Electronic Commerce Technologies

CSC 513, Spring 2008

Munindar P. Singh
singh@ncsu.edu

Department of Computer Science
North Carolina State University

Module 1: Introduction

- Scope
- Grading
- Policies

Scope of this Course

- Directed at computer science students
- Emphasizes concepts and theory
- Requires a moderate amount of work
- Includes necessary tool-specific details
- *Intensive!*

Electronic Business

- B2C: retail, finance
- B2B: supply chains (more generally, supply networks)
- Different perspectives
  - Traditionally: merchant, customer, dealmaker
  - Trends: collaboration among various parties; virtual enterprises; coalition formation

Main technical consequence: interacting across enterprise boundaries or administrative domains

Properties of Business Environments

- Traditional computer science deals with closed environments
- Business environments are *open*
  - Autonomy: independent action (how will the other party act?)
  - Heterogeneity: independent design (how will the other party represent information?)
  - Dynamism: independent configuration (which other party is it?)
    - Usually, also large scale
- Need flexible approaches and arms-length relationships

Autonomy

Independence of business partners

- Sociopolitical or economic reasons
  - Ownership of resources by partners
  - Control, especially of access privileges
  - Payments
- Technical reasons: opacity with respect to key features, e.g., precommit
  - Model components as autonomous to simplify interfaces “assume nothing”
  - Model components as autonomous to accommodate underlying exceptions

Heterogeneity

Independence of component designers and system architects

- Historical reasons
- Sociopolitical reasons
  - Differences in local needs
  - Difficulty of achieving agreement
- Technical reasons: difficulty in achieving homogeneity
  - Conceptual problems: cannot easily agree
  - Fragility: a slight change can mess it up

Dynamism

Independence of system configurers and administrators

- Sociopolitical reasons
  - Ownership of resources
  - Changing user preferences or economic considerations
- Technical reasons: difficulty of maintaining configurations by hand
  - Same reasons as for network administration
  - Future-proofing your system

Coherence

- Global information (cutting across administrative domains) is essential for coherence
  - Locations of services or agents
  - Applicable business rules
  - Data, schemas, constraints

Locality

A way to deal with openness

- Global information causes
  - Inconsistencies
  - Difficulties in maintenance
- Approach: relax global constraints
  - *Lazy:* obtain global knowledge as needed
  - *Optimistic:* correct rather than prevent violations
  - *Inspectable:* specify rules for when, where, and how to make corrections

Integration

Yields with one integrated entity

- Yields central decision making by homogeneous entity
- Requires resolving all potential inconsistencies ahead of time
- Fragile and must be repeated whenever components change

Interoperation

Ends up with the original number of entities working together

- Yields decentralized decision making by heterogeneous entities
- Resolves inconsistencies incrementally
- Potentially robust and easy to swap out partners as needed

Also termed “light integration” (bad terminology)

Example: Selling

Update inventory, take payment, initiate shipping

- Record a sale in a sales database
- Debit the credit card (receive payment)
- Send order to shipper
- Receive OK from shipper
- Update inventory

Potential Problems

- What if the order is shipped, but the payment fails?
- What if the payment succeeds, but the order was never entered or shipped?
- What if the payments are made offline, i.e., significantly delayed?

In a Closed Environment

- Transaction processing (TP) monitors ensure that all or none of the steps are completed, and that systems eventually reach a consistent state
- But what if the user is disconnected right after he clicks on OK? Did order succeed? What if line went dead before acknowledgment arrives? Will the user order again?
- The TP monitor cannot get the user into a consistent state

In an Open Environment: 1

- Reliable messaging (asynchronous communication, which guarantees message delivery or failure notification)
- Maintain state: retry if needed
- Detect and repair duplicate transactions
- Engage user about credit problems

Matter of policies to ensure compliance

In an Open Environment: 2

- Not immediate consistency
- Eventual “consistency” (howsoever understood) or just coherence
- Sophisticated means to maintain shared state, e.g., conversations

Challenges

- Information system interoperation
- Business process management
- Exception handling
- Distributed decision-making
- Personalization
- Service selection (location and assessment)

Information System Interoperation

Supply chains: manage the flow of materiel among a set of manufacturers and integrators to produce goods and configurations that can be supplied to customers

- Requires the flow of information and negotiation about
  - Product specifications
  - Delivery requirements
  - Prices

Business Processes

- Modeling and optimization
  - Inventory management
  - Logistics: how to optimize and monitoring flow of materiel

Exception Conditions

Virtual enterprises to construct enterprises dynamically to provide more appropriate, packaged goods and services to common customers

- Requires the ability to
  - Construct teams
  - Enter into multiparty deals
  - Handle authorizations and commitments
  - Accommodate exceptions
- Real-world exceptions
- Compare with PL or OS exceptions

Distributed Decision-Making: 1

Manufacturing control: manage the operations of factories

- Requires intelligent decisions to
  - Plan inflow and outflow
  - Schedule resources
  - Accommodate exceptions

Distributed Decision-Making: 2

Automated markets as for energy distribution

- Requires abilities to
  - Set prices, place or decide on others' bids
  - Accommodate risks
- Pricing mechanisms for rational resource allocation

Personalization

Consumer dealings to make the shopping experience a pleasant one for the customer

- Requires
  - Learning and remembering the customer's preferences
  - Offering guidance to the customer (best if unintrusive)
  - Acting on behalf of the user without violating their autonomy

Service Selection

What are some bases for selecting the parties to deal with?

- Specify services precisely and search for them
  - How do you know they do what you think they do (ambiguity)?
  - How do you know they do what they say (trust)?
- Recommendations to help customers find relevant and high quality services
  - How do you obtain and aggregate evaluations?

Module 2: Web Technologies in Brief

A quick look at web programming technologies

- JSP
- Servlets
- Enterprise Java Beans

Static versus Dynamic Pages

- Static: known ahead of time
- Dynamic: created on demand (or created frequently)
  - Depend on user request
  - Depend on backend data
  - Depend on processes executing in the backend system
  - Involve an application of policies

Shallow and Deep Webs: 1

Shallow Web: primarily static pages that we ordinarily access

- Links from a page to another are fixed
- Links can be followed without need of a special account
- Therefore, potentially crawled and indexed by search engines

Shallow and Deep Webs: 2

Deep Web: dynamic pages extracted from databases or applications

- Where most of the world's data is
- Typically residing in intranets or extranets
- Often require an account to access
- Require custom queries rather than following links
- Therefore, largely missed by search engines

Common Gateway Interface

CGI is a way for invoking processes from a Web server

- Create an OS process for every request
- Coded in any language; generally not safe languages (Web hosting companies may limit the functionality on shared hardware)
- Poor performance
- No support for threading

Server-Side Scripting Languages

Typified by PHP

- Powerful and popular: recall the LAMP acronym
- Produces a page, which is sent to the requester
- Lacks the type safety of Java
- Better suited when display functions dominate

Our intention is to get into message-oriented middleware

Servlet

An entry point for a service request that comes over the Web

- Capture business logic of the “controller”
- Invoke a backend component
- Generally the model part of the functionality is split off into Enterprise Java Beans

Servlet Functions

- Read in data sent and action requested by client: use a “request” object that provides a handle to the current HTTP request
- Perform necessary computations
- Produce a response for the client: use a “response” object that provides a handle to the HTTP response for the current HTTP request

Servlet Views: 1

A servlet is a Java program written according to a certain standard

- Provides certain APIs, which the program assumes
- Requires that a class HttpServlet be extended
- Requires that a method such as doGet be implemented, overriding the eponymous method in the above class

Servlet Views: 2

A servlet is a computational entity

- Analogous to a running thread of control and which might initiate one or more transactions
- Could be coded in some other method, e.g., as a JSP

Servlet Snippet

```
1 public class OrderServlet extends HttpServlet {
    public void doGet(HttpServletRequest req,
                      HttoServletResponse resp)
        throws ServletException, IOException {
      resp.setContentType("text/html");
6     PrintWriter out = resp.getWriter();
      out.println("<html>...</html>");
    }
  }
```

Java Server Pages or JSP

These describe a view (termed “page” as in a Web page) to be rendered by a client browser

- Provides support for a variety of markup (conventionally termed “tags”)
- Tags are customizable
- Separate the roles of user interface designers from programmers
- In simple terms, Java code embedded in HTML
- Alternative way to create a Servlet

JSP Snippet

```
<!DOCTYPE HTML PUBLIC "..." >
<html>
<head> ... </head>
<body>
<h2>Course Page</h2>
  <%= package.class.method(args) %>
</body>
```

Servlet Container: 1

- A system module that hosts servlets
- Corresponds to a process (or exists within an application server process); each servlet instance is a thread in the container
- Runs in conjunction with a Web server and provides
  - Remote method invocation
  - Threading
  - Connection pool management: many servlet instances access the same or few databases by sharing connection overhead

Servlet Container: 2

- Separates the functions of programmer and administrator
- Behaves like an operating system for servlets
- Shields servlets from each other, and keeps different instances apart
- Applies policies for controlling user access

Servlet Container: 3

For example, Tomcat

- Typically simpler than a full-blown application server, which also supports EJBs, for example, JBoss
- Sometimes considered a part of an application server: many containers may exist within one application server
- In terms of source code, the containment could be in the other direction: JBoss used to come packaged with Tomcat

Packaging Web Components

- Each container product can dictate its way of packaging servlets and other resources
- The package should include all the resources the servlet needs
- Never refer to external resources (that is, use no absolute paths within a servlet), yielding improved
  - Security
  - Portability
  - Good containers prevent a servlet from referring to external resources
- Put the packaging intelligence in the build script

Recommend: single archive for entire deliverable

Enterprise Java Beans

A kind of business component, meant to be hosted by a suitable container

- Capture business logic of the “model”
- Mediate between clients and backend systems
- Of three main kinds
  - Entity beans
  - Session beans
  - Message-driven beans $\approx$ interface to MoM

Containers and EJBs: 1

A container

- Is an environment on an application server that hosts Enterprise Java Beans
- Defines a contract between server vendors and EJB programmers
- Invokes specific “management” methods on EJBs, which the bean programmer must supply
  - These methods include ejbCreate() and such

Containers and EJBs: 2

- The EJB programmer can pretend that his or her EJB is the only component that is executing on the container
- A container provides important functionality to a programmer, such as
  - Remote method invocation
  - Threading
  - Thread pool management
- Write your code normally; the container supplies the thread management for free

Entity Beans

Correspond to database objects (typically tuples in relational database tables)

- Offer persistence of entities
- Long-lived
- Mapping to databases may be
  - Container-managed persistence (CMP): automatically taken care of
  - Bean-managed persistence (BMP): programmer takes care of it

Session Beans

Correspond to ongoing interactions

- Nonpersistent: short-lived
- Help manage conversations with clients
- Classically two-party conversations

Stateless Session Beans

The invocations are logically independent

- Single-method call
- No conversational state maintained by bean
- Other objects (or state information) may be referenced by the bean, e.g., to manage database connections, but the container may arbitrarily discard and recreate such information
- Easy to manage: use a pool of beans to serve clients, because they are mutually indistinguishable

Is it possible to carry out a multistep conversation using such beans?

