# Agents and the Semantic Web

Prof. Michael N. Huhns

University of South Carolina

http://www.cse.sc.edu/~huhns/

# Outline of this Tutorial

- 1: Introduction
- 2: Description
- 3: Engagement
- 4: Collaboration
- 5: Selection
- 6: Synthesis

# 1: Introduction

# The Web As Is

- Designed for people to get information
- Sources are independent and heterogeneous
- Limitations
  - HTML describes how things appear
  - HTTP is stateless
  - Processing is asynchronous client-server
  - No support for integrating information
  - No support for meaning and understanding

# Which Semantic Web?

- Version 1: Semantic Web as a Web of Data" (metadata from database schemas)

- Version 2: "Enrichment of the current Web" (metadata from NLP and automatic markup)

- Version 3: "Semantic Web as a Web of Services"

- Different use cases
- Different techniques
- Different users

# What is a Web Service?

- "… a piece of business logic accessible via the Internet using open standards…" (Microsoft)
- Encapsulated, loosely coupled, contracted software functions, offered via standard protocols over the web (Dest*i*Corp)
- A set of interfaces, which provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks (W3C)

Our working definition: A WS is functionality that can be *engaged* over the Web

# Viewpoints on Services

- *Networking:* a service is characterized by bandwidth and suchlike properties
- *Telecommunications:* Narrow telephony features such as caller ID and call forwarding, and basic connection services like narrowband versus broadband (itself of a few varieties)
- *Systems:* Services are for billing and storage and other key operational functions. These functions are often parceled up in the so-called operation-support systems
- *Web applications:* Services correspond to Web pages, especially those with forms or a programmatic interface thereto
- *Wireless:* Wireless versions of the Web, but also things like messaging, as in the popular short message service (SMS)

If there is agreement here, it is that a service is a capability that is provided and exploited, often but not always remotely

# Open Environments: Characteristics

- Cross enterprise boundaries or administrative domains
- Comprise autonomous resources that
  - Involve loosely structured addition and removal
  - Range from weak to subtle consistency requirements
  - Involve updates only under local control
  - Frequently involve nonstandard data
- Have intricate interdependencies

# Autonomy (Usage)

Independence of business partners (users)

- Political reasons
  - Ownership of resources
  - Control, especially of access privileges
  - Payments
- Technical reasons
  - Opacity of systems with respect to key features, e.g., precommit

# Heterogeneity (Construction)

Independence of component designers and system architects

- Political reasons
  - Ownership of resources
- Technical reasons
  - Conceptual problems in integration
  - Fragility of integration
  - Difficult to guarantee behavior of integrated systems

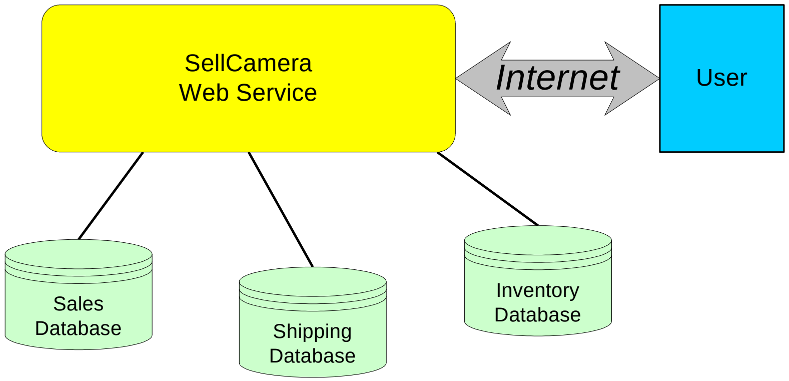
Best not to assume homogeneity

# Dynamism (Configuration)

- Independence of system administrators
- Needed because the parties change
  - Architecture and implementation
  - Behavior
  - Interactions
- Make configurations dynamic to improve service quality and maintain flexibility

# Simple B2C Web Service Example

Suppose you want to sell cameras over the Web, debit a credit card, and guarantee next-day delivery



- Your application must
  - update sales database
  - debit the credit card
  - send an order to the shipping department
  - receive an OK from the shipping department for next-day delivery
  - update an inventory database
- Problems: Some steps complete but not all

# Database Approach (Closed)

- Transaction processing (TP) monitors (such as IBM’s CICS, Transarc’s Encina, BEA System’s Tuxedo) can ensure that all or none of the steps are completed, and that systems eventually reach a consistent state
- But what if the user’s modem is disconnected right after he clicks on OK?  Did the order succeed? What if the line went dead before the acknowledgement arrives? Will the user order again?

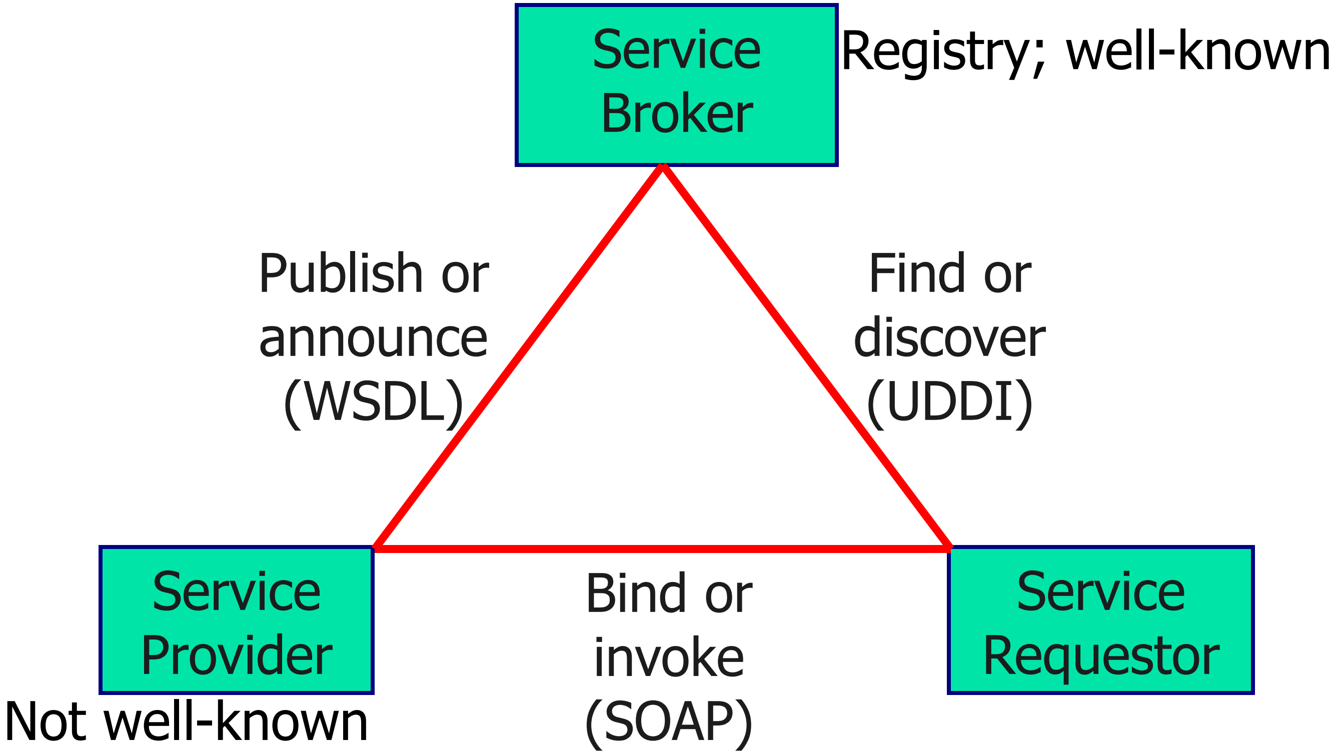
The TP monitor cannot get the user into a consistent state!

# Approach for Open Environment

- Server application could send email about credit problems, or detect duplicate transactions
- Downloaded applet could synchronize with server after broken connection was restored, and recover transaction; applet could communicate using http, or directly with server objects via CORBA/IIOP or RMI
- If there are too many orders to process synchronously, they could be put in a message queue, managed by a Message Oriented Middleware server (which guarantees message delivery or failure notification), and customers would be notified by email when the transaction is complete

*The server behaves like an agent!*

# Web Services: Basic Architecture



# SOAP: Simple Object Access Protocol

- Used to exchange messages via HTTP, SMTP, and SIP (Session Initiation Protocol for Internet telephony)
- Originally designed for remote-procedure calls (RPC)
- Works through firewalls on port 80
- Character-based, so easy to encrypt/decrypt and thus easy to secure
- Inefficient due to character, not binary, data and large headers
- Does not describe bidirectional or n-party interaction

# WSDL: Web Services Description Language

- Describes a programmatic interface to a Web service, including
  - Definitions of data types
  - Input and output message formats
  - The operations provided by the service
  - Network addresses
  - Protocol bindings

# Directory Services

- Enable applications, agents, Web service providers, Web service requestors, people, objects, and procedures to locate each other
- White pages – entries found by name
- Yellow pages – entries found by characteristics and capabilities
- A basic directory might be a simple database (passive) or a broker/facilitator (active, that provides alerts and recruits participants)
- UDDI – both white pages and yellow pages, but passive

# UDDI: Universal Description, Discovery, and Integration

- UDDI is a Web service that is based on SOAP and XML
- UDDI registers
  1. *tModels:* technical descriptions of a service’s behavior
  2. *businessEntities:* describes the specifications of multiple tModels

# Basic Profile (BP 1.0)

- The Web Services Interoperability Organization (WS-I) has specified the following Basic Profile version 1.0:
  - SOAP 1.1
  - HTTP 1.1
  - XML 1.0
  - XML Schema Parts 1 and 2
  - UDDI Version 2
  - WSDL 1.1

# Example of Current SOA Success

Amazon.com:

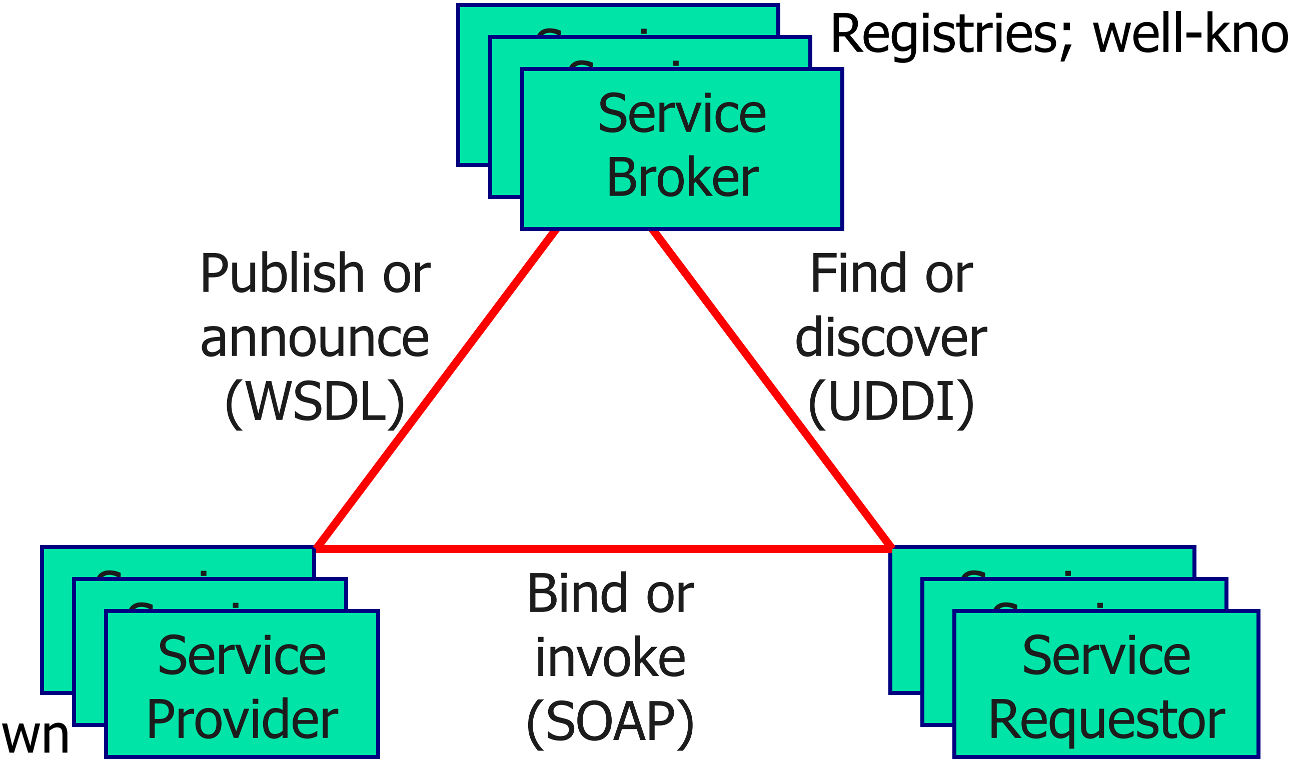
- Converted monolithic application into 100’s of services
- Applications for customer service, selling, Amazon’s Web pages, and hosted applications invoke the services as needed

Enables unanticipated 3rd-party applications:

- Shopping with a camera phone (uses reviewing service, comparable product service, and current price service)
- Mechanical Turk is a Web service that allows developers to post questions to a large group of people to gain their insight on a particular issue. Simple Storage Service (S3) and Elastic Compute Cloud (EC2) are Web services that let Amazon sell excess storage and excess compute capacity, respectively, to third-party developers
- FBA (Fulfillment by Amazon) makes Amazon’s warehouse, customer service, and pick, pack, and ship machinery available to sellers
- Already, an AJAX/S3 Wiki uses S3 for code and data

# A Source for Research Ideas:
# *What Are the Limitations of the WS Triangle?*

## Consider each vertex and edge:



Registries; well-known

Not well-known

# Still Missing…

- How to discover appropriate services?
- How to compose services dynamically?
- How to support programming-in-the-large?
- How to engineer for functionality?
- How to engineer for maintainability?
- How to scale for survivability?
- How to replicate for robustness?

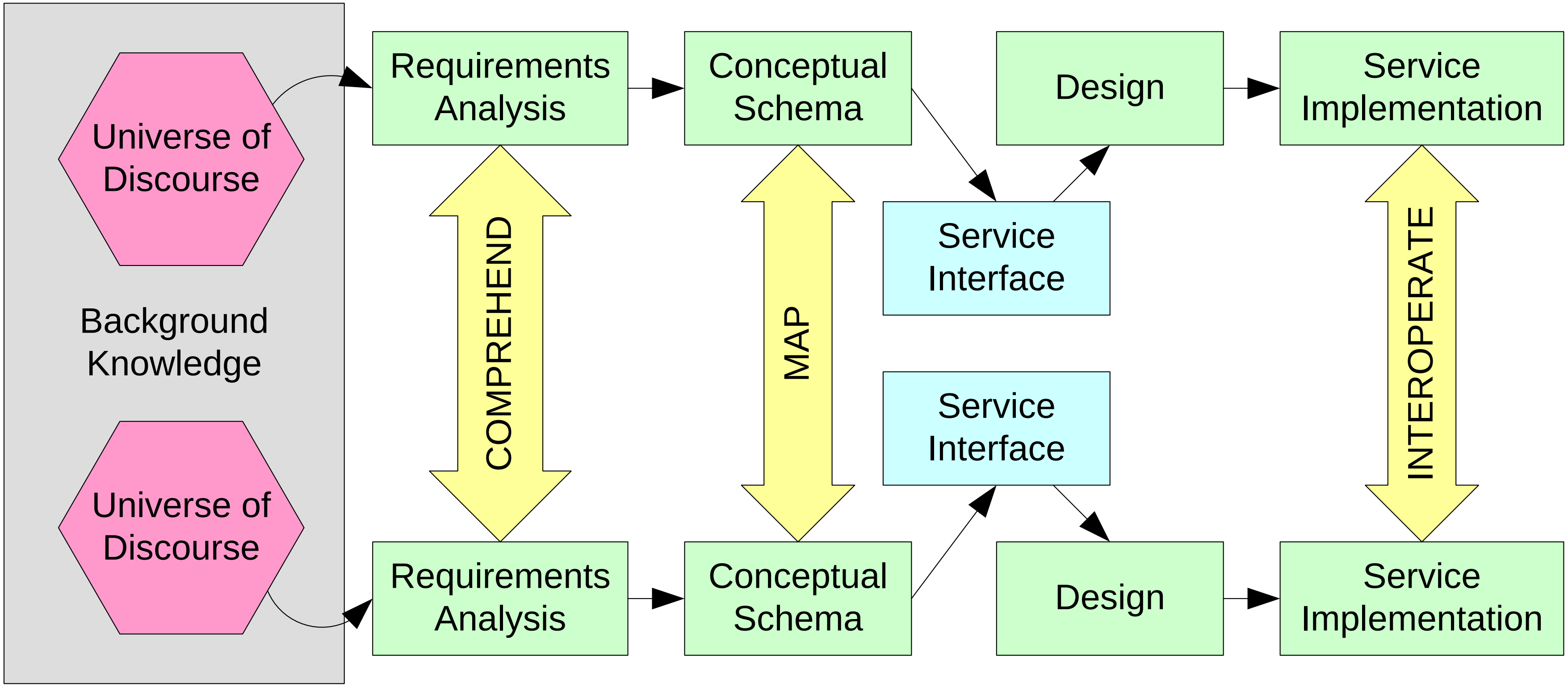
© Michael N. Huhns

# Why Agents for Web Services?

- Convergence between the two
- Similarities in key features
  - Dynamism => autonomy
  - Openness and compliance => ability to enter into and obey contracts
  - Trustworthiness => ethical behavior and social models of reputation
- Trend: Web services, formerly like objects, are becoming interactive (Web 3.0)

