Robust Agents for Services

Prof. Michael N. Huhns
Center for Information Technology
University of South Carolina

What the Buzzwords Portend

- Multicore processors → Threads
- Autonomic computing → Robustness
- Open-source software → Robustness, security, autonomy (non-proprietary)
- Web services → Programming-in-the-large (really large!)
- Unlimited bandwidth → Distributed computation; Grid computing
  ==>

<u>All require research, and all lead to agents!</u>

Computing is Changing...

- Old: individual computations
- New: *interactions* among computations

Due to new applications:

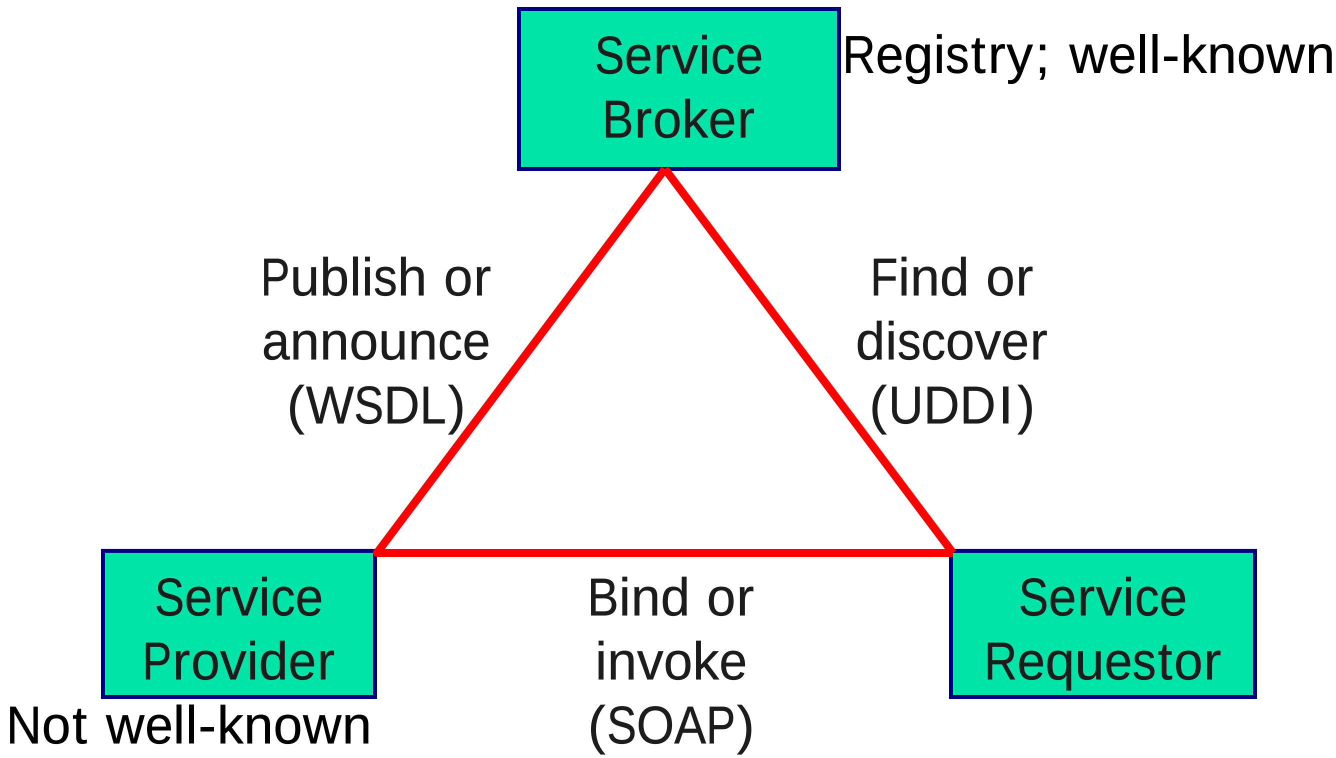
- Ubiquitous information access
- Electronic commerce
- Digital libraries
- Supply-chain automation

Requirements for a Methodology

A methodology must support

- Distributed development
- Distributed execution
- Software reuse
- Robustness
- Support for programming-in-the-large

Web Services: Basic Architecture

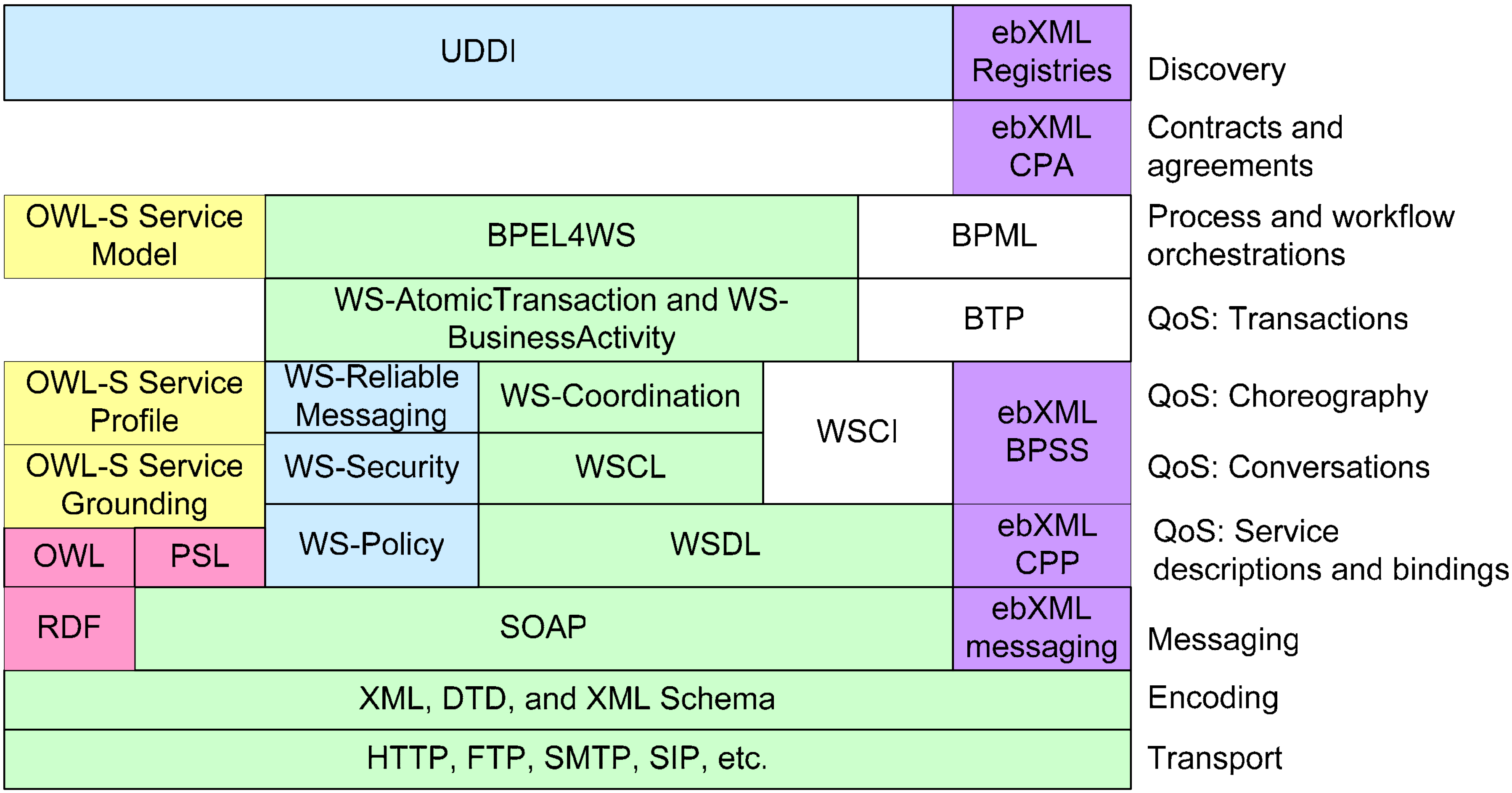


XML Web Service Foundation

Open and with broad industry support

- Publish, Find, Use Services
  - UDDI
- Service Interactions
  - SOAP
- Universal Data Format
  - XML
- Description Language
  - WSDL
- Ubiquitous Communications
  - TCP/IP, HTTP, SMTP, SIP, Reliable messaging
- Security (authentication and authorization)
  - WS-Security, SAML