Stateful Session Beans

As in shopping carts stored on a server

- Multistep conversational state
- Suitable for things like shopping carts
- Harder to manage: imagine a server implemented on a cluster

How many parties can there be to such a conversation?

Context

- Encapsulates the computational environment in which the bean functions
- Could be used to get a handle on transactional (such as whether this bean method is being invoked within a transaction) or security objects (such as who is the principal behind the current request)

The container calls methods such as setSessionContext, which are provided by the bean (often trivially implemented)

Using EJBs: 1

Mediated by two main proxy objects

- *Stub:* client-side proxy
- *Skeleton:* server-side proxy
- Each implements the *remote* interface of the EJB

Also a *local* interface to save network overhead when not needed

Using EJBs: 2

A factory or *home* object

- Create
- Find, if already created (and with a persistent identity)
- Remove

Also a *local home* interface to save network overhead when not needed

JNDI

Java Naming and Directory Interface

- To use a bean, our code must call an object whose identity and location are established only at runtime
- Hence, need for a directory system
- JNDI is the Java approach for directories; usable for purposes besides beans
- Needs a context within which it performs a search: usually boilerplate code

Important Methods for Session Beans

- ejbCreate(): required; can also define versions with arguments for stateful beans
- ejbPassivate() and ejbActivate(): trivial for stateless, but for stateful, these save and restore state
- ejbRemove(): free all resources

Important Methods for Entity Beans

- ejbCreate()
- ejbLoad() and ejbStore(): help synchronize bean with database
- ejbFindByPrimaryKey(): find or create bean
- getPrimaryKey(): to identify the underlying database object

EJB Trend

- Way too much complexity in the present (up to 2.1) standards
- Movement toward *POJOs*: Plain Old Java Objects
- EJB 3.0 is heading toward a greatly simplified standard

Module 3: Architecture

In the sense of information systems

- Web architectures
- Enterprise architectures
- Interoperation architectures
- Message-oriented middleware

Architecture Conceptually

- How a system is organized
- An over-used, vaguely defined term
  - Software architecture
  - Standards, e.g., Berners-Lee's "layer cake"
  - May include processes
  - May include human organizations

Understanding Architecture

- Two main ingredients of a system
  - Components
  - Interconnections
- *Openness* entails specifying the interconnections cleanly
  - Physical components disappear
  - Their logical traces remain
- *Information environments* mean that the interconnections are protocols

Understanding Protocols

- Protocols encapsulate interactions
  - *Connect:* conceptual interfaces
  - *Separate:* provide clean partitions among logical components
- Wherever we can identify protocols, we can
  - Make interactions explicit
  - Enhance reuse
  - Improve productivity
  - Identify new markets and technologies
- Protocols yield standards; their implementations yield products

Architectural Examples

When viewed architecturally, each logical component class serves some important function

- Power: UPS
- Network connectivity
- Storage: integrity, persistence, recovery
- Policy management
- Decision-making
- Knowledge and its management

What are some products in the above component classes?

IT Architectures

The term is used more broadly in serious IT settings

- The organization of a system
- The human organization in a system taken broadly
- The extensibility and modification of a system
- Even the processes by which a system is updated or upgraded
- Sometimes even nontechnical aspects, such as flows of responsibility

Enterprise Models: 1

- Capture static and dynamic aspects of enterprises
- Document information resources
  - Databases and knowledge bases
  - Applications, business processes, and the information they create, maintain, and use

Enterprise Models: 2

- Capture organizational structure
- Document business functions
  - Rationales behind designs of databases and knowledge bases
  - Justifications for applications and business processes

Enterprise Models: 3

By being explicit representations, models enable

- Integrity validation
- Reusability
- Change impact analysis
- Automatic database and application generation via CASE tools

Enterprise Architecture Objectives

At the top-level, to support the business objectives of the enterprise; these translate into

- Accommodating change by introducing new
  - Applications
  - Users
  - Interfaces and devices
- Managing information resources
  - Preserving prior investments, e.g., in legacy systems
  - Upgrading resources
- Developing blueprints for IT environment: guiding resource and application installation and decommissioning

Enterprise Architecture Observations

Continual squeeze on funds, staffing, and time available for IT resources

- Demand for rapid development and deployment of applications
- Demand for greater ROI
- Essential tension
  - Need to empower users and suborganizations to ensure satisfaction of their local and of organizational needs
  - Ad hoc approaches with each user or each suborganization doing its own IT cause failure of interoperability

Enterprise Architecture Principles

Business processes should drive the technical architecture

- Define dependencies and relationships among users and suborganizations of an organization
- Message-driven approaches are desirable because they decouple system components
- Event-driven approaches are desirable because they help make a system responsive to events that are potentially visible and significant to users

Architecture Modules: Applications

Often most visible to users

- Application deployment
- Data modeling and integrity
- Business intelligence: decision support and analytics
- Interoperation and cooperation
  - *Ontologies:* representations of domain knowledge
- Component and model repositories
- Business process management

Architecture Modules: Systems

Functionality used by multiple applications

- Middleware: enabling interoperation, e.g., via messaging
- Identity management
- Security and audit
- Accessibility
- Policy repositories and engines

Architecture Modules: Infrastructure

- Connectivity
- Platform: hardware and operating systems
- Storage
- System management

Enterprise Functionalities: 1

It helps to separate the key classes of functionality in a working software system

- Presentation: user interaction
  - A large variety of concerns about device constraints and usage scenarios
- Business logic
  - Application logic
  - General rules

Enterprise Functionalities: 2

- Data management
  - Ensuring integrity, e.g., entity and referential integrity (richer than storage-level integrity)
  - Enabling access under various kinds of problems, e.g., network partitions
  - Supporting recovery, e.g., application, operating system, or hardware failures

Enterprise Functionalities: 3

Bases for choosing the above three-way partitioning as opposed to some other

- Size of implementations
- Organizational structure: who owns what and who needs what
- Staff skill sets
  - User Interface: usability and design
  - Programming
  - Database
  - Policy tools
- Products available in the marketplace

One-Tier and Two-Tier Architectures

- One tier: monolithic systems; intertwined in the code base
  - Historically the first
  - Common in legacy systems
  - Difficult to maintain and scale up
- Two-tier: separate data from presentation and business logic
  - Classical client-server (or fat client) approaches
  - Mix presentation with business rules
  - Change management

Three-Tier Architecture: 1

- Presentation tier or frontend
  - Provides a view to user and takes inputs
  - Invokes the same business logic regardless of interface modalities: voice, Web, small screen, . . .
- Business logic tier or middle tier
  - Specifies application logic
  - Specifies business rules
    - Application-level policies
    - Inspectable
    - Modifiable

Three-Tier Architecture: 2

- Data tier or backend
  - Stores and provides access to data
  - Protects integrity of data via concurrency control and recovery

Multitier Architecture

Also known as n-tier (sometimes treated synonymously with three-tier)

- Best understood as a componentized version of three-tier architecture where
  - Functionality is assembled from parts, which may themselves be assembled
  - Supports greater reuse and enables greater dynamism
  - But only if the semantics is characterized properly
- Famous subclass: service-oriented architecture

Architectural Tiers Evaluated

The tiers reflect logical, not physical partitioning

- The more open the architecture the greater the decoupling among components
  - Improves development through reuse
  - Enables composition of components
  - Facilitates management of resources, including scaling up
  - Sets boundaries for organizational control
- In a narrow sense, having more moving parts can complicate management
- But improved architecture facilitates management through divide and conquer

XML-Based Information System

Let's place XML in a multitier architecture

How About Database Triggers?

- *Pros:* essential for achieving high efficiency
  - Reduce network load and materializing and serializing costs
  - Leave the heavy logic in the database, under the care of the DBA
- *Cons:* rarely port well across vendors
  - Difficult to introduce and manage because of DBA control
  - Business rules are context-sensitive and cannot always be applied regardless of how the data is modified

Implementational Architecture: 1

Centered on a Web server that

- Supports HTTP operations
- Usually multithreaded

Implementational Architecture: 2

Application server

- Mediates interactions between browsers and backend databases: runs computations, invoking DB transactions as needed
- Provides a venue for the business logic
- Different approaches (CGI, server scripts, servlets, Enterprise JavaBeans)

Implementational Architecture: 3

Database Servers

- Hold the data, ensuring its integrity
- Manage transactions, providing
  - Concurrency control
  - Recovery

Transaction monitors can manage transactions across database systems, but within the same administrative domain

Data Center Architecture

- Demilitarized zone (DMZ)
  - External router
  - Load balancer
- Firewall: only the router can contact the internal network
  - Internal network
  - Web servers
  - Application servers
  - Database servers

Web Architecture

Principles and constraints that characterize Web-based information systems

- URI: Uniform Resource Identifier
- HTTP: HyperText Transfer Protocol
- Metadata must be recognized and respected
  - Enables making resources comprehensible across administrative domains
  - Difficult to enforce unless the metadata is itself suitably formalized

Uniform Resource Identifier: 1

- URIs are abstract
- What matters is their (purported) uniqueness
- URIs have no proper syntax per se
- Kinds of URIs include
  - URLs, as in browsing: not used in standards any more
  - URNs, which leave the mapping of names to locations up in the air

Uniform Resource Identifier: 2

Good design requirements

- Ensure that the identified resource can be located
- Ensure uniqueness: eliminate the possibility of conflicts through appropriate organizational and technical means
- Prevent ambiguity
- Use an established URI scheme where possible

HTTP: HyperText Transfer Protocol

Intended meanings are quite strict, though not constrained by implementations

- Text-based, stateless
- Key verbs
  - Get
  - Post
  - Put
- Error messages for specific situations, such as resources not available, redirected, permanently moved, and so on

ReST: Representational State Transfer

Representational State Transfer

ReST is an architectural style for networked systems that constrains the connectors

- Models the Web as a network of hyperlinked resources, each identified by a URI
- Models a Web application as a (virtual) state machine
- A client selecting a link effects a state transition, resulting in receiving the next page (next state) of the application

Characteristics of ReST

- Client-Server
- Statelessness: requests cannot take advantage of stored contexts on a server
  - What is an advantage of statelessness?
  - Where is the session state kept then?
- Uniform Interface: URIs, hypermedia
- Caching: responses can be labeled as cacheable

Basic Interaction Models

Interactions among autonomous and heterogeneous parties

- Adapters: what are exposed by each party to enable interoperation
  - Sensors $\Leftarrow$ information
  - Effectors $\Rightarrow$ actions
- Invocation-based adapters
- Message-oriented middleware
- Peer-to-peer computing

Invocation-Based Adapters: 1

Distributed objects (EJB, DCOM, CORBA)

