Electronic Commerce Technologies

CSC 513

Spring 2011

Munindar P. Singh, Professor
singh@ncsu.edu

Department of Computer Science
North Carolina State University

Mechanics

- Scope
- Grading
- Policies
  - Especially, academic integrity

Scope of this Course

- Directed at computer science students
- Emphasizes concepts and theory
- Requires a moderate amount of work
- Fairly easy if you don't let things slip

Outline

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

Outline

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

Think of this as an alternative to consistency

- There may be no state (of the various databases) that can be considered consistent
    - Maintaining consistency of multiple databases is difficult
    - Unexpected real-world events can knock databases out of sync with reality
- What matters is
    - Are organizational relationships preserved?
    - Are processes followed?
    - Are appropriate business rules applied?

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

Obsolete way of thinking: tries to achieve consistency (and fails)

Locality and Interaction

A way to maintain coherence in the face of openness

- Have each local entity look after its own
    - Minimize dependence on others
    - Continually have interested parties verify the components of the state that apply to them
- Approach: replace global constraints with protocols for interaction
    - *Lazy:* obtain global knowledge as needed
    - *Optimistic:* correct rather than prevent violations
    - *Inspectable:* specify rules for when, where, and how to make corrections

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 Pertaining to Functionality

- 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?

Architectural Considerations

Architecture is motivated by additional considerations besides functionality

- Instance level, nonfunctional properties such as the availability of a specific service instance
  - What if the payments are made offline, i.e., significantly delayed?
- Metalevel properties such as the maintainability of the software modules and the ease of the upgradability of the system

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 operations
- 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 Operations

Modeling and optimization

- Inventory management
- Logistics: how to optimize and monitoring flow of materiel
- Billing and accounts receivable
- Accounts payable
- Customer support

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: Closed

Manufacturing control: manage the operations of factories

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

Distributed Decision-Making: Open

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?

Outline

The Evolution of IT

- **Applications:** Control of computations hidden in code; integration a nightmare
- **Workflows:** Control abstracted out; integration still difficult
- **Standards-driven orchestration:** Integration improved; limited support for autonomy
- **Messaging:** Integration simplified by MoM and transformations; limited support for autonomy
- **Choreography:** Model conversations over messages; limited support for autonomy
- **Governance:** Administer resources via interactions among autonomous parties

Technical Service

- Generally, an abstraction of a computational object
  - Traditional, as in web or grid services
  - Improved: Abstraction of a “capability”
- Well encapsulated, i.e., a black box
- Interface defined at the level of methods or messages

Service Engagement

An aggregation of business relationships

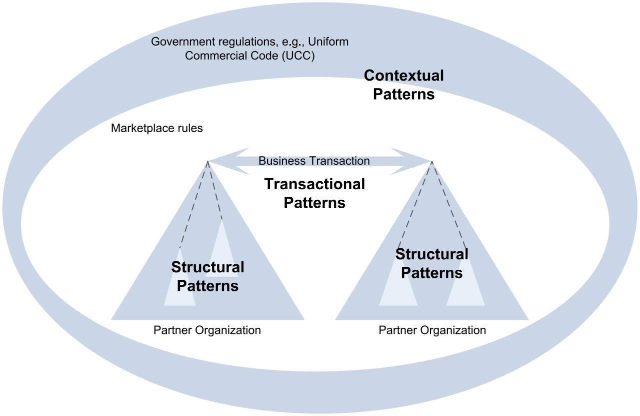
- Trillions of dollars worth of commerce conducted every year
- Characterized by
  - Independence of business partners
  - Coproduction
    - Participation by all, though not at the same level
    - Symmetric relationships: complementary capabilities and goals
  - Complex contracts among the partners
  - Participants are not black boxes

Business Service

Participant in a service engagement

- Characterized by transfer of value, not bits
- Typically long-lived with on demand enactments
- Instantiated on the fly
  - Unlike a product
  - Though may be constructed using products or about products

Conceptual Elements of a Service Engagement



- Transactional: main purpose and enactment, specifying value exchanged
- Structural: partnerships and contracts
- Contextual: setting of the engagement

Traditional Technical Approaches

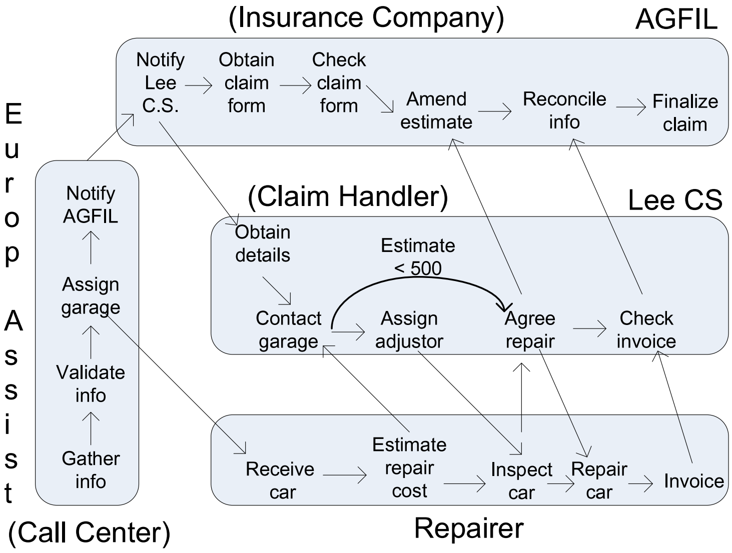
Quite unlike a real-life service engagement

- Take participants flows (e.g., in BPEL, BPMN) as units of abstraction
  - Mix private policies and public interactions
  - Proprietary: may not be available for reuse
  - Context-laden: even when available, cannot be readily reused
- Focus on low-level (e.g., WS-CDL) or data-level meanings (e.g., OWL)
  - Ignore business-level significance of messages
  - Ambiguous; not verifiable

BPEL, BPMN, WS-CDL, OWL are well-known standards

A Real-Life Service Engagement

Operationally over-specified as interacting flows



Outline

Architecture in IT

In the sense of information systems

- Important themes
  - Conceptualizing architecture
  - Enterprise architectures
  - Tiered architectures
  - Architecture as a basis for governance (next section)
- Not quite the same as conventional software architecture, though the topics are converging

Architecture Conceptually

As opposed to the description of a system via a blueprint

- How a system is organized
- An over-used, vaguely defined term
  - Software architecture
  - Standards, e.g., Berners-Lee's "layer cake" and the networking standards
  - May include processes
    - That exercise the system
    - By which the system is built and maintained
    - By which the system is administered
  - May include human organizations

Understanding Architecture: 1

- 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

Exercise: Examples of Architecture

Identify the main components and interconnections

- Buildings
- Plumbing
- Power systems

Understanding Architecture: 2

- Components and interconnections are not sufficient to characterize an architecture
- Two additional ingredients of an architecture
  - Constraints on the components and interconnnections
  - Patterns involving the components and interconnnections
- *Openness* entails the constraints
  - Do not apply on the physical components directly

Exercise: Examples of Architecture

Identify the main constraints and key patterns

- Buildings
- Plumbing
- Power systems

Understanding Protocols

- Protocols encapsulate IT interactions, i.e., interconnections over which information is the main thing that flows
  - *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

Examples of Logical Architectural Components

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?

