of modeling complex services

• Constraint-based
  – Support partial specification of desired behaviors
  – Non-determinism OK when building composition
First Impressions: Topology

Two common approaches:

- Mediated, or “hub and spoke” (typical of BPEL)

- Peer-to-peer
First Impressions: Enactments

- “Enactment” = the execution of multiple steps in a (composite) service, corresponding to a single instance of a (possibly complex) business process
- Nested enactments: one \textit{authorize}, several \textit{orders}
- Typical to focus modeling and reasoning on individual enactments
Compositions vs. Complex Individual Services

• **Re-usability of component parts**
  - For individual services, this is a design goal, but not enforced
  - For compositions, this is foundational assumption

• **World view**
  - Components of individual service can “see” the rest of the service, modulo scoping, etc.
  - Services in a composition have a limited interface to “see” other services (typically via messages only)
    - Implications on transactional aspects

• **Management of different enactments**
  - Individual service: Details of enactment management are hidden
  - Composite service: Need mechanism for associating activities of component services with appropriate global enactment
    - BPEL uses the phrase “correlation sets”
Service “Glue” Languages and Workflow Management

- “Computerised facilitation or automation of a business process, in whole or part” [WfMC]

- Centralized control
  - State of conversation maintained by WF manager

- Delegation of “almost everything” to the app.s
  - E.g., application data is not accessible to WF manager

- Workflow standardization has mixed success
  - Web services must interoperate ⇒ standards likely
  - Should focus on interfaces, not internals

Web services world is much more “open-ended”
Selected results on Automated Composition

Three families of results, each based on a different model

- **Mealy model**: Emphasis on message passing
  - Primarily useful for characterizations

- **Roman model**: Elegant approach in restricted framework
  - Using automata with emphasis on activities
  - Automata-based and logic-based proofs

- **“Semantic”**: Simple compositions driven by achieving goals “in the world”
  - Two approaches for combining atomic OWL-S services in a systematic fashion
## Selected Models

<table>
<thead>
<tr>
<th>Model</th>
<th>Emphasize Individual service</th>
<th>Emphasize Composite service</th>
<th>Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mealy/Conversation</td>
<td>yes</td>
<td>yes</td>
<td>Automata, message-based</td>
</tr>
<tr>
<td>Roman</td>
<td></td>
<td>yes</td>
<td>Automata, activity-based</td>
</tr>
<tr>
<td>Situation Calculus/PSL</td>
<td>yes</td>
<td>yes</td>
<td>First-order logic, activity-based</td>
</tr>
<tr>
<td>Data-Driven</td>
<td>yes</td>
<td></td>
<td>Automata (specified by rules)</td>
</tr>
<tr>
<td>CTR/CTR-S</td>
<td></td>
<td>yes</td>
<td>Stylized logic</td>
</tr>
<tr>
<td>Commitment</td>
<td></td>
<td>yes</td>
<td>Constraints, message-based</td>
</tr>
<tr>
<td>AZTEC, (Minoan)</td>
<td></td>
<td>yes</td>
<td>Emphasizes asynchronous events, telecom apps</td>
</tr>
</tbody>
</table>
Outline

- Introduction
- Standards
- Perspectives on “Composition”
  - Mealy machine/conversations
  - Roman
  - “Semantic”
- Analysis and Verification
  - A coordination model for telecom
- A Glimpse of the Future
A composition framework based on messaging

- A peer: autonomous process executing an e-service
- Assume reliable communication
E-Composition Schema

- An **ec-schema** is a triple \((P, C, M)\)
  Specifies the infrastructure of composition

- \(P\) : finite set of peers (e-services)
- \(C\) : finite set of peer-to-peer channels
- \(M\) : (finite) set of message classes

- Fixing an ec-schema is strong restriction
  - It would be useful to study a relaxation of this
What about inside the web service?

- In the most general case, a peer can be a Turing machine

- For analysis and optimization, useful to study more focused languages

```
Do until halt
  nondeterministic choice:
    read an input;
    send an output to some other peer;
    halt;
  end choice
```

- input messages
- local store
- message log
- to other e-services
Behavioral signatures via Mealy peers

- Finite State Automata with input/output
  - Follows spirit of process algebras, communicating processes

- Can model single or sequenced enactments
- Advantages for analysis, e.g., verifying temporal properties, characterizing global behavior
• Peer fsa’s begin in their start states
Executing a Mealy Composition (cont.)

- STORE produces letter a and sends to BANK
Executing a Mealy Composition (cont.)

- BANK consumes letter a
- Execution successful if all queues are empty and fsa’s in final state
“State” and “Conversation”

• The state of the composition is based on
  – state of each peer
  – contents of the queues

• Conversation: one enactment of global process
  – Can have “sub-conversations” of a conversation
  – Little known about formal properties of conversations
Important Choices for Composition Model

- Representation formalism for peer implementations
- Expressive power of peer implementations
- Bounded vs unbounded queues
- Several queues vs one queue vs heap vs ...
- Open vs closed
- Restricted topologies/control: peer-to-peer, hub-and-spoke, hierarchical, ...
Qualitative Analysis of Compositions for Message-based Models

- “Conversation Languages” [Bultan et.al. WWW’03]
- Assume a “watcher” that observes all messages sent
  - So, the “observables” here are simply the messages sent
- Language of peer implementation is set of words formed by successful executions of the implementation
Example Conversation Language

- **Language:** \( \text{ak SH( } (o_1 r_1 b_1 p_1)^*, (o_2 \text{ SH}(r_2,b_2 p_2))^* \text{ )} \)
  - First \( \text{ak} \), then a shuffle of orders against Supplier1 and orders against Supplier2
  - Supplier1 is “cautious” and Supplier2 is “trusting”

- This language is regular
- Same language for bounded or unbounded queues
Conversation Realizability in Mealy model

• In the example, the system realizes a regular language:

\[ a k \shuffle (o_1 r_1, b_1p_1)^*, (o_2(\shuffle (r_2, b_2p_2))^* ) \]

• Design question:

Given a (regular) language \( L \), can we design Mealy services so that their conversation language is \( L \)?
A Quick Answer: Some Regular Languages are not Realizable

- Very simple language \{abcde\}
  - Every Mealy composition allowing conversation abcde will also allow acbde