- *Synchronous:* blocking method invocation
- *Asynchronous:* nonblocking (one-way) method invocation with callbacks
- *Deferred synchronous:* (in CORBA) sender proceeds independently of the receiver, but only up to a point

Invocation-Based Adapters: 2

Execution is best effort: application must detect any problems

- At most once
- More than once is
  - OK for idempotent operations
  - Not OK otherwise: application must check

Message-Oriented Middleware: 1

- *Queues:* point to point, support posting and reading messages
- *Topics:* logical multicasts, support publishing and subscribing to application-specific topics; thus more flexible than queues
- Can offer reliability guarantees of delivery or failure notification to sender
  - Analogous to store and forward networks
- Some messages correspond to event notifications

Message-Oriented Middleware: 2

- Varies in reliability guarantees
- Usually implemented over databases
- Can be used through an invocation-based interface (i.e., registered callbacks)

Peer-to-Peer Computing

- *Symmetric client-server:* (callbacks) each party can be the client of the other
- *Asynchrony:* while the request-response paradigm corresponds to pull, asynchronous communication corresponds to push
  - Generally to place the entire intelligence on the server (pushing) side
- *Federation of equals:* (business partners) when the participants can enact the protocols they like

Application Servers

Architectural abstraction separating business logic from infrastructure

- Load balancing
- Distribution and clustering
- Availability
- Logging and auditing
- Connection (and resource) pooling
- Security

Separate programming from administration roles

Middleware: 1

Components with routine, reusable functionality

- Abstracted from the application logic or the backend systems
- Any functionality that is being repeated is a candidate for being factored out into middleware
- Enables plugging in endpoints (e.g., clients and servers) according to the stated protocols
- Often preloaded on an application server
- Simplify programmer's task and enable refinements and optimizations

Middleware: 2

Software components that implement architectural interfaces, e.g., transaction, persistence, . . .

- *Explicit:*
  - Invoke specialized APIs explicitly
  - Difficult to create, maintain, port
- *Implicit:*
  - Container invokes the appropriate APIs
  - Based on declarative specifications
  - Relies on request interceptions or reflection

Containers

- Discussed above in connection with EJBs
- Architectural abstraction geared for hosting business components
  - Remote method invocation
  - Threading
  - Messaging
  - Transactions

Message-Driven Beans

A standardized receiver for messages

- Clients can't invoke them directly; must send messages to them
- No need for specialized interfaces, such as home, remote, . . .
- Easy interface to implement: mainly onMessage(), but limited message typing
- Stateless: thus no conversations

Methods for Message-Driven Beans

- onMessage(): define what actions to take when a message arrives on the destination this bean is watching

Module 4: XML Representation

- Concepts
- Parsing and Validation
- Schemas

What is Metadata?

Literally, data about data

- Description of data that captures some useful property regarding its
  - Structure and meaning
  - Provenance: origins
  - Treatment as permitted or allowed: storage, representation, processing, presentation, or sharing
- Markup is metadata pertaining to media artifacts (documents, images), generally specified for suitable parsable units

Motivations for Metadata

Mediating information structure (surrogate for meaning) over time and space

- Storage: extend life of information
- Interoperation for business
- Interoperation (and storage) for regulatory reasons
- General themes
  - Make meaning of information explicit
  - Enable reuse across applications: *repurposing* compare to screen-scraping
  - Enable better tools to improve productivity

Reduce need for detailed prior agreements

Markup History

How much prior agreement do you need?

- No markup: significant prior agreement
- Comma Separated Values (CSV): no nesting
- Ad hoc tags
- SGML (Standard Generalized Markup L): complex, few reliable tools; used for document management
- HTML (HyperText ML): simplistic, fixed, unprincipled vocabulary that mixes structure and display
- XML (eXtensible ML): simple, yet extensible subset of SGML to capture *custom* vocabularies
  - Machine processible
  - Comprehensible to people: easier debugging

Uses of XML

Supporting arms-length relationships

- Exchanging information across software components, even within an administrative domain
- Storing information in nonproprietary format
- XML documents represent semistructured descriptions:
  - Products, services, catalogs
  - Contracts
  - Queries, requests, invocations, responses (as in SOAP): basis for Web services
- Relational DBMSs work for highly structured information, but rely on column names for meaning

Example XML Document

```
<?xml version="1.0"?> <!-- processing instruction -->
<topelem attr0="foo"> <!-- exactly one root -->
 <subelem attr1="v1" attr2="v2">
  Optional text (PCDATA) <!-- parsed character data -->
  <subsubelem attr1="v1" attr2="v2"/>
 </subelem>
 <null_elem/>
 <short_elem attr3="v3"/>
</topelem>
```


Exercise

Produce an example XML document corresponding to a directed graph

Compare with Lisp

List processing language

- S-expressions
- Cons pairs: car and cdr
- Lists as nil-terminated s-expressions
- Arbitrary structures built from few primitives
- Untyped
- Easy parsing
- Regularity of structure encourages recursion

Exercise

Produce an example XML document corresponding to

- An invoice from Locke Brothers for 100 units of door locks at $19.95, each ordered on 15 January and delivered to Custom Home Builders
- Factor in certified delivery via UPS for $200.00 on 18 January
- Factor in addresses and contact info for each party
- Factor in late payments

XML Namespaces: 1

- Because XML supports custom vocabularies and interoperation, there is a high risk of name collision
- A namespace is a collection of names
- Namespaces must be identical or disjoint
  - Crucial to support independent development of vocabularies
  - MAC addresses
  - Postal and telephone codes
  - Vehicle identification numbers
  - Domains as for the Internet
  - On the Web, use URIs for uniqueness

XML Namespaces: 2

```
<!-- xml* is reserved -->
<?xml version="1.0"?>
<arbit:top xmlns="a URI" <!-- default namespace -->
  xmlns:arbit="http://wherever.it.might.be/arbit-ns"
  xmlns:random="http://another.one/random-ns">
 <arbit:aElem attr1="v1" attr2="v2">
   Optional text (PCDATA)
   <arbit:bElem attr1="v1" attr2="v2"/>
 </arbit:aElem>
 <random:simple_elem/>
 <random:aElem attr3="v3"/>
 <!-- compare arbit:aElem -->
</arbit:top>
```

Uniform Resource Identifier

- URIs are abstract
- What matters is their (purported) uniqueness
- URIs have no proper syntax per se
- Kinds of URIs
  - URLs, as in browsing: not used in standards any more
  - URNs, which leave the mapping of names to locations up in the air
- Good design: the URI resource exists
  - Ideally, as a description of the resource in RDDL
  - Use a URL or URN

RDDL

Resource Directory Description Language

- Meant to solve the problem that a URI may not have any real content, but people expect to see some (human readable) content
- Captures namespace description for people
  - XML Schema
  - Text description

Well-Formedness and Parsing

- An XML document maps to a parse tree (if well-formed; otherwise not XML)
  - Each element must end (exactly *once*): obvious nesting structure (one root)
  - An attribute can have at most one occurrence within an element; an attribute's value must be a quoted string
- Well-formed XML documents can be parsed

XML InfoSet

A standardization of the low-level aspects of XML

- What an element looks like
- What an attribute looks like
- What comments and namespace references look like
- Ordering of attributes is irrelevant
- Representations of strings and characters

Primarily directed at tool vendors

Elements Versus Attributes: 1

- Elements are essential for XML: structure and expressiveness
  - Have subelements and attributes
  - Can be repeated
  - Loosely might correspond to independently existing entities
  - Can capture all there is to attributes

Elements Versus Attributes: 2

- Attributes are not essential
  - End of the road: no subelements or attributes
  - Like text; restricted to string values
  - Guaranteed unique for each element
  - Capture adjunct information about an element
  - Great as references to elements

Good idea to use in such cases to improve readability

Elements Versus Attributes: 3

```
<invoice>
 <price currency='USD'>
  19.95
 </price>
</invoice>
```

Or

```
<invoice amount='19.95' currency='USD'/>
```

Or even

```
<invoice amount='USD 19.95'/>
```


Validating

Verifying whether a document matches a given grammar (assumes well-formedness)

- Applications have an explicit or implicit syntax (i.e., grammar) for their particular elements and attributes
  - Explicit is better have definitions
  - Best to refer to definitions in separate documents
- When docs are produced by external software components or by human intervention, they should be validated

Specifying Document Grammars

Verifying whether a document matches a given grammar

- Implicitly in the application
  - Worst possible solution, because it is difficult to develop and maintain
- Explicit in a formal document; languages include
  - Document Type Definition (DTD): in essence obsolete
  - XML Schema: good and prevalent
  - Relax NG: (supposedly) better but not as prevalent

XML Schema

- Same syntax as regular XML documents
- Local scoping of subelement names
- Incorporates namespaces
- (Data) Types
  - Primitive (built-in): string, integer, float, date, ID (key), IDREF (foreign key), . . .
  - simpleType constructors: list, union
  - Restrictions: intervals, lengths, enumerations, regex patterns,
  - Flexible ordering of elements
- Key and referential integrity constraints

XML Schema: complexType

- Specifies types of elements with structure:
  - Must use a compositor if ≥ 1 subelements
  - Subelements with types
  - Min and max occurrences (default 1) of subelements
- Elements with text content are easy
- EMPTY elements: easy
  - Example?
  - Compare to nulls, later

XML Schema: Compositors

- *Sequence:* ordered list
  - Can occur within other compositors
  - Allows varying min and max occurrence
- *All:* unordered
  - Must occur directly below root element
  - Max occurrence of each element is 1
- *Choice:* exclusive or
  - Can occur within other compositors

XML Schema: Main Namespaces

Part of the standard

- xsd: http://www.w3.org/2001/XMLSchema
  - Terms for defining schemas: schema, element, attribute, . . .
  - The schema element has an attribute targetNamespace
- xsi: http://www.w3.org/2001/XMLSchema-instance
  - Terms for use in instances: schemaLocation, noNamespaceSchemaLocation, nil, type
- targetNamespace: user-defined

XML Schema Instance Doc

```
<!-- Comment -->
<Music xmlns="http://a.b.c/Muse"
  xmlns:xsi="the standard-xsi"
  xsi:schemaLocation="schema-URI schema-location-URL">
 <!-- Notice space character in above string -->
 ...
</Music>
```

Define null values as

```
<aElem xsi:nil="true"/>
```

XML Schema: Nillable

- An xsd:element declaration may state nillable='true'
  - An instance of the element might state xsi:nil="true"
  - The instance would be valid even if no content is present, even if content is required by default

Creating XML Schema Docs: 1

Included into the same namespace as the including doc

```
<xsd:schema xmlns:xsd="the-standard-xsd"
        xsd:targetNamespace="the-target">
 <include xsd:schemaLocation="part-one.xsd"/>
 <include xsd:schemaLocation="part-two.xsd"/>
  <!-- schemaLocation as in xsd, not xsi -->
</xsd:schema>
```