# 2: Description

# Modeling and Composing Services



# Dimensions of Abstraction: 1

Information resources are associated with abstractions over different dimensions. These may be thought of as constraints that must be discovered and represented

- Data
  - domain specifications
  - value ranges, e.g., Price ≥ 0
  - allow/disallow null values

# Dimensions of Abstraction: 2

- Structure
  - schemas and views, e.g., securities are stocks
  - specializations and generalizations of domain concepts, e.g., stocks are a kind of liquid asset
  - value maps, e.g., S&P A+ rating corresponds to Moody’s A rating
  - semantic data properties, sufficient to characterize the value maps, e.g., some stock price databases consider daily averages; others closing prices
  - cardinality constraints
  - integrity constraints, e.g., each stock must have a unique SEC identifier

# Dimensions of Abstraction: 3

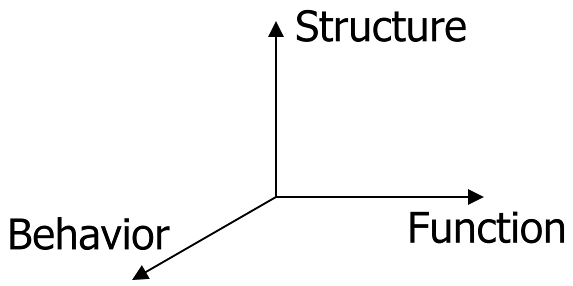
- Process
  - procedures, i.e., how to process information, e.g., how to decide what stock to recommend
  - preferences for accesses and updates in case of data replication (based on recency or accuracy of data)
  - preferences to capture view update semantics
  - contingency strategies, e.g., whether to ignore, redo, or compensate
  - contingency procedures, i.e., how to compensate transactions
  - flow, e.g., where to forward requests or results
  - temporal constraints, e.g., report tax data every quarter

# Dimensions of Abstraction: 4

- Policy
  - security, i.e., who has rights to access or update what information? (e.g., customers can access all of their accounts, except blind trusts)
  - authentication, i.e., a sufficient test to establish identity (e.g., passwords, retinal scans, or smart cards)
  - bookkeeping (e.g., logging all accesses)

# Description Dimensions for a Web Service

| **Description of Web Service** | **Current Representation Standard/Technique** |
|---|---|
| Structure: syntactic | WSDL |
| Structure: semantic | WSDL-S, OWL-S, WSMO |
| Function | WSDL-S, OWL-S, WSMO |
| Behavior (including QoS) | *Agile Unit Testing* |



© Michael N. Huhns

# Behavioral Constraints for Stock Quote Service

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<scenario>
   <parameter>
       <name>stockName</name>
       <value>IBM</value>
   </parameter>
</scenario>
<constraints>
<constraint>
   <id>B1</id>
   <parameter>price</parameter>
   <type>double</type>
   <range>
       <min>60</min>
       <max>110</max>
   </range>
   <relevance>2</relevance>
</constraint>
<constraint>
   <id>B2</id>
   <parameter>responseTime</parameter>
   <type>integer</type>
   <range>
       <min>0</min>
       <max>5000</max>
   </range>
   <relevance>1</relevance>
</constraint>
</constraints>
```

# Java Client for Stock Purchase

```
package com.invesbot.ws;
public class Client {
 public double getQuote(String symbol) {
   double priceValue = 0.0;
   try {
      StockQuoteLocator service = new StockQuoteLocator();
      StockQuoteSoap quoteService = service.getStockQuoteSoap12();
      priceValue = quoteService.getQuote(symbol);
   } catch (Exception e) {
      System.out.println(e.getMessage());
      e.printStackTrace();
   }
   return priceValue;
   }
}
```

# JUnit Test for Stock Quote Behavior

```
public class StockQuoteTest {
 com.invesbot.ws.Client client;
 double price;
 int responseTime;
 @Before public void setUp() {
   price = 0.0;
   responseTime = 0;
   client = new com.invesbot.ws.Client();
 }
 @Test public void testPrice() {
   Assert.assertEquals(0.0, price);
   price = client.getQuote(“IBM”);
   Assert.assertTrue(price >= 60.0 && price <= 110.0);
 }
@Test public void testResponseTime() {
   Date d1 = new Date();
   Assert.assertEquals(0, responseTime);
   price = client.getQuote(“IBM”);
   Date d2 = new Date();
   long responseTime = d2.getTime() – d1.getTime();
   Assert.assertTrue(responseTime >= 0 && responseTime <= 5000);
 }
 @After public void tearDown() {
   price = 0.0;
   responseTime = 0;
 }
 public static junit.framework.Test suite() {
   return new JUnit4TestAdapter(StockQuoteTest.class);
 }
}
```

© Michael N. Huhns

# Ontology

- A specification of a conceptualization or a set of knowledge terms for a particular domain, including
  - the vocabulary
  - the semantic interconnections
  - some simple rules of inference and logic
- Some representation languages for ontologies:
  - Uniform Modeling Language (UML)
  - Resource Description Framework Language Schema (RDFS)
  - Web Ontology Language (OWL)
- Some ontology editors: Protégé, Webonto, OilEd

# Exercise: Which Conceptualization Has More Expressive Power?

- awg22SolidBlueWire(ID5)
- blueWire(ID5, AWG22, Solid)
- solidWire(ID5, AWG22, Blue)
- wire(ID5, AWG22, Solid, Blue)
- wire(ID5)^size(AWG22)^type(solid)^color(Blue)

# RDF Statements

- An RDF *statement* (aka *triple*) mimics a simple sentence in natural language:
  - Subject (a resource – known by a URI)
  - Object (a resource or a value)
  - Predicate (a property – known by a URI)
- Uses XML namespace syntax
- Special namespace defined by the standard – typically called rdf

# RDF Types and Example

- Collections (containers)
  - rdf:Bag
  - rdf:Sequence
  - rdf:Alternatives

- RDF Example

```
<?xml version='1.0' encoding='UTF-8'?>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:dc="http://purl.org/dc/elements/1.1/">
   <rdf:Description rdf:about="http://www.wiley.com/SOC">
     <dc:title>Service-Oriented Computing</dc:title>
     <dc:creator>Munindar</dc:creator>
     <dc:creator>Michael</dc:creator>
     <dc:publisher>Wiley</dc:publisher>
   </rdf:Description>
</rdf:RDF>
```

# Reification of Statements

- *Reify:* to make referenceable
  - Needed to quote statements (e.g., to agree or disagree with them); assert modalities
  - Make a statement into a resource; then talk about it
  - rdf:Statement is the class whose rdf:type the given statement (object) is; additional properties such as rdf:subject, rdf:object, and rdf:predicate

# RDF Schema

- Analogous to an object-oriented type system built on top of RDF.  Defines
  - rdfs:Class, rdfs:subClassOf
  - rdfs:Resource, rdfs:Literal
  - rdfs:Property, rdfs:subPropertyOf
  - rdfs:range, rdfs:domain
  - rdfs:label, rdfs:comment, rdfs:seeAlso

# Web Ontology Language (OWL)

- Provides the ability to specify classes and properties in a form of description logic with the terms in its expressions related using Boolean operators analogous to *and*, *not*, and *or*, as well as the constraints on various properties
- OWL has 3 dialects: OWL Full, OWL DL, and OWL Lite

# Subclasses and Properties

```
<owl:Class rdf:ID="Mammal">
  <rdfs:subClassOf rdf:resource="#Animal"/>
  <owl:disjointWith rdf:resource="#Reptile"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="hasParent">
  <rdfs:domain rdf:resource="#Animal"/>
  <rdfs:range rdf:resource="#Animal"/>
</owl:ObjectProperty>
```

# Constructing OWL Classes

- Explicitly (as the examples above) or
- Anonymously, using
  - intersectionOf, unionOf, complementOf, someValuesFrom, allValuesFrom, minCardinality, and maxCardinality, e.g.,

```
<owl:Class rdf:ID='SugaryBread'>
  <owl:intersectionOf rdf:parseType='Collection'>
    <owl:Class rdf:about='#Bread'/>
    <owl:Class rdf:about='#SweetFood'/>
  </owl:intersectionOf>
</owl:Class>
```

# OWL Restrictions

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasFather"/>
  <owl:maxCardinality
   rdf:datatype="xsd:nonNegativeInteger">
    1
  </owl:maxCardinality>
</owl:Restriction>

<owl:Restriction>
      <owl:onProperty rdf:resource='#bakes'/>
      <owl:someValuesFrom rdf:resource='#Bread'/>
</owl:Restriction>
```

# OWL Axioms

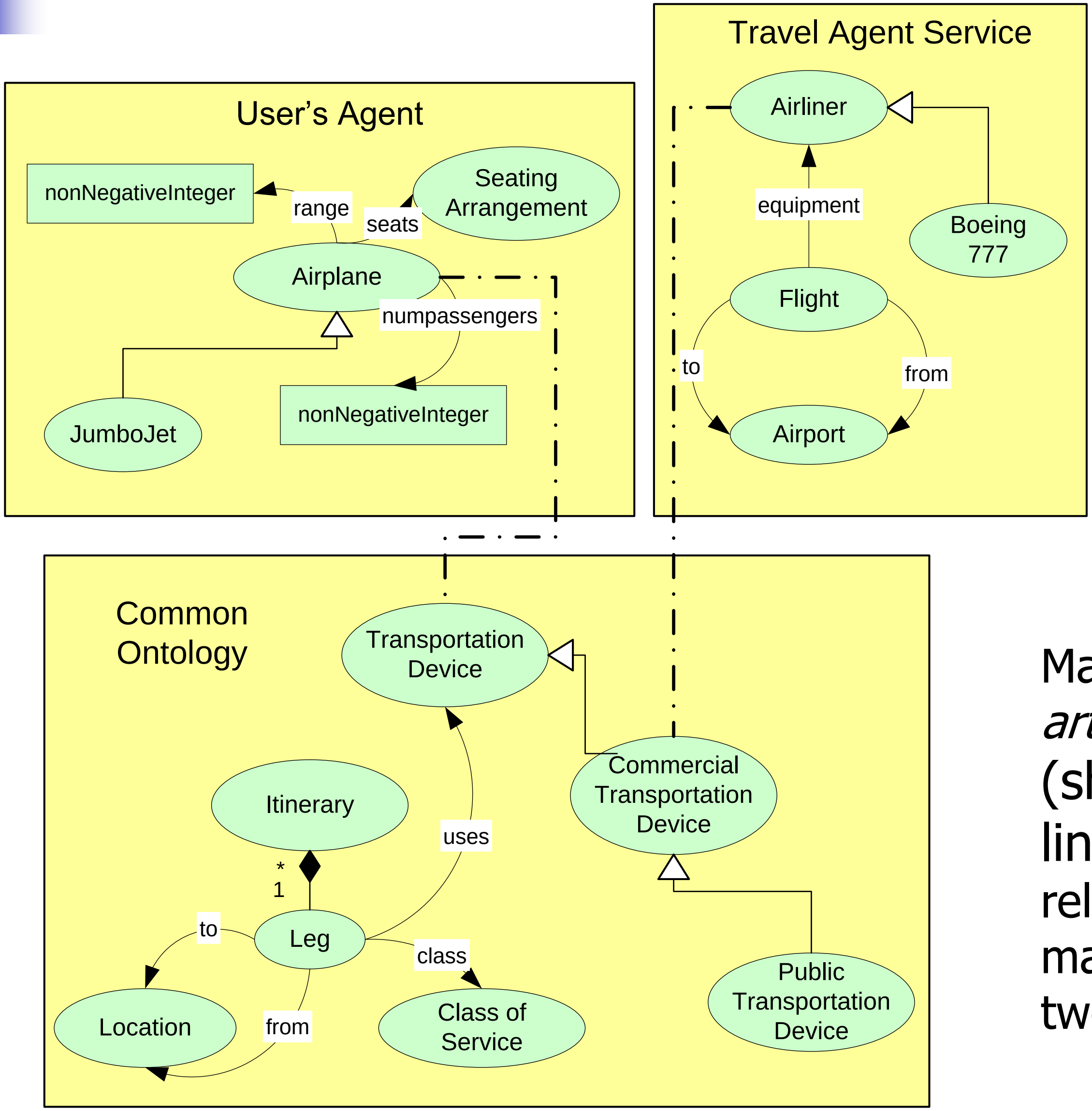
```
<owl:AllDifferent>
  <owl:distinctMembers rdf:parseType='Collection'>
   <ex:Country rdf:ID='Russia'/>
   <ex:Country rdf:ID='India'/>
   <ex:Country rdf:ID='USA'/>
  <owl:distinctMembers/>
</owl:AllDifferent>

<ex:Country rdf:ID='Iran'/>
<ex:Country rdf:ID='Persia'>
  <owl:sameIndividualAs rdf:resource='#Iran'/>
</ex:Country>
```

# Common Ontologies

- A shared representation is essential to successful communication and coordination
  - For humans: physical, biological, and social world
  - For computational agents: common ontology (terms used in communication)
- Representative efforts are
  - Cyc (and Opencyc)
  - WordNet (Princeton)
  - Several *upper-level* ontologies, e.g., IEEE SUMO and MILO

# Ontologies and Articulation Axioms



Mappings, i.e., *articulation axioms* (shown by dotted lines), describe the relationships between matching concepts in two ontologies

# Mappings among Ontologies

- Term-to-term (one-to-one), e.g.,

  hookupWire$_{O1}$ = wire$_{O2}$

- Many-to-one, e.g.,

  solidWire$_{O1}$(x, size, color) Æ strandedWire$_{O1}$(x, size, color)
  = wire$_{O2}$(x, size, color, (Stranded|Solid))

- Many-to-many, e.g.,

  solidBlueWire$_{O1}$(x, size) Æ
  solidRedWire$_{O1}$(x, size) Æ
  strandedBlueWire$_{O1}$(x, size) Æ
  strandedRedWire$_{O1}$(x, size)
  =
  solidWire$_{O2}$(x, size, (Red|Blue)) Æ
  strandedWire$_{O2}$(x, size, (Red|Blue))

# 3: Engagement

# Transactions

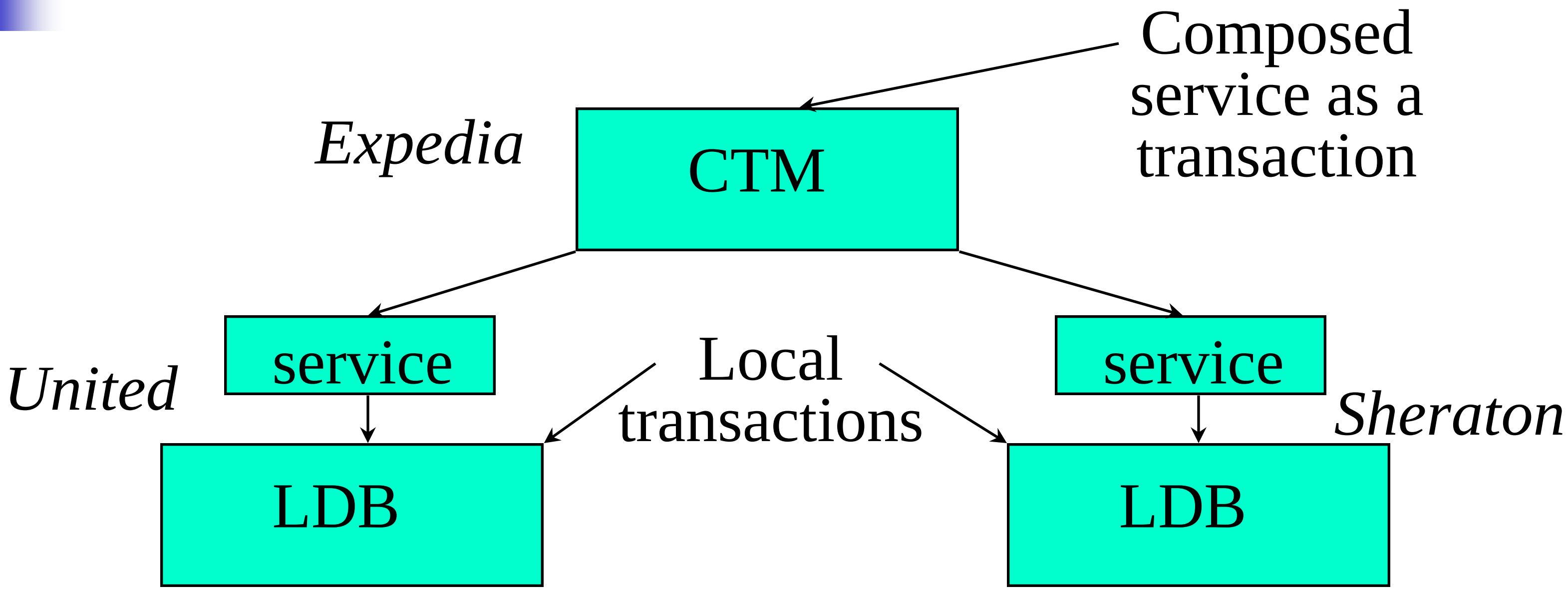
- A transaction is a computation (i.e., program *in execution*) that accesses and possibly modifies a DB:
  - Can be interleaved with other transactions
  - But guarantees certain properties

The purpose of the transaction concept is to avoid the problems that may arise from interleaving

# ACID Properties

- These formalize the notion of one operation
  - (Failure) Atomicity—all or none—if failed then no changes to DB or messages
  - Consistency—don't violate DB integrity constraints: execution of the op is correct
  - Isolation (Atomicity)—partial results are hidden
  - Durability—effects (of transactions that "happened" or committed) are forever

# Transactions over Composed Services



Two main kinds of service agreements are possible:

- *execution*, e.g., LDB retains full control on execution even if in conflict with CTM
- *communication*, e.g., LDB decides what (control) information to release

# Compositional Serializability

Transactions throughout the system should be serializable.