Outline

IT Architectures

The term *architecture* is used more broadly in IT settings

- The organization of an IT system
- The extensibility and modifiability of a system
- Even the governance of a system, which inevitably accommodates the human organization where the system is deployed

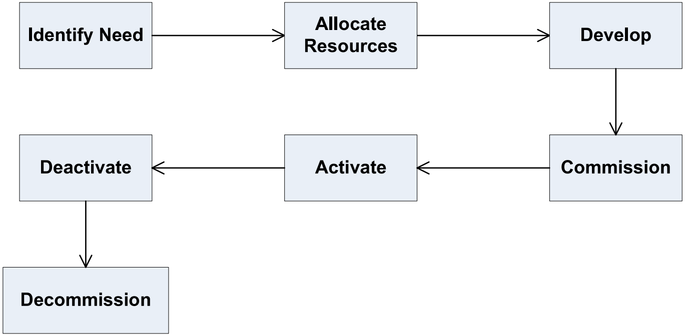
IT and SOA Governance

The human administration of an IT system

- IT Governance: How IT resources are administered
- SOA Governance: How services are created, deployed, removed, . . .
- Goes hand-in-hand with architecture
  - Incorporates
    - The human organization of a system
    - The processes through which a system is updated or upgraded
    - Nontechnical aspects, such as flows of responsibility
  - Sometimes confused with architecture, but distinct

Governance in the Service Life Cycle

Key determinations



Enterprise Models: Information Resources

Capture static and dynamic aspects

- Statically
    - Databases and knowledge bases
    - Applications, business processes, and the information they create, maintain, and use
- Through explicit representations, dynamically enable
    - Integrity validation
    - Reusability
    - Change impact analysis
    - Software engineering: Automatic database and application generation via CASE tools

Enterprise Models: Rationales

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

Enterprise Architecture Objectives

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

- Accommodating *change* by introducing new
  - Users
  - Applications
  - Ways of interaction (e.g., ongoing push toward mobility)
- Managing information resources
  - Preserving prior investments by interoperating with legacy systems
  - Upgrading resources
- Developing blueprints to guide 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 other relationships among stakeholders (including 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 directly visible to users
  - Application deployment
  - Data modeling and integrity
  - Business intelligence: decision support and analytics
- More technical but indirectly visible to users
  - 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, e.g., ID across a system to support Single Sign On
- Security and audit
- Accessibility
- Policy repositories and engines

Architecture Modules: Infrastructure

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

Functionalities in a Working Enterprise System

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

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

Outline

One-Tier and Two-Tier Architectures

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

Three-Tier Architecture

- 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
- 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

- 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 governance, including scaling up of resources
  - 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

Outline

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 (and some others)
  - 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: in terms of sessions
  - What is an advantage of statelessness?
  - Where is the session state kept then?
- Focus on resources being manipulated and their representations being transferred
- Uniform Interface: URIs, hypermedia
- Caching: responses can be labeled as cacheable

Outline

Middleware Practically

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 Conceptually

Components that implement important architectural interfaces

Key examples: transaction and persistence

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

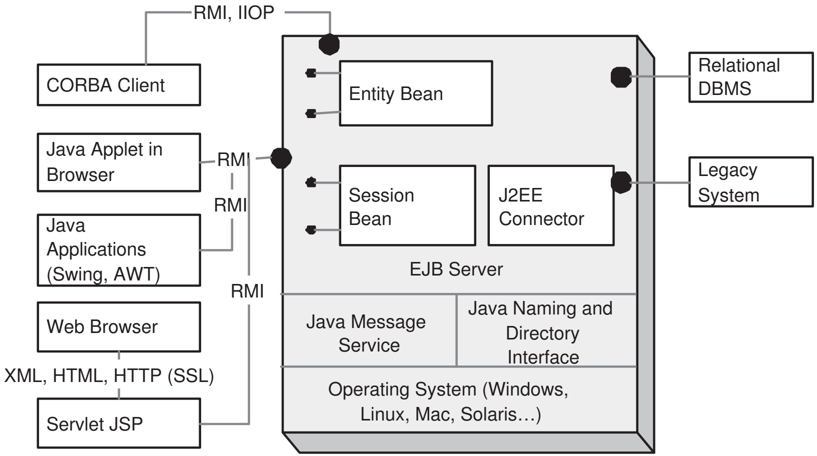
Containers

Distributed object management

- Architectural abstraction geared for hosting business components (objects)
  - Remote method invocation
  - Threading
  - Messaging
  - Transactions
- Implementations for JEE and .NET

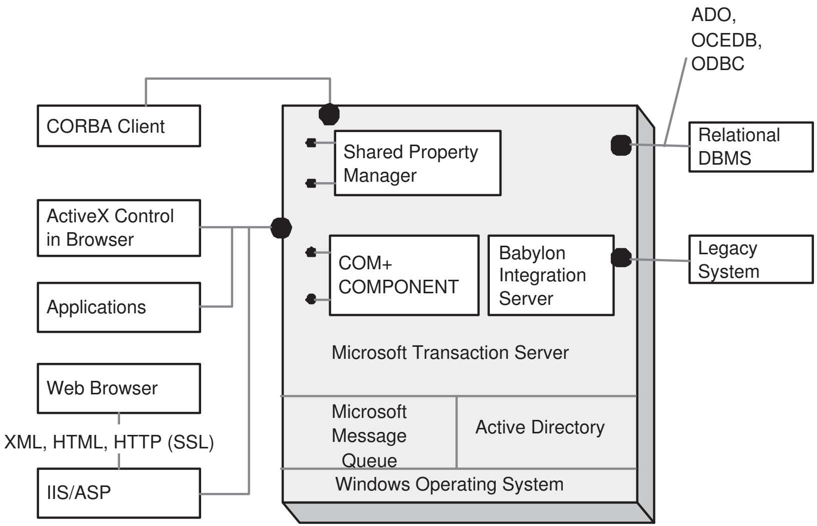
JEE Technology

Vernadat



.NET Technology

Vernadat



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: Types

Realized in distributed objects, e.g., 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: Execution

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

- Main varieties
  - *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
  - Usually implemented over databases

A standardized callback for messages

- Accessed only via messages, not invoked
- No need for specialized interfaces, such as **home**, **remote**, . . .
- Easy interface to implement: **onMessage()**
    - Programmer defines what actions to take on receipt of a message
    - Limited message typing
- Stateless: thus no conversations

Peer-to-Peer Computing

Traditionally, hard coded into specific applications, such as file sharing

- *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
    - Business protocols being defined in terms of business interactions, not low-level messaging

Enterprise Best Practices

- Enterprise Service Bus
  - Builds on top of messaging
  - Provide orchestration as a way to realize business processes
  - Shields programmer from transport and location considerations
- Asset repository
  - Building on directory services
- Data dictionary
  - Evolving into one based on ontologies

Outline

Deployment Architecture: Web Server

- Frontend
- Supports HTTP operations
- Usually multithreaded

Deployment Architecture: 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) with tradeoffs in
    - Overhead: OS processes versus threads
    - Scalability
    - Security

Application Server as an Architectural Abstraction

Separates business logic from infrastructure

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

Separate programming from administration roles

Deployment Architecture: Database Server

- Holds the data, ensuring its integrity
- Manages 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

Outline

Contract

Binding agreement specifying each party's expectations on the others

- A contract structures interactions among autonomous parties
  - People and corporations
  - Governmental agencies
- Unlike a contract in programming
- Key questions: how to create, modify, perform, or monitor a contract

Motivation for Contracts

- Provide a basis for service agreements
- Crucial in open environments
  - Emphasize behavior: observable by others
  - Constrain behavior: limit autonomy
  - Except where needed, generally disregard internal implementations, thus facilitating heterogeneity

What is a Contract?