- Three reasons why a language may not be realizable:
  - Impact of local views — Projection-Join
  - Impact of send delays — Preponing
  - Impact of unbounded queues — can yield context-sensitive languages
Local View and Join

- Local view of a peer $\pi_{\text{peer}}(L)$: the part of conversation the peer participates (receives or sends)
- Given languages $L_i$ over $\Sigma_i$, $1 \leq i \leq n$
  \[ \bigotimes_i L_i = \left\{ w \mid \forall 1 \leq i \leq n, \pi_{\Sigma_i}(w) \in L_i \right\} \]
- Mealy conversation languages $L$ are closed under “projection-join”:
  \[ \bigotimes_{\text{peers}} \pi_{\text{peer}}(L) \subseteq L \]
Delayed Send and Prepone

- If the global watcher sees $w = \ldots a \ b \ldots$

- $\pi_p(w)$ should also allow $w' = \ldots b \ a \ldots$

- $w'$ is an element of the prepone of $w$

- Mealy conversation languages are closed under prepone
Unbounded Queues ⇒ Unexpected Behaviors

- Abstract versions of previous e-services
  - But, no “handshakes” for messages
- Conversation language $L$:
  - $L \cap a^*b^* = \{ a^nb^n | n \geq 0 \}$, i.e., $L$ is not regular
How bad is it?

- In general, conversation language with Mealy peers and unbounded queues is context-sensitive
  - Accepted by a quasi-realtime automaton with 3 queues

- Take away: “Bottom up” design of compositions may lead to undesirable global behaviors

- For hierarchical ec-schemas:
  
  Each peer is a Mealy implementation $\Rightarrow$ Conversation language is join-prepone closure of a regular language
Realizing the Closure of a Regular Language

- Given a regular language $L$, we can always find a Mealy composition that realizes the following closure:
  \[ \bowtie_{\text{peers}} \text{LocalPrepone}^* (\pi_{\text{peer}}(L)) \]
  - Based on a projection construction

- Provides an approach to “top-down” design

- Reminiscent of “synthesis” results from automata
A Sufficient Condition for “true” Realizability [Fu et al CIAA ’03]

- Let $L$ be a regular language, accepted by Mealy machine $A$
  - Let $A_1, ..., A_n$ be the “projections” of $A$ to peers 1, ..., $n$

- Three key conditions:
  1. Lossless join: $\text{JOIN}(\pi_1(L), ..., \pi_n(L)) = L$
  2. Queues are optional: can construct a product machine from the $A_1, ..., A_n$ in which every message sent is ready to be read immediately
  3. Each $A_i$ is autonomous: in each state, $A_i$ can do only sends, or only receive, or terminate

- The conjunction of these conditions implies that $L$ is realizable
Outline

• Introduction

• Standards

• Perspectives on “Composition”
  – Mealy machine/conversations
  – Roman
  – “Semantic”

• Analysis and Verification
  – A coordination model for telecom

• A Glimpse of the Future
“Roman” model: A Focus on human-machine

[Berardi et. al. ICSOC 03]

- Model of human-machine web services (e.g., Amazon)
- Focus on activities

- Abstract behavior of the Service:
  - Client selects next activity

Do until Client selects “End”
- 1. Give Client a choice of actions to be performed
- 2. Wait for Client choice
- 3. Perform action chosen by Client
Roman Model: Automata representation

- Transitions labeled by activities
- More abstract than message-based approach
- For a given state, the out-edges represent the set of options that will be presented to the user
Roman Model: Composition Problem

Desired Service

Music store (front-end)

init search cart buy

Web store

init search cart

“UDDI++”: Available services

Juke listen

Bank buy

init search

September 9-10, 2004

Web Services Composition for EDBT Summer School
A Roman Composition

Delegator for music store

Delegator: Activity-based FSM annotated with delegations

“UDDI++”: Available services
Main result on Roman Composition

- Note: In some cases, delegator is not simply a labeling of target machine

\[ \text{Target} \]
\[ \text{Delegator} \]

A1
\[ \begin{array}{c}
\text{a} \\
\text{c}
\end{array} \]
A2
\[ \begin{array}{c}
\text{c} \\
\text{b}
\end{array} \]

“UDDI++”: Available services

- Thm: Can determine if a delegator exists, and build it, in EXPTIME [Berardi et al ICSOC’03]
  - We’ll examine 2 proof techniques
Second example of Roman Delegator

• Delegator is not simply a labeling of desired service

Desired re-usable service
An automata-based proof of the Roman composition result

[Gerede et. al. ICSOC ’04]

• Building a “product” machine (Mealy, but outputting what peers should move)

• Machine can be non-deterministic, but at most exponential in size
Proof outline

• Lemma: If there is a delegator, then there is one that is a sub-machine of the product machine
  – Use a splicing argument

• Thm: Can construct delegator, if one exists, in time
  \[ O( |Target| \times |A_1| \times \ldots \times |A_n| ) \]
  – Lemma: can remove “bad” states
  – Lemma: From what remains, any deterministic sub-machine is a delegator
Enriching the Roman Model: Look Ahead [Gerede et. al. ICSOC ’04]

- There is no composition in this case
- Can develop a theory of “k look-ahead” delegators
- This defines a non-collapsing hierarchy, and there are examples needing “infinite” look-ahead
- Can reduce existence of k look-ahead delegator to 0 look-ahead case in PTIME; so finding a k look-ahead delegator is still in EXPTIME
A Logic-based Proof of Roman Composition Result
[Berardi et. al. ICSOC’03]

- (Deterministic) Propositional Dynamic Logic (DPDL)
  - Well-known modal logic for reasoning about programs
  - [e.g., short intro in The Description Logic Handbook]

Formulas: built from “propositional letters” (or “fluents”)

\[ \varphi ::= f \mid \neg \varphi \mid \varphi \land \varphi' \mid [r] \varphi \mid \langle r \rangle \varphi \]

- “f is true in current state”
- “starting from current state, every execution of r leads to state in which \( \varphi \) is true”
- “starting from current state, some execution of r leads to state in which \( \varphi \) is true”

Programs: built from atomic Programs

\[ r ::= \text{P} \mid r \cup r' \mid r;r' \mid r^* ( \mid \varphi? \mid r \neg ) \]
Semantics for (D)PDL