- CTM ensures that the composed transactions are serializable.
- This doesn't guarantee compositional serializability, because of *indirect* conflicts:
  - CTM does T1: r1(a); r1(c)
  - CTM does T2: r2(b); r2(d)
  - LDB1 does T3: w3(a); w3(b)
  - LDB2 does T4: w4(c); w4(d)
  - Since T1 and T2 are read-only, they are serializable.
  - LDB1 sees S1=r1(a); c1; w3(a); w3(b); c3; r2(b); c2
  - LDB2 sees S2=w4(c); r1(c); c1; r2(d); c2; w4(d); c4
  - Each LDB has a serializable schedule; yet jointly they put T1 before and after T2
- Notice we would have lots of potential compositions, so the problem is worse.

# Achieving Business Interoperation

The parties must

- Know each other’s identity and location (presumes suitable directories)
- Agree on the low-level transport protocols and encoding formats
- Agree on the syntax and semantics of documents to be exchanged
- Agree on their expectations about when different documents will be sent and received
  - This specification is termed a *business protocol*
  - An instance of a business protocol is a *conversation* (but sometimes the term is used to mean protocol – watch out!)

# Process Abstractions

*Orchestration*: A process is a partial order of actions (activity graph, script) under the control of a central conductor; akin to a workflow [Global; central]

*Choreography*: A process is an exchange of messages among participants; akin to a conversation as described by WSCL and WS-CDL [Global; distributed]

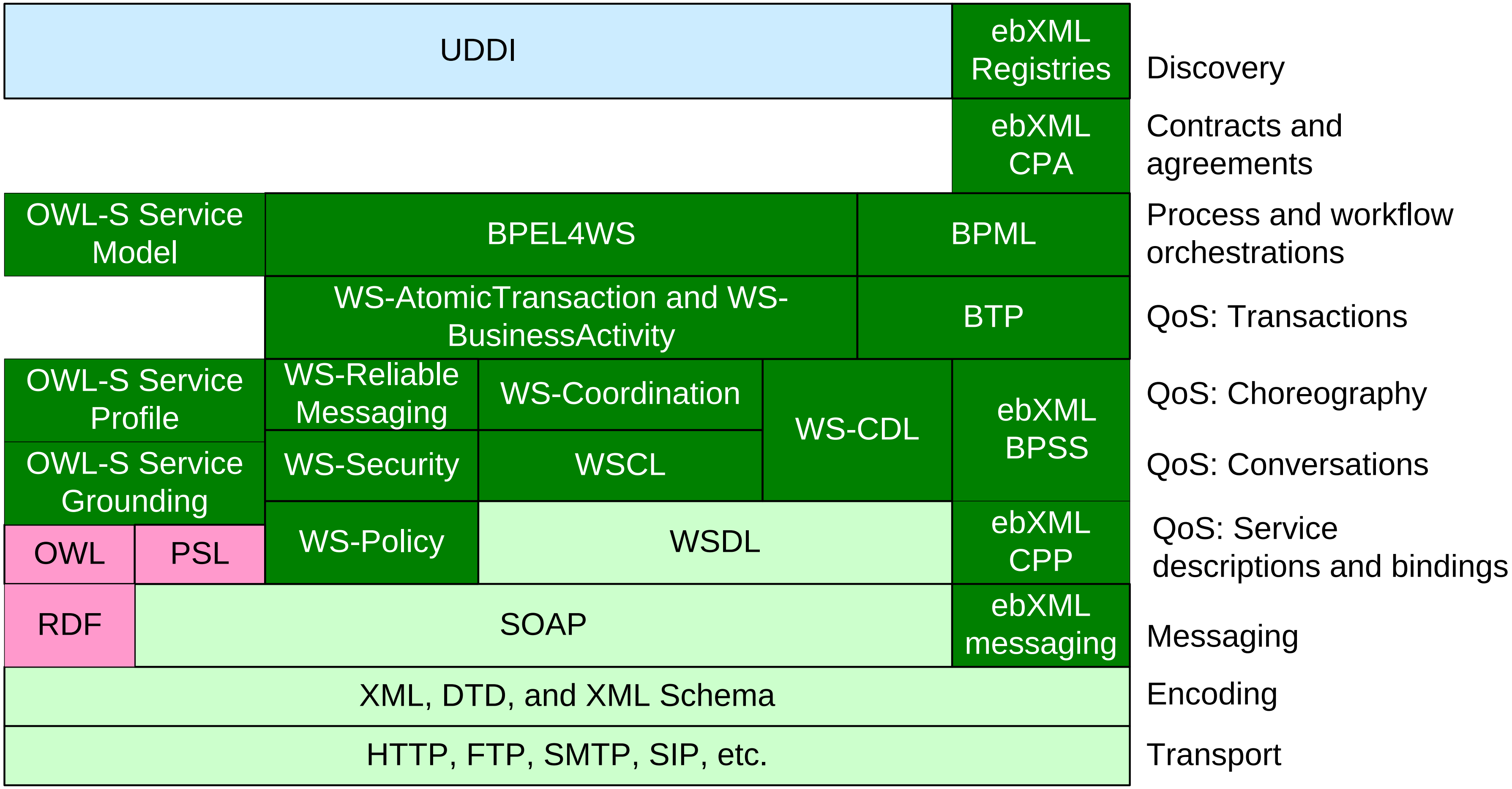
*Collaboration*: A process is a joint set of activities among business partners [Local; distributed]

*Workflow*: a narrower concept than a process, which emphasizes control flows and data flows from a central perspective; usually tool-specific

# WS-CDL

- WS-CDL describes the external observable behavior of multiple participants from a global model perspective
- Based on pi-calculus

# Processes and Workflows



UDDI | ebXML Registries — Discovery

ebXML CPA — Contracts and agreements

OWL-S Service Model | BPEL4WS | BPML — Process and workflow orchestrations

WS-AtomicTransaction and WS-BusinessActivity | BTP — QoS: Transactions

OWL-S Service Profile | WS-Reliable Messaging | WS-Coordination | WS-CDL | ebXML BPSS — QoS: Choreography

OWL-S Service Grounding | WS-Security | WSCL — QoS: Conversations

OWL | PSL | WS-Policy | WSDL | ebXML CPP — QoS: Service descriptions and bindings

RDF | SOAP | ebXML messaging — Messaging

XML, DTD, and XML Schema — Encoding

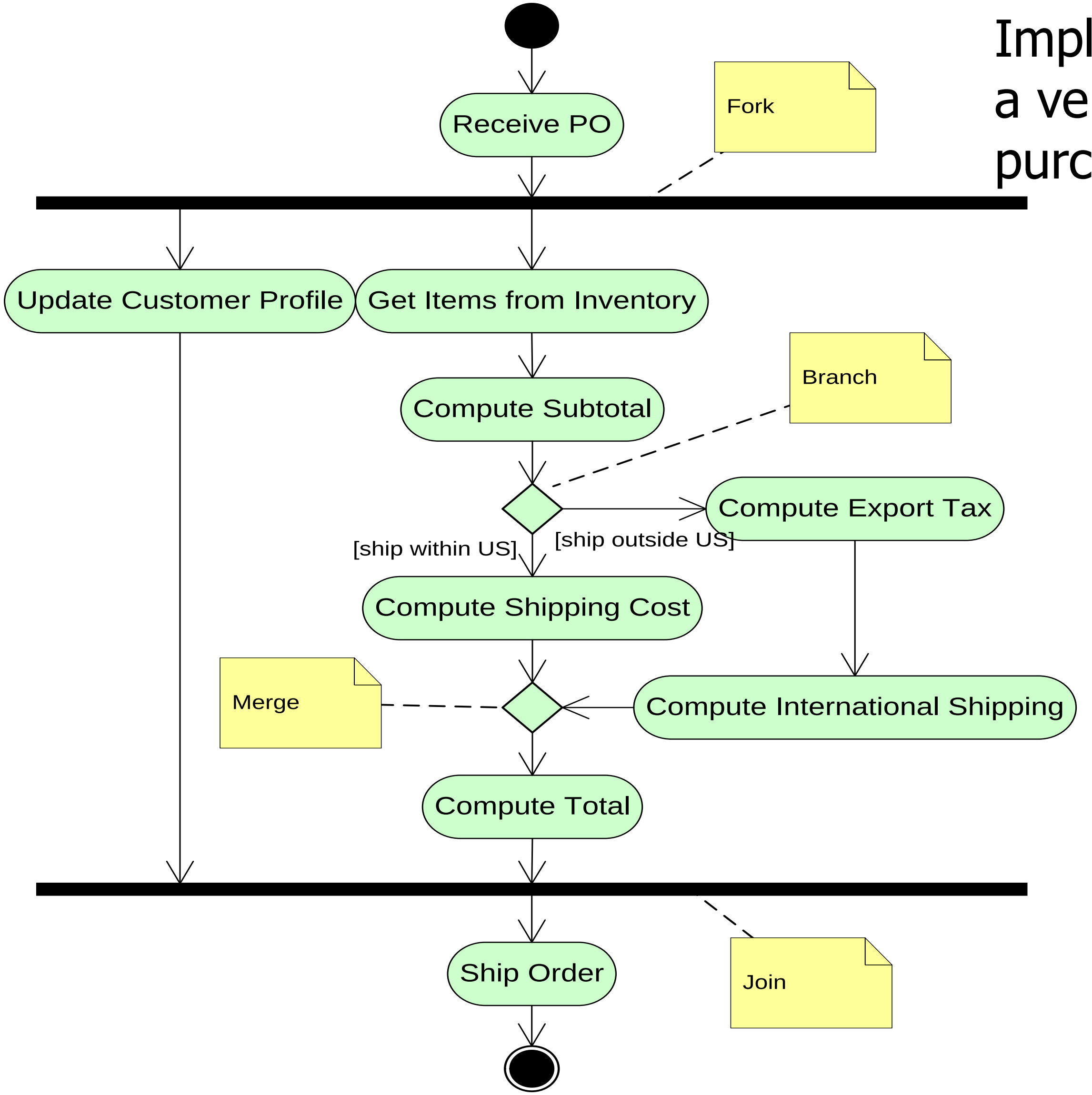
HTTP, FTP, SMTP, SIP, etc. — Transport

# Describing Dynamics with UML

UML provides graphical constructs that can be used to describe (1) actions and activities, and (2) temporal precedence and control flows. The allowable control constructs are

- *Sequence:* a transition from one activity to the next in time
- *Branch:* a decision point among alternative flows of control
- *Merge:* where two or more alternative flows of control rejoin
- *Fork:* a splitting of a flow of control into two or more concurrent and independent flows of control
- *Join:* a synchronization of two or more concurrently executing flows of control into one flow

# UML Activity Diagram

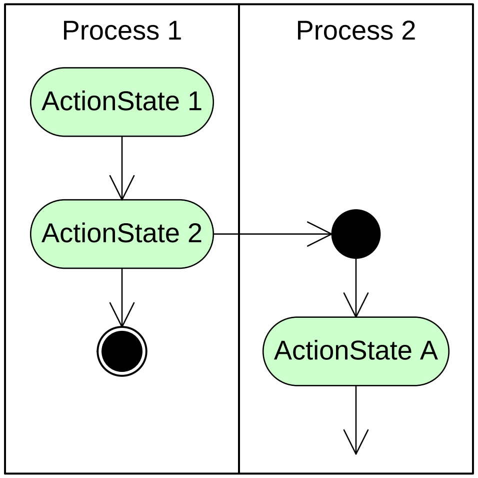


Implementation of a vendor’s purchase process

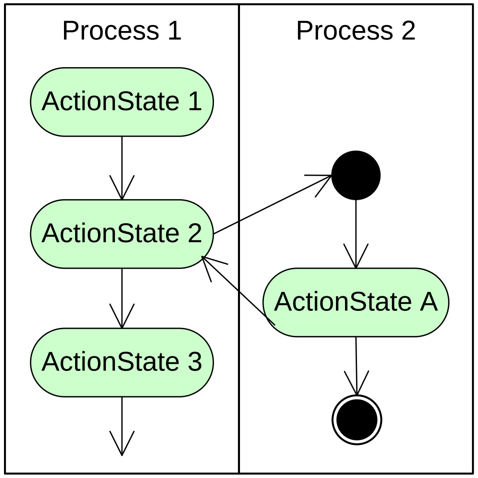
# Flow Interoperability Patterns

- Chained
- Nested
- Synchronized

Chained Interoperability

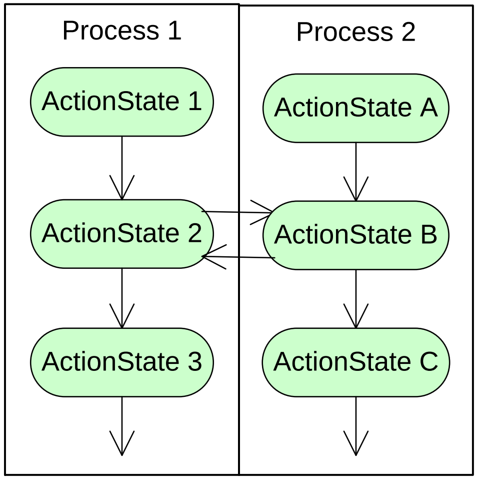


Nested Interoperability

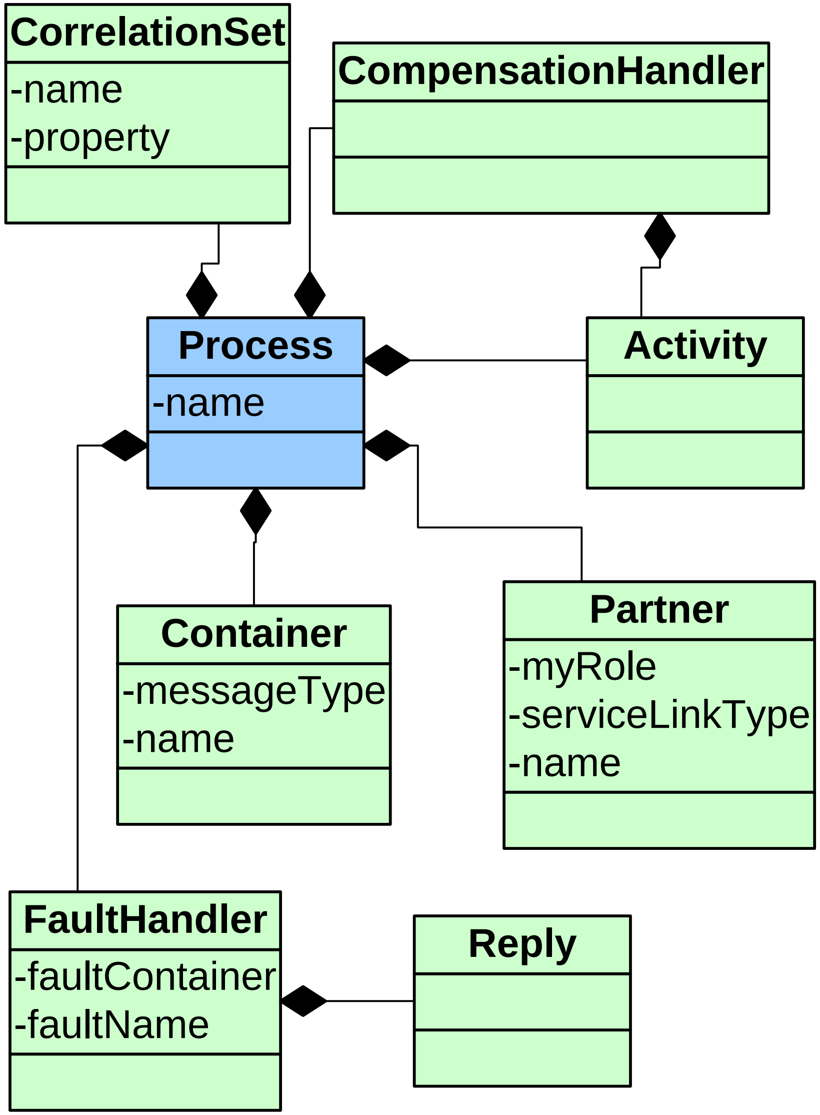


- What guarantees would you obtain from each?
- How would you accommodate exceptions in each?