A description of business-level interactions

A reusable description of an interaction understood to preserve the participants' autonomy

- Analogous to an abstract class or interface for objects
- Specifies well-defined roles
- Specifies messages among the roles and how they affect interaction state
  - Capturing commitments on a business partner playing a role
  - Setting local policies while complying with a protocol
- Stored in a repository, i.e., as an asset or resource in its own right
- Refined and composed for implementation

Importance of Governance

Stakeholders using resources to best serve their needs

- Share resources in a controlled manner
- Configure and reconfigure
- Enable unanticipated uses for resources
- Administer respecting human organizational needs

In particular, stakeholders administer themselves

Governance versus Management

Alternative approaches to administration

- *Management:* by superiors of subordinates
  - Control over managed resources
  - Necessary but not sufficient
- *Governance:* by autonomous equals of themselves
  - Collaborative decision-making among stakeholders
  - Share resources flexibly, enabling unanticipated uses
  - Administer respecting human organizational needs
- Governance is what is needed, yet metaphors and approaches deal with management

Thus, governance is hidden: manual via out-of-band communications

Difficulty of Governance

Independence of stakeholders motivates high-level normative descriptions

- *Autonomy:* Stakeholders behave independently, constrained only by their agreements
- *Heterogeneity:* Stakeholders are independently constructed, constrained only by interface descriptions
- *Dynamism:* The set of stakeholders and their mutual relationships may change continually

Understanding Governance

Philosophy

Governance is about how stakeholders administer their resources

- Focus on stakeholders
- Focus on interactions among stakeholders, framed as normative relationships
- Focus on policies (capture autonomy)
- Focus on where the policies apply (where each party acts)
- Focus on perspicuous specification of policies

Applying Contracts in IT Administration

Governance of service engagements

- Currently, humans achieve governance manually
  - Low productivity
  - Poor scalability to fine-grained, real time governance decisions
  - Hidden, implicit considerations yield low confidence in correctness and poor maintainability
- Can we address governance through contracts?
  - Applied commonly for external services: SLAs generally, cloud services
  - Apply within Org as well

Approach: Contracts and Policies

Both are centered on interaction, but . . .

- Contracts are modules of abstraction
- Policies are inherently private
- Policies lead each party to adopt a contract and decide whether and how to act given a contract
- Methodologically, we advocate going top down
  - Identify contracts
  - Identify *policy points* in a contract
  - Thus improving modularity and reusability

Outline

Prerequisites for Realizing Contracts

- Formal, computational representations
- Determining (by each party) how to act
- Monitoring, especially locally by each party
- Judging compliance
- Enforcing contracts
- Dealing with legacy contracts
- Maintaining (includes creating) contracts
- Analyzing contracts, whether to adopt or enact

Hypothesis

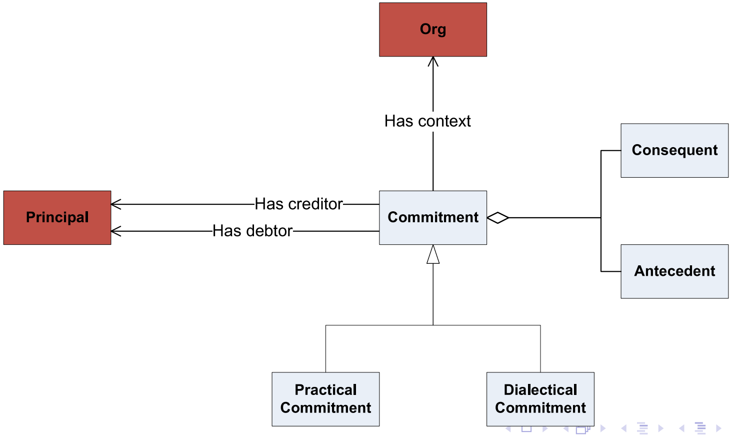
Governance is a basis for understanding contracts even outside of IT

- Each contract is governed
- Reify organization into an Org, where
  - Members are stakeholders
  - The Org itself is a stakeholder
  - The Org provides the *context* of the contract
- The Org handles
  - Identity
  - Enrollment: who can contract
  - Enforcement
- Each member handles
  - How to act: policies
  - Where to monitor
  - Whether to escalate

Commitments as Elements of a Contract

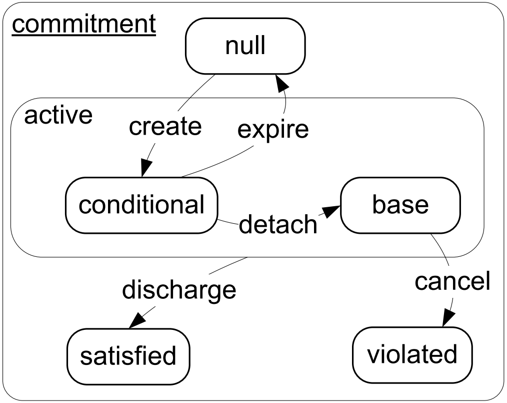
Express meanings of interactions

- Are atoms of contractual relationships
- Enable correctness checking of contracts
- Yield precise meanings and verifiability

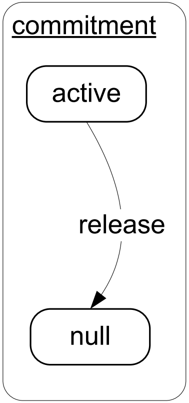


Commitment Life Cycle (and Patterns)

C(debtor, creditor, antecedent, consequent)



(a) Commit



(b) Relieve

Commitment Operations

- *create(C(x, y, p, q))* establishes the commitment
- *detach(C(x, y, p, q))* turns it into a base commitment
- *discharge(C(x, y, p, q))* satisfies the commitment
- *cancel(C(x, y, p, q))* cancels the commitment
- *release(C(x, y, p, q))* releases the debtor from the commitment
- *delegate(z, C(x, y, p, q))* replaces x by z as the debtor
  - x remains ultimately responsible (in our work)
- *assign(w, C(x, y, p, q))* replaces y by w as the creditor

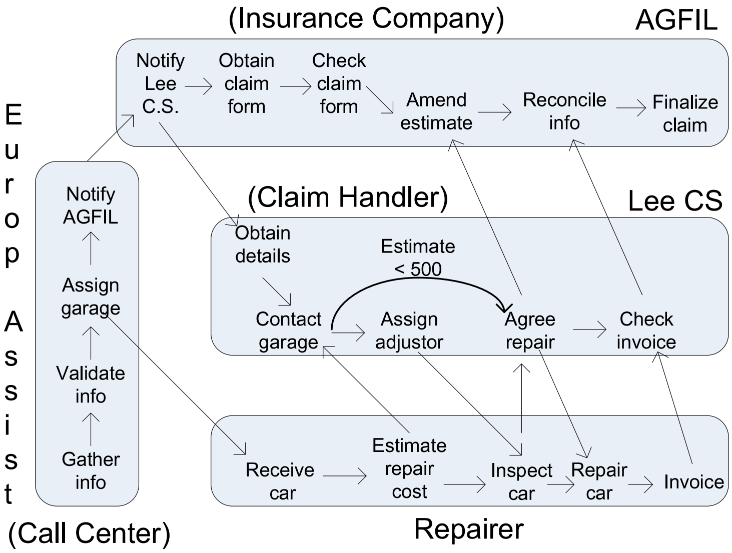
Example: Commitment Progression

C(Buyer, Seller, goods, pay)