Creating XML Schema Docs: 2

- Use import instead of include
  - Imports may have different targets
  - Included schemas have the same target
  - Specify namespaces from which schemas are to be imported
  - Location of schemas not required and may be ignored if provided

Foreign Attributes in XML Schema

XML Schema elements allow attributes that are *foreign*, i.e., with a namespace other than the xsd namespace

- Must have an explicit namespace
- Can be used to insert any additional information, not interpreted by a processor
- Specific usage is with attributes from the xlink: namespace

```
<xsd:schema>
 <xsd:element name='course' type='cT'
          xlink:role='work' ncsu:offering='true'>
4 </xsd:schema>
```

XML Schema Style Guidelines: 1

- Flatten the structure of the schema
  - Don't nest declarations as you would a desired instance document
  - Make sure that element names are not reused
  - Unqualified attributes cannot be global
  - If dealing with legacy documents with the same element names having different meanings, place them in different namespaces where possible
- Use named types where appropriate

XML Schema Style Guidelines: 2

- Don't have elements with mixed content
- Don't have attribute values that need parsing
- Add unique IDs for information that may repeat
- Group information that may repeat
- Emphasize commonalities and reuse
  - Derive types from related types
  - Create attribute groups

XML Schema Documentation

xsd:annotation

- Should be the first subelement, except for the whole schema
- Container for two mixed-content subelements
  - xsd:documentation: for humans
  - xsd:appinfo: for machine-processible data
    - Such as application-specific metadata
    - Possibly using the Dublin Core vocabulary, which describes library content and other media

Module 5: XML Manipulation

Key XML query and manipulation languages include

- XPath
- XQuery
- XSLT

Metaphors for Handling XML: 1

How we conceptualize what XML documents are determines our approach for handling such documents

- *Text:* an XML document is text
  - Ignore any structure and perform simple pattern matches
- *Tags:* an XML document is text interspersed with tags
  - Treat each tag as an “event” during reading a document, as in SAX (Simple API for XML)
  - Construct regular expressions as in screen scraping

Metaphors for Handling XML: 2

- *Tree:* an XML document is a tree
  - Walk the tree using DOM (Document Object Model)
- *Template:* an XML document has regular structure
  - Let XPath, XSLT, XQuery do the work
- *Thought:* an XML document represents a graph structure
  - Access knowledge via RDF or OWL

XPath

Used as part of XPointer, SQL/XML, XQuery, and XSLT

- Models XML documents as trees with nodes
  - Elements
  - Attributes
  - Text (PCDATA)
  - Comments
  - Root node: above root of document

Achtung!

- Parent in XPath is like parent as traditionally in computer science
- Child in XPath is confusing:
  - An attribute is not a child of its parent
  - Makes a difference for recursion (e.g., in XSLT apply-templates)
- Our terminology follows computer science:
  - e-children, a-children, t-children
  - Sets via et-, ta-, and so on

XPath Location Paths: 1

- Relative or absolute
- Reminiscent of file system paths, but *much* more subtle
  - Name of an element to walk down
  - Leading /: root
  - /: indicates walking down a tree
  - .: currently matched (*context*) node
  - ..: parent node

XPath Location Paths: 2

- @attr: to check existence or access value of the given attribute
- text(): extract the text
- comment(): extract the comment
- [ ]: generalized array accessors
- Variety of *axes*, discussed below

XPath Navigation

- Select children according to position, e.g., [j], where j could be 1 . . . last()
- Descendant-or-self operator, //
  - .//elem finds all elems under the current node
  - //elem finds all elems in the document
- Wildcard, *:
  - collects e-children (subelements) of the node where it is applied, but omits the t-children
  - @*: finds all attribute values

XPath Queries (Selection Conditions)

- Attributes: //Song[@genre="jazz"]
- Text: //Song[starts-with(.//group, "Led")]
- Existence of attribute: //Song[@genre]
- Existence of subelement: //Song[group]
- Boolean operators: and, not, or
- Set operator: union (|), which behaves like choice
- Arithmetic operators: $>$, $<$, . . .
- String functions: contains(), concat(), length(), starts-with(), ends-with()
- distinct-values()
- Aggregates: sum(), count()

XPath Axes: 1

Axes are addressable node sets based on the document tree and the current node

- Axes facilitate navigation of a tree
- Several are defined
- Mostly straightforward but some of them order the nodes as the reverse of others
- Some captured via special notation
  - current, child, parent, attribute, . . .

XPath Axes: 2

- preceding: nodes that precede the start of the context node (not ancestors, attributes, namespace nodes)
- following: nodes that follow the end of the context node (not descendants, attributes, namespace nodes)
- preceding-sibling: preceding nodes that are children of the same parent, in reverse document order
- following-sibling: following nodes that are children of the same parent

XPath Axes: 3

- ancestor: proper ancestors, i.e., element nodes (other than the context node) that contain the context node, in reverse document order
- descendant: proper descendants
- ancestor-or-self: ancestors, including self (if it matches the next condition)
- descendant-or-self: descendants, including self (if it matches the next condition)

XPath Axes: 4

- Longer syntax: child::Song
- Some captured via special notation
  - self::*:
  - child::node(): node() matches all nodes
  - preceding::*
  - descendant::text()
  - ancestor::Song
  - descendant-or-self::node(), which abbreviates to //
  - Compare /descendant-or-self::Song[1] (first descendant Song) and //Song[1] (first Songs (children of their parents))

XPath Axes: 5

- Each axis has a *principal node kind*
  - attribute: attribute
  - namespace: namespace
  - All other axes: element
- * matches whatever is the principal node kind of the current axis
- node() matches all nodes

XPointer

Enables pointing to specific parts of documents

- Combines XPath with URLs
- URL to get to a document; XPath to walk down the document
- Can be used to formulate queries, e.g.,
  - Song-URL#xpointer(//Song[@genre="jazz"])
  - The part after # is a *fragment identifier*
- Fine-grained addressability enhances the Web architecture

High-level “conceptual” identification of node sets

XQuery

- The official query language for XML, now a W3C recommendation, as version 1.0
- Given a non-XML syntax, easier on the human eye than XML
- An XML rendition, XqueryX, is in the works

XQuery Basic Paradigm

The basic paradigm mimics the SQL (SELECT–FROM–WHERE) clause

```
1 for $x in doc('q2.xml')//Song
  where $x/@lg = 'en'
  return
   <English-Sgr name='{$x/Sgr/@name}' ti='{$x/@ti}'/>
```

FLWOR Expressions

Pronounced “flower”

- For: iterative binding of variables over range of values
- Let: one shot binding of variables over vector of values
- Where (optional)
- Order by (sort: optional)
- Return (required)

Need at least one of for or let

XQuery For Clause

The for clause

- Introduces one or more variables
- Generates possible bindings for each variable
- Acts as a mapping functor or iterator
  - In essence, all possible combinations of bindings are generated: like a Cartesian product in relational algebra
  - The bindings form an ordered list

XQuery Where Clause

The where clause

- Selects the combinations of bindings that are desired
- Behaves like the where clause in SQL, in essence producing a join based on the Cartesian product

XQuery Return Clause

The return clause

- Specifies what node-sets are returned based on the selected combinations of bindings

XQuery Let Clause

The let clause

- Like for, introduces one or more variables
- Like for, generates possible bindings for each variable
- Unlike for, generates the bindings as a list in one shot (no iteration)

XQuery Order By Clause

The order by clause

- Specifies how the vector of variable bindings is to be sorted before the return clause
- Sorting expressions can be nested by separating them with commas
- Variants allow specifying
  - descending or ascending (default)
  - empty greatest or empty least to accommodate empty elements
  - stable sorts: stable order by
  - collations: order by $t collation collation-URI: (obscure, so skip)

XQuery Positional Variables

The for clause can be enhanced with a positional variable

- A positional variable captures the position of the main variable in the given for clause with respect to the expression from which the main variable is generated
- Introduce a positional variable via the at $var construct

XQuery Declarations

The declare clause specifies things like

- Namespaces: declare namespace pref='value'
  - Predefined prefixes include XML, XML Schema, XML Schema-Instance, XPath, and local
- Settings: declare boundary-space preserve (or strip)
- Default collation: a URI to be used for collation when no collation is specified

XQuery Quantification: 1

- Two quantifiers some and every
- Each quantifier expression evaluates to true or false
- Each quantifier introduces a bound variable, analogous to for

```
1 for $x in ...
  where some $y in ...
  satisfies $y ... $x
  return ...
```

Here the second $x refers to the *same* variable as the first

XQuery Quantification: 2

A typical useful quantified expression would use variables that were introduced outside of its scope

- The order of evaluation is implementation-dependent: enables optimization
- If some bindings produce errors, this can matter
- some: trivially false if no variable bindings are found that satisfy it
- every: trivially true if no variable bindings are found

Variables: Scoping, Bound, and Free

for, let, some, and every introduce variables

- The visibility variable follows typical scoping rules
- A variable referenced within a scope is
  - *Bound* if it is declared within the scope
  - *Free* if it not declared within the scope

```
1 for $x in ...
  where some $x in ...
  satisfies ...
  return ...
```

Here the two $x refer to *different* variables

XQuery Conditionals

Like a classical if-then-else clause

- The else is not optional
- Empty sequences or node sets, written ( ), indicate that nothing is returned

XQuery Constructors

Braces { } to delimit expressions that are evaluated to generate the content to be included; analogous to macros

- document { }: to create a document node with the specified contents
- element { } { }: to create an element
  - element foo { 'bar' }: creates <foo>Bar</foo>
  - element { 'foo' } { 'bar' }: also evaluates the name expression
- attribute { } { }: likewise
- text { body}: simpler, because anonymous

XQuery Effective Boolean Value

Analogous to Lisp, a general value can be treated as if it were a Boolean

- A xs:boolean value maps to itself
- Empty sequence maps to false
- Sequence whose first member is a node maps to true
- A numeric that is 0, negative, or NaN maps to false, else true
- An empty string maps to false, others to true

Defining Functions

```
1 declare function local:itemftop($t)
  {
   local:itemf($t,())
  };
```

- Here local: is the namespace of the query
- The arguments are specified in parentheses
- All of XQuery may be used within the defining braces
- Such functions can be used in place of XPath expressions

Functions with Types

```
1 declare function local:itemftop($t as element())
    as element()*
  {
    local:itemf($t,())
  };
```

- Return types as above
- Also possible for parameters, but ignore such for this course

XSLT

A programming language with a functional flavor

- Specifies (stylesheet) transforms from documents to documents
- Can be included in a document (best not to)