- Models have the form (a Kripke structure)

\[ M = (S, \{R_P\}, \Pi) \]

Set of “states”

For each atomic program P, a binary relation over S – “transition relation”

\[ \Pi : S \rightarrow 2^{\text{fluents}} \]

(for each state s, which fluents are true in s)

\[ M,s \models f \quad \text{if} \quad f \in \Pi(s) \]

\[ M,s \models \varphi \land \varphi' \quad \text{if} \quad M,s \models \varphi \text{ and } M,s \models \varphi' \]

\[ \ldots \]

\[ R_P \subseteq S \times S \text{ for each atomic program } P \]

\[ R_{r;r'} = R_r \circ R_{r'} \]

\[ \ldots \]
Key results for (D)PDL

- \( M \) satisfies \( \varphi \) if for some state \( s \), \( M,s \models \varphi \)

- \( \varphi \) is valid in \( M \) if for each state \( s \), \( M,s \models \varphi \)

- \( \Gamma \models \varphi \) if \( \varphi \) is valid in every \( M \) that satisfies \( \Gamma \)

- Thm:
  - Satisfiability is ExpTime complete
  - \( \varphi \) is satisfiable iff there is a model with size at most exponential in \( |\varphi| \) (and this can be constructed in exponential time)

- Tree model property: each model can be “unwound” to form a model that has a tree structure
Embedding of Roman Composition problem into DPDL

• Let \([u]\) denote \(\big(\bigcup_{a \in \Sigma} a\big)\)

• For each FSM \(A_i = (\Sigma, S_i, s^0_i, \delta_i, F_i)\) (including target)
  - Include fluents for each state, and for \(F_i\) (meaning that \(A_i\) is in a final state)
  - Also include fluent \(moved_i\), meaning that \(A_i\) just moved

• Formulas include for Target FSM \(A_0\)
  - \([u](s \rightarrow \neg s')\) for \(s, s'\) distinct
  - \([u](s \rightarrow \langle a\rangle true \land [a](s'))\) for each \(a\) such that \(\delta_0(s,a) = a'\)
  - \([u](s \rightarrow [a]false)\) if \(\delta_0(s,a)\) is undefined
  - \([u](F_0 \rightarrow \bigvee_{s \in F_0} s)\)

• Formulas for source machines \(A_i\) include, e.g.,
  - \([u](s \rightarrow [a]((moved_i \land s') \lor (\neg moved_i \land s)))\)

• Also, “all FSM’s start in start state”, “at least one source FSM moves”, “all FSM’s end in final state”
Summary on DPDL for Roman Composition

- **Thm:** The DPDL formula constructed as above is satisfiable iff there is a delegator
- **Cor:** Existence of delegator can be determined (and constructed) in EXPTIME
- **Note:** There is a close correspondence between (variants of) PDL and (variants of) Description Logics
  - Can use highly optimized tableaux-based algorithms from Description Logics to construct delegators
Enriching the “Roman” Model: Services that Delegate
[Berardi et. al.’04]

• New features
  – One service can “initiate” a request against another service
  – Target service can “ignore” a sequence of actions by the services (indicated by a $\tau$ move)

• Can generalize previous EXPTIME results
  – Using different variant of Propositional Dynamic Logic
  – Ability to work with richly incomplete information is especially appealing
Outline

• Introduction

• Standards

• Perspectives on “Composition”
  – Mealy machine/conversations
  – Roman
  – “Semantic”

• Analysis and Verification
  – A coordination model for telecom

• A Glimpse of the Future
Situation Calculi and PSL: Logics with Actions and Model of “World”

- Formal foundations for “semantic” models of services

Situation Calculi  [cf Reiter’s book “Knowledge in Action”]

- This formalism used widely in AI for reasoning about actions
  - It can provide the semantics underlying OWL-S executions

- Models: Trees whose nodes correspond to “situations”

- “Fluents”:
  - Propositions (and predicates) which hold in situation
    - f(x₁,...,xn, s) denotes that f(x₁,...,xn) holds in situation s
  - Used to test pre-conditions, record effects (e.g., for OWL-S)

- An “unraveled” model of PDL can be viewed as a propositional situation calculus model
Typical model in Situation Calculus

- Axioms to describe initial situation T0
- Action precondition axioms $Poss(a,s)$, for each primitive action
- Successor state axioms, one for each fluent $f$:
  - Describes when $f(x1...xn, do(a,s))$ holds
- Unique name axioms for the primitive actions
- Some foundational, domain independent axioms
  - Second-order closure axiom used to ensure that all branches are finite
Process Specification Language (PSL)

- An ISO standard, originally for sharing manual specs
- Models:
  - Essentially, trees whose nodes correspond to atomic actions
- “Fluents”:
  - Propositions (and predicates) which hold between actions
- Vocabulary of PSL (richer than typical situation calculi)
  - Various layers of reified predicates, e.g.,
    - `activity (a)`, `activity_occurrence (o)`, `timepoint (t)`
    - `occurrence_of (o, a)`, `min_precedes (o1, o2, a)`, `precedes(o1, o2, a)`
    - `holds (f, o)`, `prior (f, o)`
  - Activities and occurrences identified using variables and terms (e.g., `withdraw(x, y)`)
- Formal semantics given in first-order logic
  - Family of axioms associated with each layer/variant of PSL
PSL: Simple illustration of the model theory

Atomic activities:
\[
\begin{align*}
  w1 &= \text{withdraw (100, buyer)} \\
  d1 &= \text{deposit (100, seller)} \\
  w2 &= \text{withdraw (5, buyer)} \\
  d2 &= \text{deposit (5, broker)}
\end{align*}
\]

- Can add constraints, e.g., that \( w1 \) must precede \( w2 \)
- Can use FOL inference or domain-specific reasoning
Expressive power of PSL

