COLLEGE of CHARLESTON

Secure Software Development for the Cloud

Developing Secure Distributed Algorithms & Architectures

Aspen Olmsted

Cybersecurity/Software Engineering

- Confidentiality - develop algorithms and SOA architectures protecting privacy
- Integrity – develop algorithms and SOA architectures guaranteeing strong properties
- Availability - develop algorithms and SOA architectures providing high availability
- Budget – maintaining the CIA needs to be done within a budget

Secure Software Development

- Not just about protecting software from malicious users
  - Developing software to meeting non-functional requirements
  - Developing software to guarantee correctness

Outline of the Presentation

- Secure Data Engineering Lab @ College of Charleston
- Secure Software Development
- Overview of High Traffic Cloud/SOA vs. 2-Tier Architectures
- Lost Updates Example
- Related Research
- Research
  - Buddy System
  - Capacity Constraints
  - Coarse Grained Services
  - Service Constraints
  - Business Filters
  - Long Running Transactions
  - Integrating Heterogeneous Systems
  - Autonomous Process Authentication
  - Modeling Vulnerable Application Partitions
- Questions

Applied Cybersecurity Research

- Many Cybersecurity programs low level blooms (memorization/understanding)
  - Security+
  - CISSP
- Bachelor's/Master's programs

  should be higher level blooms

  (creating/evaluating/analyzing)

Mentoring Students Through SE

- Design & develop secure enterprise software for industrial clients
  - Gettysburg Foundation
  - New York Philharmonic
  - Seattle Art Museum
  - RPM Healthcare
  - Footlight Theatre
  - Pure Theatre

Funding

- Reverse junior level course in software architecture
- Direct grants to pay students to develop software
- Direct grants for security audits
- National grants for components of humanities solution (NEH – IHCADS)
- National grants for cybersecurity education

IHCADS

Individualized Humanities Collection and Dissemination System (IHCADS) is an open source cloud-based software solution designed to enhance public access to historical, archeological and artistic data and the experts who can help interpret the data.

Enterprise Software Application Audits

Security audits of industrial software solutions

- Reverse Classroom
- Sponsored Research Assistant

Outline of the Presentation

- Secure Data Engineering Lab @ College of Charleston
- Secure Software Development
- Overview of High Traffic Cloud/SOA vs. 2-Tier Architectures
- Lost Updates Example
- Related Research
- Research
  - Buddy System
  - Capacity Constraints
  - Coarse Grained Services
  - Service Constraints
  - Business Filters
  - Integrating Heterogeneous Systems
  - Autonomous Process Authentication
  - Modeling Vulnerable Application Partitions
- Questions

Secure Software Development (SSD)

- Once data is created it needs to be protected
- Once an algorithm is created it needs to be protected

Secure Software Development (SSD)

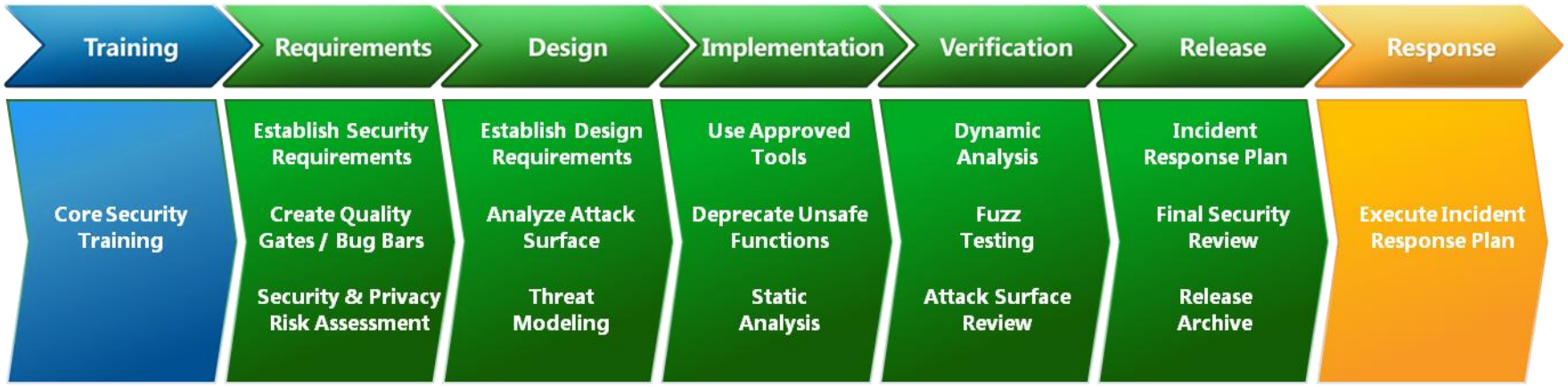
Results of vulnerabilities in SDLC

- Incorrect Data
- Missing Data
- Unavailability to Systems
- Wasted Resources
- Wasted Money
- Wasted Time

Secure Software Development (SSD)

- Vulnerabilities in Software Applications stem from the lack of
  - Proper Secure Software Development Lifecycle (SSDLC)
  - Proper programming language support for SSD
  - Proper operating system support for SSD
  - Proper training for developers in SSD

Microsoft Security Development Lifecycle



Language/OS Support Weaknesses

- Certificates (C,I)
- Authentication (C,I)
  - Process Authentication (C,I)
    - Keystore Security (C,I)
    - Datastore Security (C,I)
    - Codestore Security (I)
- ACID/CAP
  - Durability (I)
  - Constraints (I)
  - Load Balancing (A)
  - Redundancy (A,I)
- System Integration (A,I)
- Design Models for Security (C,I,A)

Certificates (C,I)

Private Key Infrastructure (PKI)

Accomplished

- Machine to machine synchronous key exchange
- Validate the integrity of messages from machines

Outstanding Challenge

- Process identification

Authentication (C,I)

Accomplished

- Something you know for humans
- Something you have for humans
- Something about you for humans
- Someplace you are for machines

Missing

- Process Authentication
  - We do have Security Assertion Markup Language (SAML) for some use cases

Keystore Security (C,I)

- Accomplished
  - Keys/certificates can be secured in OS with Android/IOS/Java
  - Keys/certificates can be secured by user in Windows
- Issues
  - Keys/certificates cannot be secured to specific processes
  - Stores are often held in insecure areas of the OS (file system or registry)

Datastore Security (C,I)

Databases have successfully protected data based on the agent accessing the data but have failed for autonomous processes

- Accomplished
  - Data can be protected by user/password/location
- Missing
  - Processes often need data from datastore
    - Anonymous web sites

Codestore Security (I)

Many hacks have involved code manipulation through an applications machine code mutation, library signature matching and unauthorized library invocation

- Accomplished
  - Code can be signed from app stores
  - Device driver signing
  - NIST forensic diskprints
  - Microsoft file checksum integrity verifier

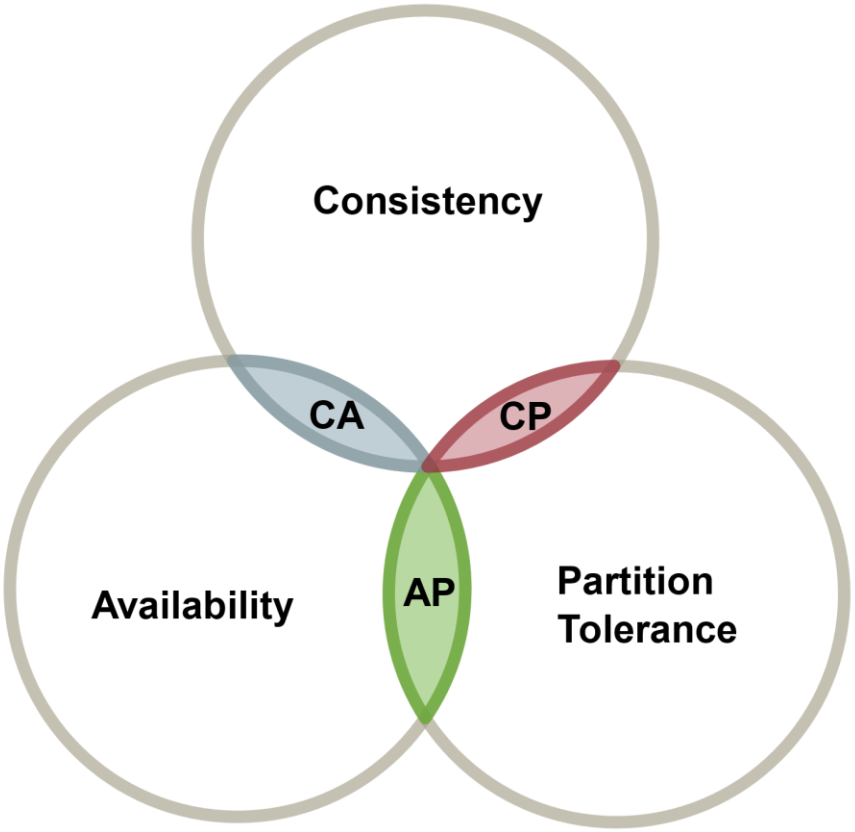
Codestore Security (I)