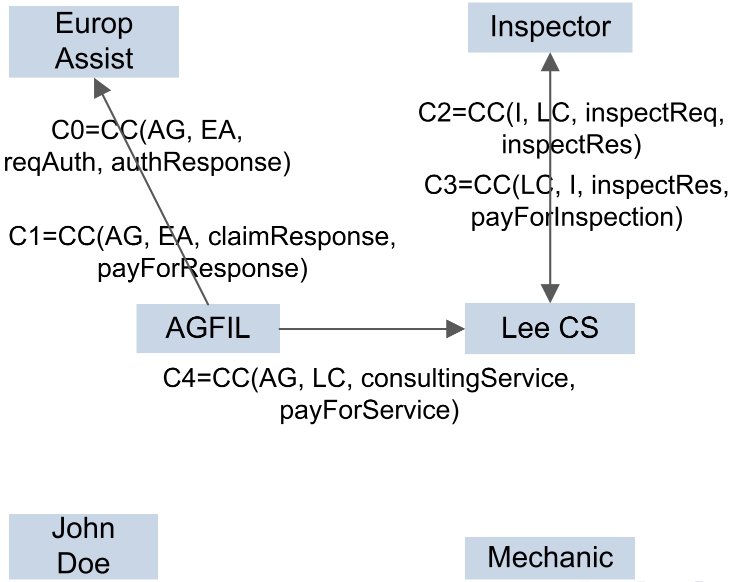
- If *goods* $\wedge$ C(Buyer, Seller, goods, pay) Then
  - C(Buyer, Seller, T, pay)
- If *pay* $\wedge$ C(Buyer, Seller, T, pay) Then
  - Satisfied
- If *pay* $\wedge$ C(Buyer, Seller, goods, pay) Then
  - Satisfied

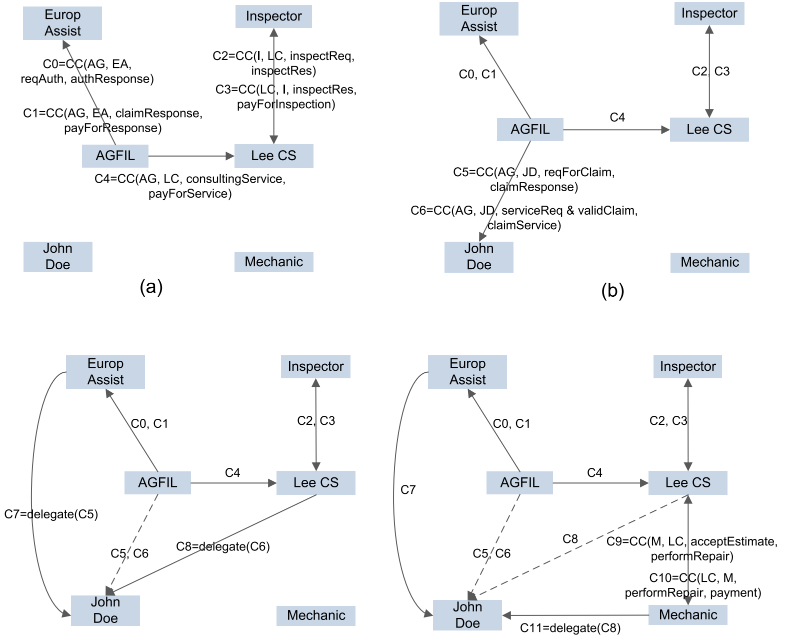
Can be nested:
C(Seller, Buyer, pay, C(Shipper, Buyer, T, deliverGoods))

A Real-Life Service Engagement (Repeated)



Example Contractual Relationships (at Outset)


Europ Assist
Inspector
C0=CC(AG, EA, reqAuth, authResponse)
C2=CC(I, LC, inspectReq, inspectRes)
C3=CC(LC, I, inspectRes, payForInspection)
C1=CC(AG, EA, claimResponse, payForResponse)
AGFIL
Lee CS
C4=CC(AG, LC, consultingService, payForService)
John Doe
Mechanic


Europ Assist
Inspector
C0=CC(AG, EA, reqAuth, authResponse)
C2=CC(I, LC, inspectReq, inspectRes)
C3=CC(LC, I, inspectRes, payForInspection)
C1=CC(AG, EA, claimResponse, payForResponse)
AGFIL
Lee CS
C4=CC(AG, LC, consultingService, payForService)
John Doe
Mechanic
(a)
Europ Assist
Inspector
C0, C1
C2, C3
AGFIL
C4
Lee CS
C5=CC(AG, JD, reqForClaim, claimResponse)
C6=CC(AG, JD, serviceReq & validClaim, claimService)
John Doe
Mechanic
(b)
Europ Assist
Inspector
C0, C1
C2, C3
AGFIL
C4
Lee CS
C7=delegate(C5)
C5, C6
C8=delegate(C6)
John Doe
Mechanic
(c)
Europ Assist
Inspector
C0, C1
C2, C3
C7
AGFIL
C4
Lee CS
C8
C9=CC(M, LC, acceptEstimate, performRepair)
C5, C6
C10=CC(LC, M, performRepair, payment)
John Doe
Mechanic
C11=delegate(C8)
(d)

Achieving Governance: Agents and Orgs

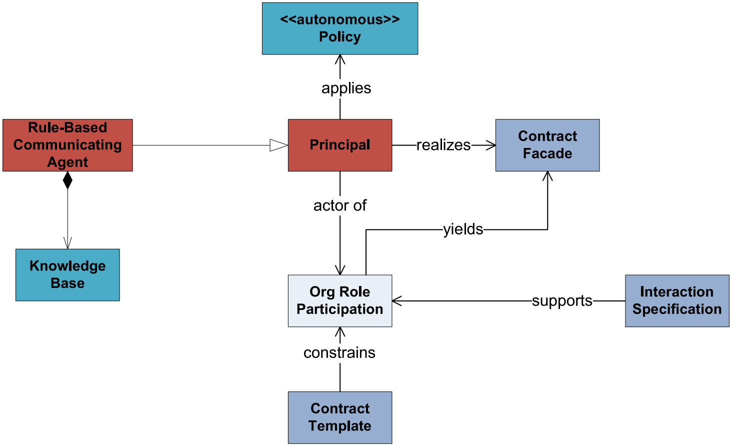
Put collaboration center stage

- Agents represent the stakeholders: people and organizations
  - Provide a locus for interaction
- Orgs are like *institutions:* have an identity and life time distinct from their members; also modeled as agents
  - Examples: NCSU, UNC System, . . .
  - Provide a locus for roles and authorizations
  - Enforce behavioral constraints on members
    - Their main hold over their members is the threat of expulsion

Duality of Contracts and Orgs

- A set of contracts define an Org
  - Roles, with their qualifications, privileges, liabilities
- An Org provides the context for defining contracts

Governance Overview

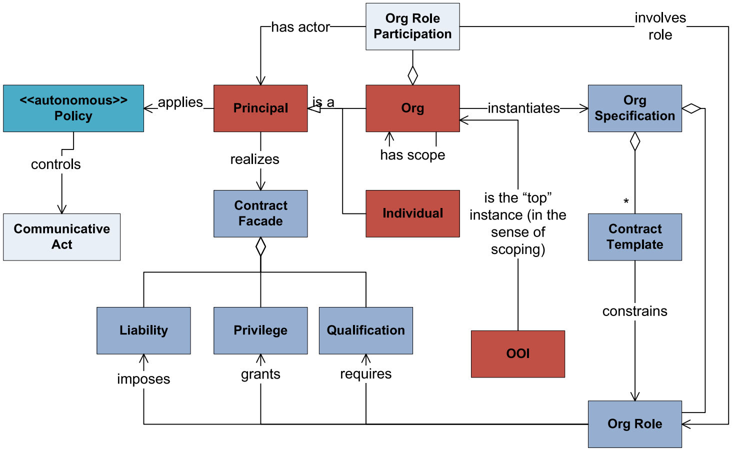


Approach

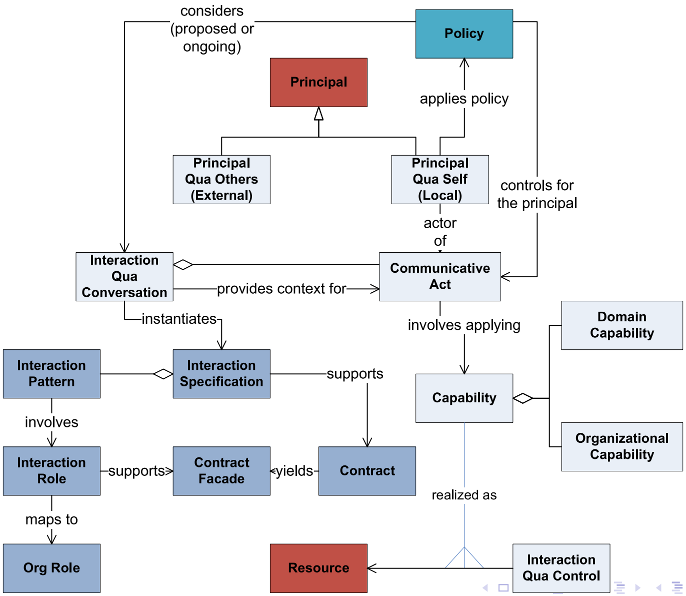
Based on a conceptual model for governance