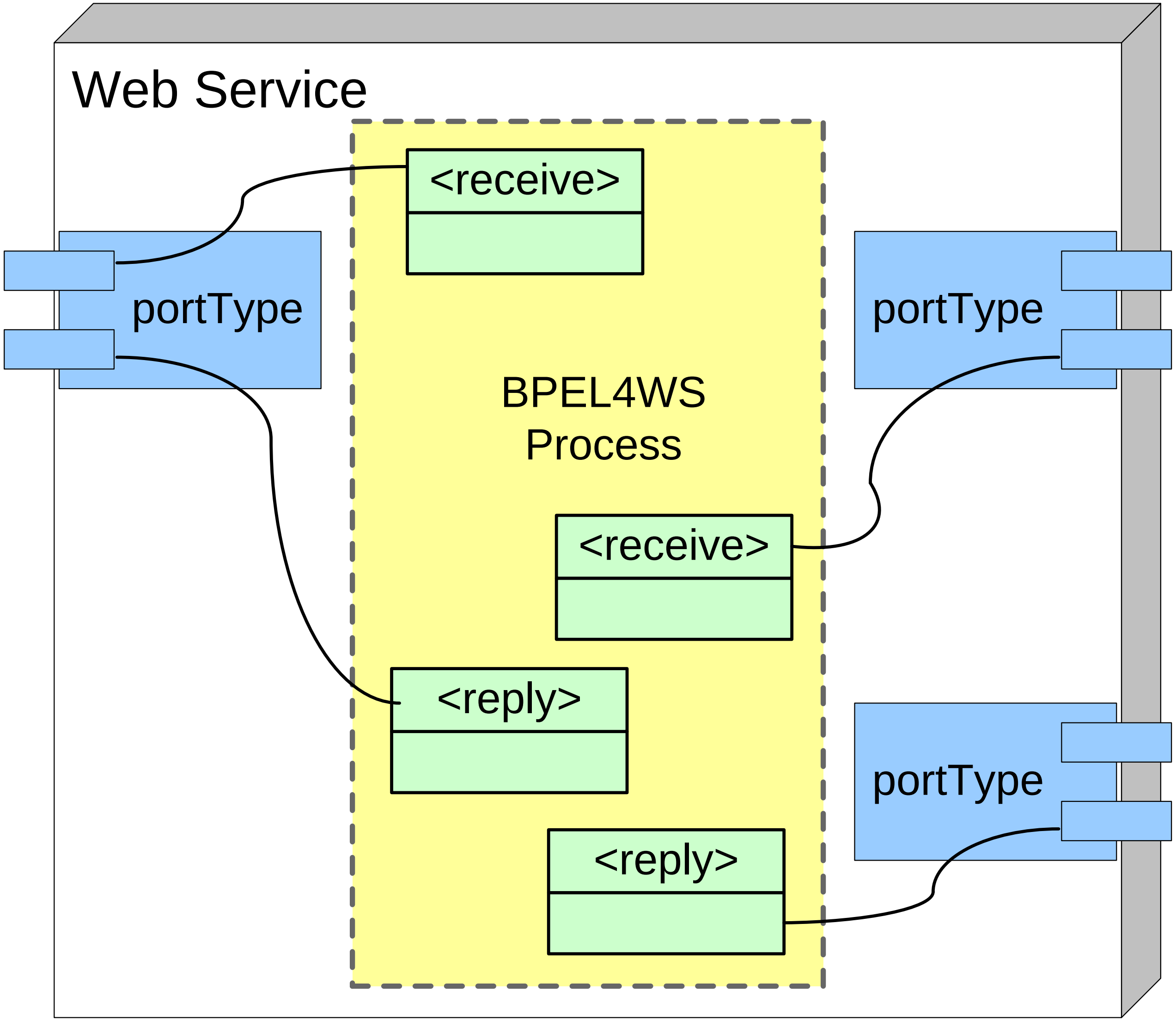
Synchronized Interoperability



# BPEL4WS Metamodel



# A BPEL4WS process is a composite Web service with a WSDL description



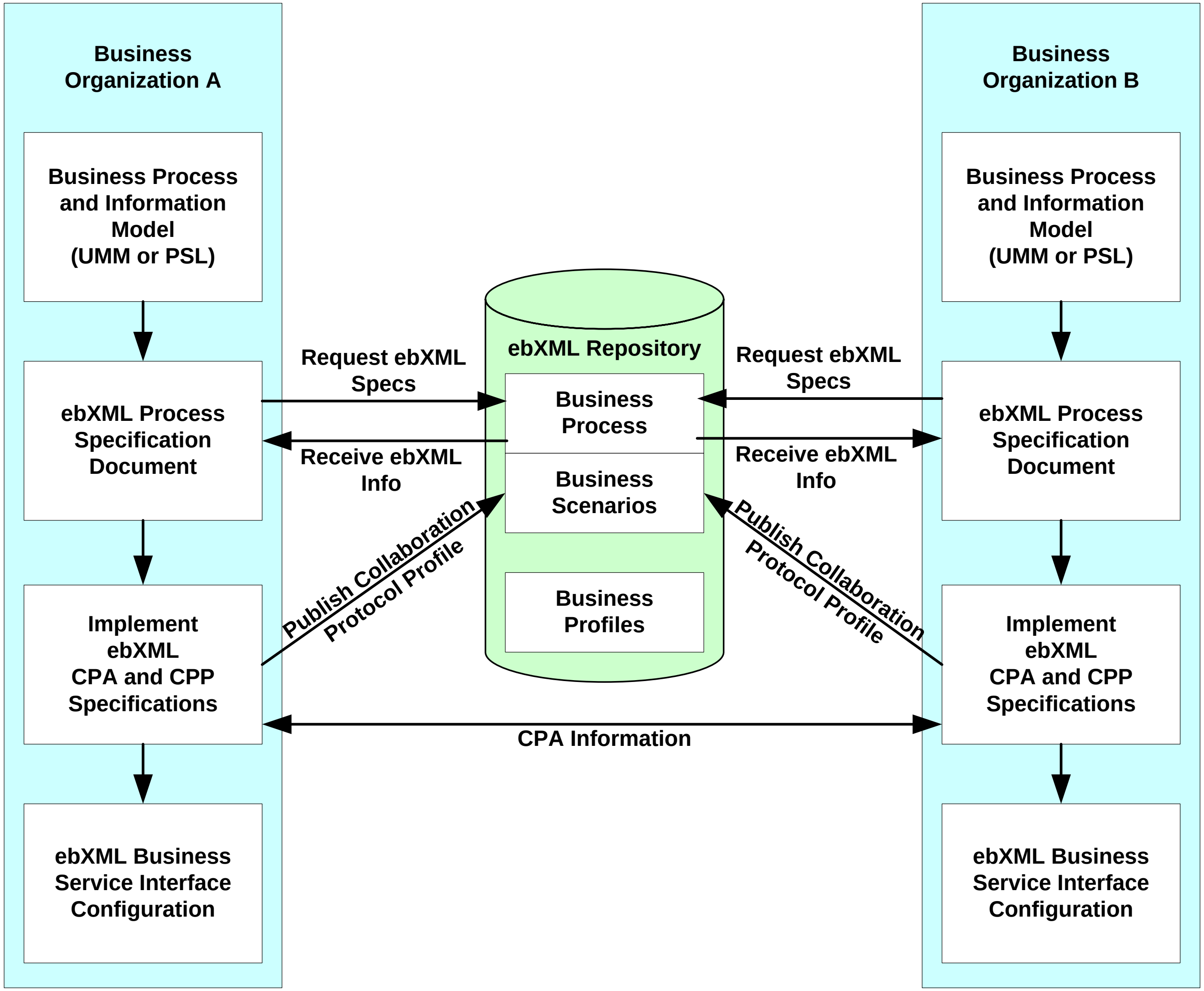
# Electronic Business Extensible Markup Language (ebXML)

- Established by UN-CEFACT (United Nations Centre for Trade Facilitation and Electronic Business) and OASIS (Organization for the Advancement of Structured Information Standards)
- Provides specifications to define standard business processes, exchange business messages and enter into trading agreements
- Motivations:
  - Global standard for companies of all sizes
  - Automate finding business partners

# ebXML Vocabulary

- Unified Modeling Methodology (UMM)
  - Specialized UML for Business Processes
- Collaboration Protocol Profile (CPP)
  - Describes a business's profile, i.e., which business processes it supports, its roles in those processes, the messages exchanged, and the transport mechanism for the messages (e.g., HTTPS)
- Collaborative Partner Agreement (CPA)
  - Intuitively, like an intersection of two CPPs
  - Technical agreement between two or more partners
  - May be legally binding

# Design of an ebXML System

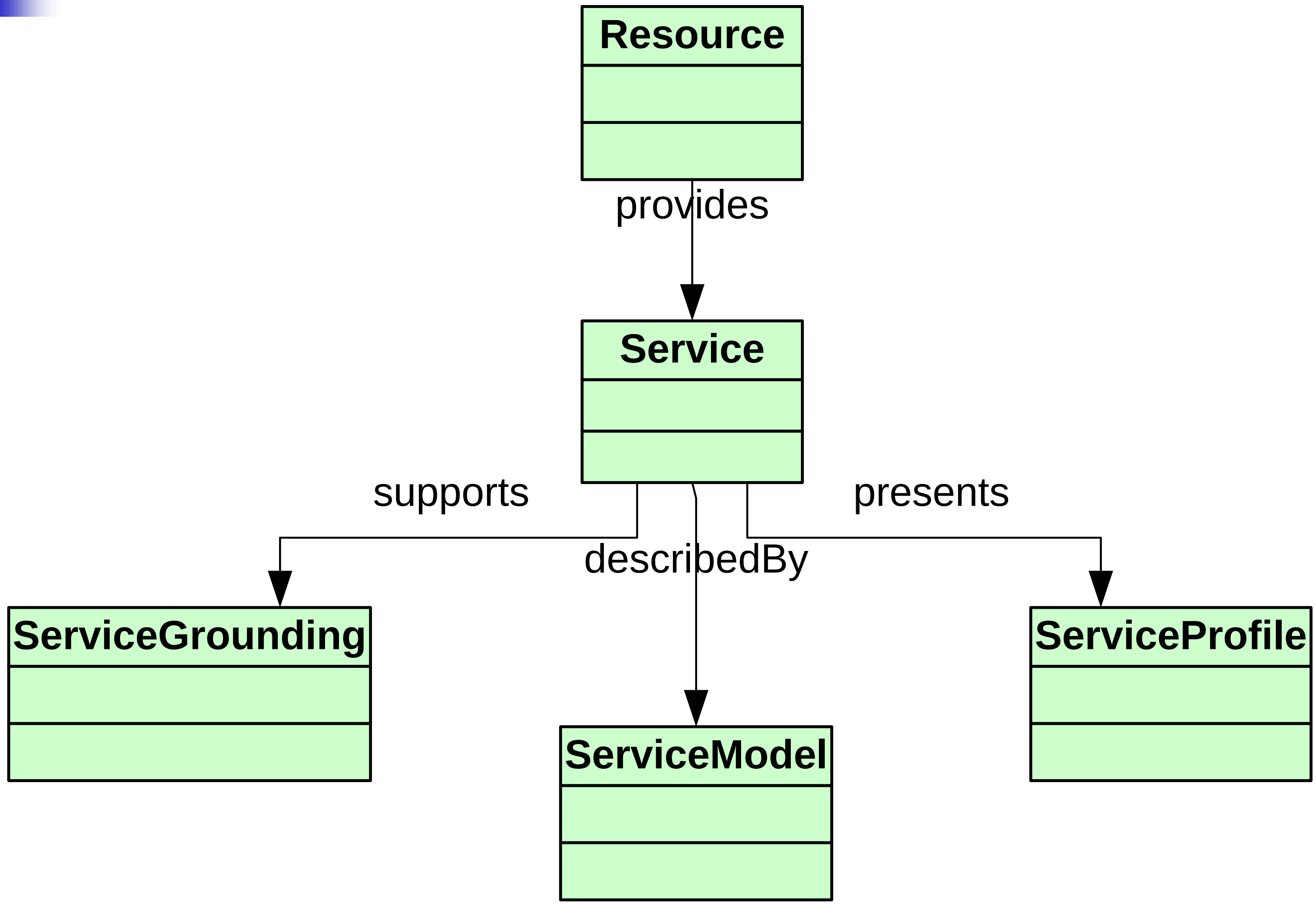

Business Organization A
Business Process and Information Model (UMM or PSL)
ebXML Process Specification Document
Implement ebXML CPA and CPP Specifications
ebXML Business Service Interface Configuration
Request ebXML Specs
Receive ebXML Info
Publish Collaboration Protocol Profile
ebXML Repository
Business Process
Business Scenarios
Business Profiles
Request ebXML Specs
Receive ebXML Info
Publish Collaboration Protocol Profile
CPA Information
Business Organization B
Business Process and Information Model (UMM or PSL)
ebXML Process Specification Document
Implement ebXML CPA and CPP Specifications
ebXML Business Service Interface Configuration

# Web Ontology Language – Services (OWL-S)

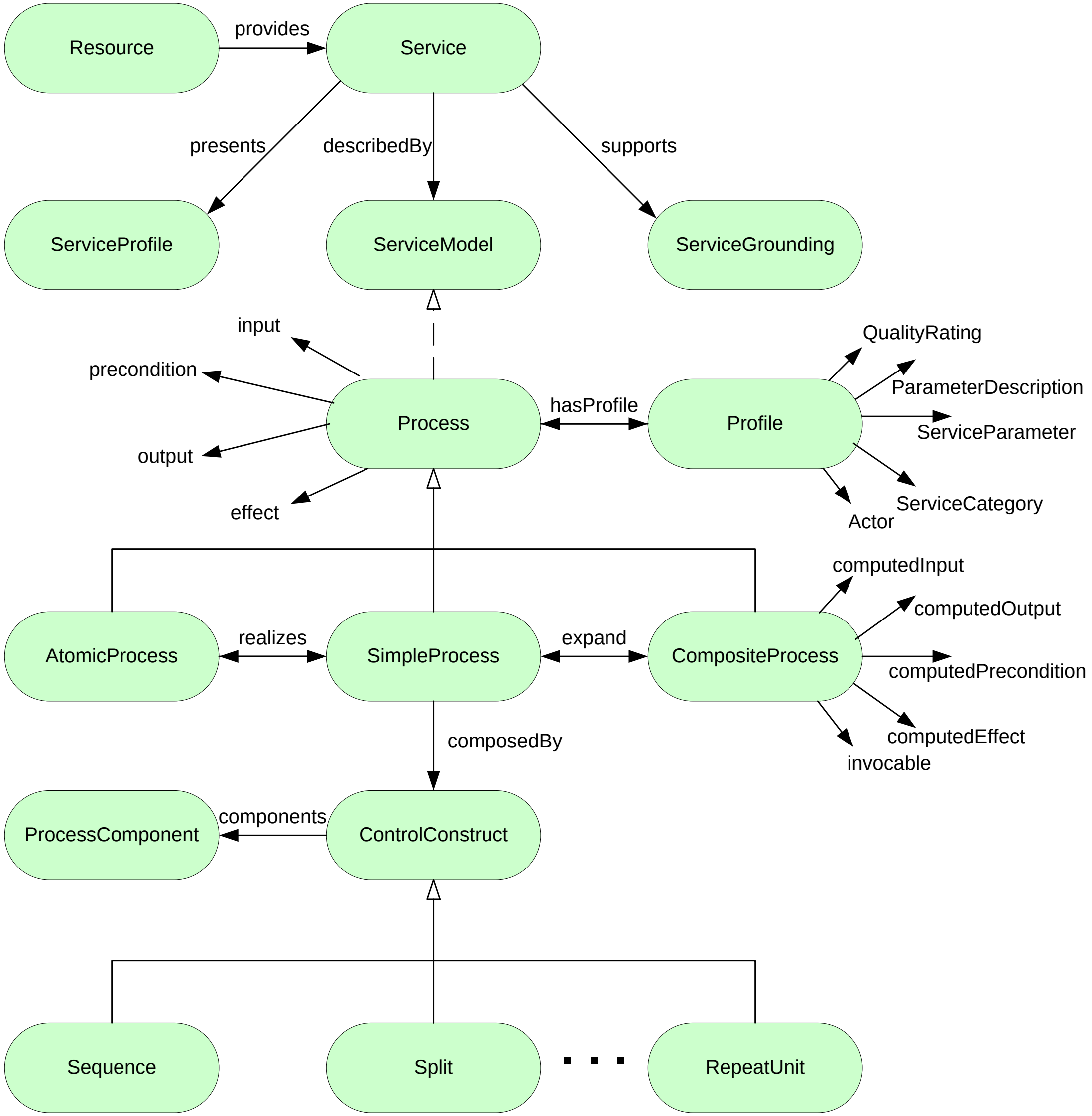
An OWL-S service description provides

- Declarative ads for properties and capabilities, used for discovery
- Declarative APIs, used for execution
- A declarative description of services
  - Based on their inputs, outputs, preconditions, and effects
  - Used for composition and interoperation