- Missing
  - Runtime checksum validation of executables
  - Library checksum validation
  - Protection of hashcode/checksum datastore

ACID vs CAP

- ACID Strong Properties
  - Atomic
  - Consistent
  - Isolated
  - Durable

ACID vs CAP (Page 2)



Durability (I)

- Durability guarantees that we do not lose data after a transaction.
  - Server partitioning requires we update many machines synchronously to avoid lose.
  - Offline stores need to resolve conflicts based on many related factors

Constraints (I)

- Constraints guarantee consistency.
- All constraints should hold before a transaction starts
- All constraints should hold after a transaction completes

Load Balancing (A)

- Applications need to be agnostic to the server they are running on

Accomplished

- Session Management

Work Needed

- Limited Resource Consumption (i.e. Locks)

System Integration (A,I)

Accomplished

- Integrating Homogenous Systems

Needs Work

- Integrating Heterogeneous Systems
  - Same logical data in different physical models
  - Related data
  - Offline

Modeling for Security (C,I,A)

- We have many UML diagrams for modeling a software application but none model the potential vulnerabilities.
- Vulnerabilities may include C, I or A

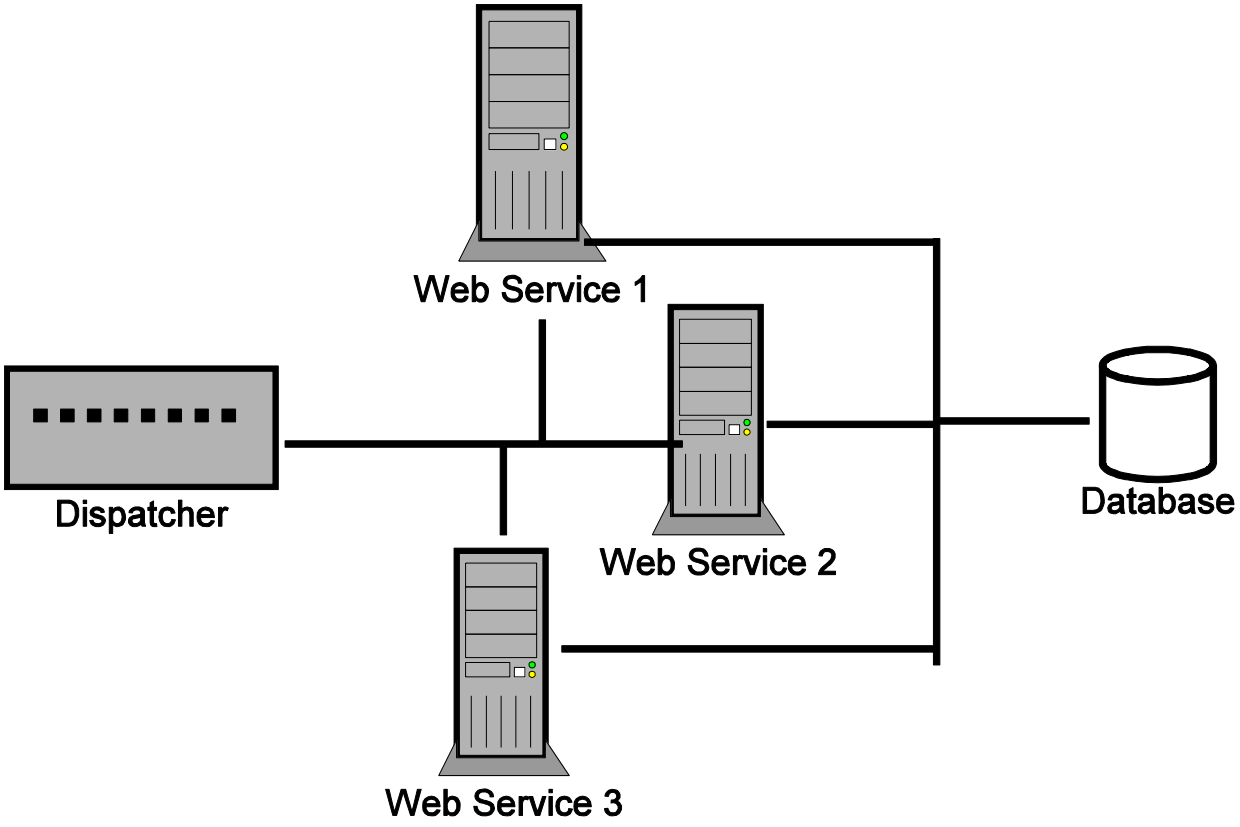
Outline of the Presentation

- Secure Data Engineering Lab @ College of Charleston
- Secure Software Development
- Overview of High Traffic Cloud/SOA vs. 2-Tier Architectures
- Lost Updates Example
- Related Research
- Research
  - Buddy System
  - Capacity Constraints
  - Coarse Grained Services
  - Service Constraints
  - Business Filters
  - Integrating Heterogeneous Systems
  - Autonomous Process Authentication
  - Modeling Vulnerable Application Partitions
- Questions

Cloud/SOA – Service Oriented Architecture

- Composition of interoperable services
- Often called from outside the domain (less control of concurrency demands)
- Web Service Farm provides redundancy in service offerings
- 2-Tier gave us ACID Guarantees but the guarantees are often lost in architecture transition