- Determine what attributes are subject to Identity Management
- Specify an execution architecture
- Specify interactive aspects building on the execution architecture
- Determine a core language for expressing governance structures, policies, and interactions
- Understand policy authoring needs

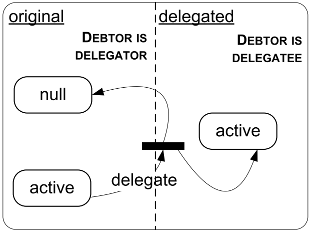
Governance Conceptually



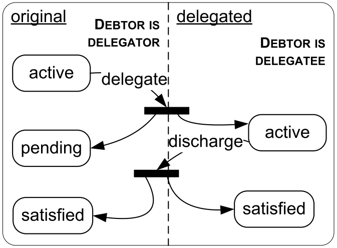
Governance Operationally



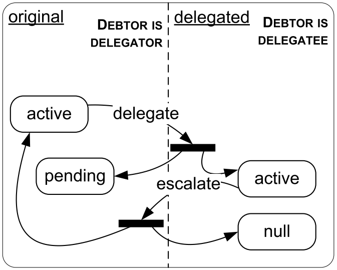
Patterns for Delegate



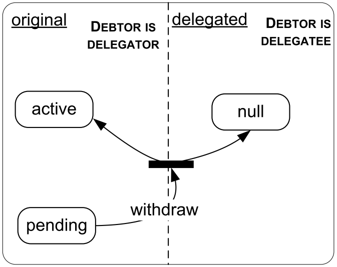
(a) Transfer responsibility



(b) Retain responsibility



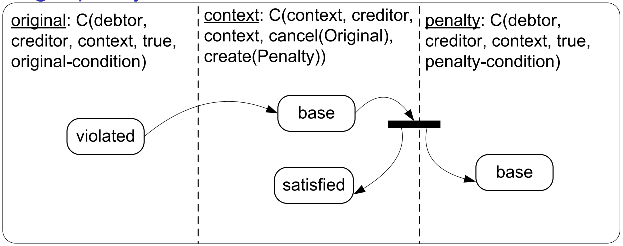
(c) Escalate



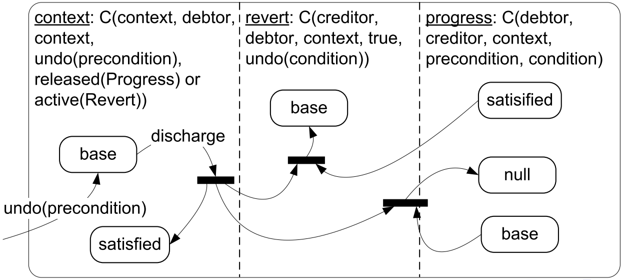
(d) Withdraw delegation

Contextual Patterns: Penalize and Revert

Introducing an Org explicitly as the context


original: C(debtor, creditor, context, true, original-condition)
context: C(context, creditor, context, cancel(Original), create(Penalty))
penalty: C(debtor, creditor, context, true, penalty-condition)
violated
base
satisfied
base

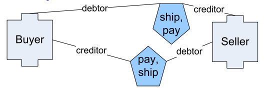
(a) Penalize


context: C(context, debtor, context, undo(precondition), released(Progress) or active(Revert))
revert: C(creditor, debtor, context, true, undo(condition))
progress: C(debtor, creditor, context, precondition, condition)
base
discharge
undo(precondition)
satisfied
base
satisified
null
base

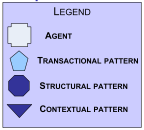
(b) Revert offer

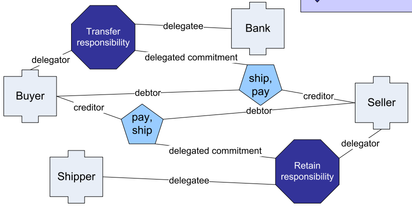
A Purchase Service Engagement

Demonstrates incremental specification as a form of stepwise refinement

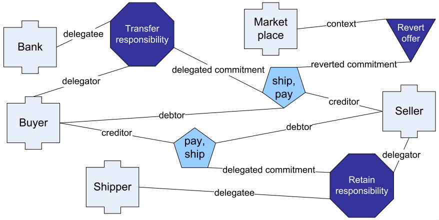


(a) Pair of conditional commitments describing purchase



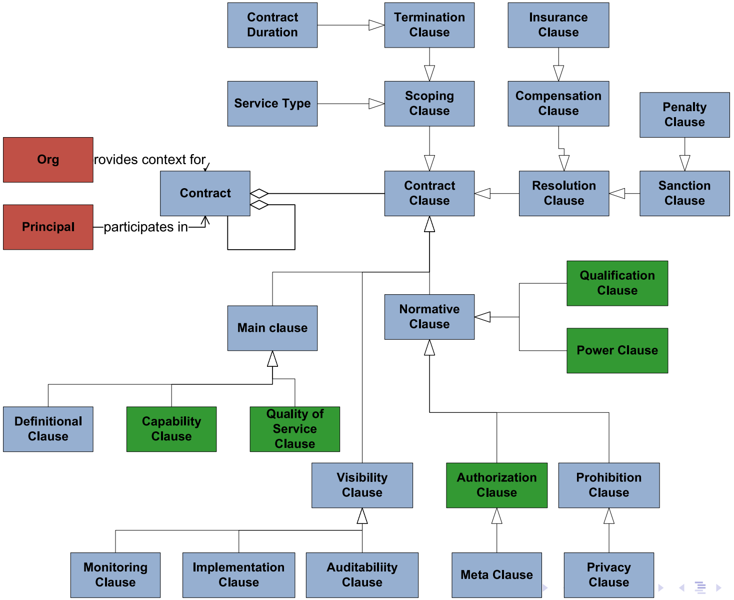


(b) Introducing bank and shipper via delegations of commitments



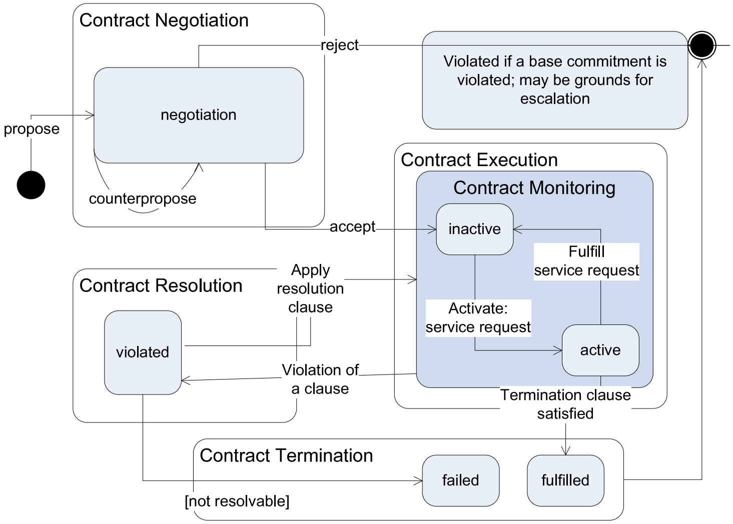
(c) Allowing buyer to skip payment or get a refund upon returning goods