# OWL-S Service Ontology

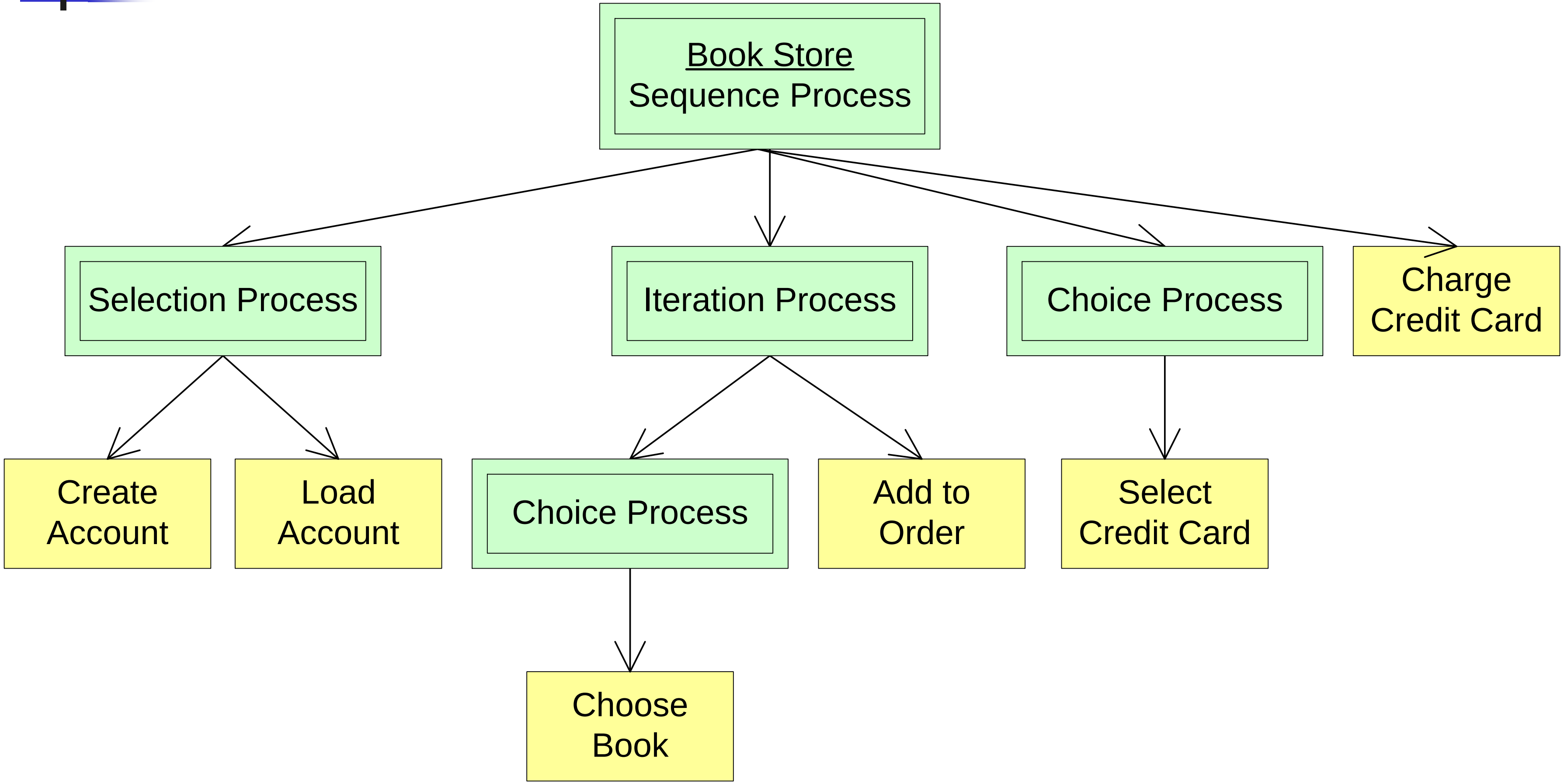


# OWL-S Service Model

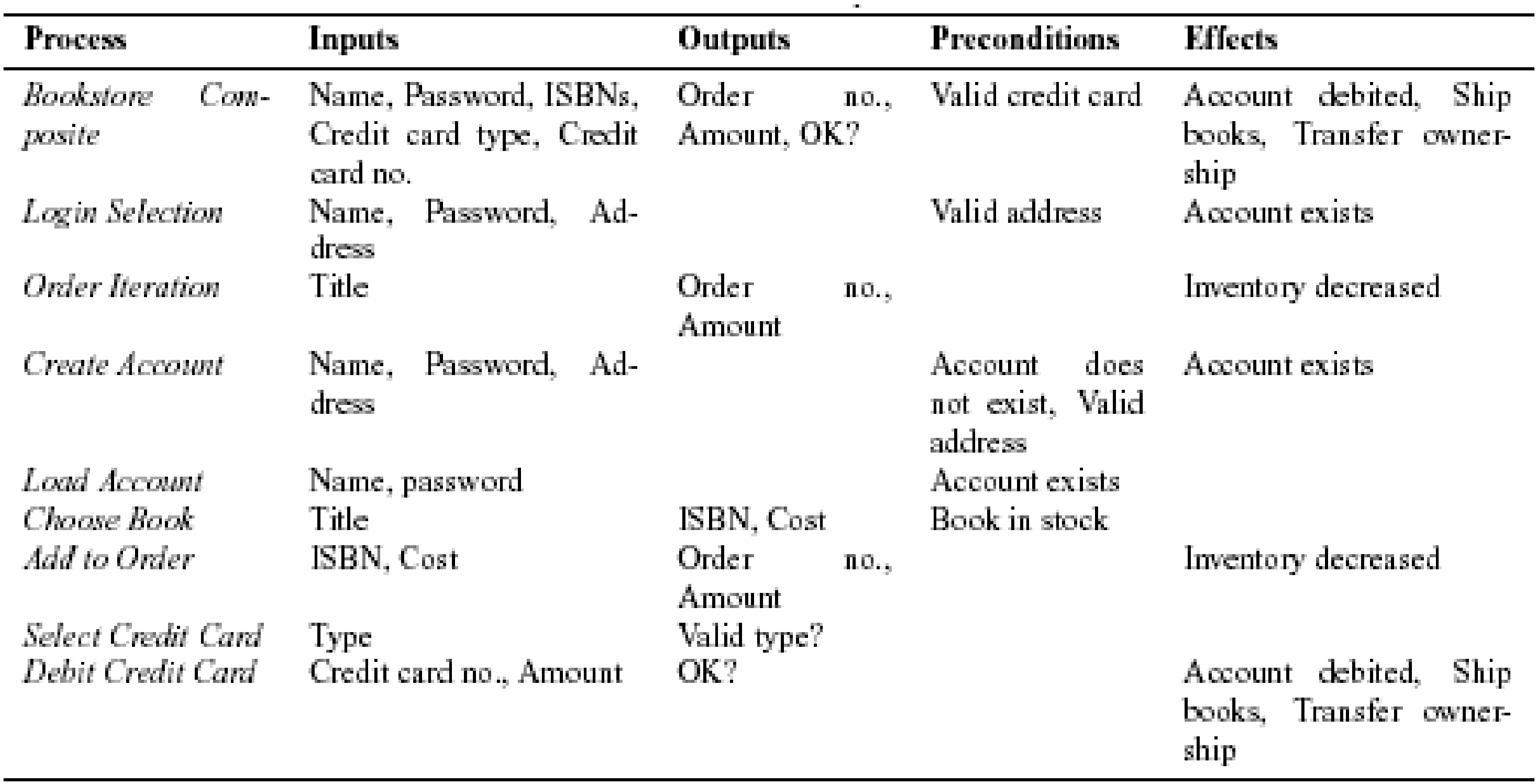


The part from Process on down is how OWL-S captures the process model of a service

# OWL-S Example: Processing Book Orders



# OWL-S IOPEs for Bookstore Example



| Process | Inputs | Outputs | Preconditions | Effects |
|---|---|---|---|---|
| *Bookstore Composite* | Name, Password, ISBNs, Credit card type, Credit card no. | Order no., Amount, OK? | Valid credit card | Account debited, Ship books, Transfer ownership |
| *Login Selection* | Name, Password, Address | | Valid address | Account exists |
| *Order Iteration* | Title | Order no., Amount | | Inventory decreased |
| *Create Account* | Name, Password, Address | | Account does not exist, Valid address | Account exists |
| *Load Account* | Name, password | | Account exists | |
| *Choose Book* | Title | ISBN, Cost | Book in stock | |
| *Add to Order* | ISBN, Cost | Order no., Amount | | Inventory decreased |
| *Select Credit Card* | Type | Valid type? | | |
| *Debit Credit Card* | Credit card no., Amount | OK? | | Account debited, Ship books, Transfer ownership |

# 5: Collaboration

# Agents and MAS for SOC

Why the interest in agents for services?

  - Need for autonomy, heterogeneity, dynamism
  - Need for high-level abstractions for engineering

Unlike objects, agents

- Know about themselves, their users, and their competitors
- Use and reconcile ontologies
- Are proactive and autonomous
- Form commitments and communicate
- Can be cooperative

# What is an Agent?

The term *agent* in computing covers a wide range of behavior and functionality.

- An agent is an active computational entity (could be implemented as an object with a thread)
  - With a persistent identity
  - Perceives, reasons about, and initiates activities in its environment
  - Communicates (with other agents) and changes its behavior based on others
- These features make agents a worthwhile metaphor in computing

# Agent Abstractions: 1

- The traditional abstractions are from AI and are *mentalistic*
  - *Beliefs*: agent’s representation of the world
  - *Knowledge*:
    - (Usually just) true beliefs
    - Justifications are sometimes considered
  - *Desires*: preferred states of the world
  - *Goals*: consistent desires
  - *Intentions*: goals adopted for action

# Agent Abstractions: 2

- The agent-specific abstractions are inherently *interactional*
  - *Social*: about collections of agents
  - *Organizational*: about teams and groups
  - *Ethical*: about right and wrong actions
  - *Legal*: about contracts and compliance

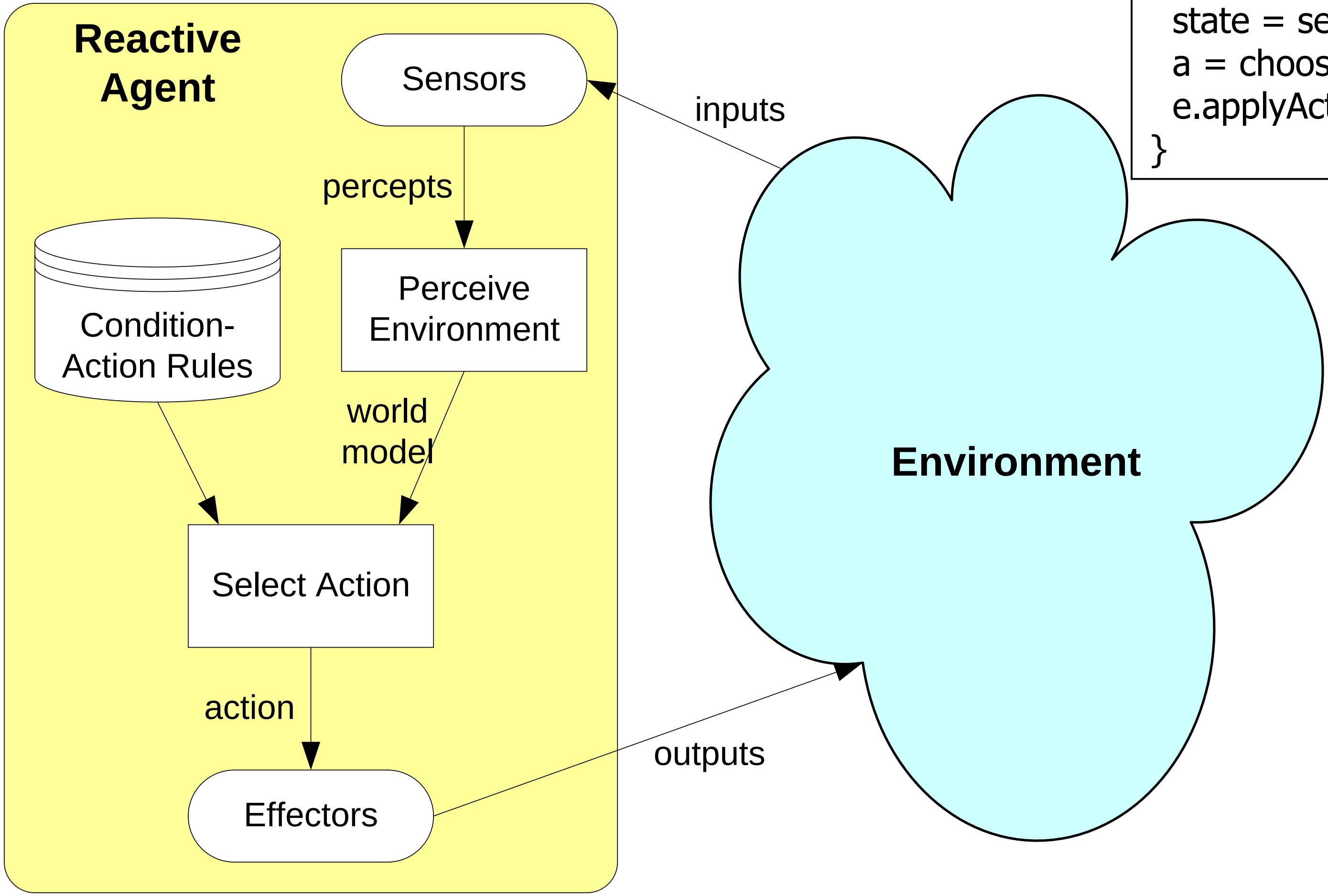
# Agent Abstractions: 3

Agents, when properly understood

- Lead naturally to multiagent systems
  - Contrary to the traditional *economic man*, Robinson Crusoe, who thinks of everything else (even people) as just a resource
- Provide a means to capture the fundamental abstractions that apply in all major applications and which are otherwise ignored by system builders

# A Reactive Agent in an Environment

```
Environment e;
RuleSet r;
while (true) {
  state = senseEnvironment(e);
  a = chooseAction(state, r);
  e.applyAction(a);
}
```



# Characteristics of Agent Environments

- *Observability:* can all aspects relative to actions be sensed?
- *Determinism:* is the next state completely determined by the current state and the agent's action?
- *History Freedom:* does action choice depend on previous episodes or just the current episode?
- *Dynamism:* can environment change while agent is deliberating?
- *Continuity:* do the agent actions, environment state variables, and time points have a continuous range of values?
- *Multiagent:* is the agent aware of others that can affect the environment?