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
   href="URL-to-xsl-sheet"?>
<main-element>
  ...
</main-element>
```


XQuery versus XSLT: 1

Competitors in some ways, but

- Share a basis in XPath
- Consequently share the same data model
- Same type systems (in the type-sensitive versions)
- XSLT got out first and has a sizable following, but XQuery has strong backing among vendors and researchers

XQuery versus XSLT: 2

- XQuery is geared for querying databases
  - Supported by major relational DBMS vendors in their XML offerings
  - Supported by native XML DBMSs
  - Offers superior coverage of processing joins
  - Is more logical (like SQL) and potentially more optimizable
- XSLT is geared for transforming documents
  - Is functional rather than declarative
  - Based on template matching

XQuery versus XSLT: 3

There is a bit of an arms race between them

- Types
  - XSLT 1.0 didn't support types
  - XQuery 1.0 does
  - XSLT 2.0 does too
- XQuery presumably will be enhanced with capabilities to make updates, but XSLT could too

XSLT Stylesheets

A programming language that follows XML syntax

- Use the XSLT namespace (conventionally abbreviated xsl)
- Includes a large number of primitives, especially:
  - <copy-of> (deep copy)
  - <copy> (shallow copy)
  - <value-of>
  - <for-each select="...">
  - <if test="...">
  - <choose>

XSLT Templates: 1

- A pattern to specify where the given transform should apply: an XPath expression
  - This match only works on the root:

```
<xsl:template match="/">
  ...
</xsl:template>
```

  - Example: Duplicate text in an element

```
<xsl:template match="text()">
  <xsl:value-of select='.'/>
  <xsl:value-of select='.'/>
</xsl:template>
```

XSLT Templates: 2

- If no pattern is specified, apply recursively on et-children via <xsl:apply-templates/>
- By default, if no other template matches, recursively apply to et-children of current node (ignores attributes) and to root:

```
1 <xsl:template match="*|/">
    <xsl:apply-templates/>
  </xsl:template>
```


XSLT Templates: 3

- Copy text node by default
- Use an empty template to override the default:

```
  <xsl:template match="X"/>
2 <!-- X = desired pattern -->
```

Confine ourselves to the examples discussed in class (ignore explicit priorities, for example)

XSLT Templates: 4

- Templates can be named
- Templates can have parameters
  - Values for parameters are supplied at invocation
  - Empty node sets by default
  - Additional parameters are ignored

XSLT Variables

- Explicitly declared
- Values are node sets
- Convenient way to document templates

Document Object Model (DOM)

Basis for parsing XML, which provides a node-labeled tree in its API

- Conceptually simple: traverse by requesting element, its attribute values, and its children
- Processing program reflects document structure, as in recursive descent
- Can edit documents
- Inefficient for large documents: parses them first entirely even if a tiny part is needed
- Can validate with respect to a schema

DOM Example

```
DOMParser p = new DOMParser();
p.parse("filename");
Document d = p.getDocument()
Element s = d.getDocumentElement();
NodeList l = s.getElementsByTagName("member");
Element m = (Element) l.item(0);
int code = m.getAttribute("code");
NodeList kids = m.getChildNodes();
Node kid = kids.item(0);
String elemName = ((Element)kid).getTagName(); ...
```

Simple API for XML (SAX)

- Parser generates a sequence of events:
  - startElement, endElement, . . .
- Programmer implements these as *callbacks*
  - More control for the programmer
- Processing program does not necessarily reflect document structure

SAX Example: 1

```
class MemberProcess extends DefaultHandler {
 public void startElement (String uri, String n,
                           String qName, Attributes attrs) {
  if (n.equals("member")) code = attrs.getValue("code")
  if (n.equals("project")) inProject = true;
  buffer.reset();
 }

 ...
```

SAX Example: 2

```
1
    ...

    public void endElement (String uri, String n,
                                        String qName) {
6     if (n.equals("project")) inProject = false;
      if (n.equals("member") && !inProject)
        ... do something ...
      }
  }
```


SAX Filters

A component that mediates between an XMLReader (parser) and a client

- A filter would present a modified set of events to the client
- Typical uses:
  - Make minor modifications to the structure
  - Search for patterns efficiently
    - What kinds of patterns, though?
- Ideally modularize treatment of different event patterns
- In general, a filter can alter the structure of the document

Creating XML from Legacy Sources

Often need to read in information from non-XML sources

- From relational databases
  - Easier because of structure
  - Supported by vendor tools
- From flat files, CSV documents, HTML Web pages
  - Bit of a black art: lots of heuristics
  - Tools based on regular expressions

Programming with XML

- Limitations
  - Difficult to construct and maintain documents
  - Internal structures are cumbersome; hence the criticisms of DOM parsers
- Emerging approaches provide superior binding from XML to
  - Programming languages
  - Relational databases
- Check pull-based versus push-based parsers

Module 6: XML Storage

The major aspects of storing XML include

- XML Keys
- Concepts: Data and Document Centrism
- Storage
- Mapping to relational schemas
- SQL/XML

Integrity Constraints in XML

- Entity: xsd:unique and xsd:key
- Referential: xsd:keyref
- Data type: XML Schema specifications
- Value: Solve custom queries using XPath or XQuery

Entity and referential constraints are based on XPath

XML Keys: 1

Keys serve as generalized identifiers, and are captured via XML Schema elements:

- *Unique:* candidate key
  - The selected elements yield unique field tuples
- *Key:* primary key, which means candidate key plus
  - The tuples exist for each selected element
- *Keyref:* foreign key
  - Each tuple of fields of a selected element corresponds to an element in the referenced key

XML Keys: 2

Two subelements built using restricted application of XPath from within XML Schema

- *Selector:* specify a set of objects: this is the scope over which uniqueness applies
- *Field:* specify what is unique for each member of the above set: this is the identifier within the targeted scope
  - Multiple fields are treated as ordered to produce a tuple of values for each member of the set
  - The order matters for matching keyref to key

Selector XPath Expression

A selector finds descendant elements of the context node

- The sublanguage of XPath used *allows*
  - Children via ./child or ./* or child
  - Descendants via .// (not within a path)
  - Choice via |
- The subset of XPath used *does not allow*
  - Parents or ancestors
  - text()
  - Attributes
  - Fancy axes such as preceding, preceding-sibling, . . .

Field XPath Expression

A field finds a unique descendant element (simple type only) or attribute of the context node

- The subset of XPath used *allows*
  - Children via ./child or ./*
  - Descendants via .// (not within a path)
  - Choice via |
  - Attributes via @attribute or @*
- The subset of XPath used *does not allow*
  - Parents or ancestors
  - text()
  - Fancy axes such as preceding, . . .

An element yields its text()

XML Foreign Keys

```
<keyref name="..." refer="primary-key-name">
 <selector xpath="..."/>
 <field name="..."/>
</keyref>
```

- Relational requirement: foreign keys don't have to be unique or non-null, but if one component is null, then all components must be null.

Placing Keys in Schemas

- Keys are associated with elements, not with types
- Thus the . in a key selector expression is bound
- Could have been (but are not) associated with types where the . could be bound to whichever element was an instance of the type

Data-Centric View: 1

```
<relation name='Student'>
 <tuple><attr1>V11</attr1>
        ...
        <attrn>V1n</attrn>
 </tuple>
 ...
</relation>
```

- Extract and store via mapping to DB model
- Regular, homogeneous structure

Data-Centric View: 2

- Ideally, no mixed content: an element contains text *or* subelements, not both
- Any mixed content would be templatic, i.e.,
  - Generated from a database via suitable transformations
  - Generated via a form that a user or an application fills out
- Order among siblings likely irrelevant (as is order among relational columns)

Expensive if documents are repeatedly parsed and instantiated

Document-Centric View

- Irregular: doesn't map well to a relation
- Heterogeneous data
- Depending on entire doc for application-specific meaning

Data- vs Document-Centric Views

- *Data-centric:* data is the main thing
  - XML simply renders the data for transport
  - Store as data
  - Convert to/from XML as needed
  - The structure is important
- *Document-centric:* documents are the main thing
  - Documents are complex (e.g., design documents) and irregular
  - Store documents wherever
  - Use DBMS where it facilitates performing important searches

Storing Documents in Databases

- Use character large objects (CLOBs) within DB: searchable only as text
- Store paths to external files containing docs
  - Simple, but no support for integrity
- Use some structured elements for easy search as well as unstructured clobs or files
- Heterogeneity complicates mappings to typed OO programming languages

Storing documents in their entirety may sometimes be necessary for external reasons, such as regulatory compliance

Database Features

- Storage: schema definition language
- Querying: query language
- Transactions: concurrency
- Recovery

Potential DBMS Types for XML: 1

- Object-oriented
  - Nice structure
  - Intellectual basis of many XML concepts, including schema representations and path expressions
  - Not highly popular in standalone products
- Relational
  - Limited structuring ability (1NF: each cell is atomic)
  - Extremely popular
  - Well optimized for flat queries

Potential DBMS Types for XML: 2

- Object relational: hybrids of above
  - Not highly popular in standalone products
- Custom XML stores or native XML databases
  - Emerging ideas: may lack core database features (e.g., recovery, . . .)
  - Enable fancier *content management systems*
  - Leading open source products:
    - Apache Xindice (server; XPath)
    - Berkeley DB XML (libraries; XQuery)

XML to Relational Databases

- Using large objects
- Flatten XML structures
- Referring to external files

Recall that for a relational schema, its entire set of attributes is necessarily a superkey

Artificial Representation: Repetitious

Capturing an object hierarchy in a relation

- Imagine an artificial identifier for each node
- Construct a relation with three main relational attributes or columns
  - One column for the identifier
  - One column for the name of an attribute (i.e., element name)
  - One column for the value (assumes the value would fit into the same relational type: potentially this could be CLOB or BLOB)

Artificial Representation: Graph

Use four generic relations to represent a graph

- Vertices:
  - Element ID, Name
- Contents
  - Element ID, Text, number (to allow multiple text nodes)
- Attributes
  - ID, Attribute name, Attribute value
- Edges
  - Source ID, Target ID

Better typed than repetitious style because this has no nulls

Shallow Representation: 1

The "natural" approaches are based on tuple-generating elements (TGEs)

- Choose one XML element type as the TGE
  - TGE corresponds to a tuple
  - The key is based on an ID attribute or text of the TGE
- A relational attribute (column) for each subelement or attribute
- Easiest if there is an attribute for IDs and there are no other attributes

Shallow Representation: 2

- Consequences
  - Nulls for missing subelements can proliferate
  - Subelements with structure (subelements or attributes) aren't represented well
  - Ancestors cannot be searched for

Deep Representation

Also called *shredding* an XML document

- Choose a TGE as before
- A column for each descendant, except that
  - Can skip *wrapper* elements (no text, only subelements), but must reconstruct them to create an XML document
- Consequences
  - Nulls for missing subelements
  - Lots of columns in a relation
  - Ancestors cannot be searched for
  - Loses structural information

Representing Ancestors

Ancestors are the elements that are above the scope of the given TGE

- Choose a TGE as before
- A column for each descendant as before
- A column for each ancestor (that needs to be searched)
  - Appropriate attributes or text fields to make the search worthwhile
- Consequences
  - Nulls for missing subelements
  - Lots of columns in a relation

Generalized TGE

- Each element is a TGE, yielding a different relation
- A column for each terminal child: attribute or text
- A column for each ancestor to capture the entire path from root to this node
  - Must promote uniquifying content so that each TGE yields unique tuples
- Consequences
  - Nulls for missing subelements
  - Lots of relations
  - Lots of columns in a relation

Variations in Structure

- Create separate relations for each variant
- Consequences
  - Lots of possible structures to store
  - Queries would not be succinct
  - Acceptable only if we know in advance that the number of variants is small and the data in each is substantial

Semistructured Representation

Create two (sets of) relations

- *Specific part:* one (or more) relations based on one of the natural approaches
- *Generic part:* one relation based on an artificial approach

Thoughtful Design

- The above approaches are not sensitive to the meaning and motivation behind the XML structure
- Understand the XML structure via a conceptual model (in terms of entities and relationships)
- Avoid unnecessary nesting in the XML structure, if possible
- Design a corresponding relational schema by hand

This is not always possible, though

Evaluation

How does the above work for data-centric and document-centric views?

- Compare with respect to
  - Document structure
  - Document "roundtripping" (compare **&**, **&**, **#a39**)
  - Normalization
- Are the documents unique?
- Are the documents unique up to "isomorphism"?

Schema Evolution

A big problem for databases in practical settings

- For relational schemas, certain kinds of updates are simpler than others
- Can have consequences on optimization
- XML schemas can be evolved by using XSLT to map old data to new schema

From Relations to XML

Mapping a relation schema (set of relations plus functional dependencies) to an XML document

- Map relation R to an element R_E with key or unique constraints
- Map column C of R to an attribute of R_E or equivalently a child element with just text
- Map relation S with a foreign key to R to
  - A child element S_E of R_E (omit foreign key content from S_E): works if only one such R_E for S_E; OR
  - An element S_E that includes the foreign key content, and includes a keyref to R_E

Null Value: 1

A special value, not in any domain, but combinable with any domain

- Need?
- Possible meanings
  - Not applicable
  - Unknown: missing
  - Questionable existence
  - Absent (known but absent)
- Hazards of null values?

Null Value: 2

XML Schema enables developing custom null values for each domain

- Create an arbitrary value that
  - Matches the given data type
  - Is not a valid value of the domain, however
- Design applications to understand specific restricted type

XML Schema Null

- <elem/> (equivalently <elem></elem>) means that the element contains the empty string
  - This is not null
- xsi defines the attribute nil
  - Used as <elem xsi:nil="true"/> if elem is declared nillable (via nillable="true")

Quick Look at SQL

Structured Query Language

- Data Definition Language: CREATE TABLE
- Data Manipulation Language: SELECT, INSERT, DELETE, UPDATE
- Basic paradigm for SELECT