Standards for Web Services

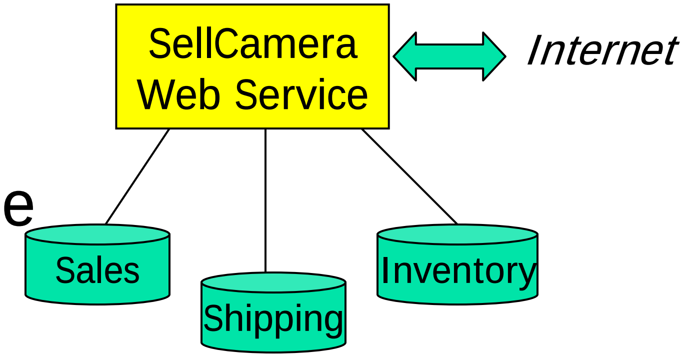


Foundation for Web Services

- Current Web services have a database or a programmatic basis. Both are unsatisfactory...

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
  - record a sale in a 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

“Traditional” B2C Problems

- What if the order is shipped, but the debit fails?
- What if the debit succeeds, but the order was never entered or shipped?

Database Approach

A traditional database approach works only for a closed environment:

- 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,* or the user's agent, into a consistent state!

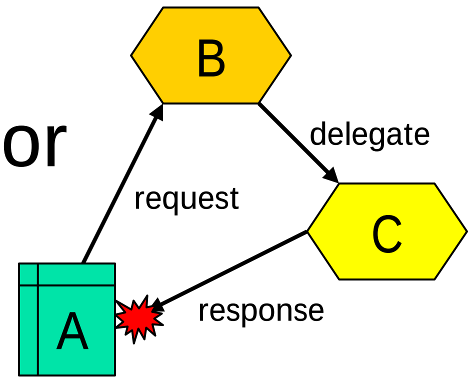
Approach for Open Environment

- 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
- Server could use 2PC, *IF* the user's agent understands that protocol

The server behaves like an agent!

Programming Basis for Web Services

- Under a programming approach, software is partitioned into composable services, which are invoked by an application using, for example, RMI
- Suppose application A invokes service B, but B is busy and delegates the request to service C. When service C sends a response to A, A *fails* because it expected a response from B



Trends...

First Generation Web

Hardware:

- Connect all PCs

Software:

- Create a Web page for everybody and everything

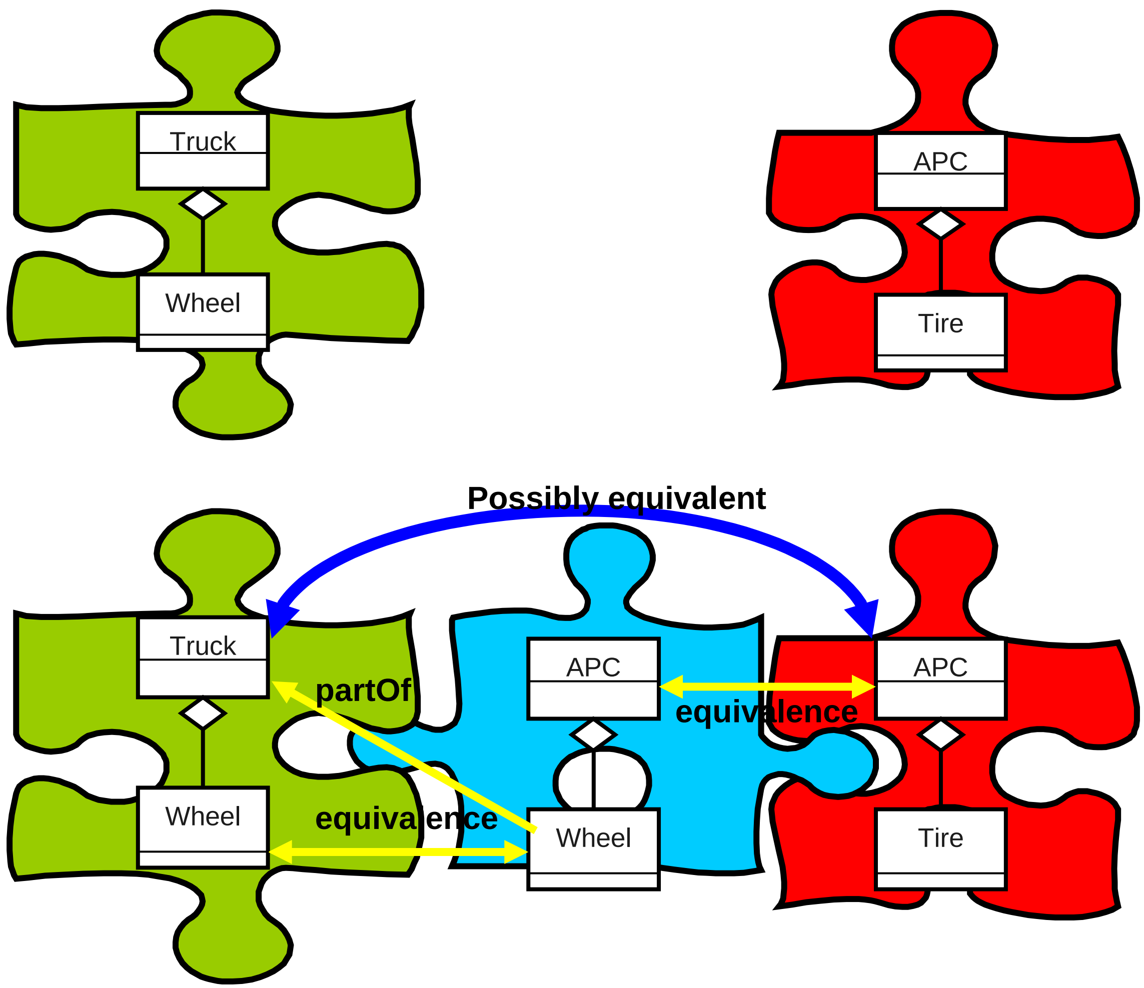
Use Google to get information, e.g., population of USA

How do we get consensus information?

