

Consistency for Service Oriented Systems

Presenter: Surya Nepal

The Team: Alan Fekete (Sydney University), Paul Greenfield, Julian Jang, Dean Kuo, Surya Nepal

**Distributed Systems Research
Networking Technologies Laboratory
CSIRO ICT Centre**



CeNTIE is supported by the Australian Government through the Advanced Networks Program (ANP) of the Department of Communications, Information Technology and the Arts



- Web services vision
- Robustness problem
- Consistency approaches
- Concluding remarks

- Building large-scale distributed applications from components called services that are loosely-coupled, autonomous, opaque, stateful, and communicate solely by exchanging asynchronous messages.
 - Complete standard distributed computing platform
 - Discovery, description, invocation, security, reliability, ...
 - Applications are components providing services
 - Custom applications, packages, 3rd parties, partners
 - Business processes built by integrating services
 - Integrating 3rd party services into applications
 - Integrate Visa, FedEx, Tax into your application
 - New businesses providing & enhancing services
 - Brokering best prices for commodities/services

Nice Vision But

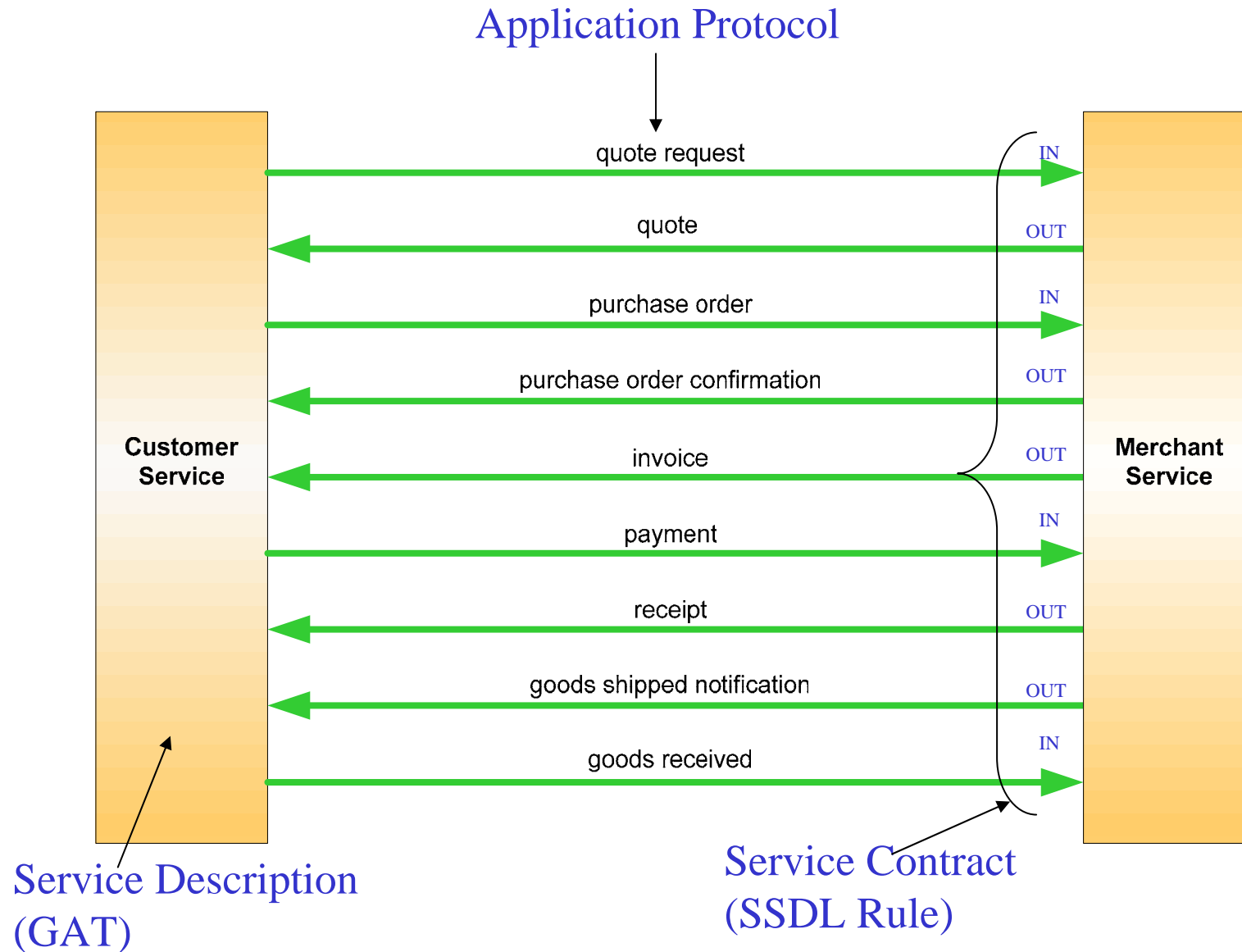
- Difficult or impossible to realise today
 - Problems of robustness, architecture, information sharing & distribution, scalability, security, ...
 - ... arising from both immaturity & technology gaps
 - ... left to application programmers or systems
- Technology gaps requiring research
 - **Robustness of distributed systems**
 - Reducing cost & complexity
 - Information storage, sharing & distribution
 - Performance & scalability