Examples of PSL

- Activities as terms: \( \forall x,y,z \text{ activity}( \text{transfer}(x,y,z) ) \)
- Composition relationships:
  - \( \forall x,y,z \text{ subactivity}(\text{withdraw}(w,y), \text{transfer}(x,y,z)) \)
  - \( \forall a,y ( a = \text{buy_product}(y) \supset x,z \text{ subactivity}(\text{transfer}(x,y,z), a) ) \)
- Process description for \text{buy_product}
  - \( \forall o,x \text{ occurrence_of}(o, \text{buy_product}(x)) \supset \exists o1,o2,y,z,w,v (\text{occurrence_of}(o1, \text{transfer}(y,x,z) \wedge \text{occurrence_of}(o2, \text{transfer}(w,x,v) \wedge \text{subactivity_occurrence}(o1, o) \wedge \text{subactivity_occurrence}(o2, o) ) ) \)

Situation Calculi typically less expressive

- Cannot have variables for composite activities/occurrences
- Cannot have variable ranging over fluents

But: PSL based on first-order entailment; Situation calculi typically use a second-order closure axiom
(Some) sentences characterizing a financial example

- **Activities as terms**
  \[ \forall x \text{ activity}( \text{buy_products}(x) ) \]
  \[ \forall x, y, z \text{ activity}( \text{transfer}(x, y, z) ) \]
  \[ \forall x, y \text{ activity}( \text{withdraw}(x, y) ) \]
  \[ \forall x, y \text{ activity}( \text{deposit}(x, y) ) \]

- **Composition relationships**
  \[ \forall a, y \ ( a = \text{buy_product}(y) ) \supset x, z \text{ subactivity}( \text{transfer}(x, y, z), a ) \]
  \[ \forall x, y, z \text{ subactivity}( \text{withdraw}(x, y), \text{transfer}(x, y, z) ) \]
  \[ \forall x, y, z \text{ subactivity}( \text{deposit}(x, z), \text{transfer}(x, y, z) ) \]

- **Process description for buy_product**
  \[ \forall o, x \text{ occurrence_of}(o, \text{buy_product}(x)) \supset \exists o_1, o_2, y, z, w, v \text{ occurrence_of}(o_1, \text{transfer}(y, x, z) \land \text{occurrence_of}(o_2, \text{transfer}(w, x, v) \land \text{subactivity_occurrence}(o_1, o) \land \text{subactivity_occurrence}(o_2, o)) \]

- **Can represent**
  - Other composite activities
  - Pre-conditions (e.g., transfers only if sufficient funds)
  - Effects (e.g., of a transfer)
PSL: Layers of Ontology and Axioms

- Base layer has only activities, occurrences, ordered time
- Additional layers build out, e.g., "composite activity", duration, legal, possible, ...
- Axioms included for each layer

| T_{psl-core} | activity(a) | a is an activity |
| T_{activity} | activity_occurrence(o) | o is an activity occurrence |
| T_{timepoint} | tpoint(t) | t is a timepoint |
| T_{object} | object(x) | x is an object |
| T_{occurrence} | occurrence_of(o, s) | o is an occurrence of s |
| T_{begin} | begin(t) | the beginning timepoint of o |
| T_{end} | end(t) | the ending timepoint of o |
| T_{before} | before(t_1, t_2) | timepoint t_1 precedes timepoint t_2 on the timeline |

- T_{subactivity} | subactivity(e_1, e_2) | e_1 is a subactivity of e_2 |
- T_{primitive} | primitive(a) | a is a minimal element of the subactivity ordering |

- T_{atomic} | atomic(a) | a is either primitive or a concurrent activity |
- T_{conc} | conc(a_1, a_2) | the activity that the concurrent composition of a_1 and a_2 |

- T_{occurrence} | successor(a, s) | the element of an occurrence tree that is the next occurrence of a after the activity occurrence s |
- T_{legal} | legal(s) | s is an element of a legal occurrence tree |
- T_{initial} | initial(s) | s is the root of an occurrence tree |
- T_{earlier} | earlier(s_1, s_2) | s_1 precedes s_2 in an occurrence tree |
- T_{pos} | pos(a, s) | there exists a legal occurrence of a that is a successor of s |

- T_{time-state} | holds(f, s) | the fluent f is true immediately after the activity occurrence s |
- T_{prior} | prior(f, s) | the fluent f is true immediately before the activity occurrence s |

- T_{complex} | min_precedes(s_1, s_2, a) | the atomic subactivity occurrence s_1 precedes the atomic subactivity occurrence s_2 in an activity tree for a |
- T_{root} | root(s, a) | the atomic subactivity occurrence s is the root of an activity tree for a |

- T_{subactivity} | subactivity_occurrence(e_1, e_2) | e_1 is a subactivity occurrence of e_2 |
- T_{root_occ} | root_occ(a) | the initial atomic subactivity occurrence of a |
- T_{leaf_occ} | leaf_occ(a, o) | o is the final atomic subactivity occurrence of a |

- T_{duration} | timeduration(d) | d is a timeduration |
- T_{duration} | duration(t_1, t_2) | the timeduration whose value is the "distance" from timepoint t_1 to timepoint t_2 |
- T_{lesser} | lesser(d_1, d_2) | the linear ordering relation over timedurations |

[Backup]
[Gruninger '03]
“Programming” as constraints

• Golog: a “programming language” for the situation calculus
  – “Constructs” such as
    • Sequence: $\delta_1 ; \delta_2$
    • Conditional: $\text{if } \phi \text{ then } \delta_1 \text{ else } \delta_2 \text{ endif}$
    • Loop: $\text{while } \phi \text{ do } \delta \text{ endWhile}$
  – Interpreted as temporal constraints on permitted paths
    • E.g., “$w_1; w_2$” is satisfied by 3 of the 6 branches
  – ConGolog: generalization to incorporate concurrent activities

• Two-tier “program” specification
  – First tier: use the “constructs” from above
    • Identifies a set of possible execution sequences
  – Second tier: arbitrary constraints
    • Further restricts set of possible execution sequences
Examples of Golog expressions

• Golog program fragments

  while \( \exists x ( \text{hotel}(x) \land \text{goodLoc}(x)) \) do
  \hspace{1em} \text{checkAvailability}(x, \text{dest}, \text{dDate}, \text{rDate})
  endwhile

• Personalization constraints

  if \( \text{hotelAvailable}(\text{dest}, \text{dDate}, \text{rDate}) \) then
  \hspace{1em} \text{bookB&B}(\text{cust}, \text{dest}, \text{dDate}, \text{rDate})
  endif