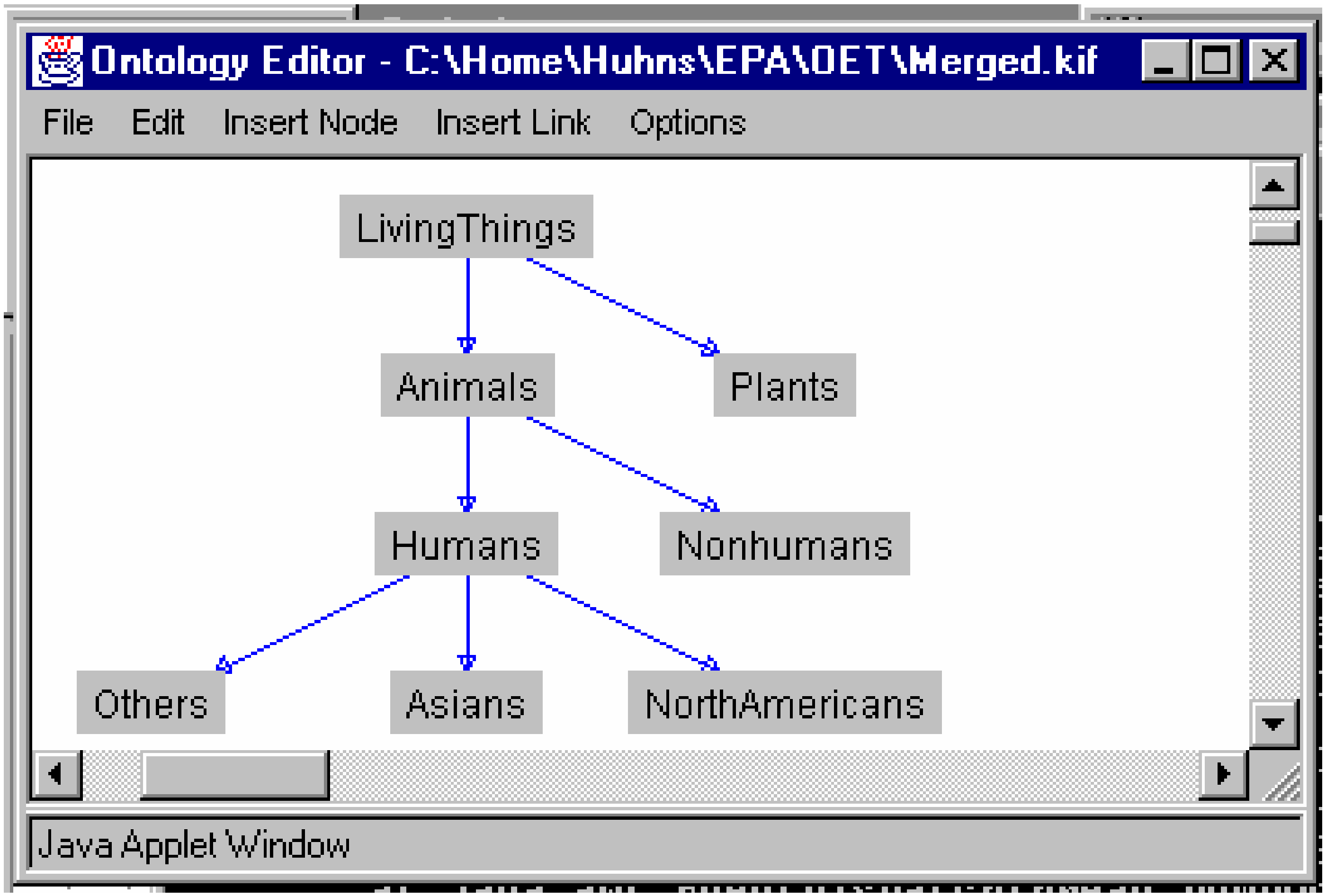
Mutual Understanding Project

- Assumptions:
  - No global ontology
  - Individual sources have individual ontologies
  - Ontologies are heterogeneous and inconsistent
- Hypothesis:
  - Large numbers of individual ontologies can be related by exploiting redundancy