- Systems built from autonomous stateful parts ...
 - ... must always reach globally consistent outcomes
 - ... despite failures, concurrency and errors
- Reaching globally consistent outcomes
 - Global business rules always satisfied at completion
 - Goods shipped & received; Full payment made & received
 - Always have to complete in acceptable distributed state
 - ACID transactions partly address just this problem
 - But can't be used in this new world...
 - ... depend on assumptions of trust and timeliness
 - Consistency part of ACID is left to application programmer
 - And no replacement theory and related technologies

- **Goal: robust distributed applications**
 - Distributed processes always complete
 - No deadlocks, unprocessed messages; all participants completed
 - Safe concurrent execution
 - No problems caused by races or contention for shared data
 - Terminate in acceptable distributed state
 - Participants finish in agreed compatible states
 - Completeness of components
 - Can always handle any message in any state

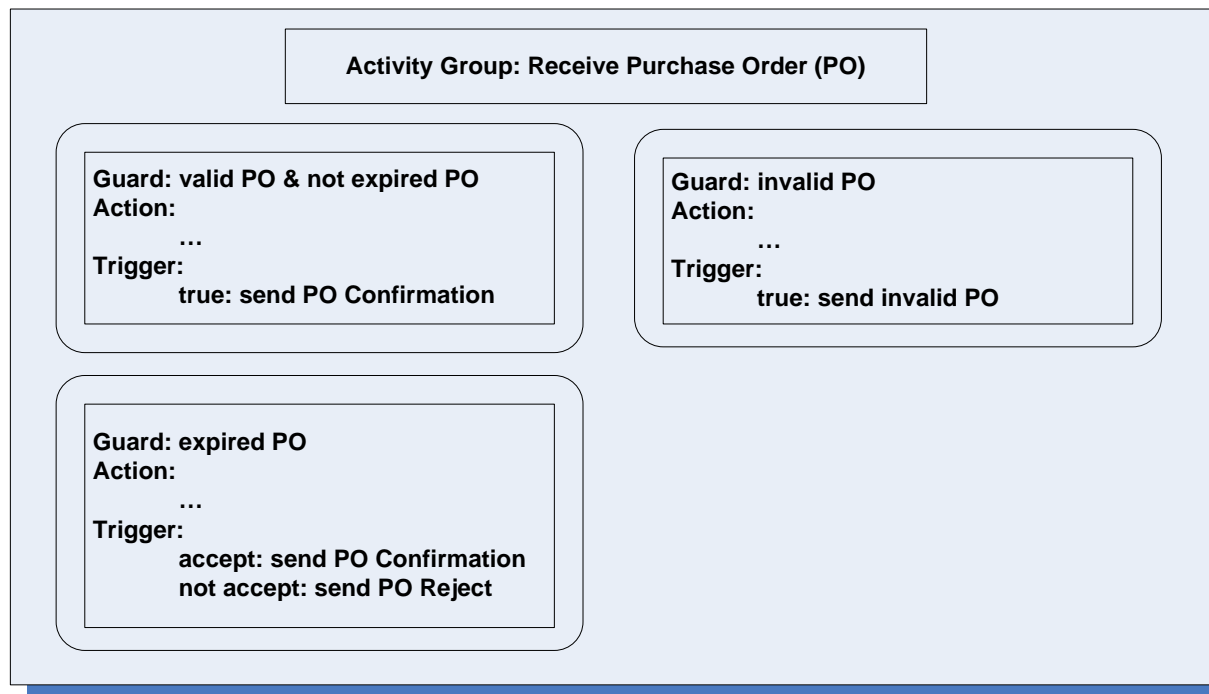
- **Service Description**
 - GAT – a service-oriented workflow language
 - Features for designing correct services
- **Service Contract**
 - SSDL rule-based protocol for defining stateful message exchanging behaviours
 - A tool to allow automatic checking that a service in GAT conforms to the SSDL rule based service contract
- **Application Protocol**
 - Relationships between consistent outcomes, states & protocols
 - Correctness properties of application protocols including consistent outcomes
 - A tool to allow automatic checking that service contracts participating in a distributed application satisfies the correctness properties

An Interaction Between Two Services



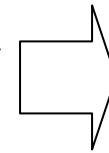
Event Based Programming Model (GAT)

- A process is defined by a set of “activity groups”
 - A set of activity groups
 - A activity group consists of activities
 - An activity consists of a guard, an action & a set of triggers group
 - Acceptable termination states