# Reactive Architecture

- Seeks to produce intelligent behavior without explicit
  - Symbolic representations
  - Abstract reasoning
- Intelligence is an emergent property of certain complex systems (depends on the environment too, not just the agent)
  - Cannot plan to drive a car to full detail
  - Reactively avoiding collisions while heading toward an attractor indicates intelligence

# A Rational Agent

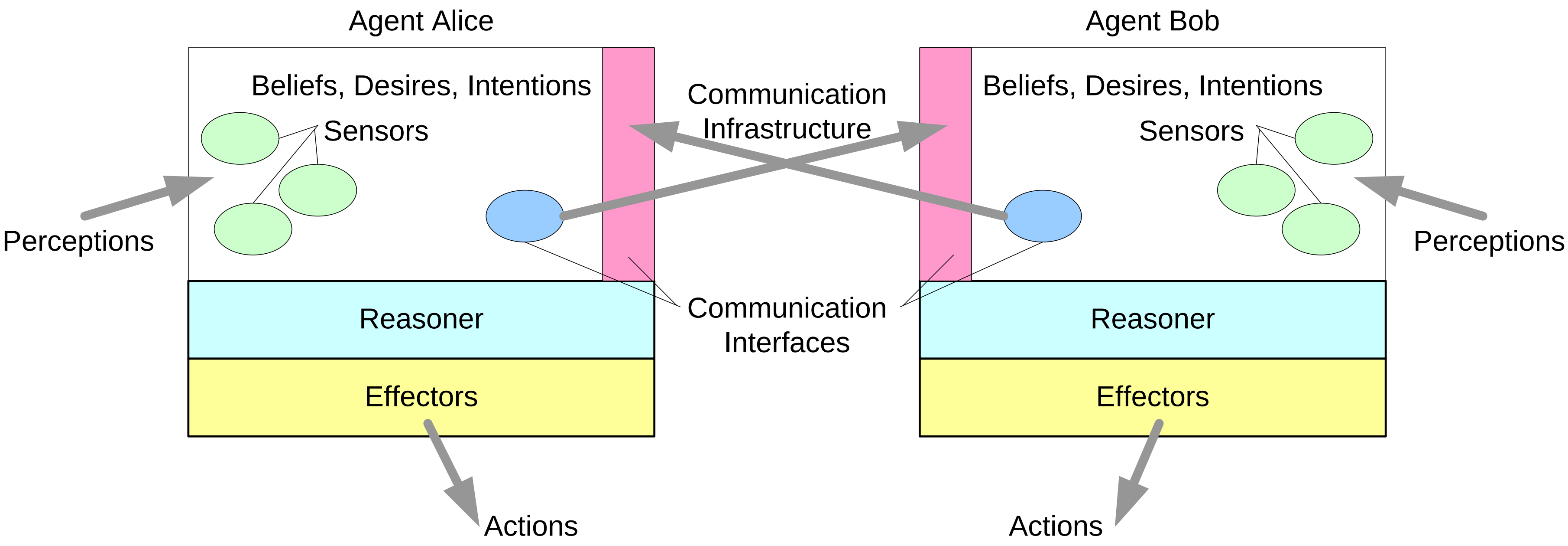
Rationality depends on...

- The performance measure for success, usually taken as utility
- What the agent has perceived so far
- What the agent knows about the environment
- The actions the agent can perform

An **ideal rational agent:** *for each possible percept sequence, it acts to maximize its expected utility, on the basis of its knowledge and the evidence from the percept sequence*

# Cognitive Architecture for an Agent

Called a BDI (beliefs, desires, intentions) architecture



Like the reactive architecture at a coarse level, but with two differences:

•Cognitive representations

•Deeper reasoning based on the above representations

# BDI: A Cognitive *Single-Agent* Architecture

- Beliefs: constitute an agent's representation of the world
- Knowledge: (usually) true beliefs
- Desires: an agent's preferred states of the world
- Goals: consistent desires
- Intentions: goals adopted for action, i.e., what the agent *has chosen* to do

© Michael N. Huhns

# Properties of Cognitive Theories

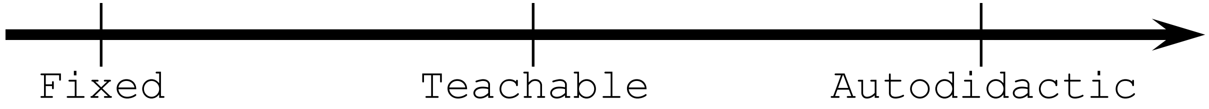
- Beliefs are mutually consistent (this can be a demanding property to realize in a practical system and usually requires an agent's beliefs to be restricted in some way)
- An agent will intend an action only while it believes the action is possible
- An agent need not intend something that would happen anyway

Designers ascribe these properties to an agent, and then link them to the agent's sensors and effectors, while considering the relationships from the sensors and effectors to the environment

© Michael N. Huhns

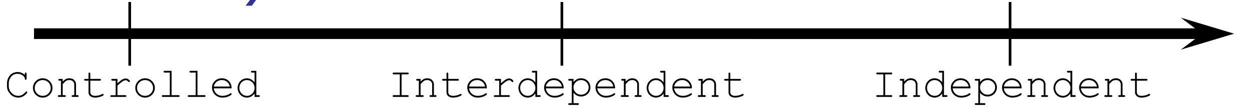
# Dimensions of MAS: Agent

*Adaptivity (the ability of an agent to learn):*



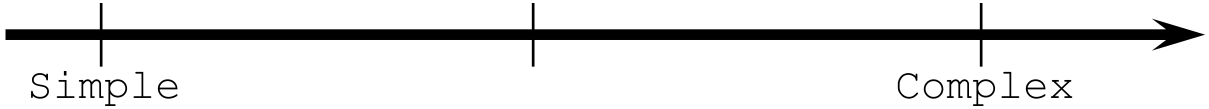
Fixed Teachable Autodidactic

*Autonomy:*



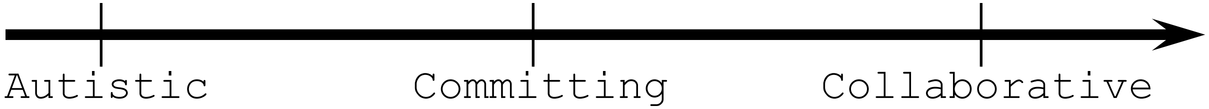
Controlled Interdependent Independent

*Interactions:*



Simple Complex

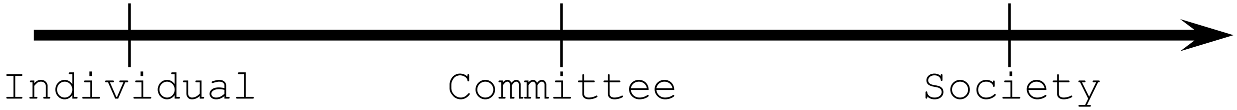
*Sociability (awareness):*



Autistic Committing Collaborative

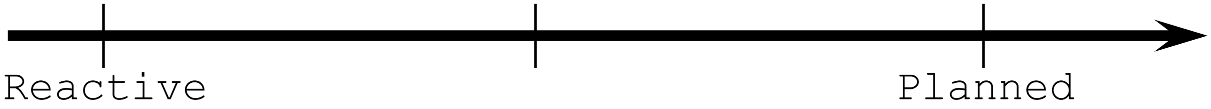
# Dimensions of MAS: System

*Scale (the number of agents):*



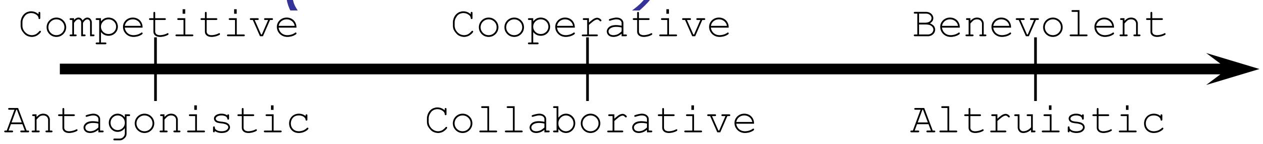
Individual Committee Society

*Interactions:*



Reactive Planned

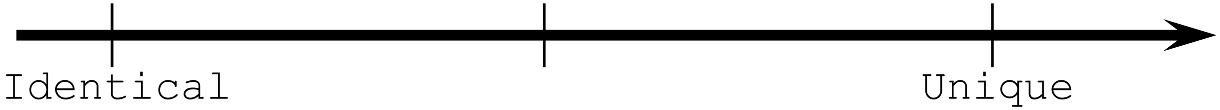
*Coordination (self interest):*



Competitive Cooperative Benevolent

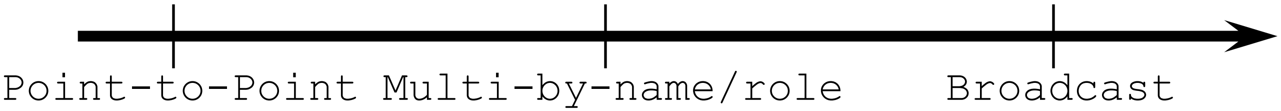
Antagonistic Collaborative Altruistic

*Agent Heterogeneity:*



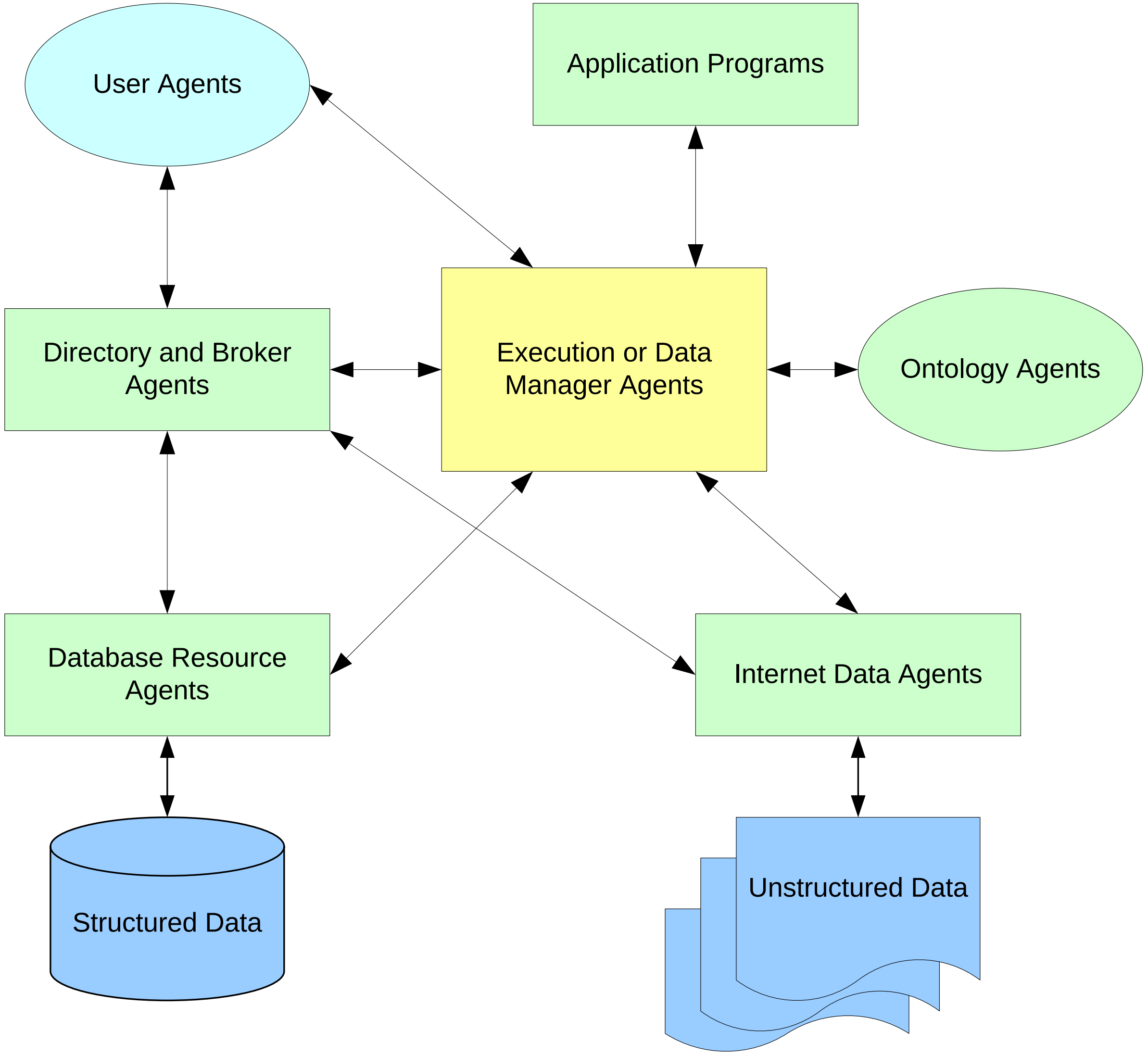
Identical Unique

*Communication Paradigm:*



Point-to-Point Multi-by-name/role Broadcast

# (de facto) Standard Agent Types



# FIPA

- FIPA was the Foundation for Intelligent Physical Agents (www.fipa.org)
- Now an IEEE standards group
- Specifies standards for heterogeneous, interoperating agent-based systems.
- Concerned with *agency* as it relates to
  1. Autonomy (goal-driven)
  2. Communal integration; mostly communication, but also cooperation.

# Consistency Maintenance across Services

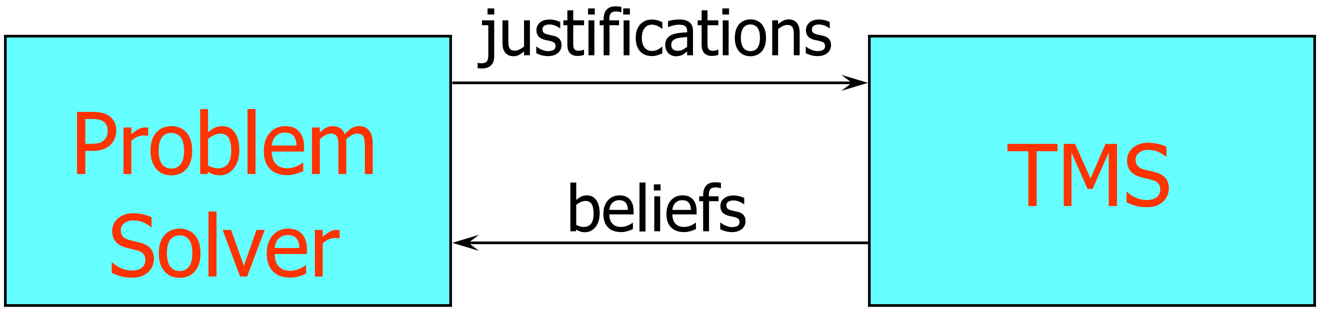
A truth maintenance system (TMS)

- Maintains justifications and explains the results of its deductions
- Updates KB (knowledge base) incrementally when data are added or removed
- Performs a form of propositional deduction

TMSs are important because they

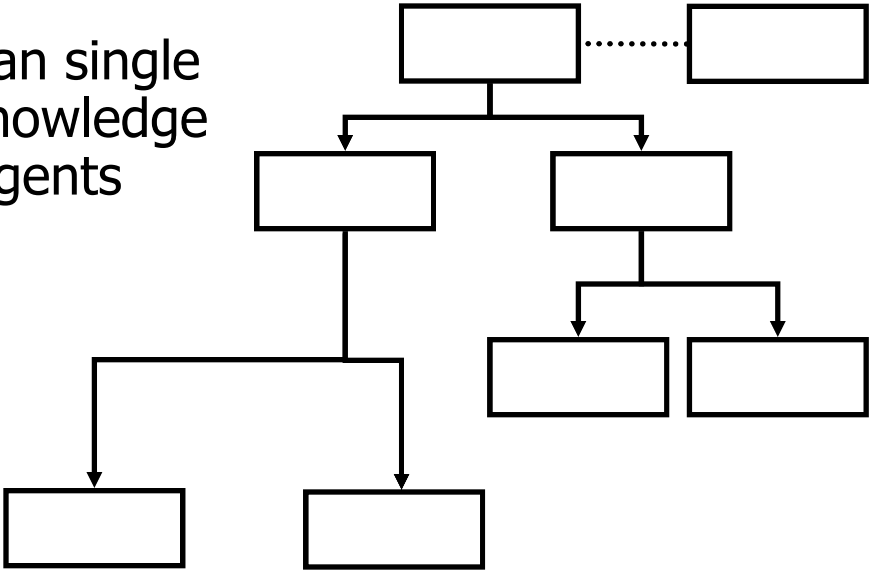
- deal with atomicity: all required changes are made to the KB before anyone can read it
- deal with the frame problem: the parts of the KB that are not affected by a revision are not modified
- lead to efficient search: by using justifications to perform dependency-directed backtracking

# Architecture of TMS-Based Agent



- The problem solver represents domain knowledge in the form of rules, procedures, etc. and chooses what to focus on next
- The TMS keeps track of the current state of the search for a solution. It uses constraint satisfaction to maintain consistency in the inferences made by the problem solver