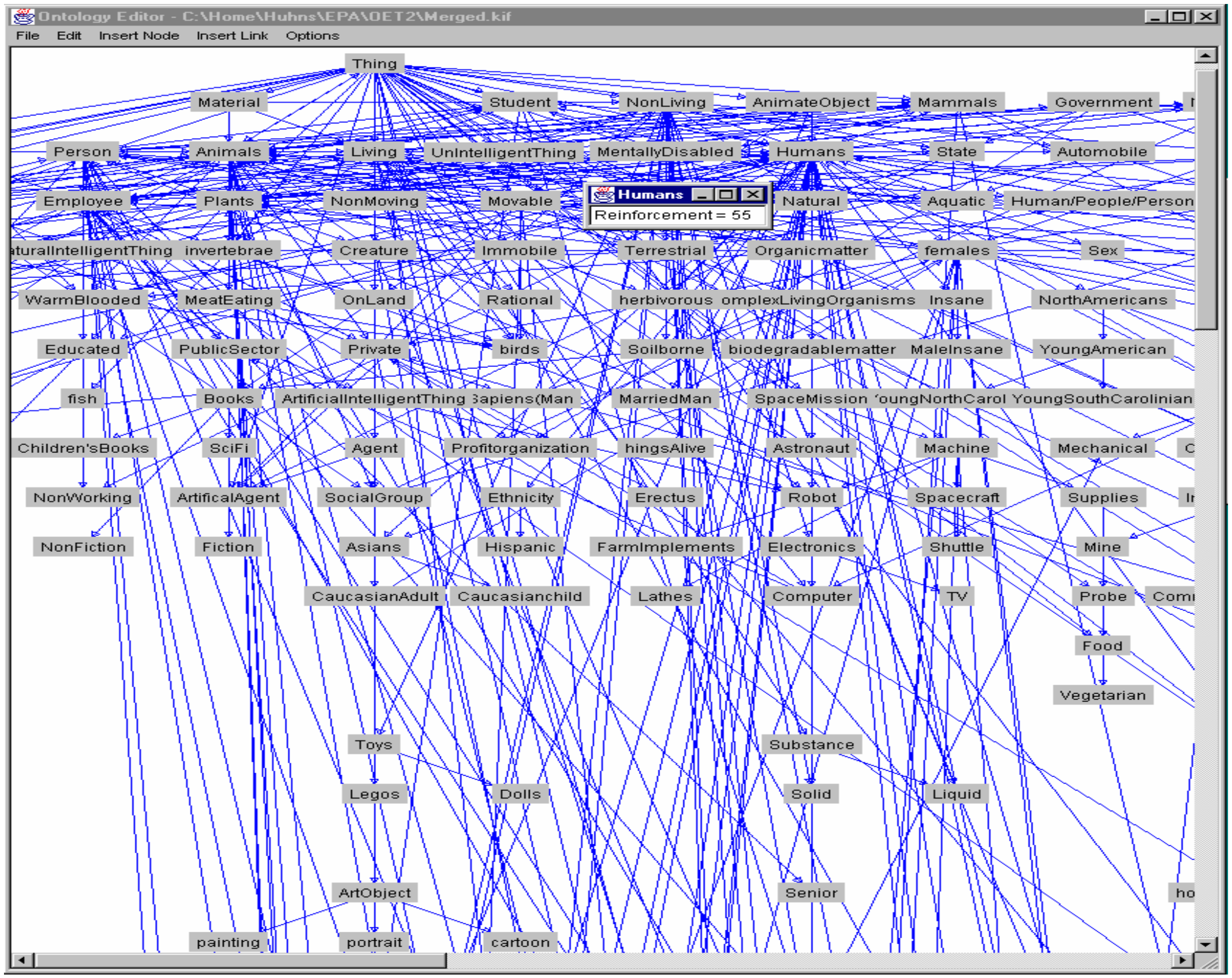
Relating Ontologies



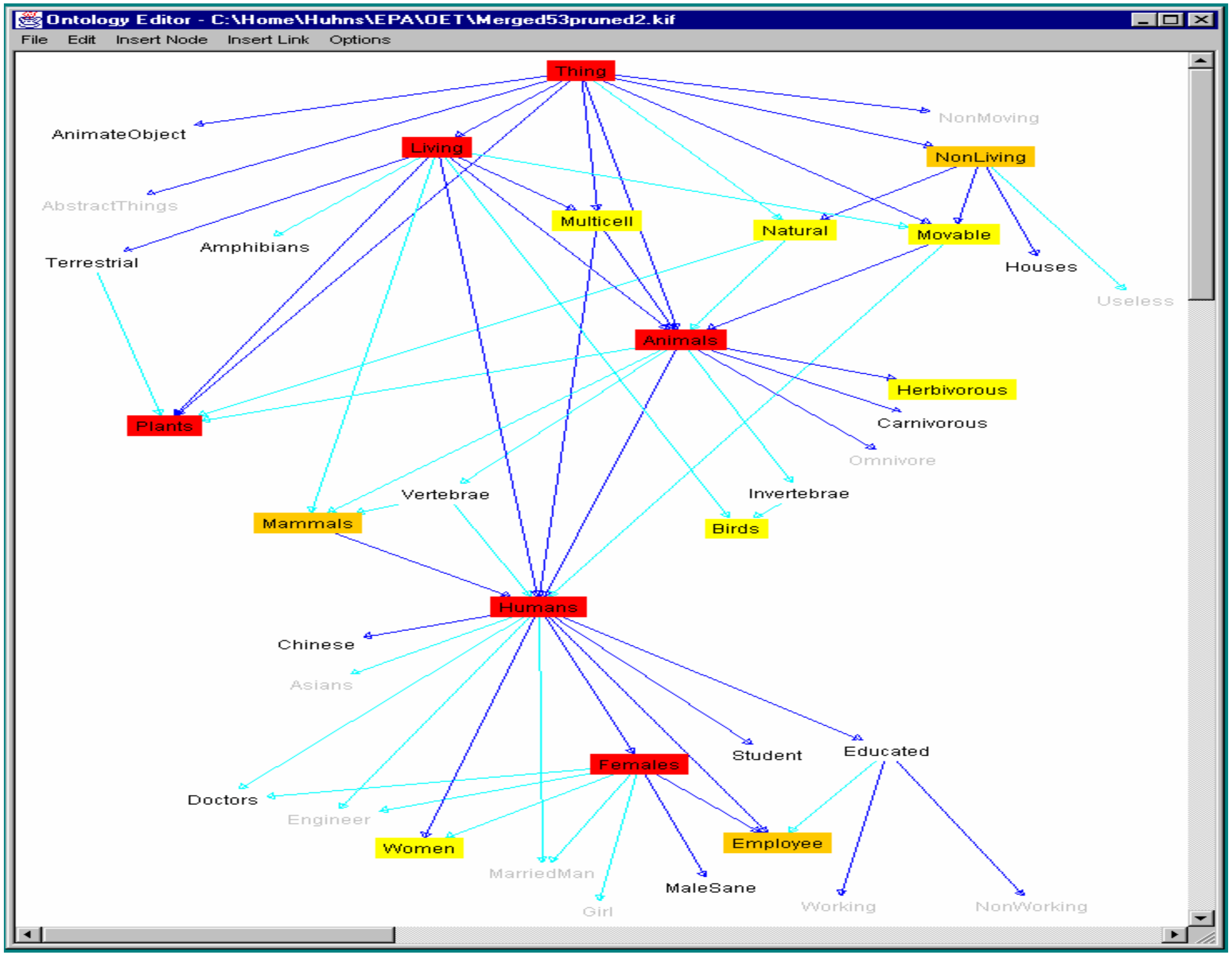
Initial Experiment: 55 Individual Ontologies



55 Merged Ontologies



Consensus Ontology



Trends...

Second Generation Web

Hardware:

- Connect all devices, sensors, objects, (and PCs)

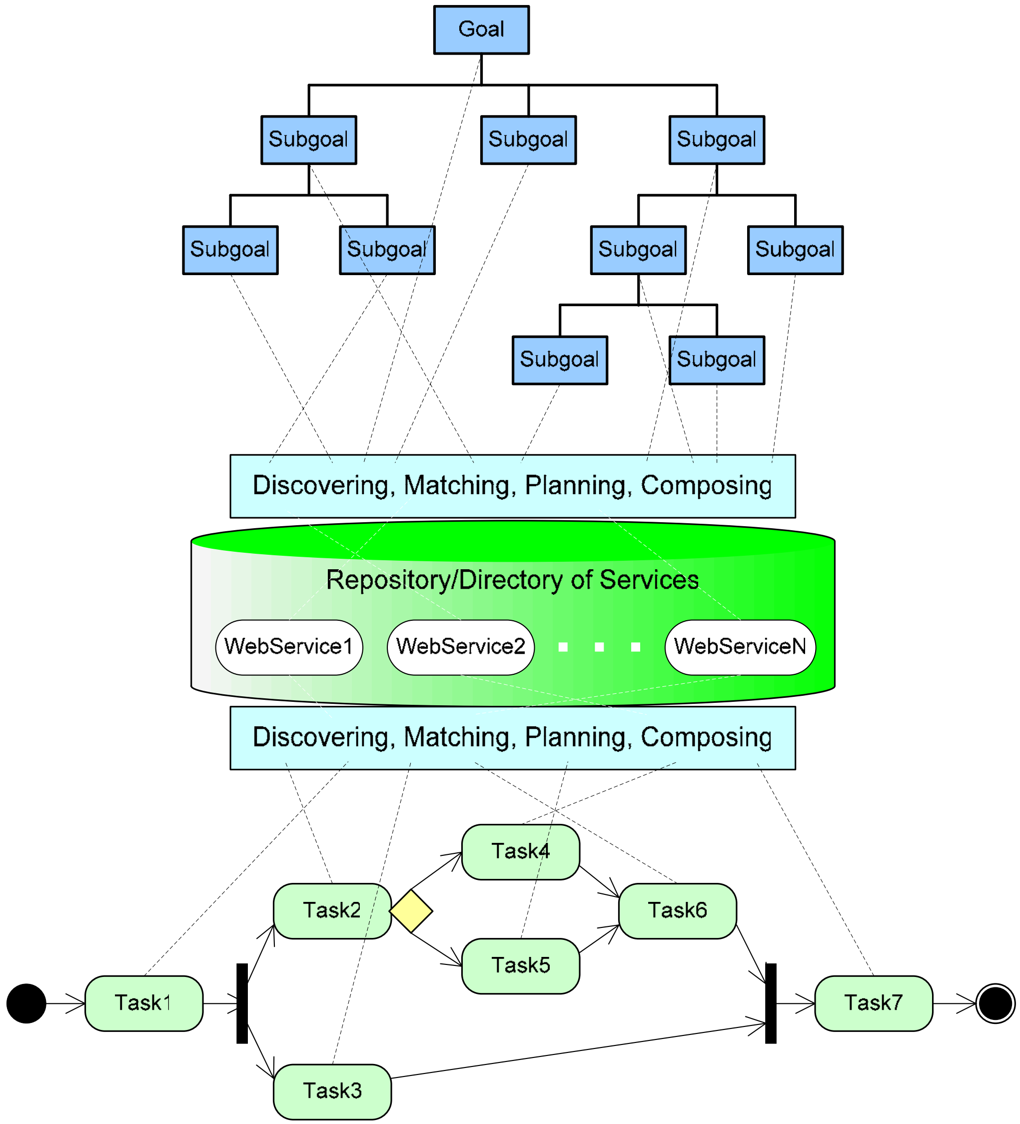
Software:

- Create a Web service for each

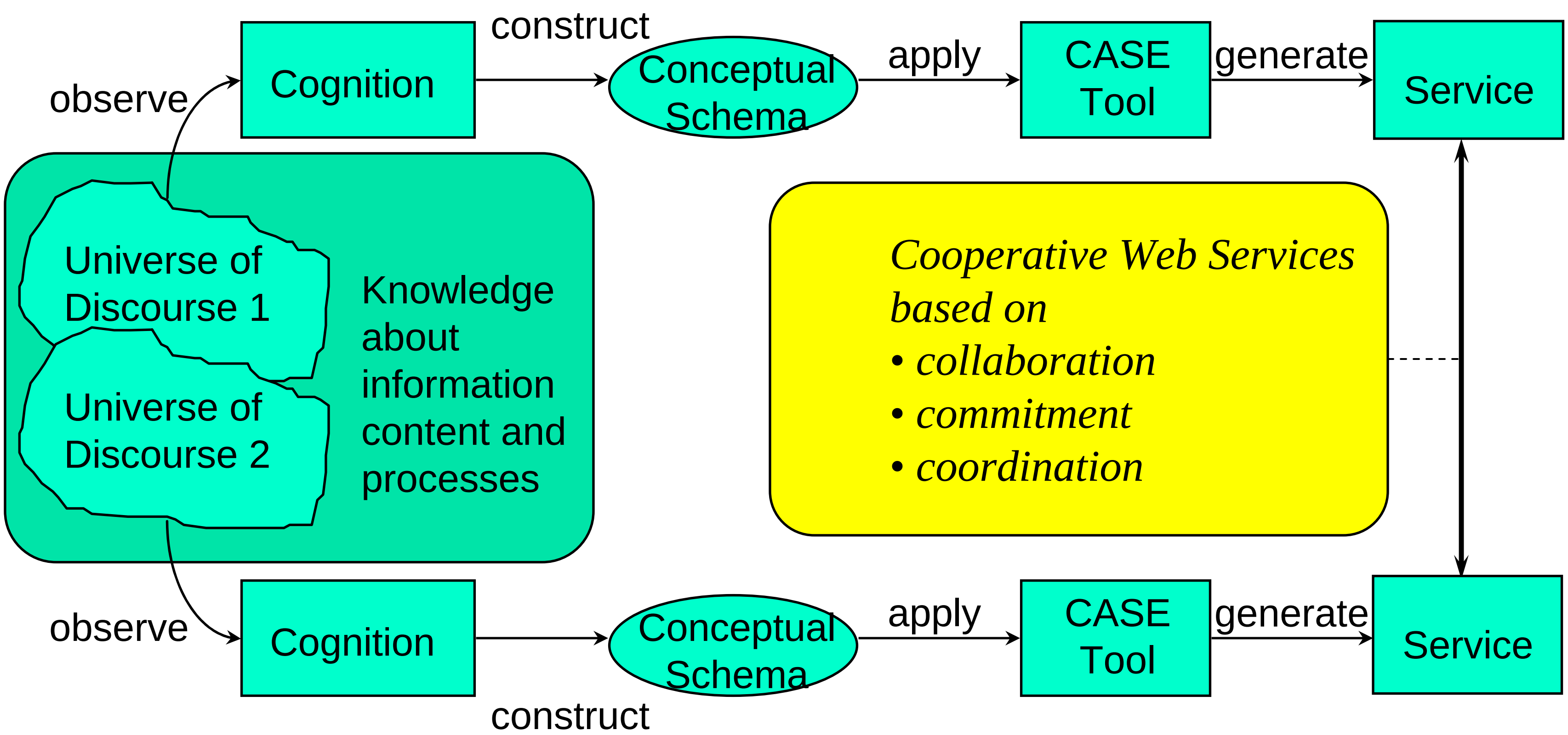
Use Semantic Web (OWL-S) to glue behaviors into a composite; this is programming-in-the-large (or SOC)