Contracts


Contract Duration
Termination Clause
Insurance Clause
Service Type
Scoping Clause
Compensation Clause
Penalty Clause
Org
rovides context for
Contract
Contract Clause
Resolution Clause
Sanction Clause
Principal
participates in
Qualification Clause
Main clause
Normative Clause
Power Clause
Definitional Clause
Capability Clause
Quality of Service Clause
Visibility Clause
Authorization Clause
Prohibition Clause
Monitoring Clause
Implementation Clause
Auditabiliity Clause
Meta Clause
Privacy Clause

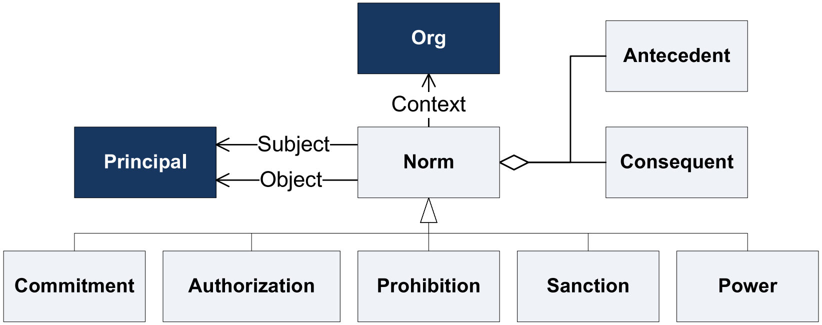
Contracts Life Cycle

Vázquez-Salceda et al.



Additional Norms

Directionality and context are key features

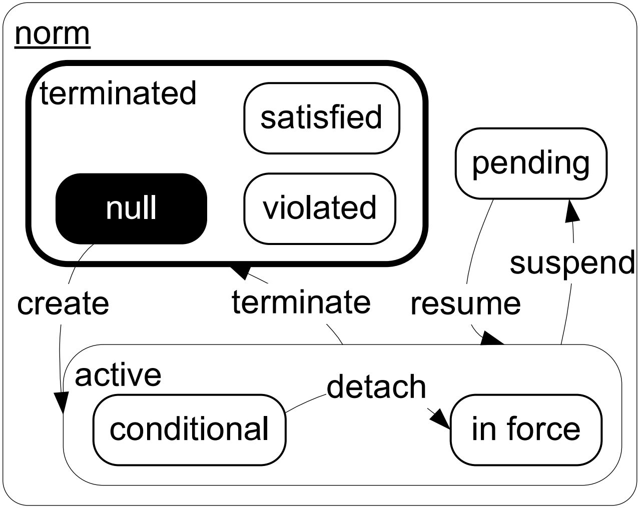


Permission versus prohibition and sanction

Norms and Façades

Normative Concept	**Subject's Façade**	**Object's Façade**
Commitment	Liability	Privilege
Authorization	Privilege	Liability
Power	Privilege	Liability
Prohibition	Liability	Privilege
Sanction	Liability	Privilege

Norm Life Cycle: 1



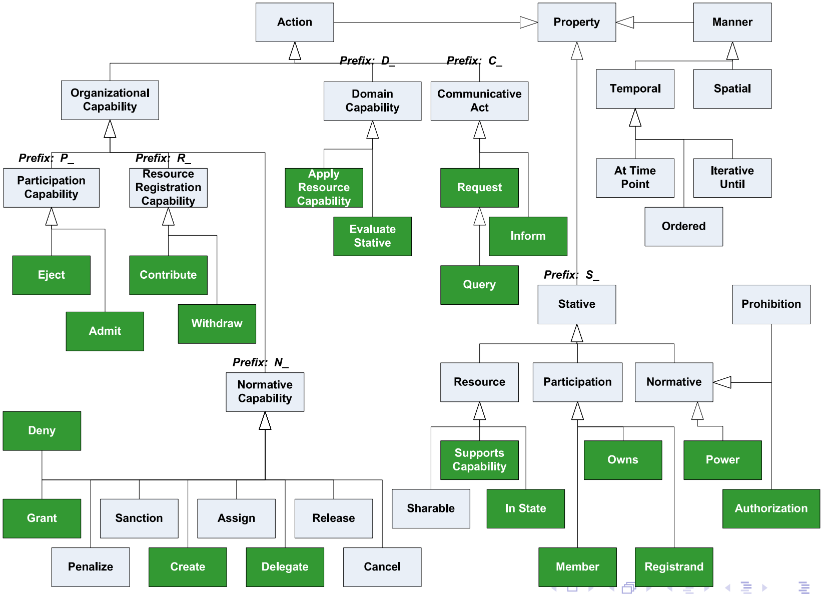
Norm Life Cycle: 2

Computing the substate of a terminated norm

If terminated in		**Then**				
ant	**con**	**Com**	**Aut**	**Pro**	**San**	**Pow**
false	false	null	null	null	null	null
false	true	sat	vio	null	null	null
true	false	vio	null	sat	null	vio
true	true	sat	sat	vio	sat	sat

In the case of a power, a vio occurs upon the failure of an attempt to bring about the consequent.

Contracts Vocabulary



States, Events, and Processes

A Linguistic Perspective

- Verbs versus nouns
- Verbs: states or events
  - Can often lift events to states: maintaining maps to being in the maintenance phase
  - Can identify events from state transitions
- Nouns: objects
  - Count: discrete units
  - Mass: dense or continuous substances

State

How things are

- What the user believes or knows
- What a business partner is committed to performing
- On
- Off
- Busy
- Idle
- Hibernating

Telicity

From Greek "telos"

- Telic
  - Refers to goal-directed events
- Atelic
  - Refers to nondirected events

Graduality

Events naturally map over objects

- Gradual
  - Proper parts of an event are of the same type
  - Walking in the park
  - Working on a project

Achievement

Immediate and directed at a goal

- Opening a door
- Completing a phase
- Committing a database transaction
- Reaching a project milestone

Activity

Ongoing (thus conceptually long-lived) and undirected

- Taking a walk
- Marketing a product
- Maintaining a software product
- Monitoring a network
- Tuning system performance
- Listening to a socket (for incoming requests)
- Searching for interesting tidbits
- Innovating

Accomplishment

Long-lived and directed toward a goal

- Building a house
- Gathering requirements
- Searching for specific information
- Finishing a project within a deadline
- Executing an algorithm
- Completing a business process
- Obtaining certification

Semelfactive

Immediate but not directed as such

- Sneeze
- Power failure
- Network failure
- Incoming bot attack

Outline

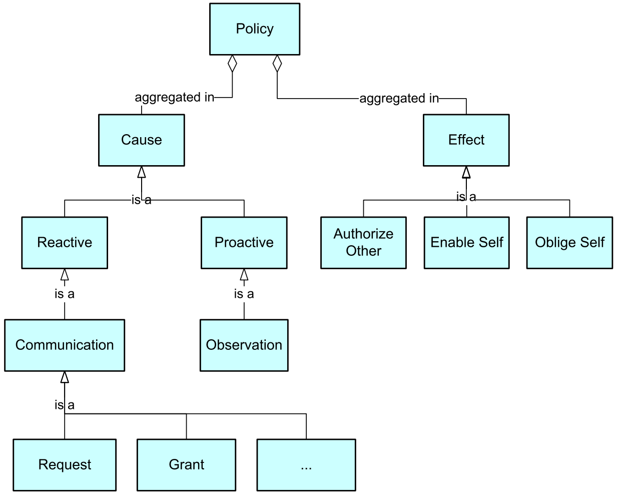
Challenges for Policies and Decision Making