• Procedure definition

  proc \text{Travel}(\text{cust}, \text{origin}, \text{dest}, \text{dDate}, \text{rDate}, \text{purpose});
  \hspace{1em} if \text{registrationRequired} \text{ then } \text{Register};
  \hspace{1em} \text{BookTransport}(\text{cust}, \text{origin}, \text{dest}, \text{dDate}, \text{rDate});
  \hspace{1em} \text{BookAccommodations}(\text{cust}, \text{dest}, \text{dDate}, \text{rDate});
  \hspace{1em} \text{UpdateExpenseClaim}(\text{cust});
  \hspace{1em} \text{Inform}(\text{cust});
  endproc
Composition with OWL-S and ConGolog interpreter

[McIlraith+Son ’02]

Framework based on two tiers:

- Generic programs and Customization via constraints

- Start with family of atomic OWL-S services, with pre-conditions and effects

- Write Golog program capturing constraints on generic flow of control and parameter passing

- Write additional constraints (in situation calc) to capture personalization
  - Typically express them as Horn formulas

- Use ConGolog interpreter to find one (or more) branches in situation calc tree that satisfies all constraints
  - "Middle-Ground Optimization" based on gathering data in advance of world-altering activities
Automatic Composition for OWL-S via Petri nets [Narayanan+McIlraith ‘02]

• Can simulate OWL-S via 1-safe Petri nets
  1. For set of atomic e-services, create Petri Net that represents all possible (single-use) combinations of them
  2. Specify desired goal as a state of this Petri Net
  3. Determine if this goal state is reachable

• In this framework reachability is PSPACE-complete in size of Petri net
  – Different complexities depending on OWL-S constructs permitted
  – Petri net itself may be exponential in size of atomic e-services
  – Heuristics can be used to avoid full construction
Illustration of OWL-S automated composition via Petri nets

• Simplifying assumption for this example
  – Assume only propositional variables

• Given:
  – Finite set of OWL-S atomic services $S_1, \ldots, S_n$
  – Pre-condition (e.g., in DNF)
  – Goal – the desired post-condition (again in DNF)

• Determine whether there is a sequence $S_{i_1}, \ldots, S_{i_j}$ that:
  – When started under the pre-condition it achieves the goal
  – No repeats of an atomic service
  – Note: in an execution, some services in the sequence may fail

• Technique: Build a Petri-net that captures
  – Impact of pre-conditions
  – Impact of (attempted) execution of an atomic service
  – Satisfaction of post-condition as some place in the Petri-net

• If the post-condition can be reached, then goal achievable
Block for one atomic service

- Assume just propositional variables $a$, $b$
- Assume service $S_1$ has
  - Pre-condition: $a \land \neg b$
  - Post-condition: if $a$ then $b$ becomes true

\[ S_1 \text{ succeed} \]
\[ S_1 \text{ fail} \]

Size is $O(2^{|\text{var}|})$
Assembling the full Petri-net

- Size is $O(n!) = O(2^n) \times$ size of service block;
  Overall size is $O(2|\text{problem statement}|)$
- Reachability in 1-safe Petri-nets is PSPACE
- Overall decision procedure is EXPSPACE
2 Flavors of Composition

- Example: reserving and paying for a plane ticket
  - If cheap, use CreditCard; if expensive use MoneyOrder

“All Knowledge in Advance”
- Know in advance whether the ticket is cheap or not

```
R
 CC
R
 MO
```

• Can describe in a linear temporal logic
  - “After R there is CC or MO”

• Can use “transition system” as underlying framework

“Knowledge obtained along the way”
- Find out in middle of execution whether ticket is cheap or not
- Might be learned from external world, or from user

```
R
 CC
R
 MO
```

• Need a branching logic to describe
  - “After R there is a branch with CC and a branch with MO”

• Need to use something richer
• Cf. the k-look-ahead results
Outline

• Introduction

• Standards

• Perspectives on “Composition”

• Analysis and Verification
  – A coordination model for telecom

• A Glimpse of the Future
Analysis of Web Services

• Service properties:
  – Statements on functional logic, service guarantees, ...
  – Statement on execution (deadlock, safety, ...)

• Analysis may be in more demand:
  – Dynamic composition
  – Difficulties in testing
  – Immature service oriented development environments

• Possible approaches:
  – Static: model checking, theorem proving, ...
  – Runtime monitoring
Model Checking

- The target of interest is given as a state transition system.
- Properties are specified in some temporal logic, e.g., linear temporal logic (LTL), branching time logic (CTL*),...
  - Model checking within $2^{|\phi| |\text{transition system}|}$
- The entire state space is examined systematically
  - Explicit (automata techniques): e.g., SPIN, CWB, ...
  - Symbolic, using forward or backward fixpoint: e.g., SMV
    - BDDs can be used to symbolically represent sets of states
- Model checking and compositions: The challenge is to map a composition analysis problem to a model checking problem
  - May need an approximation of the composition model
Approaches Examined

- **Model checking**
  - With finite state machines
    - [Fu et al WWW’04] [Foster et al ASE’03]
  - With process algebra [Koshikina-van Breugel 2003]

- **OWL-S services analysis with Petri nets**
  - [Narayanan-McIlraith WWW’02]

- **Rule-based services** [Deutsch et al PODS’04]

- **Analysis of a Natural Fragment of CTR**
  - [Davulcu et al PODS’98]

- **Dynamic verification of protocol compliance in commitments** [Venkatraman and Singh ’99]

- **An approach for queues and spawning, based on counters** [a work in progress]
Explicit Model Checking

- Transition system $R$ and initial states $I$ → automaton $M_R$

- Property $P \rightarrow \neg P$ → automaton $M_{\neg P}$

- Question: $M_R \cap M_{\neg P} = \emptyset$ ?
Symbolic Model Checking: Forward Fixpoint

Initial states \( I \), Transition relation \( R \), Property \( P \)

\[ F \subseteq P \]
Symbolic Model Checking: Backward Fixpoint

Initial states $I$, Transition relation $R$, Property $P$

Question: $B \cap I = \emptyset$ ?
Verifying Temporal Properties of Mealy Compositions

- Label states with propositions
  - Level of indirection between states and “observables”
- Express temporal formulas, e.g.,
  - “shipment just made” only after “line-of-credit avail”
Results on Temporal Verification