How do we get our desired aggregate behavior?

Engineering a SOC System



Cooperating Web Services



Requirements for Service-Oriented Computing

- Loose coupling
- Implementation neutrality
- Flexible configurability
- Long lifetime
- Coarse granularity
- Teams

Must capture patterns of semantic and pragmatic constraints governing service compositions

Complementary Features

Service-Oriented Computing	Multiagent-Based Computing
▪ Encapsulation ▪ Autonomy ▪ Distribution ▪ Interaction via messaging ▪ Mutual understanding ▪ Ontologies for objects and processes	▪ Encapsulation ▪ Autonomy ▪ Distribution ▪ Interaction via messaging ▪ Proactivity ▪ Negotiation ability

Fundamental Abstractions

Concept	Procedural Language	Object Language	Agent-Oriented Language	Multiagent-Based Service-Oriented Language
Abstraction	Type	Class	Agent type	Service / Interaction
Building Block	Instance and Data	Object	Agent	Commitment
Computation model	Procedure/Call	Method/Message	Perceive/Reason/Act	Commitment algebra
Design Paradigm	Tree of procedures	Interaction patterns	Speech acts, goals beliefs, intentions	Cooperative & normative behavior; QoS
Architecture	Functional decomposition	Inheritance and Polymorphism	Managers and peers	Requestors and providers
Modes of Behavior	Coding	Designing and using	Tasking	Contracting and performing
Terminology	Implement	Engineer	Activate	Engage

Trends...

Third Generation Web

Hardware:

- Now that everything is connected...

Software:

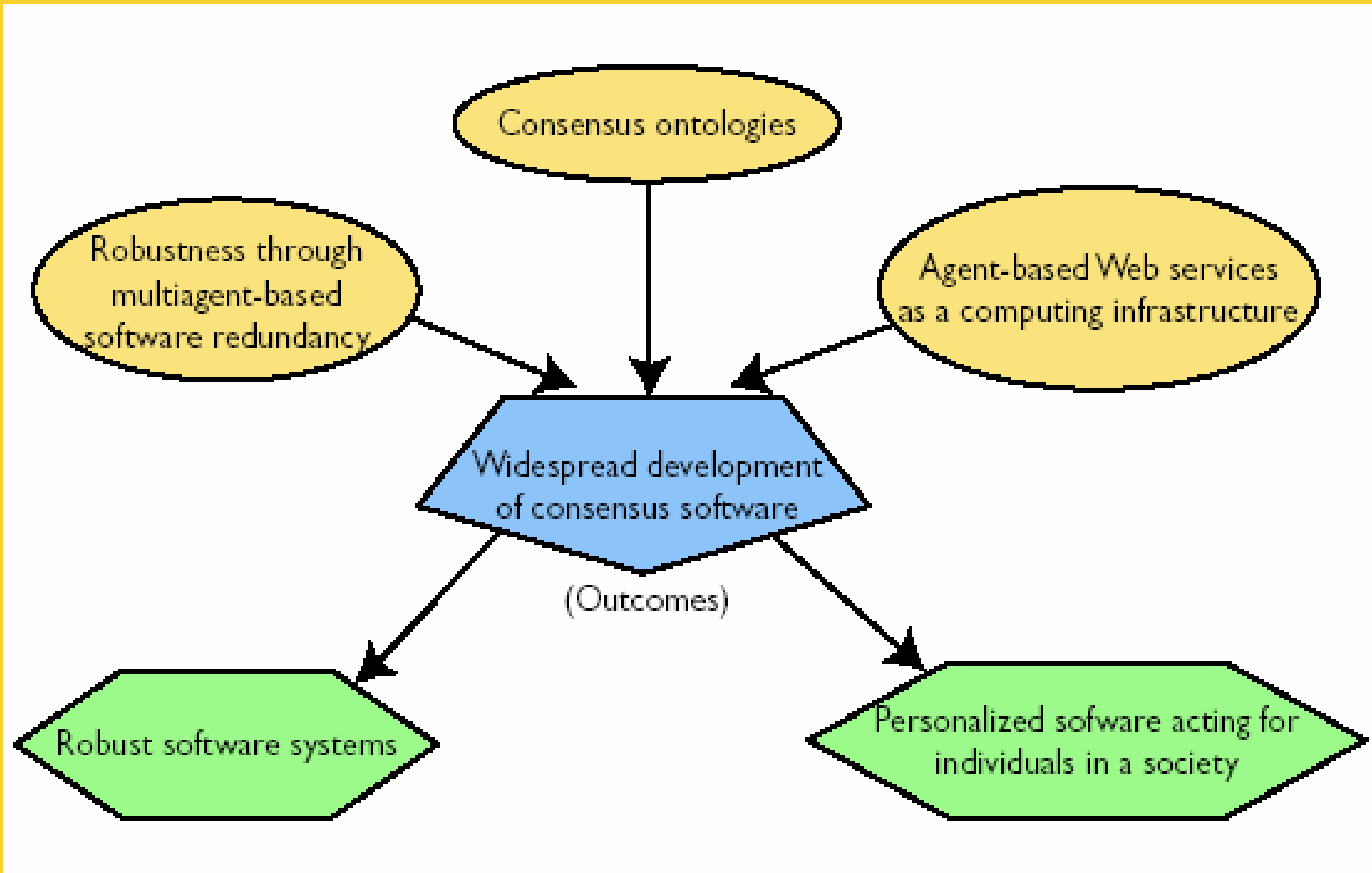
- Create an agent to wrap and represent every Web page and Web service

Use ??? to create new behaviors dynamically and form a new consensus about each; e.g., who programs the lights at a highway intersection?

How do we get consensus behavior?

How do we get consistent and correct behavior?