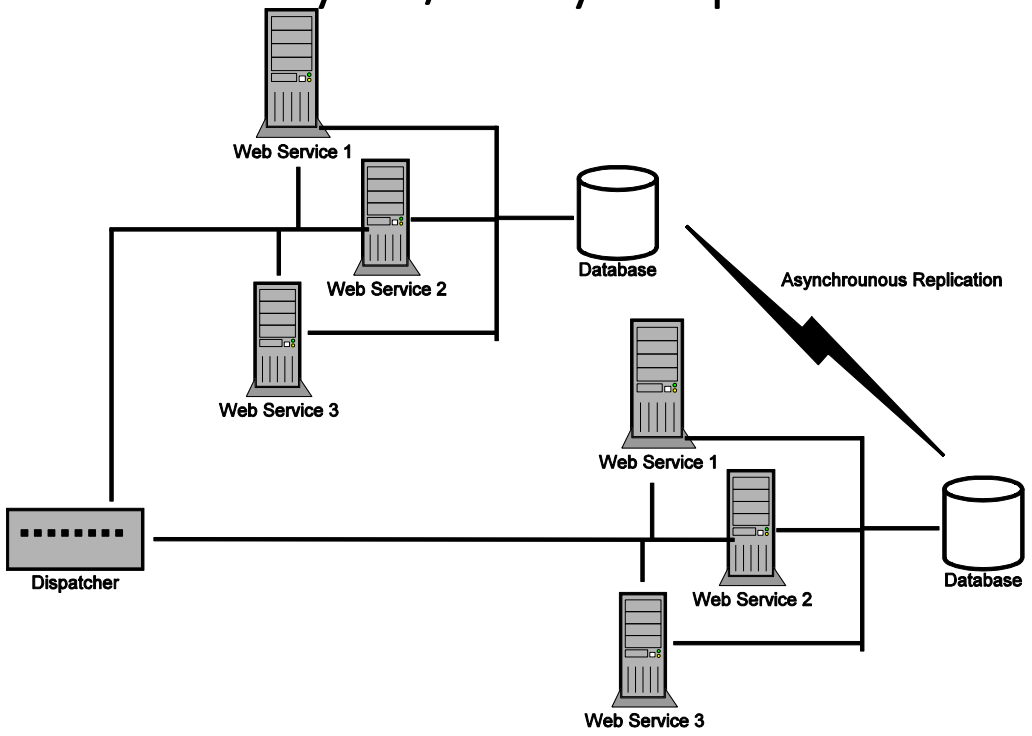
Simple Web Service Farm



Increasing Availability Through Replication

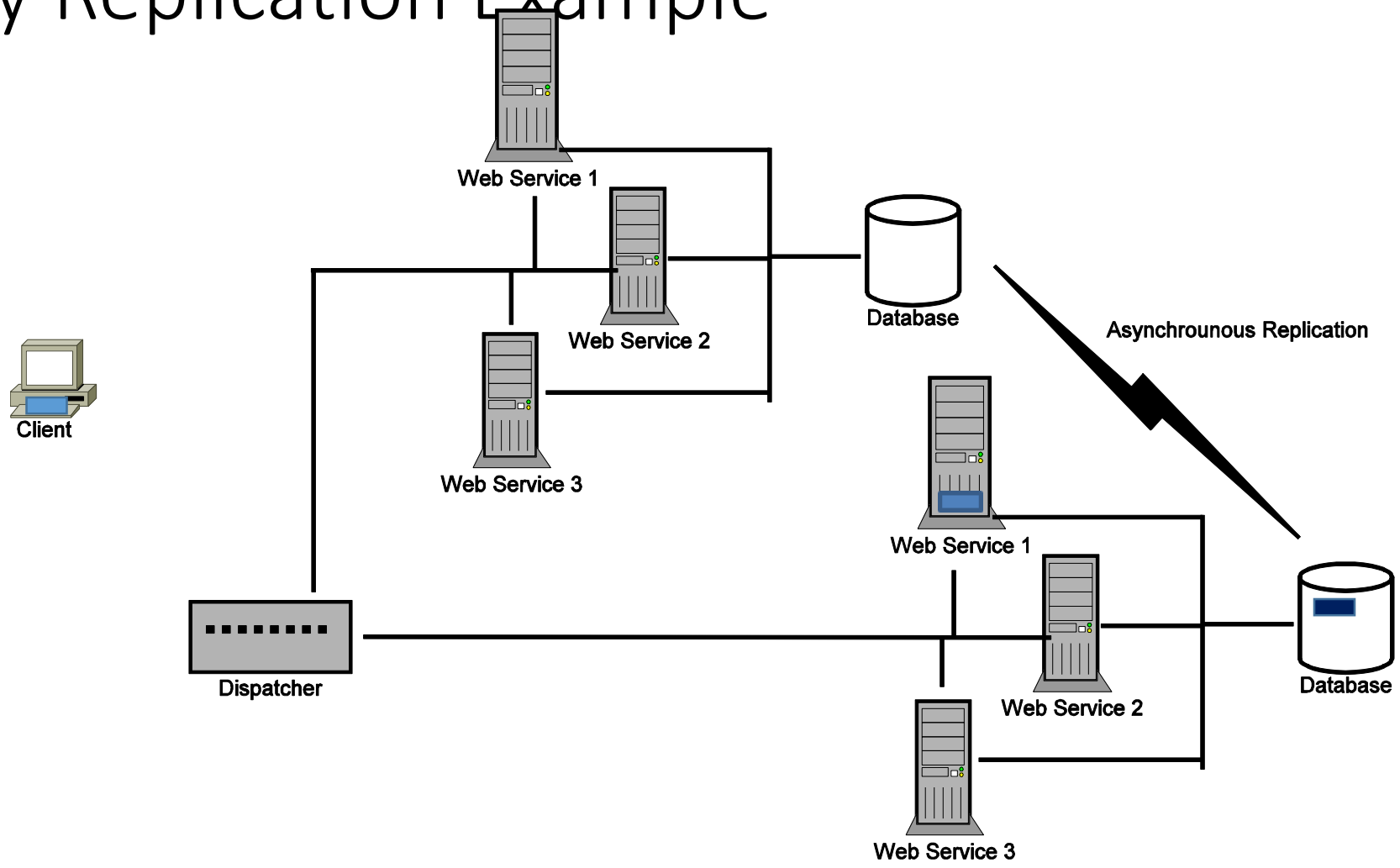
- Replication Types
  - Strict – A transaction updates at all cluster or none
  - Lazy – One cluster is the master for a data item and it will asynchronously update other clusters after a transaction

Increasing Availability w/Lazy Replication



- Pros: Higher Availability
- Cons: Lower Consistency, Lower Durability, Lower Atomic

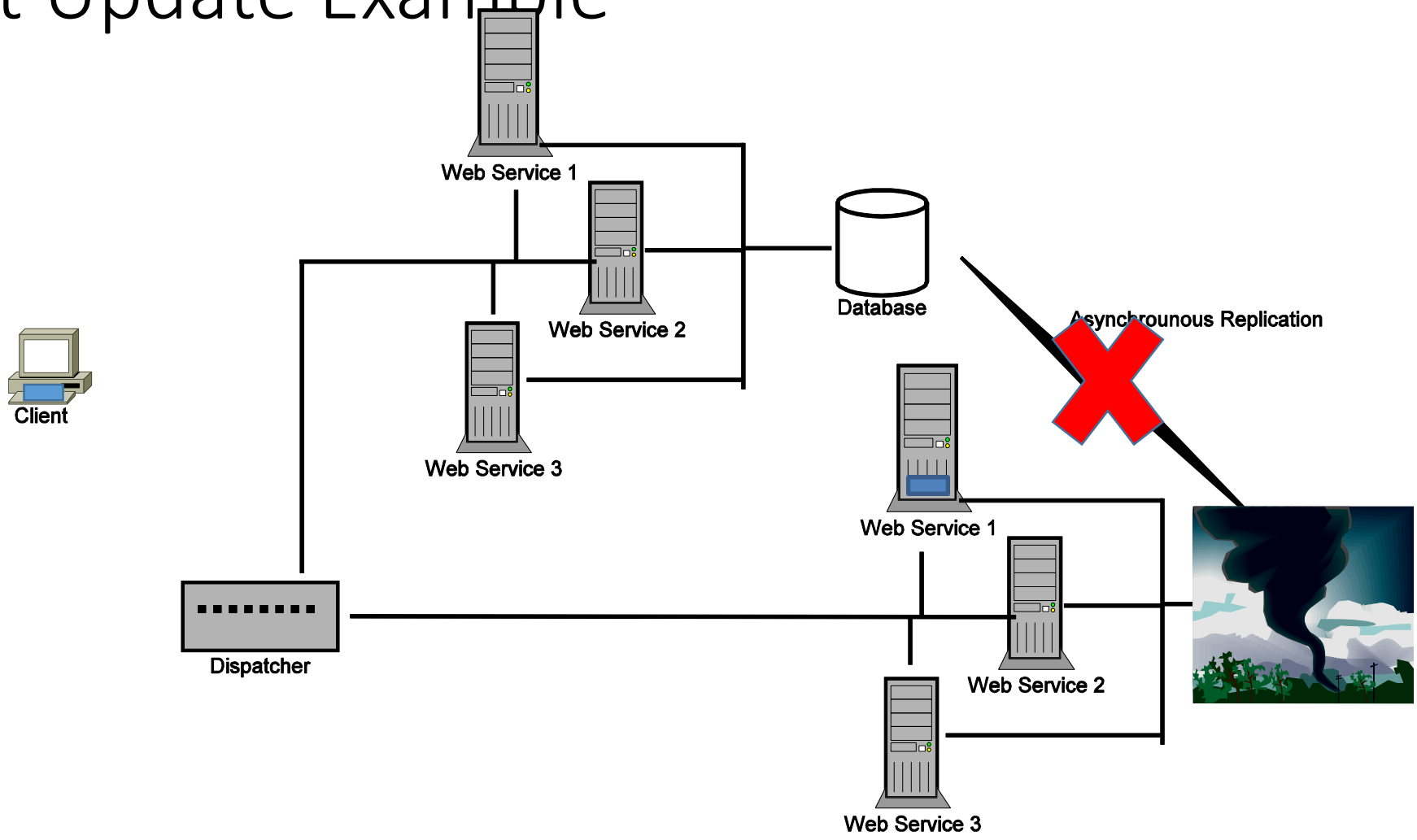
Lazy Replication Example



Outline of the Presentation

- Secure Data Engineering Lab @ College of Charleston
- Secure Software Development
- Overview of High Traffic Cloud/SOA vs. 2-Tier Architectures
- Lost Updates Example
- Related Research
- Research
  - Buddy System
  - Capacity Constraints
  - Coarse Grained Services
  - Service Constraints
  - Business Filters
  - Integrating Heterogeneous Systems
  - Autonomous Process Authentication
  - Modeling Vulnerable Application Partitions
- Questions

Lost Update Example



Outline of the Presentation

- Secure Data Engineering Lab @ College of Charleston
- Secure Software Development
- Overview of High Traffic Cloud/SOA vs. 2-Tier Architectures
- Lost Updates Example
- Related Research
- Research
  - Buddy System
  - Capacity Constraints
  - Coarse Grained Services
  - Service Constraints
  - Business Filters
  - Long Running Transactions
  - Integrating Heterogeneous Systems
  - Autonomous Process Authentication
  - Modeling Vulnerable Application Partitions
- Questions

Related Work

- Improving Availability of Strict Replication
  - Snap Isolation Replication (Fekete, et al.)
- Improving Consistency of Lazy Replication
  - Frameworks with Lazy Consistency Guarantees (Breitbart & Korth)
- Hybrid Systems
  - Hybrid Majority Systems (Jajodia & Mutchler)
  - NoSQL Systems; Casandra, BigTable, etc.