• Long history, e.g., [Clarke et.al. ’00]
• E.g., verification for fsa’s and propositional LTL
  – Complexity: PSPACE in size of formula + fsa
  linear time in size of fsa
• Application to Mealy compositions
  – Results apply to open and closed case
  – Bounded queues
    • Composition can be simulated as Mealy machine
    • Verification is decidable
    • Standard techniques to reduce cost
  – Unbounded queues
    • In general, undecidable
    • Approximation techniques can be applied
Verification of Web services

- Model execution as finite state machines
  [Foster et al ASE ’03] [Fu et al WWW ’04]
- [Fu et al WWW ’04] Verifying conversation among a set of BPEL composite services

- Approach: model BPEL services as "guarded automata"
  - Mealy machines + conditions on transitions + XML messages + XML local data
  - Translated to Promela (input language of SPIN)

LTL properties: Every authorize followed by some bill?
BPEL to Guarded Automata

[Fu et al WWW ’04]

- Each atomic activity → an automaton with single entry, single exit

```xml
<receive ...
  operation = “approve”
  variable = “request” />
```

```xml
<invoke operation=“approve”,
  invar=“request”,
  outvar=“aprvInfo” >
  <catch faultname=“loanfault“>
    < ... handler1 ... />
  </catch>
</invoke>
```
BPEL to Guarded Automata

- Control flow constructs: assemble Mealy machines

```xml
<sequence .../>
  <... act1.../>
  <... act2.../>
</sequence .../>
```

```
<flow .../>
  <... act1 ...>
    <source linkname = “link1”
      condition = “cond1” />
  </act1>
  <... act2 ...>
    <target linkname = “link1” />
  </act2>
</flow .../>
```
Partial and Complete Verification (with SPIN)

- Promela: input language of SPIN
  - Concurrent processes communicating with bounded queues
- Each guarded automaton $\Rightarrow$ a Promela program
  - Bound the queue length $\Rightarrow$ partial verification
- Synchronizable $\Rightarrow$ complete verification

Conversation:

\[
\begin{array}{cccccccc}
\text{store} & \text{bank} & \text{warehouse}1 & \text{warehouse}2 \\
\text{authorize} & \text{ok} & \text{order} & \text{payment1} & \text{payment1} & \text{bill1} & \text{receipt1} & \text{bill2} & \text{receipt2} \\
\text{order} & \text{payment2} & \text{bill2} & \text{receipt2} & \text{payment2} & \text{bill1} & \text{receipt1} & \text{order} & \text{authorize} \\
\end{array}
\]

LTL properties: Every authorize followed by some bill?
# Synchronizability Analysis Results

<table>
<thead>
<tr>
<th>Source</th>
<th>Web service</th>
<th>Name</th>
<th>#msg</th>
<th>#states</th>
<th>#trans.</th>
<th>Synchronizable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISSTA'04</td>
<td>SAS</td>
<td>9</td>
<td>12</td>
<td>15</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>IBM Conv. Support Project</td>
<td>CvSetup</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MetaConv</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Chat</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Buy</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Haggle</td>
<td>8</td>
<td>5</td>
<td>8</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AMAB</td>
<td>8</td>
<td>10</td>
<td>15</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>BPEL spec</td>
<td>shipping</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Loan</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Auction</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>Collaxa.com</td>
<td>StarLoan</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Caution</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>yes</td>
<td></td>
</tr>
</tbody>
</table>
Automata → Promela

• Translate to Promela (input lang. of SPIN)
  – 1 web service composition → 1 Promela specification
  – 1 Guarded Automaton → 1 Promela Process
  – Each Promela process has one Channel
  – Channels are bounded!

• Challenge: handling XML Data [ISSTA’04]
  – (bounded) XML Schema Type → Type system in Promela
  – Symbolic emulation of XPath expression
  – Special handling of position() and last() functions
    • Map each function to an integer variable
Verification Experience

• Handling of XPath helps find intricate bugs!
  – Stock Analysis Service [Fu et. al. ISSTA’04]

• Experience with other examples
  – Exhaustive search takes a lot of time
  – Performance satisfactory when finding errors
  – SPIN does not scale well with data domains
  – Use of Symbolic Verification may help

```
Investor
  regList of stockIDs

Broker

Research Dept.
  1 stockID per request
```

```
Find stockID in last Request;
If last ID in regList exit;
Else
  send its subsequent stockID;
```
Web Service Analysis Tool (WSAT)
[Santa Barbara group WWW’04, ISSTA’04, CAV’04]

- An aid for debugging web service compositions
  - Peer-to-peer or mediated

WS-Choreography, OWL-S

Front End

Guarded Automata

Back End

Sync. Analysis

Complete verification

SPIN

Partial verification

SPIN

other verification tools

Interacting BPEL Web Services

LTL properties

Interacting BPEL Web Services

LTL properties
Verification with Process Algebra

[Koshikina-van Breugel 2003]

• BPEL control structures $\rightarrow$ BPE-calculus
• BPE-calculus $\rightarrow$ PAC and then to CWB
• Concurrency Workbench [Cleaveland and Sims CAV’96]
  – Model checking tool for CCS and CSP

• Checking if a BPEL composition is deadlock-free
  – In general, temporal properties
• Message contents and local data contents are not modeled
Verification of OWL-S services
[Narayanan-McIlraith WWW’02]

Analyzing and automated composition of OWL-S

• Analysis: Does an OWL-S service satisfy some property?