Consensus Software and Services



Coherence and Commitments

- Coherence is how well a system behaves as a unit. It requires some form of organization, typically hierarchical
- Social commitments among agents are a means to achieve coherence. An agent's commitment to another agent or to its society
  - Is unidirectional
  - Arises within a well-defined scope or *context*
  - Is revocable with restrictions
  - Enables coordination through the ordering and occurrence of actions by the agents

Commitments for Contracts

Commitments capture contracts. Importantly, commitments are

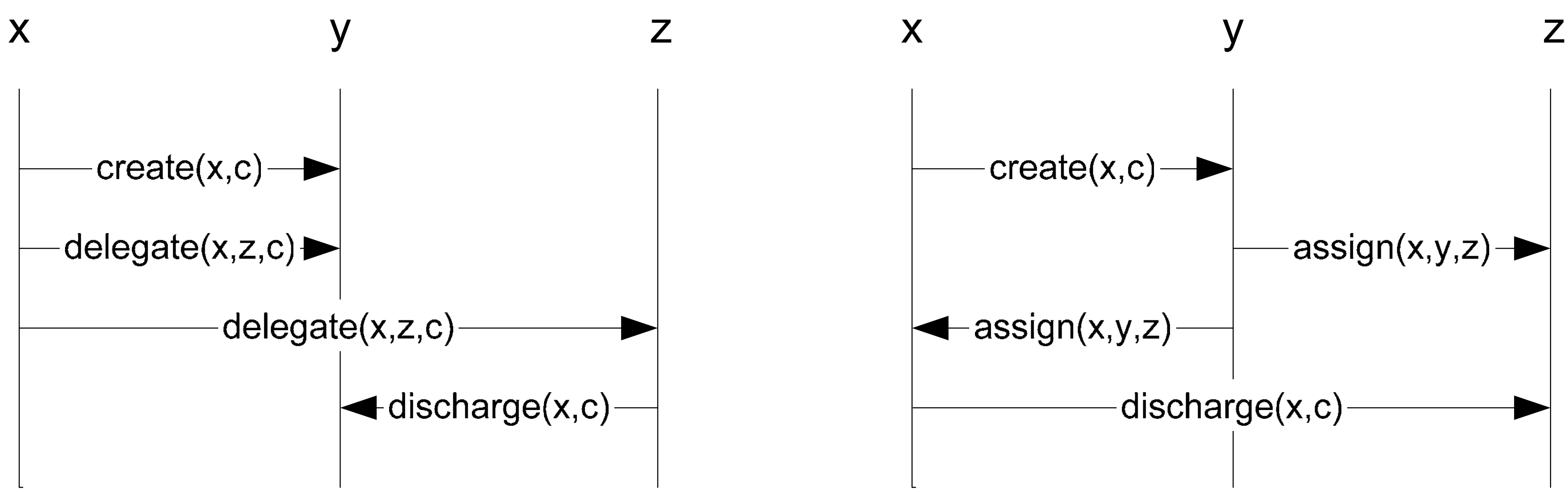
- Public (unlike beliefs and intentions)
- Can be used as the basis for compliance
- Contracts apply between parties, in a context
- Other approaches:
  - Are single-agent focused, e.g., deontic logic
  - Don’t handle organizational aspects of contracts
  - Don’t accommodate manipulation of contracts

Manipulating Commitments

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

Message Patterns for Commitment Operations

Ensure that information about commitment operations flows to the right parties, to enable local decisions



SoCom: Sphere of Commitment

An organization that provides the context or scope of commitments among relevant roles (*abstract*) or agents (*concrete*)

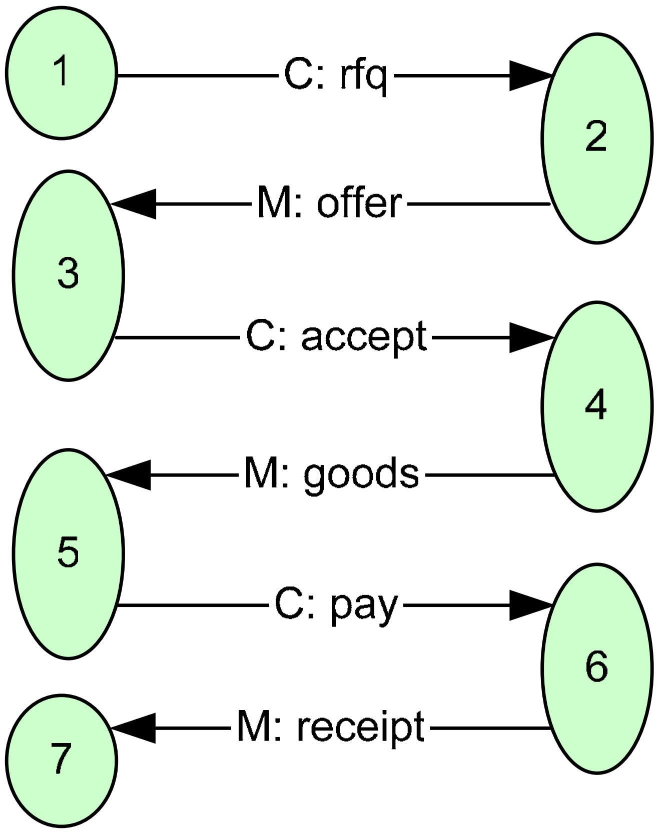
- Serves as a *witness* for the commitment, i.e., knows that the commitment exists
- Helps validate commitments and test for compliance
- Offers compensations to undo members' actions

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)

Example: NetBill Protocol

FSM Representation

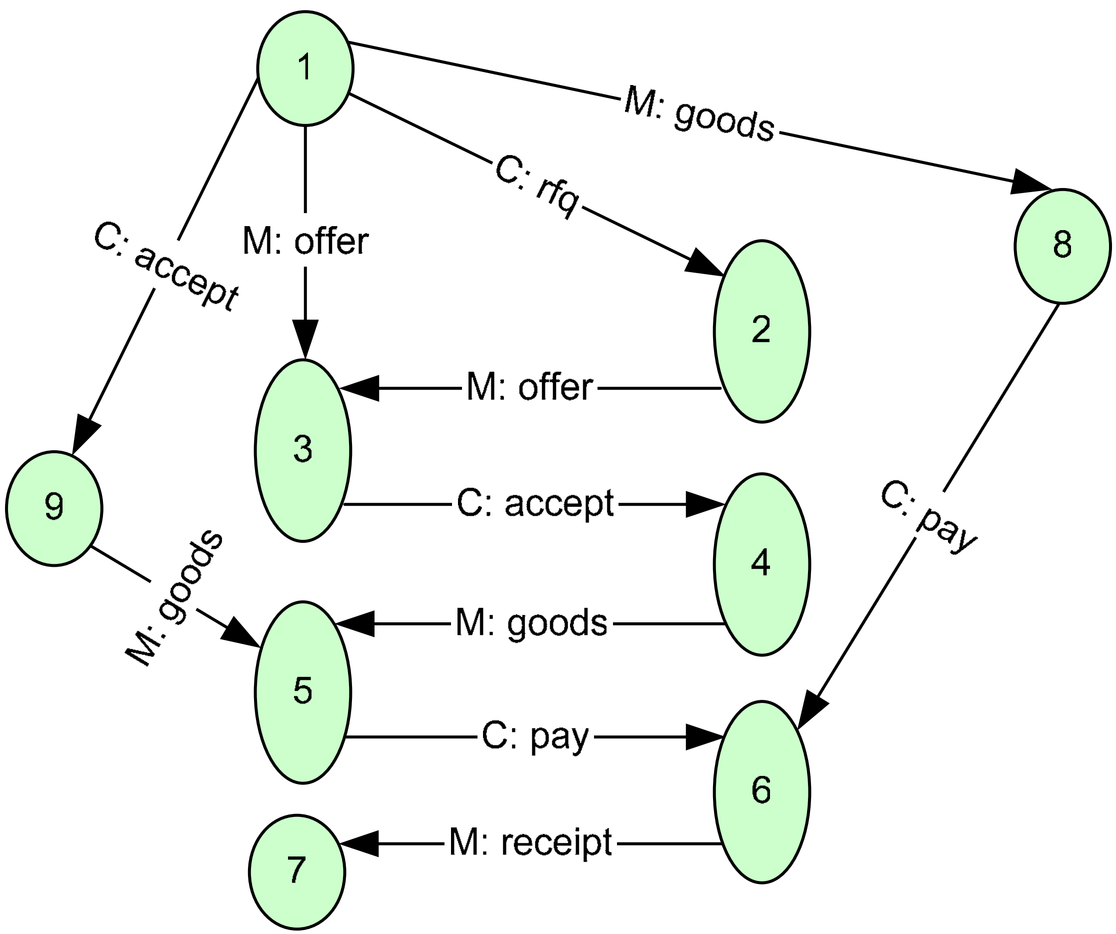


Some variations:

- The merchant may start the protocol by sending a quote
- The customer may send an accept prior to offer
- The merchant may send the goods prior to accept

These variations are not allowed by the FSM

NetBill Enhanced by CMs



Meanings:

1. true
2. request
3. offer
4. $C_m$goods ∧ accept ∧ promiseReceipt
5. goods ∧ $C_c$pay ∧ promiseReceipt
6. goods ∧ pay ∧ $C_m$receipt
7. goods ∧ pay ∧ receipt

8. goods ∧ promiseReceipt

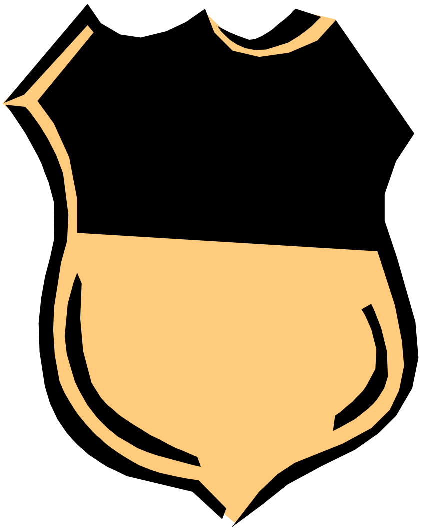
9. accept

Final state: No open commitments remain

Compliance with Protocols

In an open environment, agents are contributed by different vendors and serve different interests

- How can an application 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



Verifying Compliance with Commitment Protocols

- Specification
  - models based on *potential causality*
  - commitments based on branching-time TL
- Run-time Verification
  - respects design autonomy
  - uses TL model-checking
  - local verification based on observed messages

Run-Time Compliance Checking

- An agent can keep track of
  - its pending commitments
  - commitments made by others that are not satisfied
- It uses this local model to see if a commitment has been violated
- An agent who benefits from a commitment can always determine if it was violated

Social Commitments and Law

Social commitments are a *legal abstraction*, because they subsume directed obligations as well as the Hohfeldian concepts. Importantly, they are

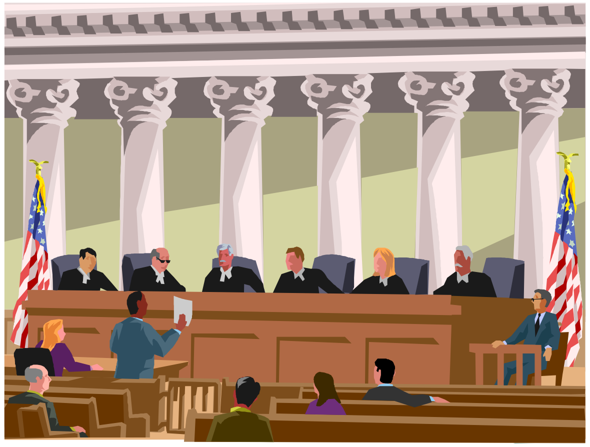
- Public
- Can be used as the basis for compliance

The law involves the interactions of legal entities (people, corporations, governmental agencies) with one another, often concerning the creation and manipulation of contracts

Ethical Abstractions

A large collection of agent-based Web services can be viewed as a society, where its member agents must have an ethics and a philosophy. This requires the development of components for

- Deontological or teleological ethics
- Consequentialism
- Duties
- Obligations
- Applying ethics



Motivation

The ethical abstractions help us specify agents who act appropriately

- Intuitively, ethics is just the basic way of distinguishing right from wrong
- Unfortunately, ethical theories are useful for justification, not deliberation
- It is difficult to separate ethics entirely from legal, social, or even economic considerations

Applying Ethics

- The deontological theories
  - Are narrower
  - Ignore practical consideration
  - But are only meant as incomplete constraints (out of all right actions, the agent can choose any)
- The teleological theories
  - Are broader
  - Include practical considerations
  - But leave fewer options for the agent, who must always choose the best available alternative

Trusted Behavioral Framework

Start with a layered deliberative reasoning architecture

Provides a framework for reasoning about behavior

Middle layers guide agent's interactions with others

For handling immediate events

Philosophical Principles

- Philosophical principles should provide guidance with respect to:
  - Choices affecting the overall mission
  - Choices affecting interaction among agents
  - Choices affecting the agent’s welfare
  - Choices affecting efficiency

Philosophical Principles (adapted from Asimov)

Seven proposed principles:

1. Do not harm the mission
2. Do not harm mission participants
3. Do not harm yourself
4. Make rational decisions
5. Follow established conventions
6. Make rational progress towards goals
7. Operate efficiently

Philosophical Principles

- We must design agents whose action choices are guided by their ethical philosophy
- Possible approaches:
  - deontological approach (morally right)

    "Agents should act as they think other agents should act"
  - teleological approach (ends justify the means)

    "Agents should act to maximize the good of the outcome"

Long-Term Research Objectives

- Robust software constructed out of large numbers of agents with redundant functionality
- Consensus software for societal systems
  - Anyone can publish information as Web pages
  - Anyone should be able to contribute to the behavior of societal systems, such as for power distribution, traffic control, telecommunications regulation, etc.
- Large-scale ethical and philosophical agent societies

Motivation: Robust Software and Services

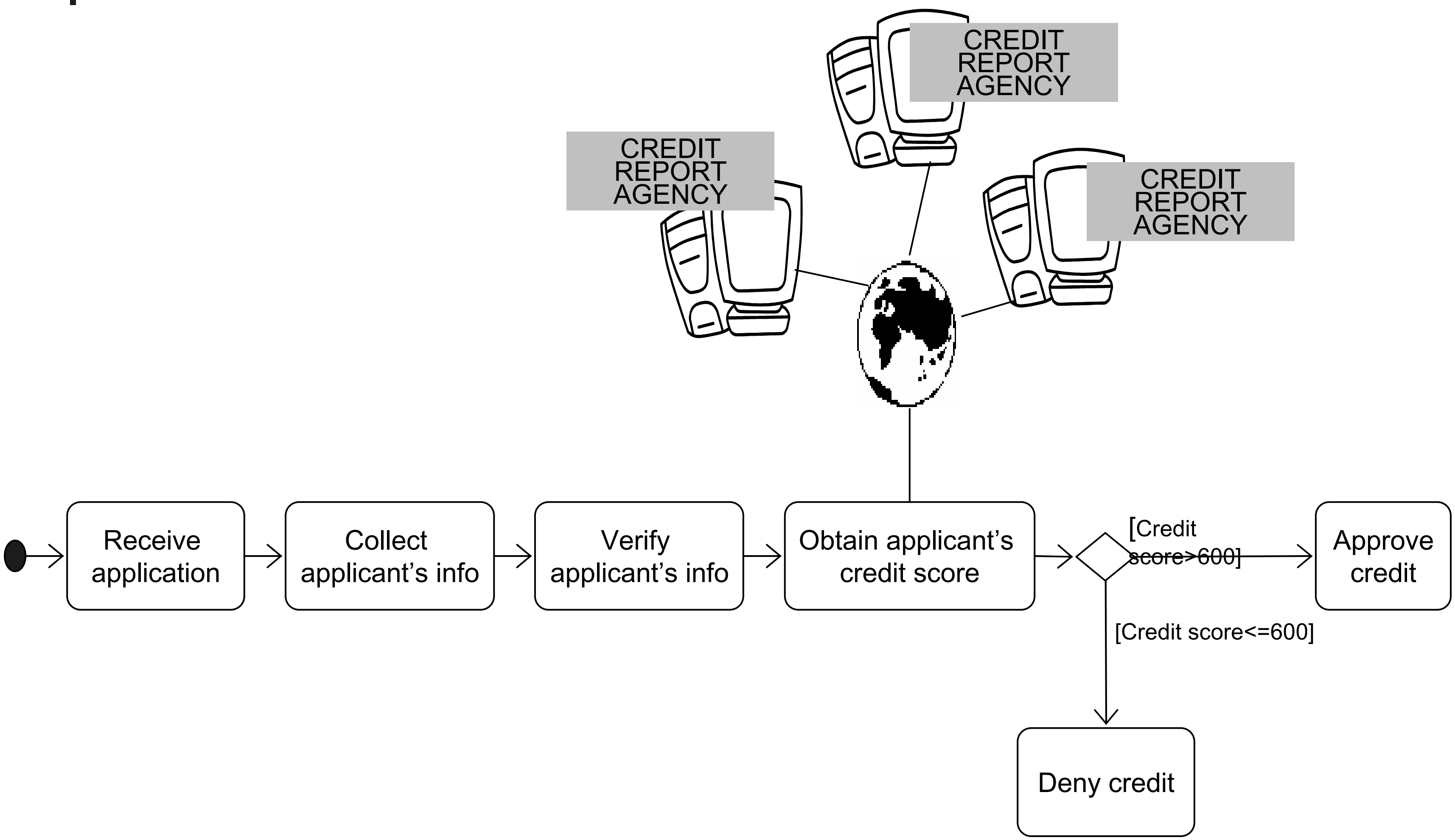
- My neighbor is rebuilding a wall that fell down. To keep it from falling down again, he is using twice as many bricks