- *No unique locus:* separate policies for each autonomous participant
- *Dependence on business relationships*
- *Complexity of modeling*
  - Specifying vocabulary pertinent to service engagements
  - Determining where policy decisions apply
- *Idea:* Architecture for governance centered on interactions viewed as conversations
  - *Domain-specific policies:* Incorporate monitoring and responding to events
  - *Generic policies:* Altering business relationships

Policy Model: Types

Provide a basis for achieving governance; hence must go beyond traditional access control

- Each policy can be understood in terms of its cause and its effect
- Cause
  - *Reactive:* triggered by a request from another stakeholder
  - *Proactive:* triggered by local observations
- Effect
  - *Authorization* of action to be taken on behalf of requester
  - *Enablement* of action, which would otherwise not be taken
  - *Obligation* of action, which would now be performed


Policy
aggregated in
aggregated in
Cause
Effect
is a
is a
Reactive
Proactive
Authorize Other
Enable Self
Oblige Self
is a
is a
Communication
Observation
is a
Request
Grant
...

Policy Model: Information

Each policy relies upon certain information in order to produce a decision

- Attributes of the parties involved
  - Qualifications, affiliations
- Attributes of the capabilities involved
  - Interactions to be carried out upon resources
  - Collated as interaction types and resource types
- Attributes of the relationships among the parties involved
  - Participations in different Orgs
  - Arrangements among institutions (captured as participations)
  - Ongoing conversations

Outline

DoDAF and Friends

Department of Defense Architecture Framework

- A standardized way to organize an enterprise architecture
- Lists 26 *views* organized into four categories
- Roughly, a software methodology
  - How to capture requirements: user activities
  - How to develop solutions: meet performance criteria
  - How to consider technical standards
- Best for large systems with lifetimes of decades
- Relates to MoDAF and TOGAF and to OMG's MDA

DoDAF View Sets

- All Views (AVs)
  - Two overall views
  - Include the terminology used in the remainder of the views
- Operational views (OVs)
  - Seven
  - Describe the working of the system as visible to its stakeholders
  - Formalized using technical notations such as UML Activity Diagrams and UML Message Sequence Diagrams
- Systems and services views (SVs)
  - Seventeen
  - Describe service interfaces, their performance and physical schemas, and the functioning of the system close to an implementation level
- Technical standards views (TVs)
  - Two
  - Relate the system to potentially relevant standards and technologies

Methodology for Developing an Enterprise Architecture

- Identify stakeholders
- Understand the stakeholders' requirements
  - Begin with the customers
- Identify interactions
- Identify the business services as offered by the enterprise
  - Business processes
  - Ongoing activities
- Identify components needed to support the above services.
  - Organization
  - Services
  - Data

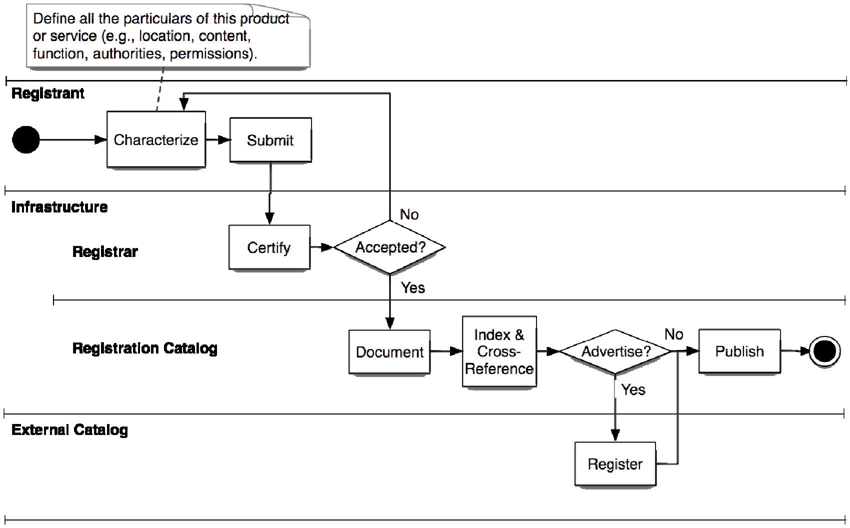
Ongoing Studies

Ocean Observatories Initiative (OOI)

- Primary: Operational Activity Model (OV5) document describing the entire life cycle via several use cases
  - Resources being created
  - Resources being registered and published
  - Resources being commissioned and decommissioned
  - Several more . . .
- Secondary: OOI Concept of Operations document

The OV5 Register Activity Diagram

Developed by others

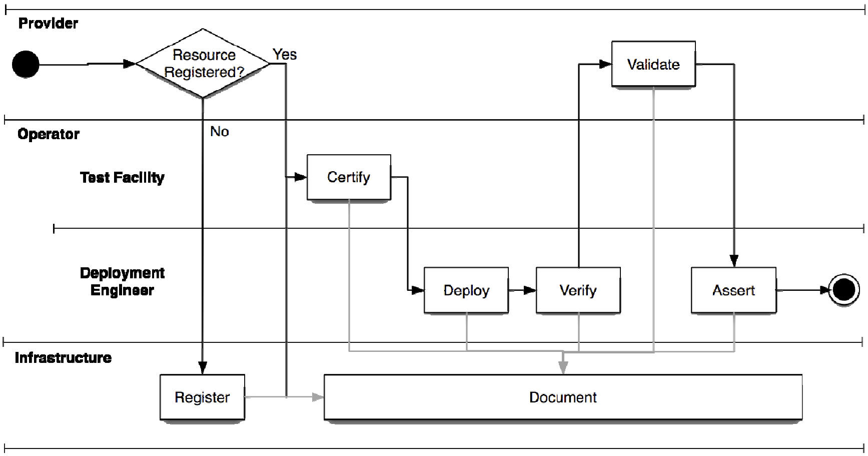


What We Extract from the OV5 Register Activity

- Roles
  - Registrar (e.g., facility administrator)
  - Registrant (e.g., a researcher)
- Main interactions
  - Registrant registers a new resource (e.g., a data stream) to make it available to others
  - Registrar advertises a registered resource
- Policy points for the registrar
  - Whether to accept the registrant's request
  - Whether to advertise a registered resource