• Approach:
  – Simulating $S$ using a Petri net
  – Conduct Petri net reachability analysis

• DAML-S (v0.5) analysis is PSPACE complete
  – Reachability of 1-safe nets (each place is marked 1 or less)
Mapping OWL-S to Petri Net

• Petri nets:
  – Places: hold tokens
  – Transitions: consume input tokens and produce output tokens
  – Marking: a snapshot
  – Reachability: one marking to another via transitions

• OWL-S to Petri net mapping:
  – Conditions (in situation calculus): places
  – Atomic services: transitions
    • Pre- and post-conditions

• Inductive mapping from services to Petri nets
OWL-S to Petri Net

- Control structures “glue” pieces together
If-Then-Else and Loops
Data Driven Web Service

[Deutsch et al PODS’04]

- Emphasis is on interaction between control flow and database contents
- Transitions resemble Datalog rules and update the database
- Rules + DB can be used to simulate control structures

```
Home page(HP)
Customer page(CP)
Product index page(PIP)
Product detail page(PP)
Confirmation page(CoP)
Error message page(MP)
Past Order (POP)
Order status(OSP)
Order status(COP)
Customer page(CP)
Desktop search
Laptop search
```

```
Effects (updateable)
```

```
Read only
```

```
Data Driven Web Service
[Deutsch et al PODS’04]

- Emphasis is on interaction between control flow and database contents
- Transitions resemble Datalog rules and update the database
- Rules + DB can be used to simulate control structures

```
Database
```

```
Bell Labs
Innovations for Lasting Technologies
```

```
September 9-10, 2004 Web Services Composition for EDBT Summer School
```

```
Data Driven Web Service
[Deutsch et al PODS’04]

- Emphasis is on interaction between control flow and database contents
- Transitions resemble Datalog rules and update the database
- Rules + DB can be used to simulate control structures

```
Data Driven Web Service
[Deutsch et al PODS’04]

- Emphasis is on interaction between control flow and database contents
- Transitions resemble Datalog rules and update the database
- Rules + DB can be used to simulate control structures

```
Database
```

```
Bell Labs
Innovations for Lasting Technologies
```

```
September 9-10, 2004 Web Services Composition for EDBT Summer School
```

```
Data Driven Web Service
[Deutsch et al PODS’04]

- Emphasis is on interaction between control flow and database contents
- Transitions resemble Datalog rules and update the database
- Rules + DB can be used to simulate control structures

```
Data Driven Web Service
[Deutsch et al PODS’04]

- Emphasis is on interaction between control flow and database contents
- Transitions resemble Datalog rules and update the database
- Rules + DB can be used to simulate control structures

```
Database
```

```
Bell Labs
Innovations for Lasting Technologies
```

```
September 9-10, 2004 Web Services Composition for EDBT Summer School
```

```
Data Driven Web Service
[Deutsch et al PODS’04]

- Emphasis is on interaction between control flow and database contents
- Transitions resemble Datalog rules and update the database
- Rules + DB can be used to simulate control structures

```
Data Driven Web Service
[Deutsch et al PODS’04]

- Emphasis is on interaction between control flow and database contents
- Transitions resemble Datalog rules and update the database
- Rules + DB can be used to simulate control structures

```
Database
```

```
Bell Labs
Innovations for Lasting Technologies
```

```
September 9-10, 2004 Web Services Composition for EDBT Summer School
```

```
Data Driven Web Service
[Deutsch et al PODS’04]

- Emphasis is on interaction between control flow and database contents
- Transitions resemble Datalog rules and update the database
- Rules + DB can be used to simulate control structures

```
Data Driven Web Service
[Deutsch et al PODS’04]

- Emphasis is on interaction between control flow and database contents
- Transitions resemble Datalog rules and update the database
- Rules + DB can be used to simulate control structures