# Organizations



- Organizations are larger-scale than single agent, goal-oriented, and with knowledge and memory beyond individual agents
- Organizations help overcome the limitations of agents in
  - Reasoning
  - Capabilities
  - Perception
  - Lifetime and persistence
- Concretely, organizations consist of agents acting coherently
- Abstractly, organizations consist of roles and commitments among the roles – these form a *sphere of commitment*

# Legal Concepts

- Traditional AI has a single-agent slant
- Because law involves the interactions of citizens with one another and with the government, the legal abstractions have been rich in multiagent concepts
- Traditional formalisms for legal reasoning, however, are often single-agent in orientation, e.g., deontic logic (the logic of obligation, "obliged to do p")

# Contracts

- Much of the law is about the creation and manipulation of contracts among legal entities
  - People
  - Corporations
  - Governmental agencies

*The law is the study of how to break contracts!*

# Motivation

The legal abstractions provide a basis for agents to enter into contracts, e.g., service agreements, with each other

- Contracts
  - Are about behavior: restrict autonomy
  - Important in open environments
    - About behavior
    - Generally not about implementations

# Commitments: A Basis for Multiple Agents

- Binary relationships binding two agents
  - ‘Debtor’ agent
  - ‘Creditor’ agent
- Represent the agreements between agents

© Michael N. Huhns

# Commitments for Contracts

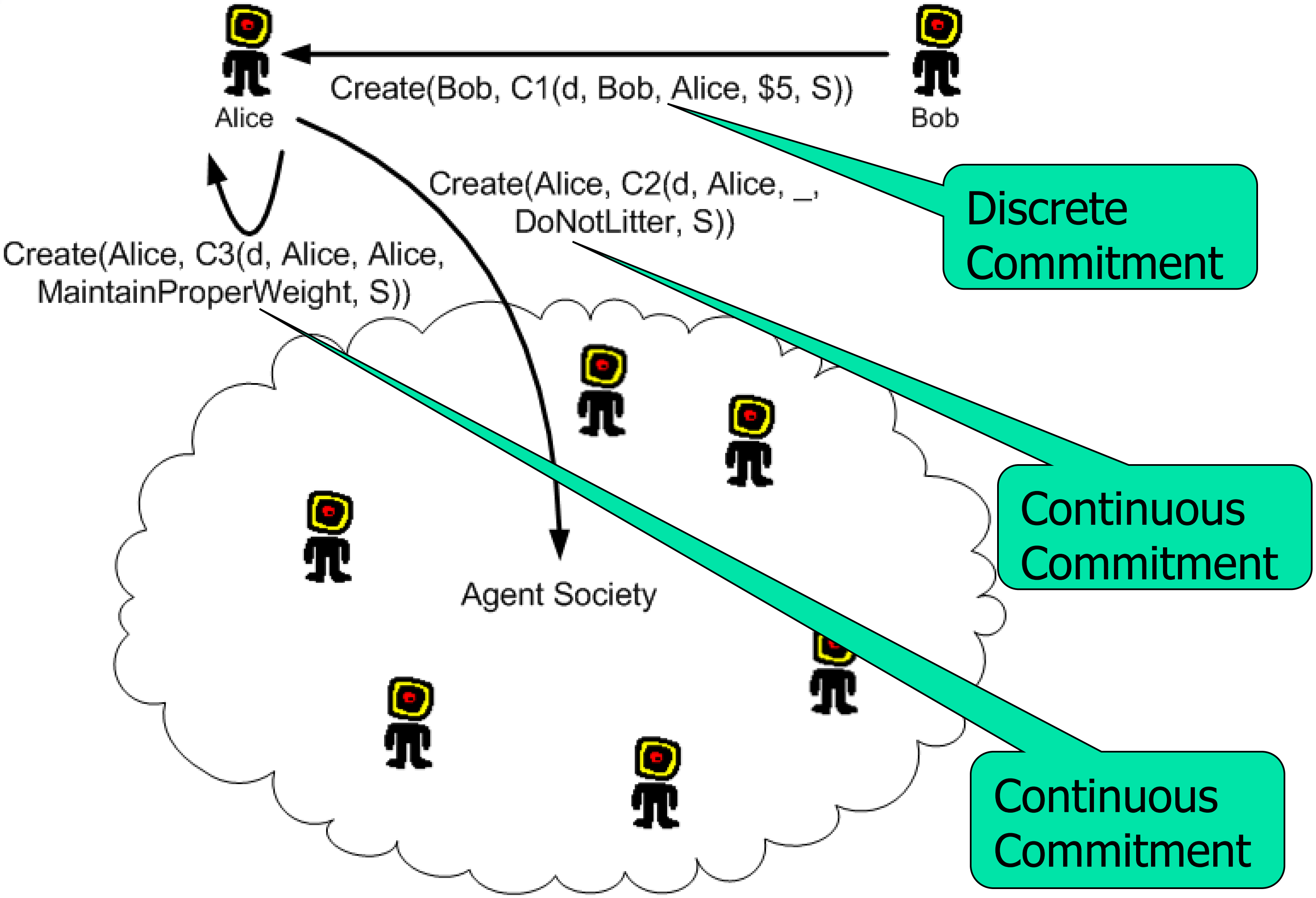
Commitments capture contracts. Importantly, commitments are

- Public (unlike beliefs and intentions)
- Can be used as the basis for compliance
- Contracts apply between parties, in a context
- Other approaches are:
  - Single-agent focused, e.g., deontic logic
  - Don’t handle organizational aspects of contracts
  - Don’t accommodate manipulation of contracts

# Manipulating Commitments

- Operations on commitments:
  - Create
  - Discharge (satisfy)
  - Cancel
  - Release (eliminate)
  - Delegate (change debtor)
  - Assign (change creditor)
- Metacommitments constrain the manipulation of commitments

# Commitment Types


Alice
Bob
Create(Bob, C1(d, Bob, Alice, $5, S))
Create(Alice, C2(d, Alice, _, DoNotLitter, S))
Create(Alice, C3(d, Alice, Alice, MaintainProperWeight, S))
Agent Society

Discrete Commitment

Continuous Commitment

Continuous Commitment

© Michael N. Huhns

# $BDI_{CTL*}$ - Syntax and Semantics

Kripke Structure:

$M = \langle S, R, B_a, D_a, I_a, L \rangle$

- $S$ is a set of states
- $R$ is a binary relation $R \subseteq S \times S$
- $L : S \rightarrow PowerSet(AtomicPropositions)$ is a labeling that associates with each state $s$ an interpretation $L(s)$ of all atomic propositions at state $s$

The relations $B_a$, $D_a$, and $I_a$ map the agent's current situation to its belief, desire, and intention-accessible worlds

# Our Formalization

Creating a Commitment,

Create(a, C(d, a, b, p, S))

- M $\models_m$ Create(a, C(d, a, b, p, S)) ⇒ AB$_a$((XG(active(C))) U (satisfied(C) V breached(C) V canceled(C)))

  For all paths, agent *a* believes that from the next moment onwards commitment C will be active until it is either satisfied or breached or canceled

# Creating a Commitment

- $M \models_m Create(a, C(d, a, b, p, S)) \Rightarrow AB_a F(satisfied(C))$

  For all paths, agent *a* believes that commitment C will eventually be satisfied

- $M \models_m Create(a, C(d, a, b, p, S)) \Rightarrow AXG((I_a(C))\ U\ (satisfied(C) \lor breached(C) \lor canceled(C)))$

  For all paths from the next moment onwards, agent *a* intends the commitment C until it is either satisfied or breached or canceled

# Creating a Commitment

- $M \models_m Create(a, C(d, a, b, p, S)) \Rightarrow AB_a((XG(D_b(C))) U (satisfied(C) V canceled(C)))$

  For all paths, agent *a* believes that from the next moment onwards agent *b* desires commitment C until it is either satisfied or canceled

- $M \models_m Create(a, C(d, a, b, p, S)) \Rightarrow AB_b((XG(active(C))) U (satisfied(C) V breached(C) V canceled(C)))$

  For all paths, agent *b* believes that from the next moment onwards commitment C will be active until it is either satisfied or breached or canceled

© Michael N. Huhns

# Creating a Commitment

- $M \models_m \text{Create}(a, C(d, a, b, p, S)) \Rightarrow AB_b((XG(I_a(C))) \text{ U } (\text{satisfied}(C) \vee \text{breached}(C) \vee \text{canceled}(C)))$

  For all paths, agent *b* believes that from the next moment onwards agent *a* intends commitment C until it is either satisfied or breached or canceled

- $M \models_m \text{Create}(a, C(d, a, b, p, S)) \Rightarrow AB_bF(\text{satisfied}(C))$

  For all paths, agent *b* believes that commitment C will eventually be satisfied

# Creating a Commitment

- M $\models_m$ Create(a, C(d, a, b, p, S)) $\Rightarrow$ AXG((D$_b$(C)) U ($\neg$active(C)))

  For all paths from the next moment onwards, agent *b* desires commitment C until it becomes inactive
- Note: agent *b* cannot intend C to be satisfied, because it has no control over C

Other commitment operations: similar

© Michael N. Huhns

# SoCom: Sphere of Commitment

- An organization that provides the context or scope of commitments among
  - Roles (*abstract SoCom*) at design time
  - Agents (*concrete SoCom*) at run time
- A SoCom, especially at run time
  - Serves as a *witness* for the commitment, i.e., knows that the commitment exists
  - Helps validate commitments and test for compliance
  - Offers compensations to undo members' actions, e.g., to handle exceptions

# Policies and Structure

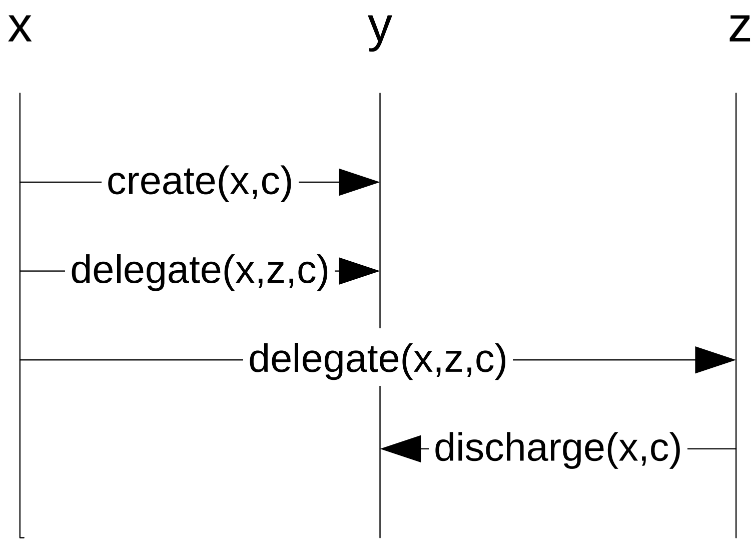
- Spheres of commitment (SoComs)
  - Abstract specifications of societies
  - Made concrete prior to execution
- Policies apply on performing social actions
- Policies relate to the nesting of SoComs
- Role conflicts can occur when agents play multiple roles, e.g., because of nonunique nesting

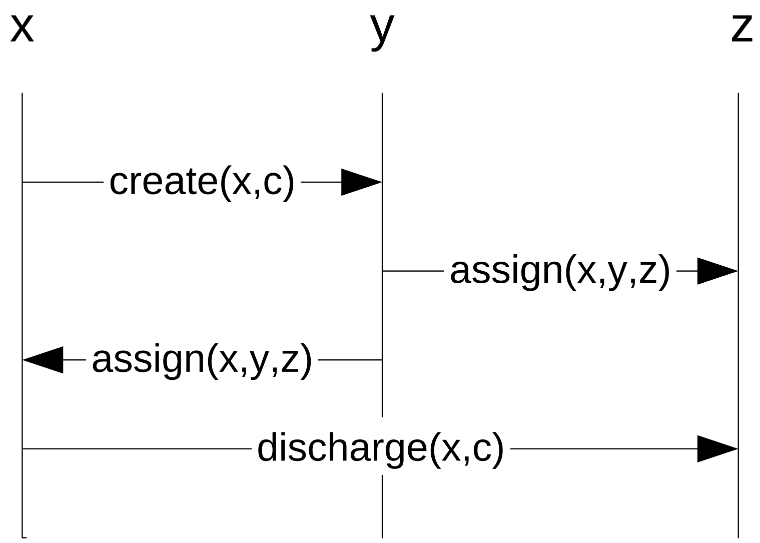
# Commitment Protocols

- Protocols enable open systems to be constructed
- Interaction protocols expressed in terms of
  - Participants’ commitments
  - Actions for performing operations on commitments (to create and manipulate them)
  - Constraints on the above, e.g., captured in temporal logic
- *Examples:* escrow, payment, RosettaNet (107 request-response PIPs)

# Message Patterns for Commitment Operations

- For efficient checking, ensure that the discharge of a commitment is reachable from its create (discharge has a greater vector timestamp than create)
- That is, ensure that information about commitment operations flows to the right parties
- The patterns below accomplish this by sending extra messages for delegate (add message x to y) and assign (add message y to z)





# Compliance with Protocols

Compliance means all commitments are taken care of (discharged directly or indirectly)

- How can we check if the agents *comply* with specified protocols?
  - Coordination aspects: traditional techniques
  - Commitment aspects: representations of the agents’ commitments in temporal logic
- Commitment protocols are specified in terms of
  - Main roles and sphere of commitment
  - Roles essential for coordination
  - Domain-specific propositions and actions

# Negotiation

Negotiation is central to adaptive, cooperative behavior

- Negotiation involves a small set of agents
- Actions are propose, counterpropose, support, accept, reject, dismiss, retract
- Negotiation requires a common language and common framework (an abstraction of the problem and its solution)

# Negotiation Mechanism Attributes

- Efficiency
- Stability
- Simplicity
- Distribution
- Symmetry

e.g., sharing book purchases, with cost decided by coin flip

# Negotiation among Utility-Based Agents

Problem: How to design the rules of an environment so that agents interact productively and fairly, e.g.,

- Vickrey’s Mechanism:  lowest bidder wins, but gets paid second lowest bid (this motivates telling the truth?? and is best for the consumer??)

# Negotiation

- A deal is a joint plan between two agents that would satisfy their goals
- The utility of a deal for an agent is the amount he is willing to pay minus the cost to him of the deal
- The negotiation set is the set of all deals that have a positive utility for every agent. The possible situations for interaction are
  - *Conflict*: the negotiation set is empty
  - *Compromise*: agents prefer to be alone, but will agree to a negotiated deal
  - *Cooperative*: all deals in the negotiation set are preferred by both agents over achieving their goals alone

# Negotiation Mechanism

The agents follow a Unified Negotiation Protocol, which applies to any situation. In this protocol,

- The agents negotiate on mixed-joint plans, i.e., plans that bring the world to a new state that is better for both agents
- If there is a conflict, they “flip a coin” to decide which agent gets to satisfy his goal

# Agent Communication Language (ACL)

What is the semantics of queries, requests, promises?