GAT Example

Group: ProcessPayment	
Event	Payment
Activity: ProcessFullPayment	
Guard	<i>received payment is equal to the remaining amount to be paid</i>
Action	<i>update balance; set PaidInFull to be True; construct an event PaidInFull;</i>
Triggers	(true) ⇒ PaidInFull
Activity: ProcessUnderpayment	
Guard	<i>received payment is less than the remaining amount to be paid</i>
Action	<i>update balance;</i>
Triggers	<i>// do nothing</i>
Activity: ProcessOverpayment	
Guard	<i>received payment greater than the remaining amount to be paid</i>
Action	<i>update balance; construct an event overPayment;</i>
Triggers	(true) ⇒ OverPayment



IN_EVENT: *Payment* // incoming event

ACTIVITY GROUP: ProcessPayment

ACTIVITY: ProcessFullPayment

GUARD: FullPayment(payment)

ACTION: UpdateInvoice(payment)

TRIGGERS: {true}

<INT>PaidInFull(payment)

ACTIVITY: ProcessUnderPayment

GUARD: UnderPayment(payment)

ACTION: UpdateInvoice(payment)

TRIGGERS: none

ACTIVITY: ProcessOverPayment

GUARD: OverPayment(payment)

ACTION: UpdateInvoice(payment)

TRIGGERS: {true}

<INT>OverPayment(payment)

EVENT: PaidInFull

ACTIVITY GROUP: SendReceipt

ACTIVITY: SendingReceipt

GUARD: true

ACTION: PrepareReceipt(payment)

TRIGGERS: {true} // outgoing event

<OUT>CustomerRemoteService.Receipt(payment)

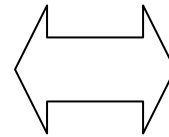
- **Uniform processing/outcomes**
 - No distinction between *normal* processing & *exception* handling
 - No inbuilt notion of return to an initial state
- **Resumption model**
 - Resumes normal cases after handling exceptional cases
- **Coverage**
 - Alternate actions are grouped together & conditions specify which of these should be executed
- **Protected action**
 - Guards specify explicitly when an *action* can execute
- **Access to state**
 - No hidden or implicit states (e.g, current position in the graph)
- **Response to non-occurrence**
 - Expected messages fail to arrive on time

- Define stateful messaging behaviour of a service
 - ... based on sent & received messages and their causal relationships
- A condition is associated with each message
 - ... is defined on already sent & received messages
- Expressed purely in terms of message states
- Possible to express consistency constraints using message states
- Simple and expressive enough for the purpose of defining service contracts for common services

www.ssd1.org (rules SSDL protocol framework)

Example Service Contract

```
<rls:rule>
  <ssdl:msgref ref="QuoteRequestMsg" direction="in"/>
  <rls:condition/> <!-- true -->
  <ssdl:msgref ref="QuoteMsg" direction="out"/>
  <rls:condition>
    <rls:and>
      <rls:not>
        <ssdl:msgref ref="QuoteRejectMsg" direction="out"/>
        <ssdl:msgref ref="CancelAcceptMsg" direction="out"/>
      </rls:not>
    </rls:and>
  </rls:condition>
  <ssdl:msgref ref="PurchaseOrderMsg" direction="in"/>
  <rls:condition>
    <rls:and>
      <rls:not>
        <ssdl:msgref ref="CancelAcceptMsg" direction="out"/>
      </rls:not>
    </rls:and>
  </rls:condition>
  .....
</rls:rule>
```



→ receive quote request;

← send quote if the quote request has not been rejected & has not been cancelled;

→ receive purchase order if the ordering process has not been cancelled.

Application Protocol Correctness



www.ict.csiro.au

- **Application protocol**
 - The set of all possible message exchange sequences between participating services
- **Correctness properties**
 - All the participating services have finished
 - No messages left to be processed
 - All of the services are in a consistent state.
- **Relationship between protocols and outcomes**
 - A path in the protocol always gives only one outcome, but an outcome may be possible from more than one path.
- **Example between merchant and customer services**
 - Both the customer and merchant services always finished
 - ...with no unprocessed messages &
 - ... with both services agreeing on whether the goods have been delivered and paid for.



- **Runtime checking**
 - Dynamic consistency checking protocol (Happiness Protocol)
 - At least we know when something has gone wrong
- **Static checking**
 - A tool for checking conformance between service description and service contract
 - Derivation of SSDL rules from a GAT specification
 - Verification of SSDL rules against a GAT specification
 - A tool to check correctness of application protocols
 - Converting SSDL rules to PROMELA
 - Converting correctness properties to LTL expression
 - Using model checkers (SPIN) for automatic verification of correctness properties

- Our overall goal
 - Making easier to build robust distributed applications from component services
 - ...including ensuring consistency at termination
- Outcomes
 - An event based programming model with guards and triggers
 - Rule-based framework for service contract description
 - Application protocol correctness for interacting services
 - Consistency checking protocols (happiness protocol)
- Other ongoing work
 - Isolation mechanisms based on patterns and protocols
 - Patterns & practices
 - Best practices to ensure consistency & robustness
 - Protocol patterns, use of WS-BA standard, ...