```
Database
```

```
Bell Labs
Innovations for Lasting Technologies
```

```
September 9-10, 2004 Web Services Composition for EDBT Summer School
```
An Abstract Perspective: Rule-based Control

- A hybrid combining automata and logic

Condition based on database query

Updates to the database

If \( \psi \) then run \( S \)

A complex service:

If \( \psi_1 \) then run \( S_1 \)
: 
If \( \psi_n \) then run \( S_n \)
Verification of Rule Based Service

[Deutsch et al PODS’04]

First order temporal properties (FO-LTL)

- Verification is in PSPACE for a restricted model (input-bounded services)
- Undecidable with slight generalization
- Use and extend technique for abstract state machines [Spielman PODS’01]

Verification of CTL properties

- Decidable for the propositional case
Outline

• Introduction

• Standards

• Perspectives on “Composition”

• Analysis and Verification
  - A coordination model for telecom

• A Glimpse of the Future
Telecom services:
2 Critical Differentiators

• Spawning of like processes, e.g.,
  – Adding participants to a teleconf
  – Re-connecting people that dropped
  – BPEL does not support internal spawning

• Graceful handling of “arbitrary” asynchronous events
  – People losing wireless connection
  – People becoming present
  – Note: BPEL requires scoping of points waiting for asynchronous events

• (Draft) “Minoan” model: combines OO, spawning, asynchronous events

• Preliminary result: Applies counter machines to verify properties of Minoan compositions
**Minoan Model**

[Benedikt, Christophides, Hull, Karvounarakis, Kumar, Tannen, Su]

- M-classes are analogous to objects from OOPLS
- Internally: guarded automata
- Messages go to existing process or spawn new one
- Various approaches to local data stores
IMS-inspired example

Two applications, written independently
- TS: Sets up, maintains teleconferences
- AS: Automatic reconnect for wireless users that drop

Mediator/Coordinator: Gives TS and AS illusion of working independently
Inside the services and mediator

- All three modules have a “root” process
- Additional M-classes in Mediator
  - Forward request to participate in teleconf to TS
    - If subscriber has paid for AS, also forward info to AS
    - Handle TS requests (e.g., telling BM to add someone to conf)
  - Handle situation where user drops connection
    - Forward to AS, and deal with AS response
- Additional M-classes in TS:
  - Manage teleconf, manage participant in teleconf
Representative M-class of Mediator

Mediator.Conference_Request (params: reqid, devid, isprem) (1 per request)

- Even in a single teleconf, there may be multiple M-instances for this M-class
- Unbounded spawning \(\Rightarrow\) verification undecidable
- As an abstraction, we will view the “tokens” moving around this FSM as uniform

Local vars: reqid, devid, confid, bid, apid, aid: ids ; is_prem: bool
A Basic Question, answered via counters

• “Bounded Stability” : For each possible state of the system, if a finite number of events come in from the deep network, will processing eventually terminate?

• Towards sufficient condition (and algorithm) for bounded stability
  – Ignore data stores and message contents
  – Assume one message type per queue
  – For each M-class, for each state of the M-class, keep a count of number of M-instances that are in that state
  – Also count the number messages queued

• “Stability Ordering” for the counters: one such that each valid move leads to decrease of counter values, according to lexicographical ordering
Results so far

• Thm: If there is a stability ordering, then the system has bounded stability

• Algorithm: searches for stability ordering, by adding constraints based on set of M-classes
  – If no cycle found, then abstract and concrete systems have bounded stability
  – If cycle found, then the abstract system does not bounded stability

• We see this as a first step towards (approximate) solution of general properties of Minoan systems
Part of system view, with queues and counters for M-class states

---

September 9-10, 2004 Web Services Composition for EDBT Summer School 130
Abstract view of (part of) system, entirely counter-based

\begin{align*}
\text{init) TR}_0 = 1 \\
\text{init) MR}_0 = 1 \\
\text{init) MCR}_0 = 1 \\
MR_0 > 0 & \text{Æ conf req1} > 0 \) \text{conf req1} --, (spawn \ (M.CR)) \text{MCR}_0 ++ \\
MR_0 > 0 & \text{Æ new conf started} > 0 \) \text{new conf started} --, \text{new conf started} 0 ++ \\
MR_0 > 0 & \text{Æ addpart2conf} > 0 \) \text{addpart2conf} --, \text{addpart2conf} 0 ++ \\
MR_0 > 0 & \text{Æ added2conf} > 0 \) \text{added2conf} --, \text{added2conf} 0 ++ \\
MCR_0 > 0 & \text{Æ MCR}_0 --, \text{MCR}_1 ++, \text{conf req2} ++ \\
MCR_1 > 0 & \text{Æ new conf started} 0 > 0 \) \text{new conf started} 0 --, \text{MCR}_1 --, \text{MCR}_2 ++, \text{setup bridge}++ \\
MCR_1 > 0 & \text{Æ added2conf} 0 > 0 \) \text{added2conf} 0 --, \text{MCR}_1 --, \text{MCR}_3 ++ \\
MCR_2 > 0 & \text{Æ bridge set} > 0 \) \text{bridge set} --, \text{MCR}_2 --, \text{MCR}_3 ++ \\
MCR_3 > 0 & \text{Æ addpart2conf} 0 > 0 \) \text{addpart2conf} 0 --, \text{MCR}_3 --, \text{MCR}_4 ++, \text{addleg2bridge}++ \\
MCR_4 > 0 & \text{Æ leg added} > 0 \) \text{MCR}_4 --, \text{leg added} --, \text{partic added}++ \\
TR_0 > 0 & \text{Æ conf req2} > 0 \) (spawn \ (TS.TM)) \text{conf req2} --, \text{TM}_0 ++ \\
TR_0 > 0 & \text{Æ conf req2} > 0 \) \text{conf req2} 0 ++ \\
TR_0 > 0 & \text{Æ partic added} > 0 \) \text{partic added} --, \text{partic added} 0 ++ \\
TM_0 > 0 & \text{Æ TM}_0 --, \text{TM}_1 ++, \text{new conf started}++ \\
TM_1 > 0 & \text{Æ TM}_1 --, \text{TM}_2 ++ (spawn \ (TS.AP)) \text{TA}_0 ++ \\
TM_2 > 0 & \text{Æ conf req2 0 >0} \) \text{TM}_2 --, \text{TM}_1 ++, \text{conf req2} 0 --, \text{added2conf}++ \\
TA_0 > 0 & \text{Æ TA}_0 --, \text{TA}_1 ++, \text{addpart2conf}++ \\
TA_1 > 0 & \text{Æ partic added} 0 > 0 \) \text{partic added} 0 --, \text{TA}_1 -- \\
BR_0 > 0 & \text{Æ setup bridge} > 0 \) \text{setup bridge} --, \text{bridge set} ++ \\
BR_0 > 0 & \text{Æ addleg2bridge} > 0 \) \text{addleg2bridge} --, \text{leg added}++
\end{align*}
Outline

• Introduction

• Standards

• Perspectives on “Composition”

• Analysis and Verification
  – A coordination model for telecom

• A Glimpse of the Future
Anatomy of Web Services Composition

- No unified model:
  - BPEL, Mealy: Strong on orchestration, info sharing
  - OWL-S: Strong on goals, activities, discovery
  - “Roman” model: Strong on activities, orchestration
Composition Models and Standards

Mealy Model
WSCL

WS-Choreography
BPEL4WS

Commitments

PSL/ Situation Calculi
ConGolog*

OWL-S
Roman
CTR-S*
Data-Driven*

*if interpreted as a composition model

Rich on messages

Rich on activities
Elements of a unified model
[based on recent discussions with D. Berardi, D. Calvanese, G. de Giacomo, M. Lenzerini, and M. Marcello]

Crucial Feature

- Usable model of the “world”
- Notion of atomic actions that examine/impact the world
- Notion of web service that wraps
  - Atomic actions
  - Message send/receive
  - Process/Enactments
- Will probably need a notion of “semantic behavior signature” to describe services
- Message-based communication between services
- Automatic composition

Possible starting point?

- Fluents, as in PSL or sit. calc.
- OWL-S atomic services
- Many choices . . .
- Guarded automata (generalization of Mealy)
- Use lessons from Mealy machines
- Generalize Roman approach? Maybe high complexity or undecidable; issue of knowledge gained during execution
- Leave for another day...
Additional concern: The structure of processes and enactments

• Typical to conceptualize a process spec, and a process instance for each request
  – Cf. object-oriented style
  – Tendency to build models, reasoning on “per enactment” basis

• But one order might have multiple sub-requests
  – E.g., one order might involve processing each item separately
  – Possible argument for “unbounded” spawning of processes

• Sub-request activities might be grouped
  – E.g., parts of several book orders may be combined into one shipment
  – Reasoning on “per enactment” basis will not be sufficient

• Telecom and “always-on” services
  – Enactments may have unbounded duration
  – Process spawning appears useful for specifying behaviors
Reasons for Optimism

• Broad and growing interest, across several disciplines
  – AI
  – Services oriented Computing
  – Databases
  – Verification, Logic in Computer Science

• Consortia have formed
  – Semantic Web Services Initiative (SWSI)
    • Re-thinking OWL-S, e.g., to overcome limitations re variable passing
  – Web Services Model Framework (WSMF)

• Government research funding
  – Substantial in Europe
  – US: DARPA’s DAML program was significant beginning; follow-on may be created

• Substantial groundwork has been laid