```
SELECT t1.column-1, t1.column-2 ... tm.column-n
FROM table-1 t1, table-m tm
WHERE t1.column-3=t4.column-4 AND ...
```

SQL 2003

Standardized by ANSI/ISO; next version after SQL 1999

- Includes SQL/XML: SQL extensions for XML (other aspects of SQL 2003 are not relevant here)
- Distinct from Microsoft's SQLXML
- SQL/XML is included in products
  - By DBMS vendors, sometimes with different low-level details (MINUS versus EXCEPT)
  - DBMS-independent products

XML Type in SQL/XML

- A specialized data type for XML content; distinct from text
- Usable wherever an SQL data type is allowed: type of column, variable, tuple cell, and so on . . .
- Value rooted on the XML Root information item (described next)

XML Root Information Item: 1

Based on the XML InfoSet document information item, this can be an

- XML root (as in SQL/XML)
- XML element
- XML attribute
- XML parsed character data (text; aka PCDATA)
- XML namespace declaration
- XML processing instruction
- XML comment

And some more possibilities from the InfoSet . . .

XML Root Information Item: 2

- Unlike the XML InfoSet root (which allows exactly one child element), this allows zero or more children
  - Partial results need not be documents
- IS DOCUMENT: a predicate that checks if the argument XML value has a single root
- An XML value can be
  - NULL, as usual for SQL
  - An XML root item, including whatever it includes

SQL/XML Builtin Operators

- xmlparse(): maps a string (char, varchar, clob) to a value of type XML (stripping whitespace by default)
- xmlserialize(): maps a value of type XML to a string
- xmlconcat(): combines values into a forest
- xmlroot(): create or modify the root node of an XML value

SQL/XML Publishing Functions: 1

These are templates that go into a SELECT query; all with names that begin “xml”

- xmlelement(name ’Song’, ·)
  - Needs a value: an SQL column or expression or an attribute or an element
  - Yields a value (an element)
  - Can be nested, of course
- xmlattributes(column [AS cname], column [AS cname],. . .)
  - Creates XML attributes from the columns
  - Inserts into the surrounding XML element

SQL/XML Publishing Functions: 2

- xmlforest()
  - Creates XML elements from columns
  - Analogous to a node-set in XPath
  - Must be placed within an element; otherwise not well-formed XML
- xmlagg(): combines a collection of rows, each with a single XML value into a single forest
- xmlnamespaces()
- xmlcomment(): comment
- xmlpi(): processing instruction

SQL/XML Example: 1

```
SELECT xmlelement(Name 'Sgr',
2          xmlattributes (z.sgrId AS student_ID),
           z.sgrName)
FROM Singer z
WHERE ...
```

yields something like

```
<Sgr student_ID='s1'>
 Eagles
</Sgr>
```

SQL/XML Example: 2

```
SELECT xmlelement(Name 'Sgr',
       xmlattributes (z.sgrId AS student_ID),
       z.sgrName,
       xmlelement(Name 'Song', 'Hotel'))
FROM Singer z
WHERE ...
```

yields something like

```
<Sgr student_ID='s1'>
  Eagles
  <Song>Hotel</Song>
</Sgr>
```


SQL/XML Mapping Rules

A number of low-level matters, which are conceptually trivial but complicate combining SQL and XML effectively; captured as *mapping rules*

- Lexical encodings in names and content
- Mapping datatypes in each direction, e.g., SQL date and XML Schema date
- Mapping SQL tables, schemas, catalogs to and from XML

Tool Support for SQL 2003

- Oracle 10g, IBM DB2, Sybase support it
- Apparently, Microsoft doesn't or won't [not sure]
- Oracle 9i release 2 supports similar constructs, but in proprietary syntax

Oracle 9i SQL/XML: 1

```
1 CREATE TABLE singer ( sgrId VARCHAR2(9) NOT NULL,
                        sgrName VARCHAR2(15) NOT NULL,
                        sgrInfo SYS.XMLTYPE NULL,
                        CONSTRAINT singer_key
                        PRIMARY KEY (sgrId));
```

Oracle 9i SQL/XML: 2

```
INSERT INTO singer VALUES ('Sgr-01', 'Eagles',
   SYS.XMLTYPE.createXML('<genre>rock</genre>'));

INSERT INTO singer VALUES ('Sgr-04', 'Beatles',
              SYS.XMLTYPE.createXML(
   '<trivia><convictions>freedom</convictions>
   <genre>rock</genre></trivia>'));

SELECT z.sgrName, z.sgrInfo.extract( '/genre/text()')
                                      .getClobVal()
FROM singer z;
```


Oracle 9i SQL/XML: 3

```
SELECT z.sgrName, z.sgrInfo.extract( '//genre/text()')
                                      .getClobVal()
FROM singer z
WHERE z.sgrInfo.extract(
     '//genre/text()').getStringVal() like 'r%';

SELECT z.sgrName, z.sgrInfo.extract( '/genre/text()')
                                      .getClobVal()
FROM singer z
WHERE z.sgrInfo.existsNode('//genre') = 1;
```

Oracle 9i SQL/XML: 4