Goals

- Develop Algorithms and Architecture that will
  - Guarantee Transaction Consistency for Distributed Transactions
  - Provide Availability of Lazy Replication
  - Provide Durability of Strict Replication

Outline of the Presentation

- Secure Data Engineering Lab @ College of Charleston
- Secure Software Development
- Overview of High Traffic Cloud/SOA vs. 2-Tier Architectures
- Lost Updates Example
- Related Research
- Research
  - Buddy System
  - Capacity Constraints
  - Coarse Grained Services
  - Service Constraints
  - Business Filters
  - Long Running Transactions
  - Integrating Heterogeneous Systems
  - Autonomous Process Authentication
  - Modeling Vulnerable Application Partitions
- Questions

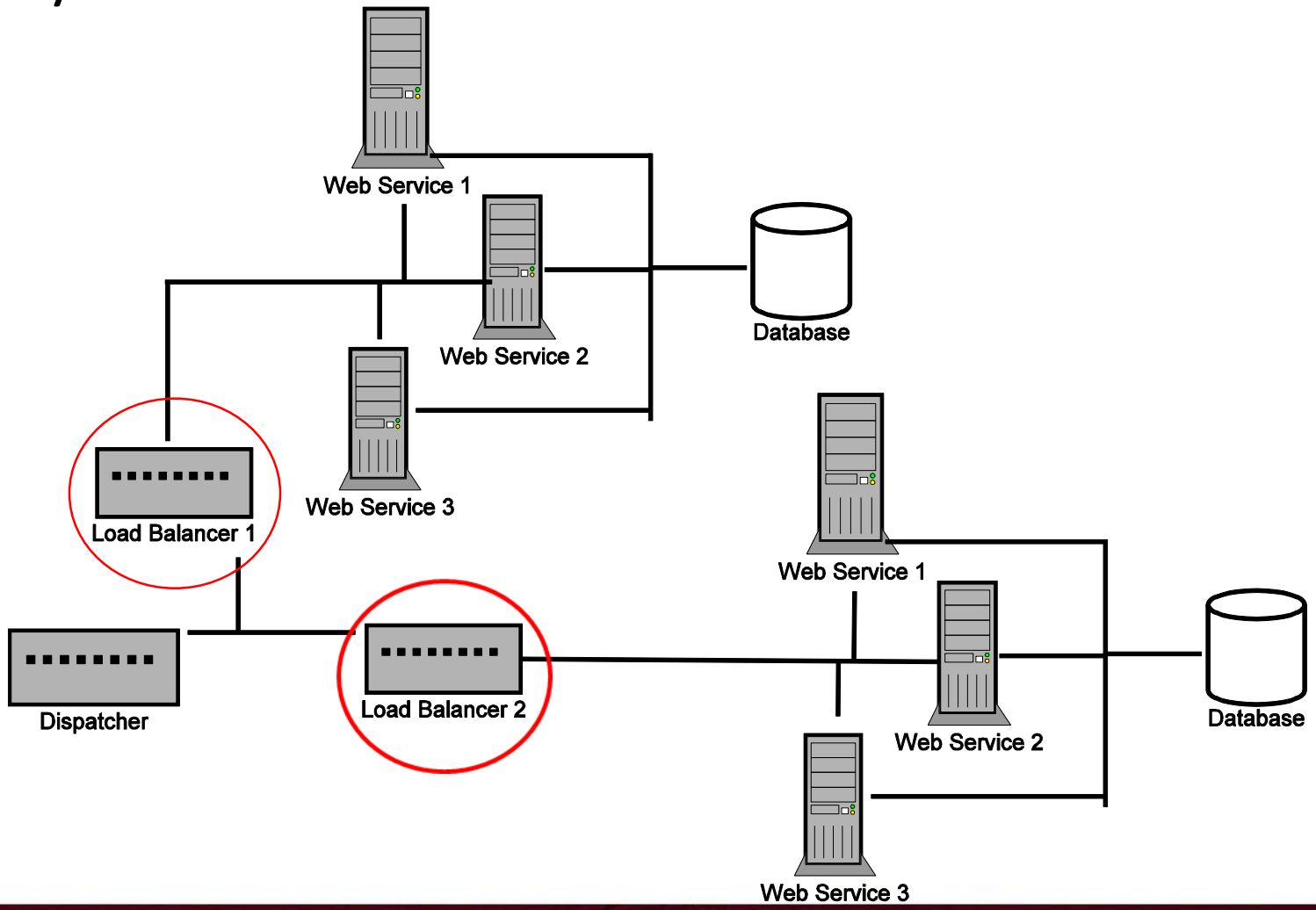
Our Solution: Buddy System

- Ensure at least two replicas are updated synchronously
- Maintain data element version so master can be any updated cluster
- Guarantees one-copy serializability

The Cost of Increased Transactional Correctness and Durability in Distributed Databases, Information Reuse and Integration (IRI), 2012 IEEE International Conference on, Publication Year: 2012 , Page(s): 441 – 448

Buddy System: Available, Consistent, Durable Web Service Transactions, Journal of Internet Technology and Secured Transactions (JITST), 3 (1/2/3/4). ISSN 2046-3723