- Microsoft put twice as much code in Windows 98 as in Windows 95, so that it would never crash

Robustness through Multiple Viewpoints

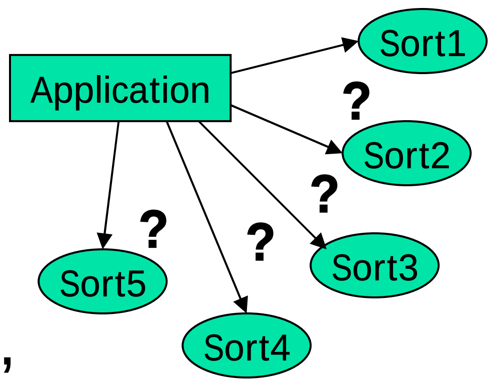
- Errors will **always** be in complex systems
- Errors can be detected and corrected through redundancy
- Hypothesis: agents are the right level of granularity and abstraction for adding redundancy
- Agents can vote, present different viewpoints, cover for agents that misbehave, provide checks and balances

Redundancy Is the Basis for Robustness



Advanced Composition

- 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 (cont.)

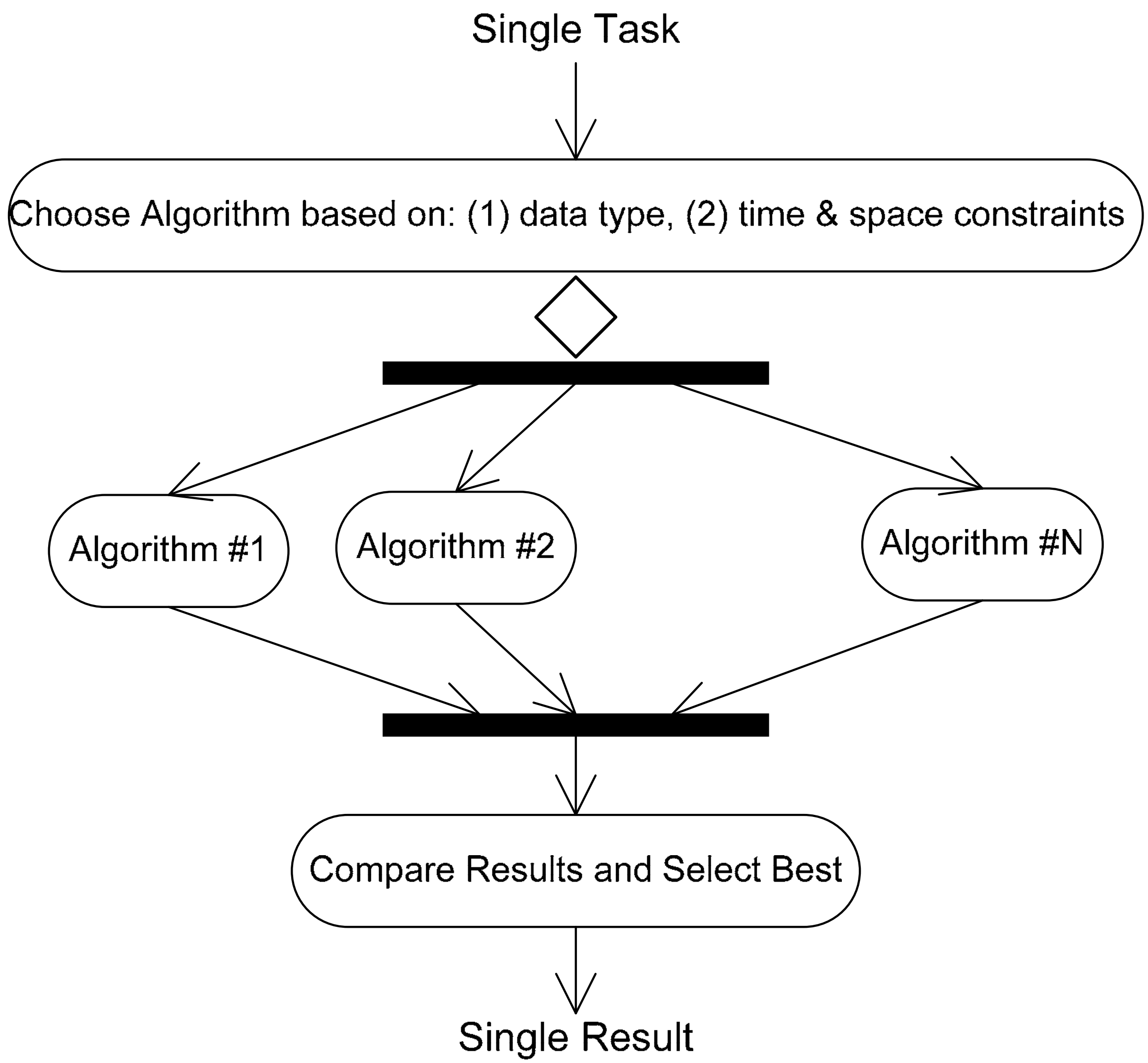
- Possible approaches
  - Application invokes services randomly until one succeeds
  - Application ranks services and invokes them in order until one succeeds
  - Application invokes all services and reconciles the results
  - Application contracts with one service after requesting bids
  - Services 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

Agent-Based N-Version Robustness

Each agent would have to know its role in this system as well as:

- Something about its own algorithm, such as its time/space complexity, and input/output data structures
- Something about other agents, such as their time and space complexity and reliability
- How to negotiate
- How to communicate
- How to compare results
- How to manage reputations and trust

Architecture for Agent-Based N-Version Robustness



Experimental Results

- We asked 25 students to each write sorting and list-reversing algorithms
- Each algorithm was converted to an agent, with a wrapper written in Jade
- The results show that
  - The same wrapper can be used for algorithms in different domains
  - More agents (algorithms) produce better results than any one alone

Self-Correcting Software

- If we consider an agent's behavior to be either correct or incorrect (binary), then, based on a notion of Hamming distance for error correcting codes, $4m$ agents can detect m - 1 errors in their behavior and can correct (m - 1)/2 errors

Obstacles

- Predicting behavior without explicit control
- Achieving controlled behavior without explicit control
- Achieving trust and acceptance without explicit control

Metrics

- Mean-time-between-failures as a function of number of bugs and degree of redundancy
- Increase in number of people contributing to societal software systems
- Degree of success in achieving mission goals in face of changes to system environment

Summary Research Hypothesis

Robust software and services can be achieved through:

- Redundancy in planning
- Redundancy in execution
- Decentralization
- Adoption of a trusted behavioral framework

Taken together this implies highly distributed autonomous decision-making and task execution guided by agent-societal laws

Advertisement for a New Book...

Munindar Singh and Michael Huhns:
“Service-Oriented Computing: Semantics, Processes, Agents,”
John Wiley & Sons, Inc., 2005.