```
SELECT SYS_XMLAGG(SYS_XMLGEN(z.sgrname),
    SYS.XMLGENFORMATTYPE.createformat('FooList'))
                                        .getClobVal()
FROM singer z
WHERE z.sgrId IS NOT NULL
GROUP BY z.sgrname;
```


Modern Information Systems

- Three legs of modern software systems
  - *Documents:* as in XML
  - *Tuples:* as in the information stored in relational databases
  - *Objects:* as in programming languages
- A lot of effort goes into managing translations among these at the level of programming
- But deeper challenges remain . . .

Limitations of XML

- Doesn't represent meaning
- Doesn't represent conceptual structure
- Enables multiple representations for the same information
  - Give an example

Transforms can be robustly specified and accurately documented only if models are known, but usually the models are not known

Directions in XML

Trends: sophisticated approaches for

- Querying and manipulating XML, e.g., XSLT and XQuery
- Sophisticated storage and access techniques in traditional relational databases
- Tools that shield programmers from low-level details
- Semantics, e.g., RDF, OWL, . . .

Module 7: Rationality

- Basis for understanding interactions among autonomous parties
- Many questions reduce to resource allocation
- What is an optimal or correct resource allocation

What is an Agent?

Abstraction to support autonomy and heterogeneity

- In general, an agent is an *active* computational entity that
  - Carries a *persistent* (i.e., long-lived) identity
  - Perceives, reasons about, and initiates activities in its environment
  - Adapts its behavior based on others' behavior
  - Communicates (with other agents)

Example: an agent in a market or supply chain

Negotiation

- The key to adaptive, cooperative behavior
- The essence of business
- Involves several nontechnical considerations
- The computer science role is to provide representations and algorithms to facilitate negotiation

Explicit Negotiation

Achieving agreement among a small set of agents, e.g., to construct a supply chain

- Based on "dialog" moves
  - Such as *propose*, *counter-propose*, *support*, *accept*, *reject*, *dismiss*, *retract*
  - Composed into protocols allowing valid "conversations"
- Presupposes a common abstraction of the problem, and a common content language

Relate the moves to resource allocations, contracts, and post-contractual events

Deals: Joint Plans Among Agents

Possible outcomes of negotiation

- Utility of a deal for an agent is its benefit minus its cost
- Negotiation set: set of deals (joint plans) with *positive* utility for each agent
  - *Conflict:* the negotiation set is empty
  - *Compromise:* each agent prefers to work alone, but will agree to a negotiated deal
  - *Cooperation:* all deals in the negotiation set are preferred by both agents over achieving their goals alone

Simple Negotiation Protocol

- Each party, in turn, proposes a joint plan
- They proceed as long as they agree
- Any conflicts are resolved randomly (e.g., "flip a coin" to decide which agent would satisfy its goal)

Example: two friends each want coffee; one of them (selected via a coin toss) will fetch coffee for both

Mechanism Design

- *Mechanism:* a set of rules of an environment under which agents operate
  - Honor systems
  - Honor systems with social censure (as a penalty)
  - Auctions
  - Paying taxes (voluntary, but with selective audits and severe penalties for violators)
- How do the above compare?
- *Mechanism design:* Creating a mechanism to obtain desired system-level properties, e.g., participating agents interact productively and fairly

Example Mechanism: Puzzle

Given two horses to be raced for a mile

- Owner of horse proved *faster* wins a reward
  - Each owner is or hires a jockey
  - The horses are raced against each other
  - The winner of the race wins
- Owner of horse proved *slower* wins a reward
  - Might consider rewarding the loser of a race, but such a race won't terminate because each rider will want to go slower than the other

Economic Abstractions

A well-established approach to capture complex systems of *autonomous* agents (people or software)

- Incomplete by themselves
- Support achieving optimal resource allocations
- Provide a basis for achieving some contractual behaviors, especially in helping
  - An individual agent decide what to do
  - Agents negotiate

How Can Trade Work?

Whether barter or using money

- Why would rational agents voluntarily participate?
- Both cannot possibly gain; or can they?
- Consider the following. Would you trade
  - A dollar bill for another dollar bill?
  - A US dollar for x Euros?
  - Money for a bottle of drinking water?
  - A bottle of drinking water for money?

It comes down to your valuations: differences in valuations make trade possible

Markets Introduced

Compare stock with specific real-estate

- Can be
  - *Public*
  - *Private:* part of restricted exchanges
- Can restrict kinds of goods traded
  - *Endogenous:* NASDAQ
  - *Exogenous:* eBay, where physical goods are traded outside the scope of the market
- Offer some form of nonrepudiation

Centrality of Prices

A price is a scalar: easy to compare

- The computational state of a market is described completely by current prices for the various goods
- Communications are between each participant and the market, and only in terms of prices
- Participants reason about others and choose strategies entirely in terms of prices being bid

Functions of a Market

- Provides this information to participants
- Takes requests (buy, sell bids) from participants, enforcing rules such as bid increments and time limits
- Decides outcome based on messages from participants, considering rules such as reserve prices, . . .

Achieving Equilibrium

When supply equals demand

- At equilibrium, the market has computed the allocation of resources
  - Dictates the activities and consumptions of the agents
- Under certain conditions, a simultaneous equilibrium of supply and demand across all goods exists
  - That is, the market "clears"
  - Reachable via distributed bidding
  - *Pareto optimal:* you cannot make the allocation better for one agent without making it worse for another

Pareto Optimality

- Allocation: how resources are allocated to different parties
- Think of a vector of allocations, one dimension for each participant
- An allocation is Pareto optimal if improvements along any dimension must be accompanied by a reduction along another dimension

Using Agents for Resource Allocation

- Consumer agents: exchange goods for money
- Producer agents: transform some goods into other goods
- Assume individual impact on market is negligible
- Both types of agents bid so as to maximize profits (or utility with respect to their valuations)

The agents' strategies can be complex, especially if time matters, goods have multiple attributes, and agents have multiple criteria

Market-Oriented Programming

An economic approach where the *market* is a mediating party and authority

- Market price mechanisms are effective for coordinating the activities of many agents with minimal direct communication
- Build computational economies to solve problems of distributed resource allocation to serve individual preferences

Contrast with direct bartering and with central control

Auctions in Markets

Computational mechanism to manage supply and demand: support dynamic pricing

- Exchange common object (money) for goods
  - Ascending (English) vs. Descending (Dutch)
  - Silent (auctioneer names a price; bids are silent) vs. outcry (bids name prices; auctioneer listens)
  - Hidden identity or not
  - Combinatorial: involve *bundles* or sets of goods

English Auction

Buyers bid for an item

- Prices start low and increase
- Highest bidder gets the object and pays the price bid
- Variations:
  - Minimum bid increment
  - Reserve price (no sale if too low)
  - Limited time

Dutch Auction

- Price "clock" or counter starts high and winds down
- First to stop the clock wins and pays the price on the clock
- In other words, the highest bidder wins and pays the price bid

Winner's Curse: 1

- If you just won an English or a Dutch auction
- You just paid $x for something
- How much can you sell it for?
- Obviously, you will be able to sell it for ...

Not quite a curse if inherently valuable, but perhaps could have obtained the item for less

Winner's Curse: 2

Sealed bid; no resale

- A group of mutually independent people estimate the values of different goods and bid accordingly
- Assume that the group is smart
  - The average is about right as an estimate of the true value
- The winner bid the maximum

Fish Market Auction

Imagined scenario is based on a Spanish fish market

- Auctioneer calls out prices
  - If two or more bidders
    - repeat with higher price
  - If no bidders
    - repeat with lower price

Suckers' Auction

Consider two bidders bidding for $1 currency

- Bid in increments of 10¢
- Highest bidder wins
- Both bidders pay (i.e., loser also pays)
- Once you are in, can you get out?
  - The myopically rational strategy is to bid
  - The outcome is not pleasant

Sealed Bid First-Price Auction

Also known as *tenders*: bidding to buy

- One-shot bidding without knowing what other bids are being placed
- Used by governments and large companies to give out certain large contracts (lowest price quote for stated task or procurement)
  - All bids are gathered
  - Auctioneer decides outcomes based on given rules (e.g., highest bidder wins and pays the price it bid)

Vickrey Auction

- Second-price sealed bid auction
- Highest bidder wins, but pays the *second* highest price

Pricing

- General theme: allocate resources to those who value them the most
- Kinds of pricing
  - Fixed: slowly changing, based on various criteria
  - Dynamic: rapidly changing, based on actual demand and supply

Fixed Pricing: Example Criteria

- *Flexibility:* (restrict rerouting or refundability in air travel)
- *Urgency:* (convenience store vs. warehouse)
- *Customer preferences* (coupons: price-sensitive customers like them; others pay full price)
- *Demographics*
- *Artificial* (Paris Metro, Delhi “Deluxe” buses)
- *Predicted demand* (New York subway, phone rates)

Dynamic Pricing Motivation

- Under certain assumptions, markets ensure a resource allocation where all the goods that can be sold (for the market price) are sold and all the goods that can be bought (for the market price) are bought
- Fixed pricing leaves some revenue that other parties exploit (e.g., in secondary markets such as black markets as in football ticket scalping)
- Participants seek to maximize individual utility; those who value something more will pay more for it

Continuous Double Auction

As in stock markets, a way to accomplish dynamic pricing

- Multiple sellers and buyers, potentially with multiple sell and buy bids each
- Buy bids are like upper bounds
- Sell bids are like lower bounds
- *Clears* continually:
  - The moment a buyer and seller agree on a price, the deal is done and the matching bids are taken out of the market
  - Possibly, a moment later a better price may come along, but it will too late then

Auction Management: Bidding

Bidding rules to govern, e.g.,

- Whose turn it is
- What the minimum acceptable bid is, e.g., increments
- What the reserve price is, if any

Compare these for outcry, silent, sealed bid, and continuous auctions

Auction Management: Information

What information is revealed to participants?

- Bid value (not in sealed bid auctions)
- Bidder identity (not in sealed bid auctions or stock exchanges)
- Winning bid or current high bid
- Winner
- How often, e.g., once per auction, once per hour, any time, and so on

Auction Management: Clearing

Bids are cleared when they are executed and taken out of the market

- What defines a deal: how are bids matched?
  - What prices? If uniform, then matching is not relevant
  - Who?
- How often?
- Until when?

M^{th} and $(M+1)^{st}$ Price Auctions: 1

- L = M+N single-unit sealed bids, not continuously cleared
  - M sell bids
  - N buy bids
- M^{th} price clearing rule
  - Price = M^{th} highest among all L bids
  - English: first price; M=1
    - Seller's reserve price is the sole sell bid (assume minimum value, if no explicit reserve price)

M^{th} and $(M+1)^{st}$ Price Auctions: 2

- $(M+1)^{st}$ price clearing rule
  - Price = $(M+1)^{st}$ highest among all L bids
  - Vickrey: second price; M=1

M^{th} and $(M+1)^{st}$ Price Auctions: 3

The M^{th} and $(M+1)^{st}$ prices delimit the equilibrium price range, where supply and demand are balanced

- Above M^{th} price: no demand from some buyers
- Below $(M+1)^{st}$ price: no supply from some sellers

Restrict to M=1

Kinds of Valuations

How do agents place values of goods?

- *Independent (and private):* Agents value goods in a manner that is unaffected by others
  - Consume or use: cake