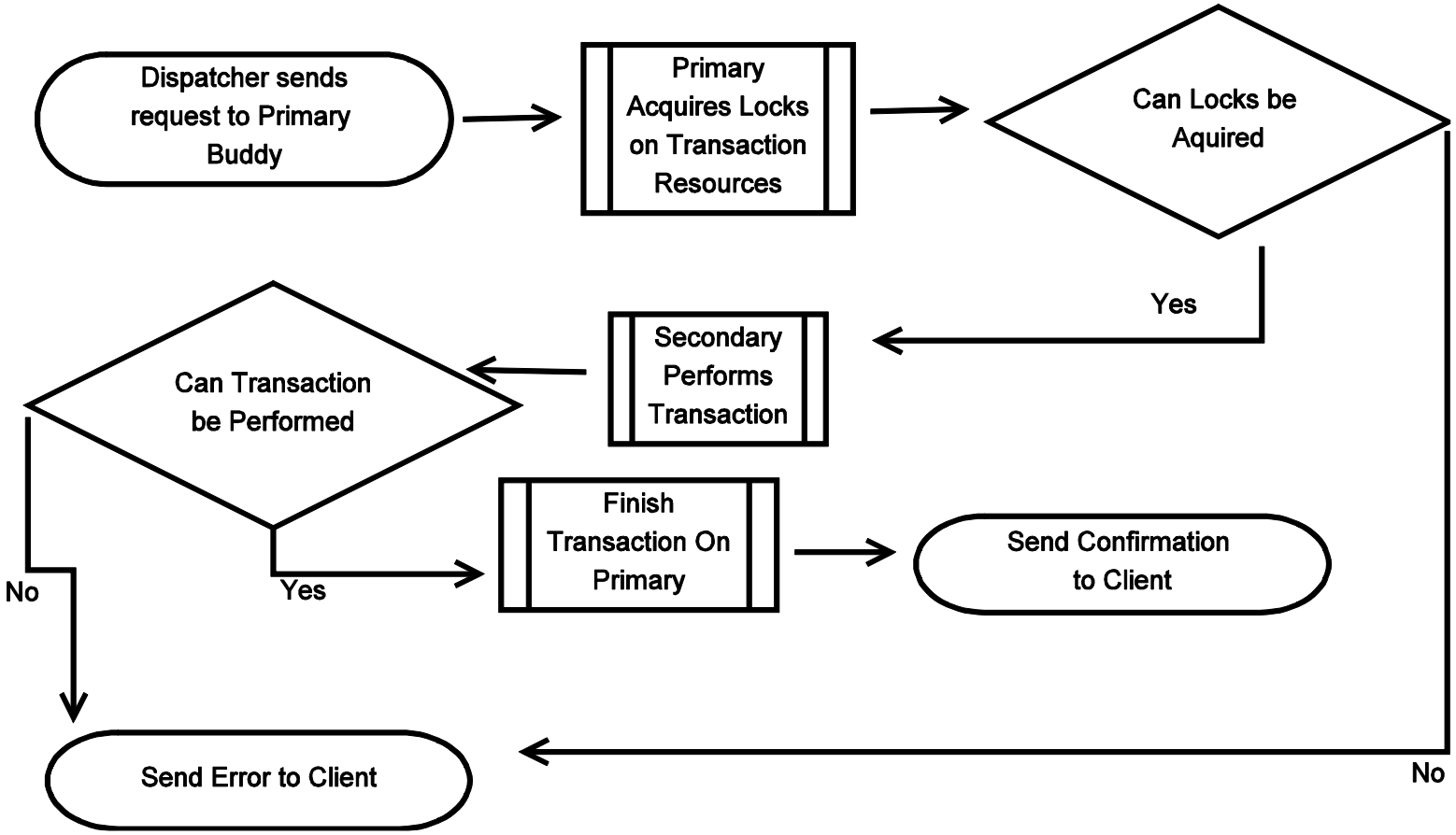
Buddy system architecture



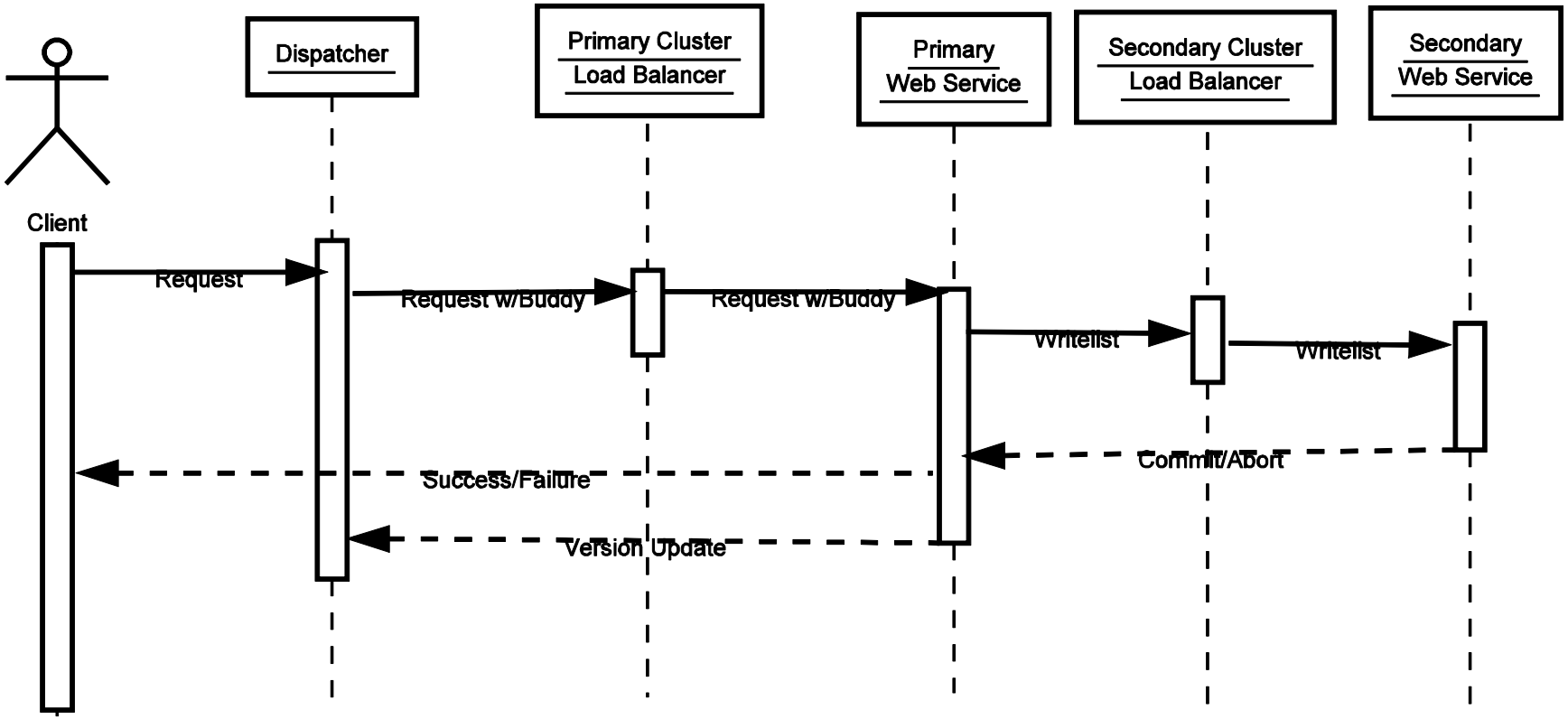
Dispatcher Intelligence

- Peek into Packet
- Maintains data structures to pick pairs of buddies who can service a request
- Supplied algorithms randomly choose buddies
- Could decorate cluster data to choose buddies based on other system and network attributes

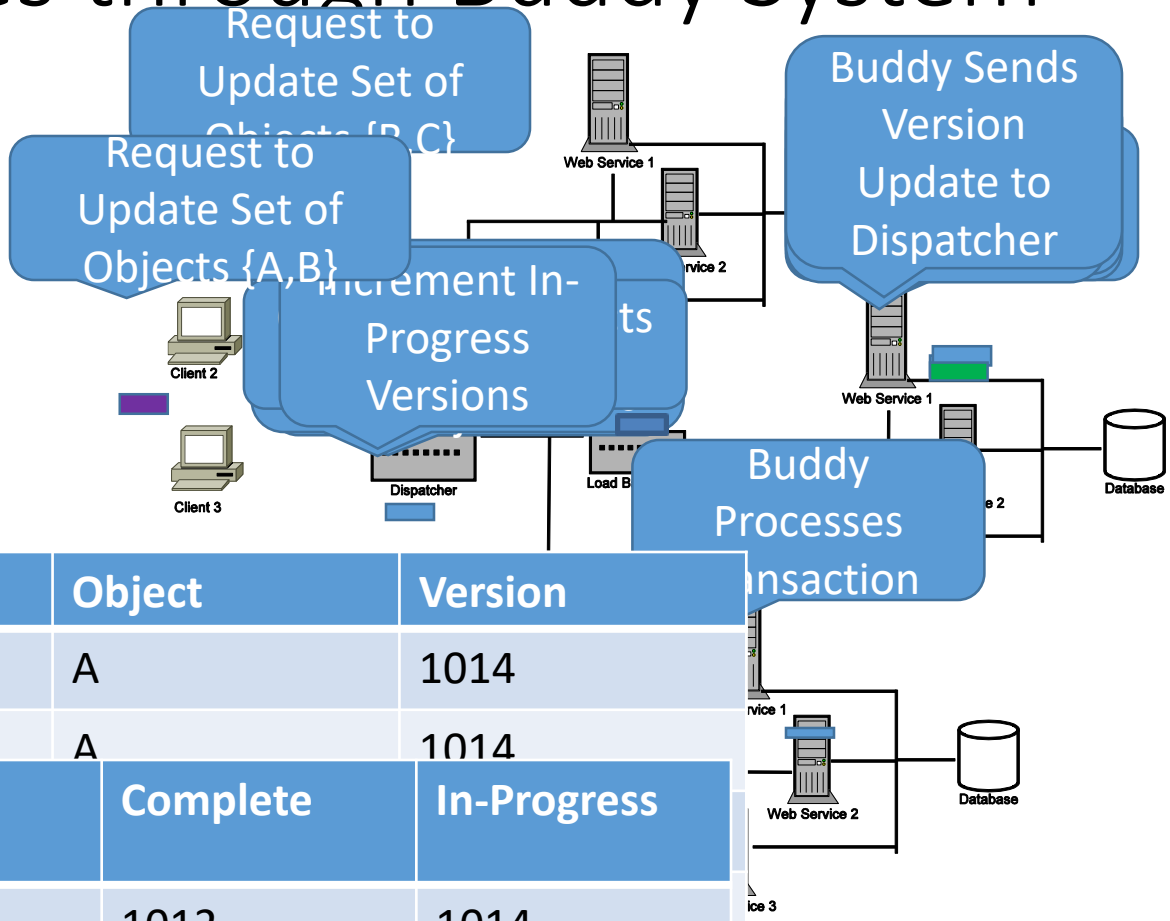
Flow under Buddy System



Buddy Sequence



Updates through Buddy System



Cluster	Object	Version
1	A	1014
2	A	1014
3		

	Object	Complete	In-Progress
1	A	1012	1014
2	B	955	955
3	C	2055	2055

Table 2 – Windows of Vulnerability

	InActive	During Transaction	After Transaction Committed
Web Server housing Web Service of primary buddy	No issue because of redundancy	2PC will roll back trans, Client will get error from primary	No issue because of redundancy
Web Server housing Web Service of secondary buddy	No issue because of redundancy	2PC will roll back trans, Client will get error from primary	No issue because of redundancy
Database Server	No issue. Dispatcher needs to be notified by web service that it is unavailable	2PC will roll back trans, Client will get error from primary	No issue because of redundancy
Dispatcher	New dispatcher, Each clusters will update dispatcher versions	After dispatch to primary no issue because primary responds directly to client	New dispatcher, Each clusters will update dispatcher versions

Results - Lost Updates

- Buddy System - Guaranteed Durability
- Strict Replication – Guaranteed Durability
- Lazy Replication – Time for replication

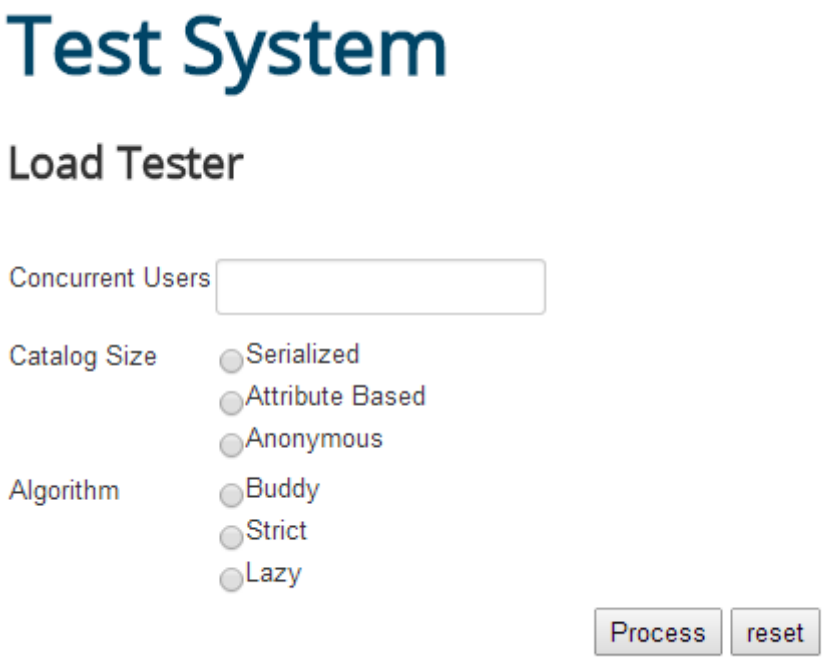
Concurrency Improvement by Item Type

- Anonymous Item Consumption - No improvement
- Attribute Item Consumption – Improvement linear to min(# of attributes,# of clusters)
- Serialized Item Consumption – Improvement linear to min(# of items,# of clusters)