- *Mentalist*: each agent has a knowledge base that its messages refer to
  - An agent promises something if it intended to make the content of that promise come true
- *Public*: semantics depends on laws, protocols, and observable behavior
  - An agent promises something if it says so in the appropriate circumstances
- **Evaluation:** For open systems, public semantics is appropriate, because a semantics without compliance doesn’t make sense

# Syntax, Semantics, Pragmatics

Consider communication as synonymous with message passing

- *Syntax*: requires a common language to represent information and queries, or languages that are intertranslatable
- *Semantics*: requires a structured vocabulary and a shared framework of knowledge-a shared ontology
- *Pragmatics*: is usually context-sensitive
  - Knowing whom to communicate with and how to find them
  - Knowing how to initiate and maintain an exchange
  - Knowing the effect of the communication on the recipient

# Speech Act Theory

Speech act theory, developed for natural language, views communication as action

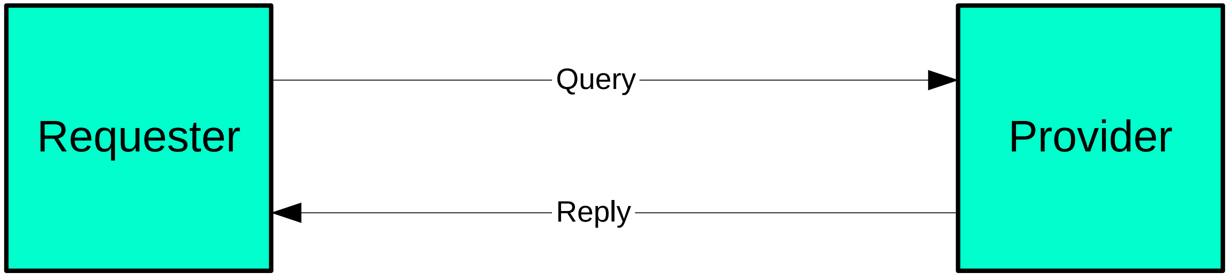
- Differs from traditional logic
- Considers three aspects of a message:
  - *Locution*, or how it is phrased, e.g., "It is hot here" or "Turn on the air conditioner"
  - *Illocution*, or how it is meant by the sender or understood by the receiver, e.g., a request to turn on the air conditioner or an assertion about the temperature
  - *Perlocution*, or how it influences the recipient, e.g., turns on the air conditioner, opens the window, ignores the speaker

Illocution is the core aspect

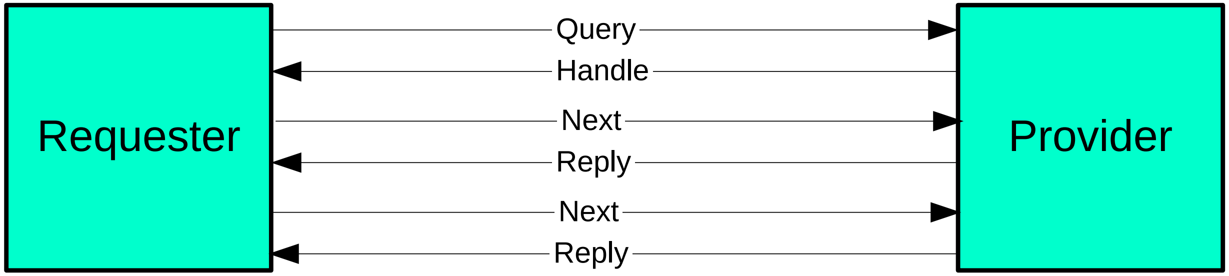
# Speech Act Theory Applied

- Classifications of illocutions motivate message types, but are typically designed for natural language
  - Rely on NL syntax, e.g., they conflate directives and prohibitives
- Most research in speech act theory is about determining the agents’ beliefs and intentions, e.g., how locutions map to illocutions
- For services and agents, determining the
  - Message type is trivial, because it is explicitly encoded
  - Agents’ beliefs and intentions is impossible, because the internal details of the agents are not known

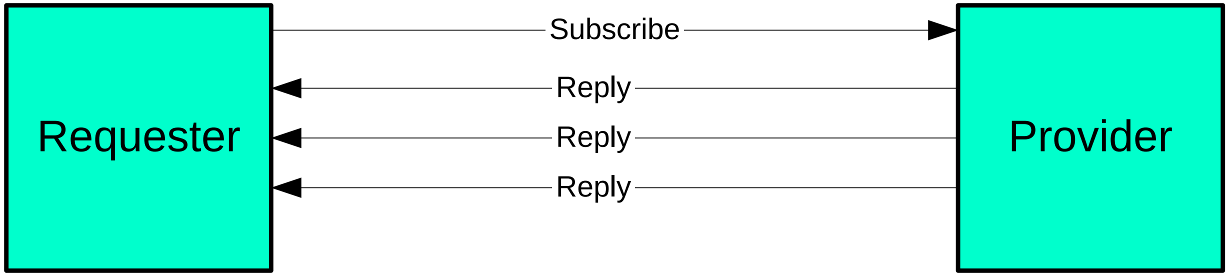
# Patterns and Protocols



Synchronous: a blocking query waits for an expected reply



Provider maintains state; replies sent individually when requested

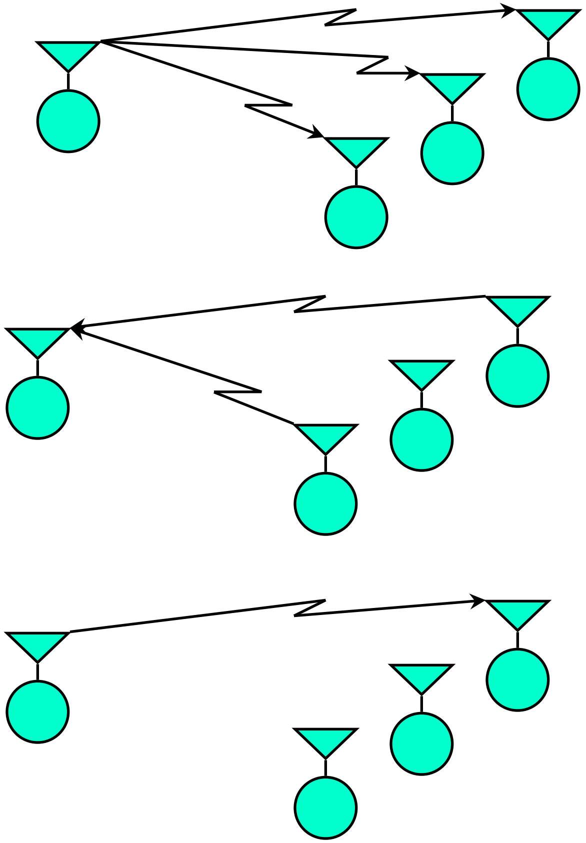


Asynchronous: a nonblocking subscribe; replies sent as available

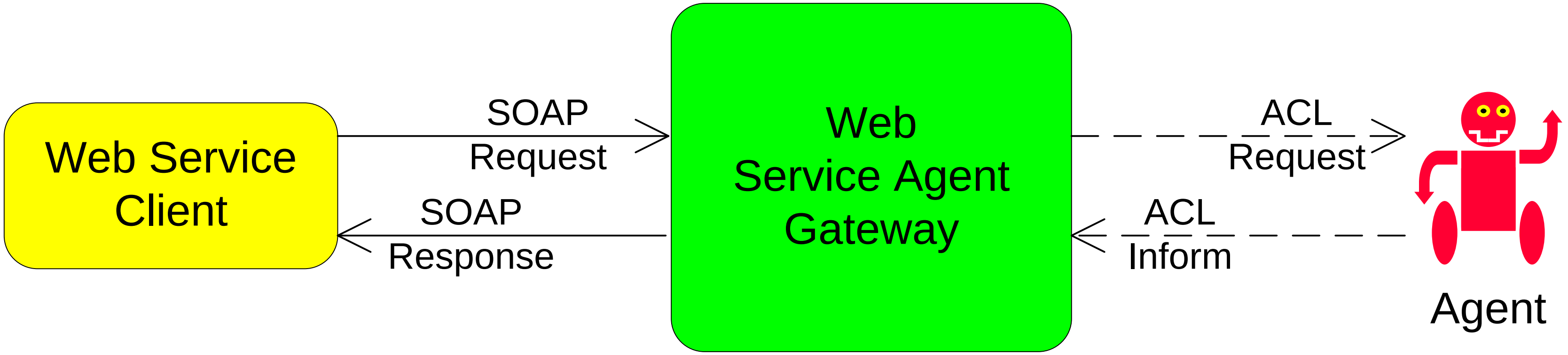
# The Contract Net Protocol

An important generic protocol

- Manager announces tasks via a (possibly selective) multicast
- Agents evaluate the announcement.  Some submit bids
- Manager awards a contract to the most appropriate agent
- Manager and contractor communicate privately as necessary



# Combining Agents with Traditional Web Services



# 5: Discovery and Selection

# Discovery versus Selection

- Often the purpose behind discovering a service is to select a good one
  - We don’t need to find all services
  - Just the one that’s best for us!
- By focusing on selection, we can
  - Reduce irrelevant results
  - Reduce irrelevant traffic and management
  - Improve the payoff

# Recommending Products vs. Services

- Products (by a product vendor)
  - The recommender is the provider
  - Votes are known to recommender
  - Votes are given prior to usage (buying)
  - Repetition is less likely (buy the same book)
- Services (by a service registry)
  - The recommender is not the provider
  - Votes are not necessarily known to recommender
  - Votes are given after usage
  - Repetition can occur but not known to registry

# Reputation

The agency (e.g., eBay) is the authority that

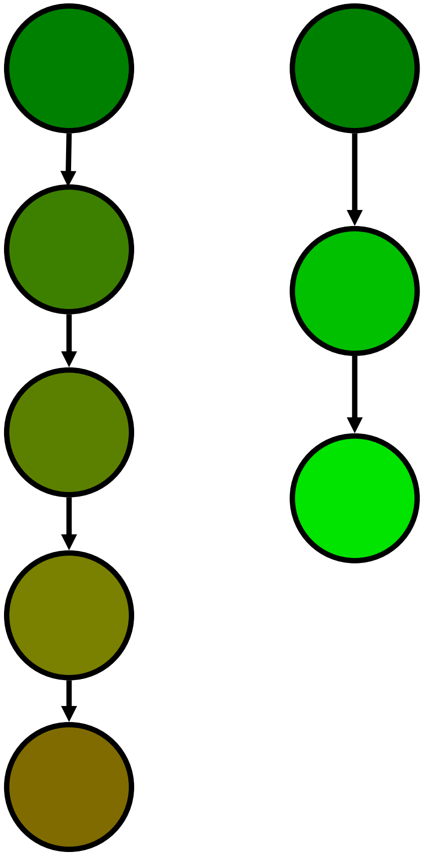
- Authenticates users
- Records, aggregates, and reveals ratings
- Provides the conceptual schema for
  - How to capture ratings (typically a number and text)
  - How to aggregate them
  - How to decay them over time

# Social Networks and Referral Chains

- Referral chains provide:
  - Way to judge the quality of an expert's advice
  - Reason for the expert to respond in a trustworthy manner

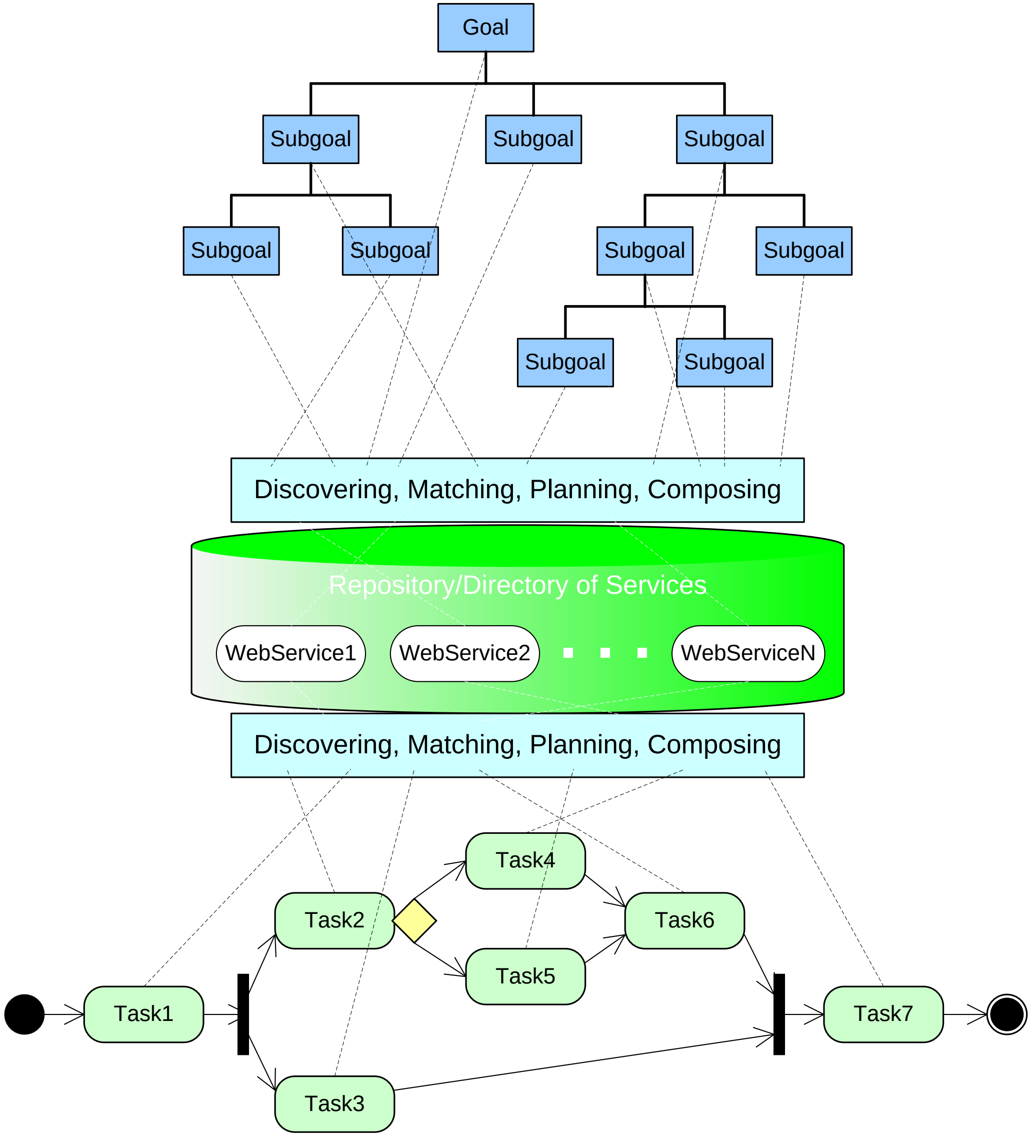
Social networks induce referral chains in which an individual may participate

- As the chains get longer
  - The trustworthiness of a recommendation decreases
  - The effort to find experts increases
- Therefore, shorter chains are better



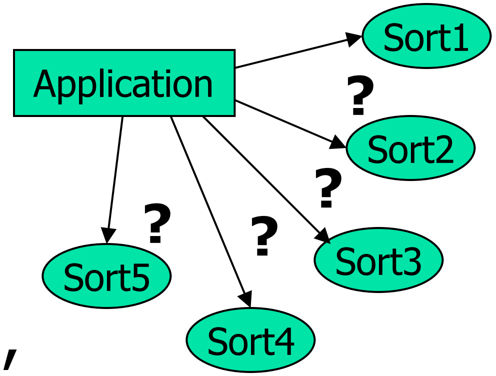
# 6: Synthesis

# Two Patterns for Engineering Service Applications



# Advanced Composition: 1

- Suppose an application needs simply to sort some data items, and suppose there are 5 Web sites that offer sorting services described by their input data types, output date type, time complexity, space complexity, and quality:
  - One is faster
  - One handles more data types
  - One is often busy
  - One returns a stream of results, another a batch
  - One costs less



# Advanced Composition: 2

- Possible approaches
  - Application invokes services randomly until one succeeds
  - Application ranks services and invokes them in order until one succeeds
  - Application invokes all services and reconciles the results
  - Person organizes all services into one service using BPEL4WS
  - Application contracts with one service after requesting bids
  - Services self-organize into a team of sorting services and route requests to the best one
- The last two require that the services behave like agents
- The last two are scalable and robust

# Elements of Service-Oriented Architectures

- *Loose coupling*: focus should be on high-level contractual relationships
- *Implementation neutrality*: the interface is what should matter
- *Flexible configurability*: late binding of components
- *Long lifetime*: components should exist long enough to be discovered, to be relied upon, and to engender trust in their behavior
- *Granularity*: interactions and dependencies should occur at as high a level as possible
- *Teams*: computation in open systems should be conceptualized as business partners working as a team

# Systemic Trust

- Fundamentally
  - The information agents retrieve must be accurate, or characterized accurately
  - The information agents contribute must be used appropriately
- Requires
  - Sources have reliability and reputation, and specify constraints on usage
  - Dependencies are preserved and maintained
- Results: information items have credibility and domains of utility; agents self-organize into *service communities*

# Trust

Ultimately, what we would like is to trust Semantic Web services. Trust involves services that

- Are understood in context
- Have the right capabilities and understanding of needs
- Follow legal contracts where specified
- Support one’s organization or society
- Follow an understood ethics
- Failing all else, behave rationally

# Summary

| Multiagent System Properties | Benefits for Service Development |
|---|---|
| Autonomous, objective-oriented behavior; agent-oriented decomposition | Autonomous, active functionality that adapts to the users’ needs; reuse of whole subsystems and flexible interactions |
| Dynamic composition and customization | Scalability |
| Interaction abstractions; statistical or probabilistic protocols | Friction-free software; open systems; interactions among heterogeneous systems; move from sophisticated and learned e-commerce protocols to dynamic selection of protocols |
| Multiple viewpoints, negotiation, and collaboration | Robustness and reliability |
| Social abstractions | High-level modeling abstractions |

# To Probe Further

- IEEE Internet Computing, *http://computer.org/internet*
- DAI-List-Request@engr.sc.edu
- (International Joint Conference and Journal) Autonomous Agents and Multiagent Systems
- Conferences on Semantic Web, Web Services, Service-Oriented Computing, Service Computing, World-Wide Web
- Book: Singh & Huhns, *Service-Oriented Computing,* John Wiley & Sons, 2005