- *Common:* Agents value goods entirely based on others' valuations, leading to symmetric valuations
  - Resale: treasury bills
- *Correlated:* Combination of above
  - Automobile or house

Concepts About Matching: 1

Buy and sell bids can be matched in various ways, which support different properties:

- *Equilibrium prices:* those at which supply equals demand, also known as *market price*
- *Individually rational:* each agent is no worse off participating than otherwise
- *Efficient:* No further gains possible from trade (agents who value goods most get them): i.e., Pareto optimal

Concepts About Matching: 2

- *Uniform price:* Multiple units, if simultaneously matched, are traded at the same price
- *Discriminatory:* Trading price for each pair of bidders can be different
- *Incentive compatible:* Agents optimize their expected utility by bidding their true valuations

Incentive Compatibility

Incentives are such it is rational to tell the truth

- Ramification: Agents can ignore subtle strategies and others' decisions: hence simpler demands for knowing others' preferences and reasoning about them
- Basic approach: payoff depends not on decisions (bids) by self
- Example: Vickrey (second-price sealed bid) auctions for independent private valuations
  - Underbid: likelier to lose, but price paid on winning is unaffected by bid
  - Overbid: likelier to win, but may pay more

Economic Rationality

- Space of alternatives or outcomes
- Each agent has some ordinal (i.e., sorted) preferences over the alternatives, captured by a binary relation, $\succ$
  - $\succ$ is a strict ordering
    - Asymmetric, Transitive (implies irreflexive)
  - $\succ$ is not total
  - Another binary relation, $\sim$, captures indifference

Lotteries

Probability distributions over outcomes or alternatives (add up to 1)

- In essence, define potential outcomes
  - Flip a coin for a dollar: [0.5: \$1; 0.5: –\$1]
  - Buy a \$10 ticket to win a car in a raffle: [0.0001: car−\$10; 0.9999: −\$10]
  - Four choices: $[p : A; q : B; r : C; 1 - p - q - r : D]$

Using Lotteries

Infer (rational) agents' preferences based on their behavior with respect to the lotteries

- What odds will a specific person accept?
- For example, [0.01: car$-$\$10; 0.99: $-$\$10]

Properties of Lotteries

- Substitutability of indifferent outcomes
  - If $A \sim B$, then $[p : A; (1-p) : C] \sim [p : B; (1-p) : C]$
- Monotonicity (for preferred outcomes)
  - If $A \succ B$ and $p > q$, then $[p : A; (1-p) : B] \succ [q : A; (1-q) : B]$
- Decomposibility (flatten out a lottery)
  - Compound lotteries reduce to simpler ones
  - $[p : [q : A; 1-q : B]; 1-p : C] = [pq : A; p-pq : B; 1-p : C]$

Expected Payoff

- Expresses the value of a lottery as a scalar (i.e., in monetary terms)
- Expected payoff is sum of utilities weighted by probability
  - Calculate for [0.0001: car−$10; 0.9999: −$10] where the car is worth $25,010

Completeness of Nonstrict Preferences

Same as indifference being an equivalence relation

- Given outcomes A and B
  - Either $A \preceq B$ or $B \preceq A$
- That is, $A \sim B$ if and only if $A \preceq B$ and $B \preceq A$
- Thus, $\sim$ is an *equivalence* relation
  - *Reflexivity:* $A \sim A$
  - *Symmetry:* $A \sim B$ implies $B \sim A$
  - *Transitivity:* ($A \sim B$ and $B \sim C$) implies $A \sim C$

Continuity of Preferences

- $A \succ B \succ C$ implies that there is a probability p, such that
  - $[p : A; 1 - p : C] \sim B$
  - Consider A, B, and C to be ice-cream, yogurt, and cookies, respectively
- Informally, this means we can price alternatives in terms of each other
- Is this reasonable in real life? Why or why not?

Utility Functions

One per agent

- Map each alternative (outcome) to a scalar (real number), i.e., money
  - U : {alternatives} $\mapsto \mathcal{R}$
- For agents with irreflexive, transitive, complete, continuous preferences, there is a utility function U such that
  - $U(A) > U(B)$ implies $A \succ B$
  - $U(A) = U(B)$ implies $A \sim B$
  - $U([p : A; 1 - p : C]) = p \times U(A) + (1 - p) \times U(C)$ (weighted sum of utilities)

Risk: 1

- According to the above, two lotteries with the same expected payoff would have equal utility
- In practice, risk makes a big difference
  - Raffles
  - Insurance
  - Business actions with unpredictable outcomes

Risk: 2

Showing utilities instead of outcomes

- Consider two lotteries
  - $L_1 = [1 : x]$
  - $L_2 = [p : y; 1 - p : z]$
  - Where $x = py + (1 - p)z$. That is, L_1 and L_2 have the same expected payoff
- An agent's preferences reflect its attitude to risk
  - *Averse:* $U(L_1) > U(L_2)$
  - *Neutral:* $U(L_1) = U(L_2)$
  - *Seeking:* $U(L_1) < U(L_2)$

Beyond Simple Utility

- Other factors besides expected payoff and risk are relevant in real life:
  - Total deal value: $10 discount for a t-shirt vs. for a car
  - Current wealth: 1^{st} million vs. 10^{th} million
  - Altruism or lack thereof
- Leads to social choice theory

Sharing Resources

Consider two scenarios for sharing—only requirement is that the parties agree on the split

- *Splitting a dollar:* relative sizes are obvious. Should splits consider the relative wealth of the splitters? Should splits consider the tax rates of the splitters?
- *Sharing a cake:* relative sizes and other attributes (e.g., amount of icing) can vary—several cake-cutting algorithms exist

Simplifying Assumptions

- Participants are risk neutral
  - Willing to trade money for any of their resources at a price independent of how much money they already have
- Participants know their valuations, which are independent and private

Pareto Optimality

- A distribution of resources where no agent can be made better off without making another agent worse off
- Example: A has goods g and values g at \$1; B values g at \$3
  - It is Pareto optimal for B to buy g at a price between \$1 and \$3, say \$2.50
  - A's gain: \$2.50–\$1 = \$1.50
  - B's gain: \$3–\$2.50 = \$0.50
- No further gains can be made from trade

Allocations

- How do we find Pareto optimal allocations in general?
  - Private valuations
  - No central control
- Design mechanisms that are efficient and where participants have an incentive to bid their private values
  - Buyers and sellers are symmetrical: may need to flip a coin

Incentive Compatibility of Vickrey

That is, buy bids equal private valuations

- Consider a single seller
- Consider two buyer agents A_1 and A_2, with private valuations v_1 and v_2, bidding b_1 and b_2
- If $b_1 > b_2$, A_1 wins and pays b_2
  - A_1's utility in that case is $v_1 - b_2$: could be positive or negative
- If $b_1 < b_2$, A_1 loses the auction: utility = 0 (assuming no bidding costs)

Argument for Incentive Compatibility

- Utility to bidder A_1: $(v_1 - b_2)$
- If $(v_1 - b_2) > 0$ (i.e., $v_1 > b_2$)
  - Then A_1 benefits by maximizing Prob($b_1 > b_2$)
    - *Underbid:* likelier to lose, but would pay the same price if it wins
  - Else A_1 benefits by minimizing Prob($b_1 > b_2$)
    - *Overbid:* likelier to win, but may pay more than the valuation
- Thus, setting the bid equal to valuation is the best strategy

Basic Idea

- If A_1 wins, what A_1 pays depends on bids by other agents
- A_1 should try to
  - Win when it would benefit by winning
  - Lose when it would suffer by winning

How do the above ideas apply when a buyer is bidding for multiple units of the same item?

M^{th} and $(M+1)^{st}$ Price Auctions

- Vickrey = $(M+1)^{st}$ price, with one unit for sale
- For single-unit buyers, $(M+1)^{st}$ price induces truthfulness
- For multiunit buyers, NO!
  - A buyer may artificially lower some bids to lower the price for other bids

Dominant Strategies

One which yields a greater payoff for the agent than any of its other strategies (regardless of what others bid)

- Under Vickrey auctions, the dominant strategy for a *buyer* is bidding according to its true value
- Under first-price auctions, the dominant strategy for a *seller* is to bid its true value

Multiunit Auctions

- Multiunit bids are *divisible* when not necessarily the whole set needs to be bought or sold
- When multiunit bids are divisible,
  - Treat multiunit bids as multiple copies of single-unit bids
  - If indivisible, e.g., sets of two or four tires, then treat as bundled goods

Desirable Properties of Markets

- Efficient: the one values it most gets it
  - If seller's valuation $<$ buyer's valuation, they trade
- Truthful
  - Rational to bid true valuation for both sellers and buyers
- Individually rational
  - No participant is worse off for participating
- Budget balanced, i.e., no subsidy from the market: Σpayment = Σrevenue
  - Seller receives what the buyer pays

Can all of the above be satisfied?

Impossibility Result

Given a sealed buy bid b and a sealed sell bid s (Meyerson & Satterthwaite)

- Valuations of each from overlapping distributions
- Ultimately buyer pays p_b and seller gets p_s
  - For truthfulness, $p_b = s$ and $p_s = b$
  - But the deal happens only if $b > s$, else irrational
  - Thus buyer pays less than the seller receives, i.e., a deficit!

That is, *subsidize* or relax another requirement

McAfee's Dual Price Auction: 1

- Let p be a price in the equilibrium range
  - That is, Mth to (M+1)st
  - Let's choose the midpoint to be specific
- Omits the lowest buyer at or above Mth and the highest seller at or below (M+1)st

Which of the above properties does the dual price auction violates?

McAfee's Dual Price Auction: 2

- Individually rational
- Promotes truthfulness
- Budget balanced
- *Inefficient*
  - Discards the lowest valued match
  - Not good if it is the only one

Kinds of Prices

- *Uniform:* all matches at the same price
  - Mth, (M+1)st, Dual
- *Discriminatory:* prices vary for each match
  - *Chronological:* prices = earlier (or later) bid
  - Each match at one of the bid prices
    - Buyer's
    - Seller's

Auction Example: 1

- A_1: (sell \$1) at time 1
- A_2: (buy \$3) at time 2
- A_3: (sell \$2) at time 3
- A_4: (buy \$4) at time 4
- Consider $(A_1 \longrightarrow A_4)$, $(A_3 \longrightarrow A_2)$, $(A_3 \longrightarrow A_4)$, $(A_1 \longrightarrow A_2)$

Auction Example: Uniform

- Mth: $(A_1 \longrightarrow A_4$: \$3), $(A_3 \longrightarrow A_2$: \$3), $(A_3 \longrightarrow A_4$: \$3), $(A_1 \longrightarrow A_2$: \$3)
- (M+1)st: $(A_1 \longrightarrow A_4$: \$2), $(A_3 \longrightarrow A_2$: \$2), $(A_3 \longrightarrow A_4$: \$2), $(A_1 \longrightarrow A_2$: \$2)
- Dual: $(A_1 \longrightarrow A_4$: \$2.5), $(A_3 \longrightarrow A_2$: NA), $(A_3 \longrightarrow A_4$: NA), $(A_1 \longrightarrow A_2$: NA)

Auction Example: Discriminatory

- *Earliest:* $(A_1 \rightarrow A_4$: \$1), $(A_3 \rightarrow A_2$: \$3), $(A_3 \rightarrow A_4$: \$2), $(A_1 \rightarrow A_2$: \$1)
- *Latest:* $(A_1 \rightarrow A_4$: \$4), $(A_3 \rightarrow A_2$: \$2), $(A_3 \rightarrow A_4$: \$4), $(A_1 \rightarrow A_2$: \$3)
- *Buyers:* $(A_1 \rightarrow A_4$: \$4), $(A_3 \rightarrow A_2$: \$3), $(A_3 \rightarrow A_4$: \$4), $(A_1 \rightarrow A_2$: \$3)
- *Sellers:* $(A_1 \rightarrow A_4$: \$1), $(A_3 \rightarrow A_2$: \$2), $(A_3 \rightarrow A_4$: \$2), $(A_1 \rightarrow A_2$: \$1)

Continuous Double Auctions (CDAs)

As in stock markets

- *Bid* quote: what a seller needs to offer to form a match
- *Ask* quote: what a buyer needs to offer to form a match
- The *bid-ask spread* represents the difference between the buyers and the sellers

Mth and (M+1)st as Bid-Ask

Generalize English, Vickrey, double auctions

- The Mth price generalizes the ask quote
- The (M+1)st price generalizes the bid quote
- When the standing bids overlap
  - Beating the ask price either matches an unmatched seller, or displaces a matched buyer

Mth and (M+1)st Winning Bid

- A buy bid b wins if either of the following holds
  - Either $b > p_{ask}$
  - Or $b = p_{ask}$ and $p_{ask} > p_{bid}$
- If $b = p_{ask} = p_{bid}$, then it is not clear if b is winning
- Symmetric result for the seller

Module 8: Summary and Directions

Collective concept map

Key Ideas

- Information system interoperation
- Architecture conceptually
- Importance of metadata
- XML technologies
- Elements of rational resource allocation

Business Environments

Theme of this course: How is computer science different for open environments?

- Autonomy
  - Messaging, not APIs
  - Markets
- Heterogeneity
  - Capturing structure of information
  - Transforming structures
- Dynamism
  - Partially addressed through above

Support flexibility and arms-length relationships

Course: Service-Oriented Computing

- Takes the ideas of this course closer to their natural conclusions
- For autonomous interacting computations
  - Basic standards that build on XML
  - Descriptions through richer representations of meaning
  - Engagement of parties in extended transactions and processes
  - Collaboration among parties
  - Selecting the right parties

How to develop and maintain flexible, arms-length relationships