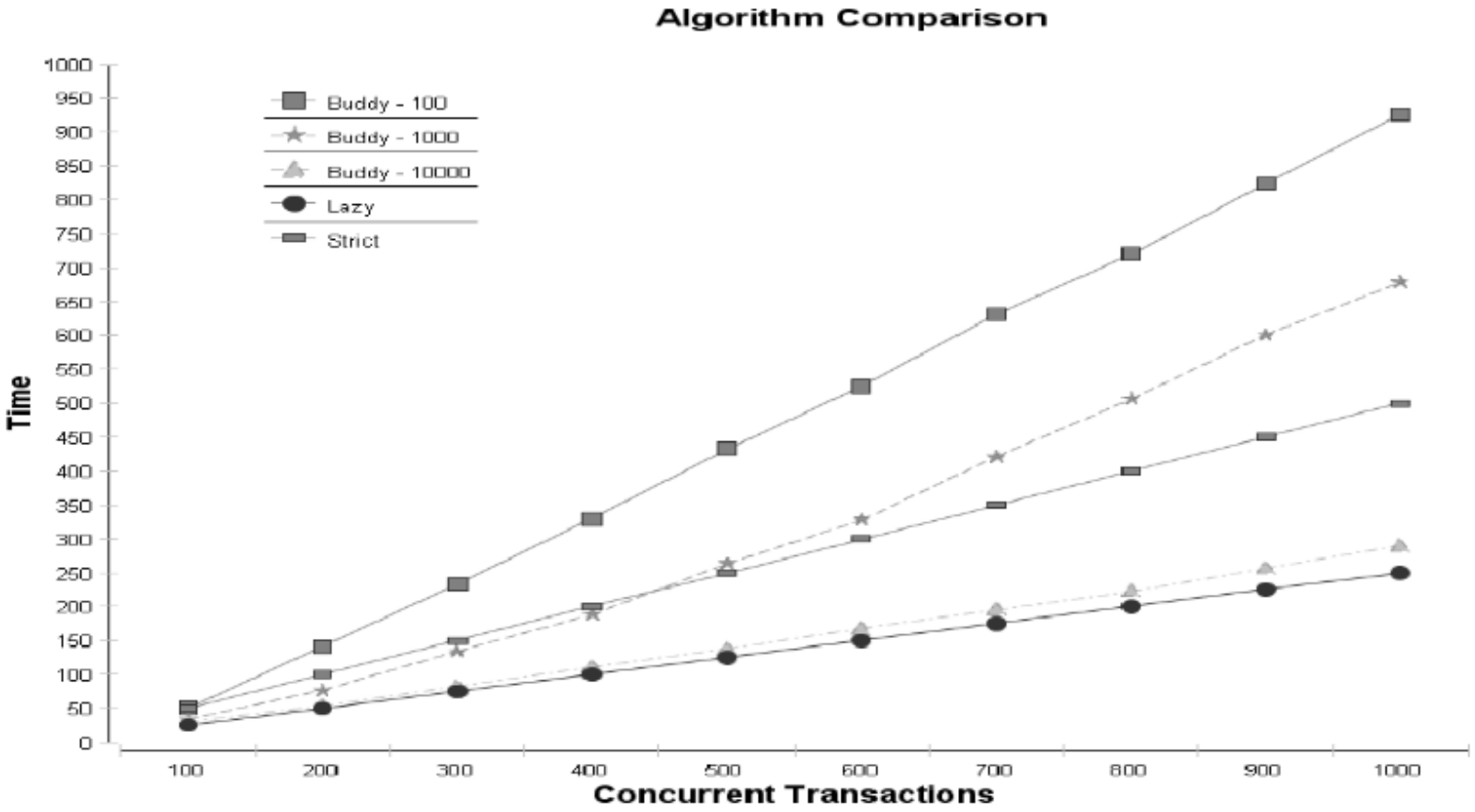
Implementation

- Java
- Synchronous Requests (Http)
- Clusters (Java EE, Tomcat, MySQL)
- Dataset sizes (100, 1000, 10000)
- Concurrent transactions (100-1000)

Load Tester


Test System
Load Tester
Concurrent Users
Catalog Size
Serialized
Attribute Based
Anonymous
Algorithm
Buddy
Strict
Lazy
Process
reset

Performance Results



Outline of the Presentation

- Secure Data Engineering Lab @ College of Charleston
- Secure Software Development
- Overview of High Traffic Cloud/SOA vs. 2-Tier Architectures
- Lost Updates Example
- Related Research
- Research
  - Buddy System
  - Capacity Constraints
  - Coarse Grained Services
  - Service Constraints
  - Business Filters
  - Integrating Heterogeneous Systems
  - Long Running Transactions
  - Autonomous Process Authentication
  - Modeling Vulnerable Application Partitions
- Questions

Capacity Constraints

- Original Buddy algorithm produced availability equivalent to lazy replication with higher durability and higher consistency for serialized resource consumption
- Add Capacity Constraint to Dispatcher to allow writes of anonymous resources to be distributed
- A Capacity Constraint acts as a counting semaphore to ensure no more then available number get consumed

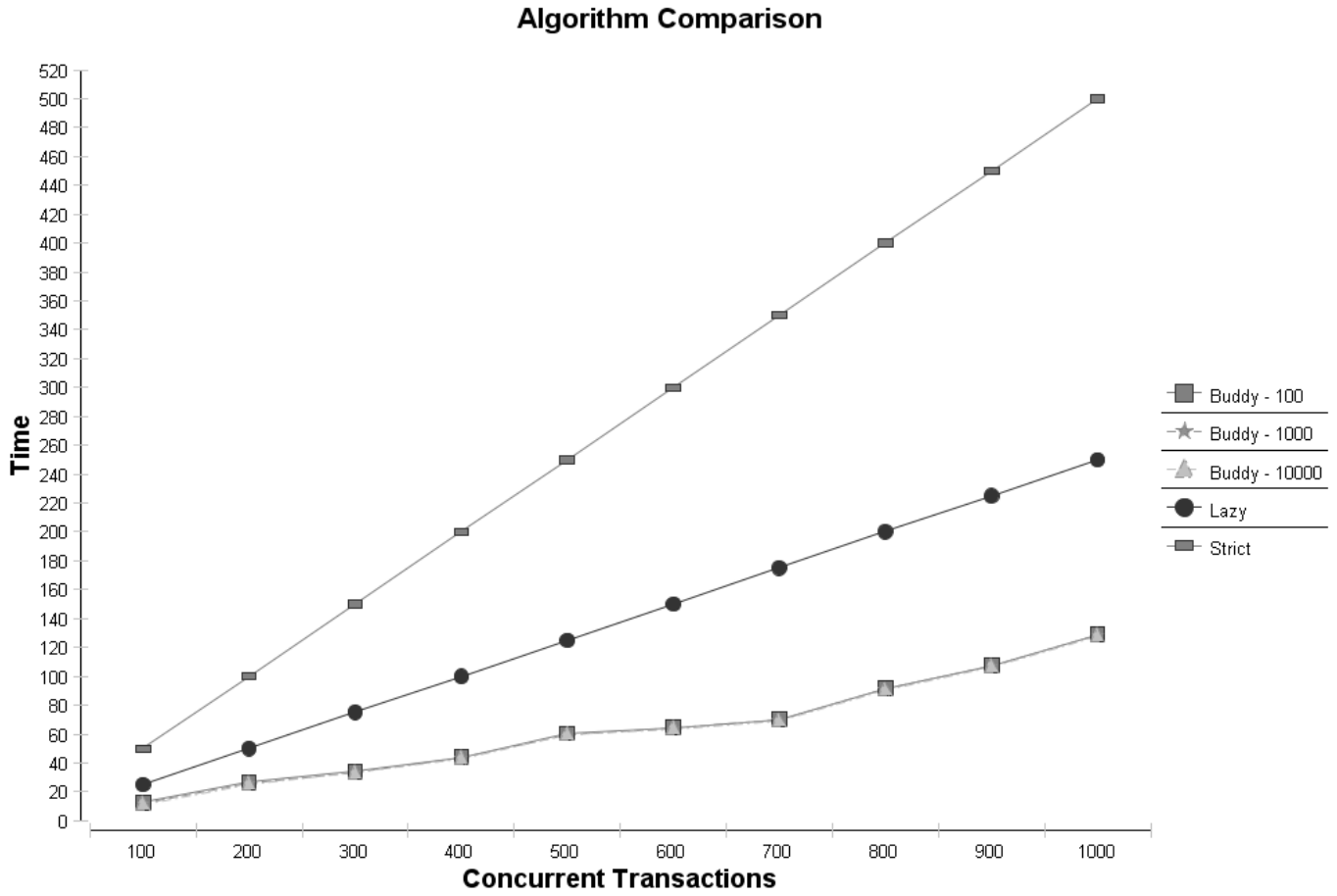
High Volume Web Service Resource Consumption, Internet Technology and Secured Transactions, 2012. ICITST 2012. International Conference for, Publication Year: 2012