The OV5 Commission Activity Diagram

Developed by others

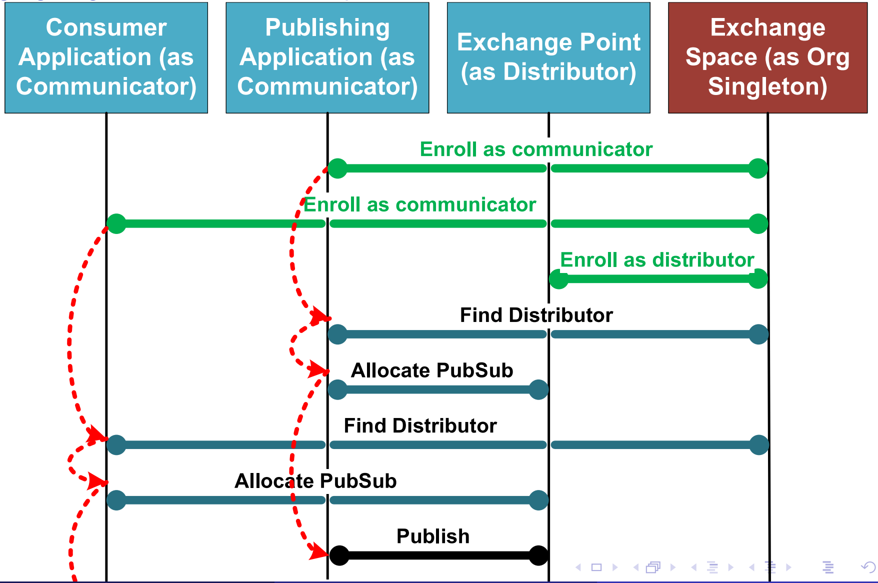


What We Extract from the OV5 Commission Activity

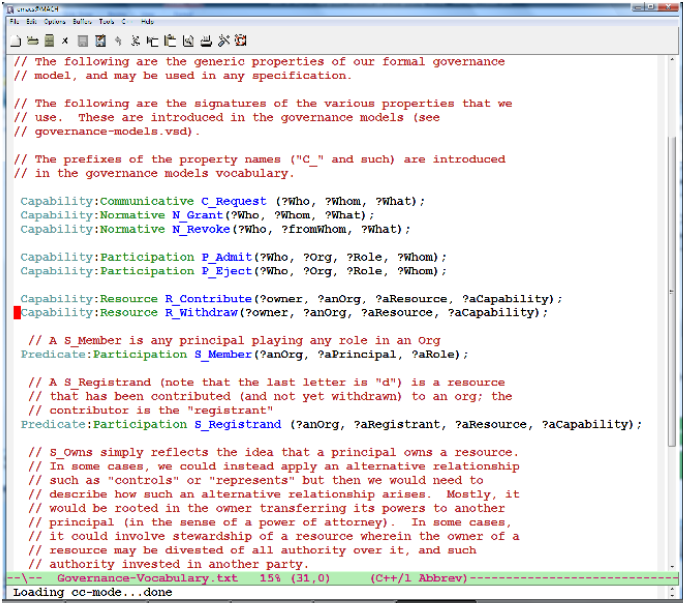
- Roles
  - Operator (e.g., a test facility or deployment engineer)
  - Provider (e.g., a researcher)
- Main interactions
  - Provider requests the operator to certify a data stream from a sensor
  - Operator completes verification of deployment of a sensor that has been requested for commissioning
- Policy point for the operator
  - Whether to accept the provider's request
- Policy point for the provider
  - Whether to proceed to validate the deployment

Governance of AMQP Exchange Space

Highlighting the business relationships

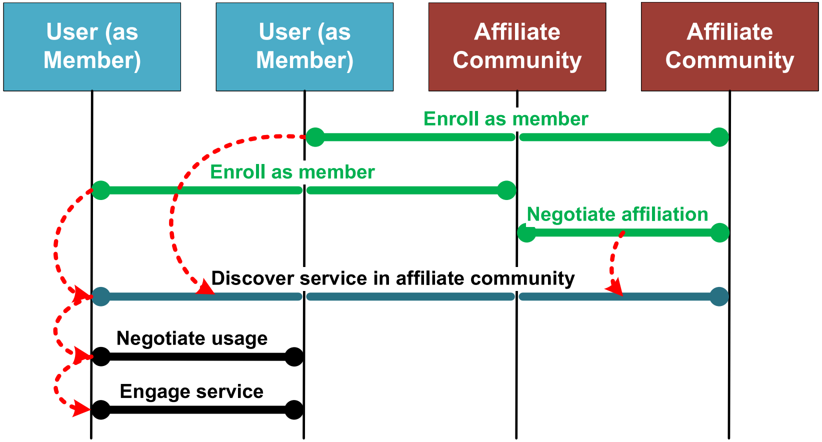

Consumer Application (as Communicator)
Publishing Application (as Communicator)
Exchange Point (as Distributor)
Exchange Space (as Org Singleton)
Enroll as communicator
Enroll as communicator
Enroll as distributor
Find Distributor
Allocate PubSub
Find Distributor
Allocate PubSub
Publish

Vocabulary Example for a Resource Sharing Community



```
// The following are the generic properties of our formal governance
// model, and may be used in any specification.

// The following are the signatures of the various properties that we
// use.  These are introduced in the governance models (see
// governance-models.vsd).

// The prefixes of the property names ("C_" and such) are introduced
// in the governance models vocabulary.

 Capability:Communicative C_Request (?Who, ?Whom, ?What);
 Capability:Normative N_Grant(?Who, ?Whom, ?What);
 Capability:Normative N_Revoke(?Who, ?fromWhom, ?What);

 Capability:Participation P_Admit(?Who, ?Org, ?Role, ?Whom);
 Capability:Participation P_Eject(?Who, ?Org, ?Role, ?Whom);

 Capability:Resource R_Contribute(?owner, ?anOrg, ?aResource, ?aCapability);
 Capability:Resource R_Withdraw(?owner, ?anOrg, ?aResource, ?aCapability);

  // A S_Member is any principal playing any role in an Org
 Predicate:Participation S_Member(?anOrg, ?aPrincipal, ?aRole);

  // A S_Registrand (note that the last letter is "d") is a resource
  // that has been contributed (and not yet withdrawn) to an org; the
  // contributor is the "registrant"
 Predicate:Participation S_Registrand (?anOrg, ?aRegistrant, ?aResource, ?aCapability);

  // S_Owns simply reflects the idea that a principal owns a resource.
  // In some cases, we could instead apply an alternative relationship
  // such as "controls" or "represents" but then we would need to
  // describe how such an alternative relationship arises.  Mostly, it
  // would be rooted in the owner transferring its powers to another
  // principal (in the sense of a power of attorney).  In some cases,
  // it could involve stewardship of a resource wherein the owner of a
  // resource may be divested of all authority over it, and such
  // authority invested in another party.
--\--  Governance-Vocabulary.txt   15%  (31,0)    (C++/l Abbrev)------------------------------
Loading cc-mode...done
```

Governance of Community Affiliation Scenario



Outline

Outline

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
- Representing semistructured descriptions:
  - Products, services, catalogs
  - Contracts
  - Queries, requests, invocations, responses (as in SOAP): basis for Web services

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

Meaning in XML

- Relational DBMSs work for highly structured information, but rely on column names for meaning
- Same problem in XML (reliance on names for meaning) but better connections to richer meaning representations

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'/>
```

Outline

XML Query and Manipulation

Main XML query and manipulation languages include

- XPath
- XQuery
- XSLT
- SQL/XML

Metaphors for Handling XML: 1

How we conceptualize XML documents determines our approach for handling them

- *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 an information model
  - 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 (—), analogous to 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

```
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**

```
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

```
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
- An empty sequence maps to **false**
- A sequence whose first member is a node maps to **true**
- A numeric that is 0 or NaN maps to **false**, else to **true**
- An empty string maps to **false**, others to **true**

Defining Functions

```
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

```
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:

XSLT Templates: 1

A pattern to specify where the given transform should apply: an XPath expression

- Match only 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:
  ```
  <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"/>
<!-- 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

Outline

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

```
  ...

 public void endElement (String uri, String n,
                                    String qName) {
  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

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

Outline

XML Modeling and Storage

The major aspects of storing XML include

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

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 . . .

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)

Outline

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',
        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

```
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;
```

Outline

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 **&**, **&amp;**, **#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

XML **key**, **unique**, **keyref** as introduced in the XML Keys section

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

Outline

XML Schema Motivated

Validation means 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 of subelements
  - Can occur within other compositors
  - Allows varying min and max occurrence
- *All:* unordered subelements
  - Must occur directly below root element
  - Max occurrence of each element is 1
- *Choice:* exclusive or: include one subelement
  - 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"/>
```

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"**)

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

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

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'>
</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

Outline

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 Constraints: 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 Constraints: 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.

Outline

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

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, . . .

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