Buddy System: Available, Consistent, Durable Web Service Transactions, Journal of Internet Technology and Secured Transactions (JITST), 3 (1/2/3/4). ISSN 2046-3723

New Improvement By Item Type

- Anonymous Item Consumption - Improvement linear to min(# of items,# of clusters)
- Attribute Item Consumption – Improvement linear to min(# of items,# of clusters)
- Serialized Item Consumption – Improvement linear to min(# of items,# of clusters)

Performance Results



Outline of the Presentation

- Secure Data Engineering Lab @ College of Charleston
- Secure Software Development
- Overview of High Traffic SOA vs. 2-Tier Architectures
- Lost Updates Example
- Related Research
- Research
  - Buddy System
  - Capacity Constraints
  - Coarse Grained Services
  - Service Constraints
  - Business Filters
  - Long Running Transactions
  - Integrating Heterogeneous Systems
  - Autonomous Process Authentication
  - Modeling Vulnerable Application Partitions
- Questions

Coarse Grained Web Services

- Previous web services have been fine grained CRUD services
- Coarse Grained Services are a black box
- Input and Output data from SOAP WSDL

Coarse-Grained Web Service Availability, Consistency, & Durability, Web Services (ICWS), 2013 IEEE International Conference on, Publication Year: 2013

Buddy System: Available, Consistent, Durable Web Service Transactions, Journal of Internet Technology and Secured Transactions (JITST), 3 (1/2/3/4). ISSN 2046-3723

Semantics from Model

- To schedule the coarse grained services we need semantics
  - Which services can be concurrent
  - Which services change data
  - What data is read by a service
  - What data is changed by a service

UML

- UML (Unified Markup Language) – standardized general purpose modeling language
- XMI (XML Metadata Interchange) – XML format for storing UML models

UML Extensibility

- Semantics can be added to UML through stereotypes
  - Classes can be stereotypes
  - Attributes can be stereotyped
- Profiles contain sets of stereotypes

Example Transaction

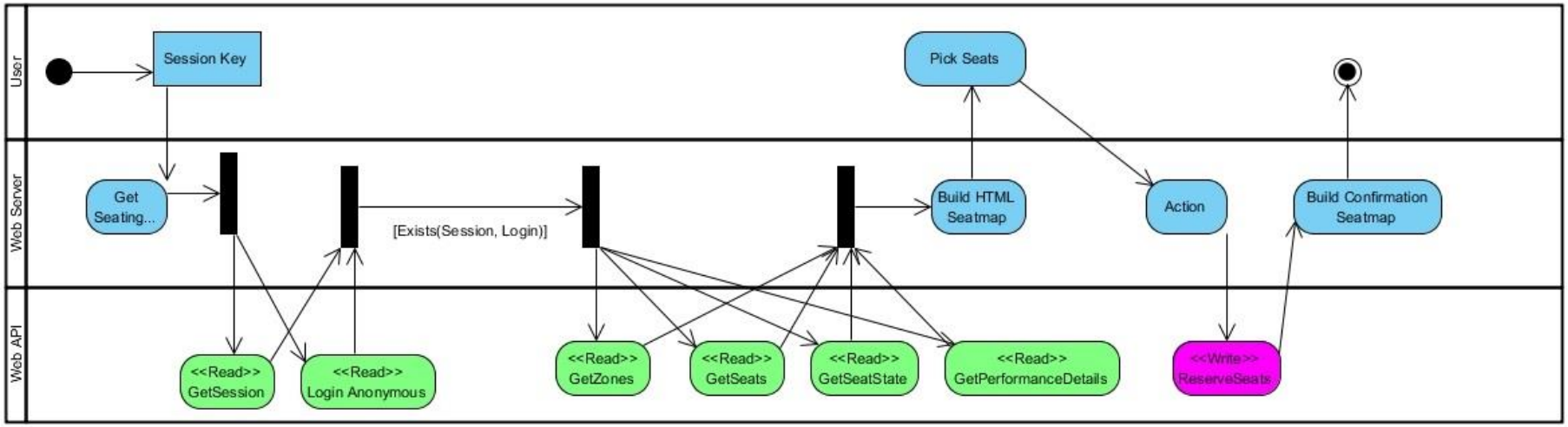
- Consider a Ticket Reservation System (TRS).
- TRS uses web services to provide a variety of functionalities to the clients.
- For example, clients may want to select a specific seat for a popular concert in the ticket reservation

Example Web-Service

- Reserve Seats
  - Input : event identifier and a collection of seats
  - Output: collection of seats with current statuses

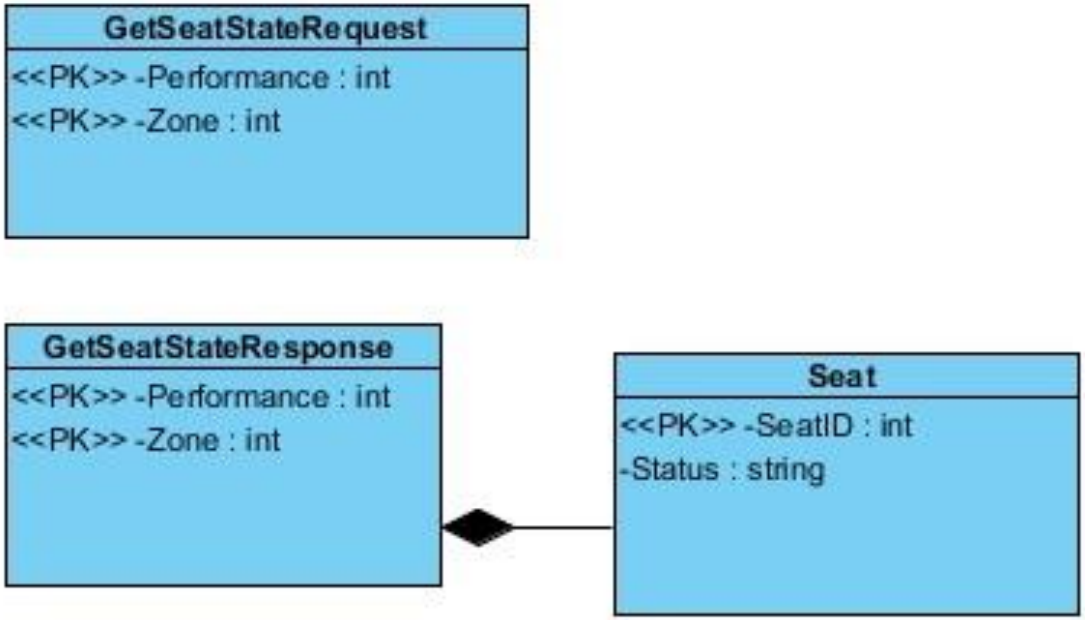
  - Data changed: seat records
  - Data read: seat records

Stereotypes in Activity Diagram



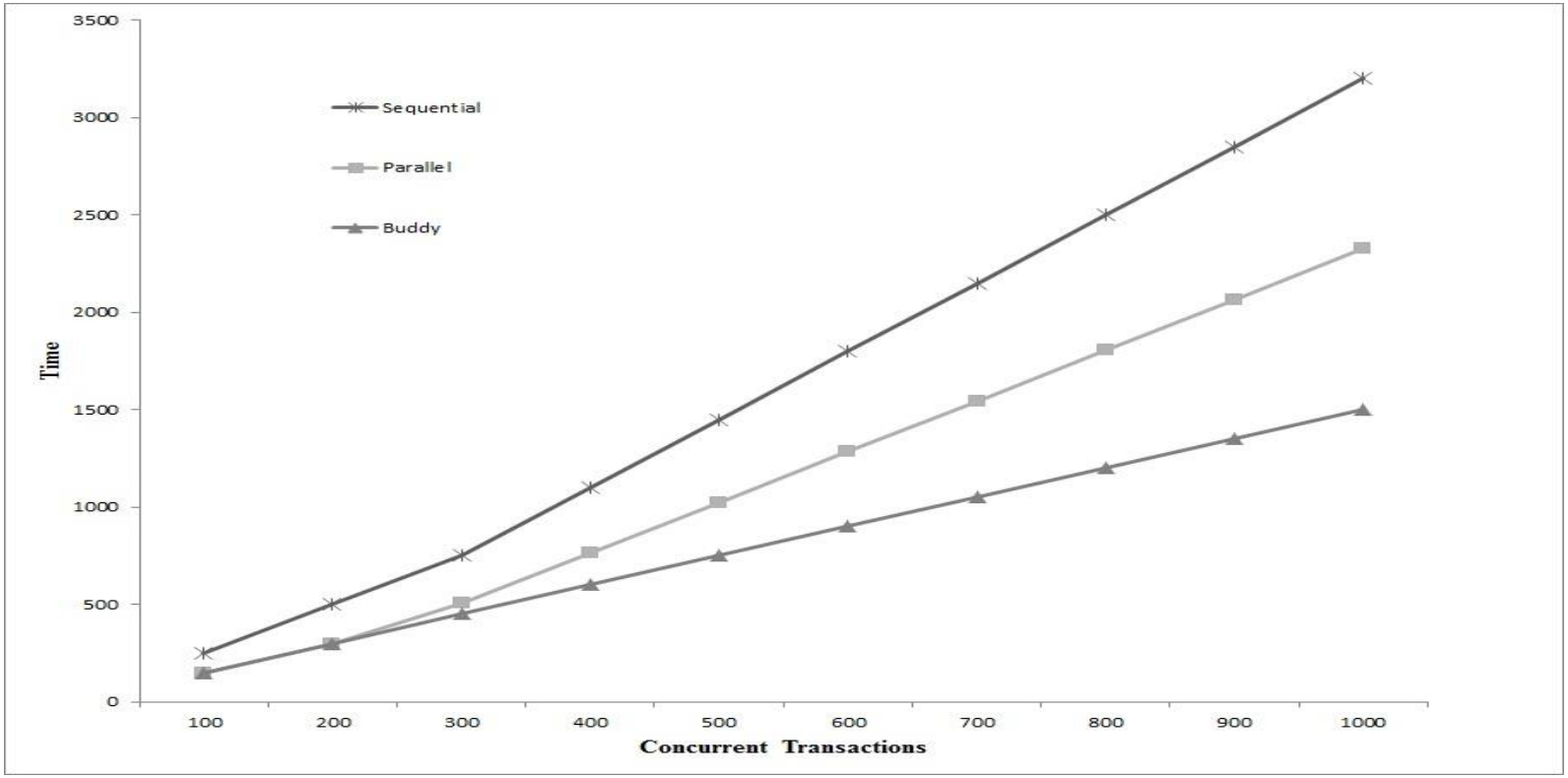
- Web Services are marked as getters or setters
- Stereotypes are packaged into profile for development tooling

Sample Class Diagrams



- Each web services in activity diagram has matching class diagram
- Attribute stereotype identifies unique identifier set

Performance Results



Outline of the Presentation

- Secure Data Engineering Lab @ College of Charleston
- Secure Software Development
- Overview of High Traffic SOA vs. 2-Tier Architectures
- Lost Updates Example
- Related Research
- Research
  - Buddy System
  - Capacity Constraints
  - Coarse Grained Services
  - Service Constraints
  - Business Filters
  - Long Running Transactions
  - Integrating Heterogeneous Systems
  - Autonomous Process Authentication
  - Modeling Vulnerable Application Partitions
- Questions

Service Constraints

- Integrity Constraints Guarantee Consistency
- Unfortunately they lower availability by limiting distribution
- We model Constraints in OCL and categorize to find those that can run in parallel (distributed)

Web Service Constraint Optimization, *Internet Technology and Secured Transactions, 2013. ICITST 2013. International Conference for, Publication Year: 2013*

Service Constraint Guarantees, International Journal of Intelligent Computing Research, Volume 5, Issues 1/2, Mar/Jun 2014, ISSN: 2042-4655

Integrity Constraint Types

- Entity
  - Codd's Entity (Seq)
- Attribute Domain
  - Codd's Domain (Par), Column(Par), Referential Integrity (Seq)
- Hierarchical
  - Codd's User Defined (Seq)

Hierarchical Constraints

- Involve more then one tuple
- Two types
  - Aggregate – Expensive calculation (min, max, sum, avg)
  - Iterative – (universal or existential)

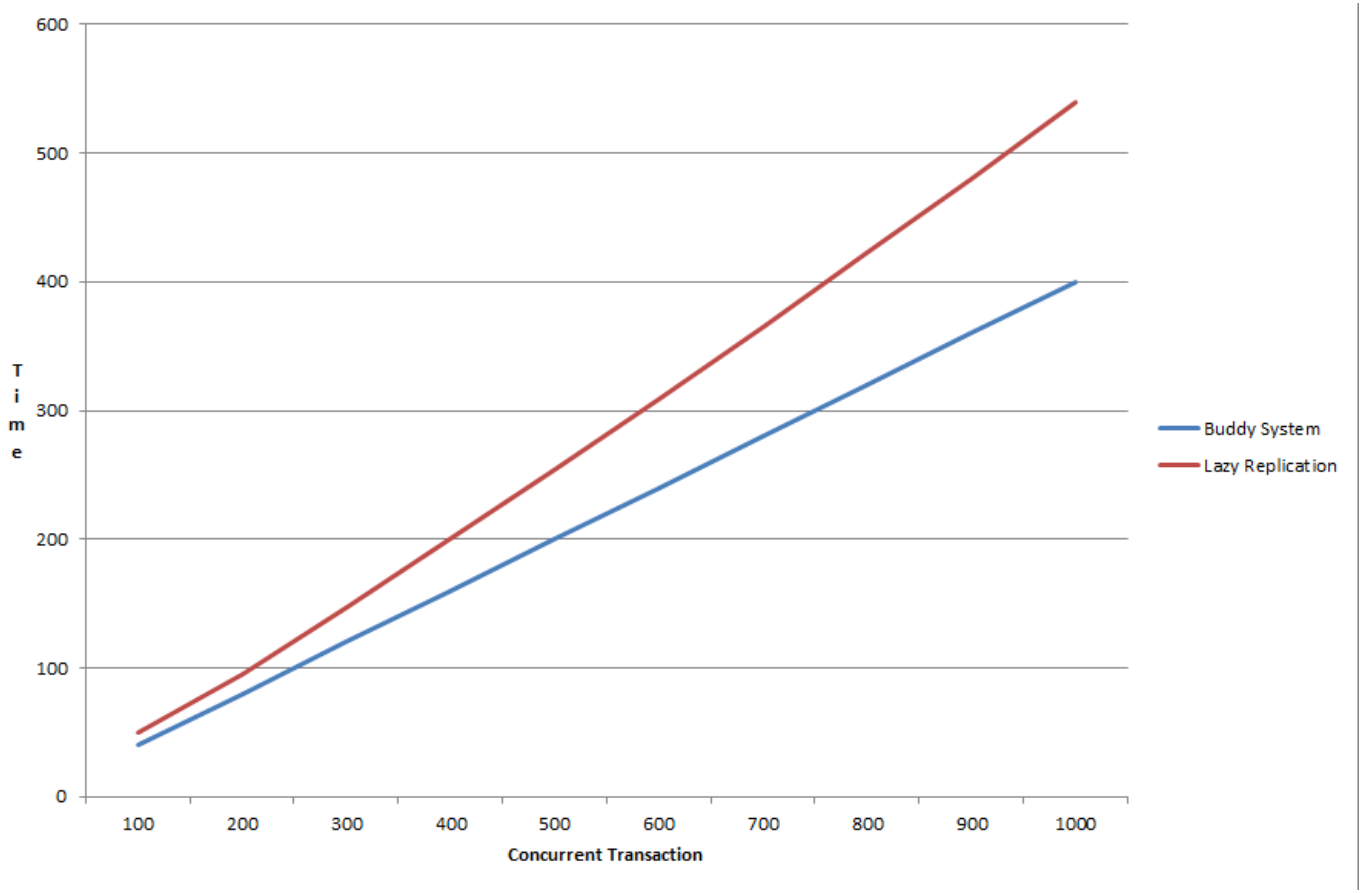
Hierarchical Constraint Materialization

- Maintain data to update aggregation on insert, update or delete

```
1 context SmarTrip::addActivity(inSequence: Integer) pre:
2     self.activities->forAll( a|a.sequence < inSequence )
```

Object	*Constraint*	*Parent*	*Value*	*Quantity*
smarTrip	sequenceOrd	1000120	408	408

Performance Results



Outline of the Presentation

- Secure Data Engineering Lab @ College of Charleston
- Secure Software Development
- Overview of High Traffic SOA vs. 2-Tier Architectures
- Lost Updates Example
- Related Research
- Research
  - Buddy System
  - Capacity Constraints
  - Coarse Grained Services
  - Service Constraints
  - Business Filters
  - Long Running Transactions
  - Integrating Heterogeneous Systems
  - Autonomous Process Authentication
  - Modeling Vulnerable Application Partitions
- Ongoing Research

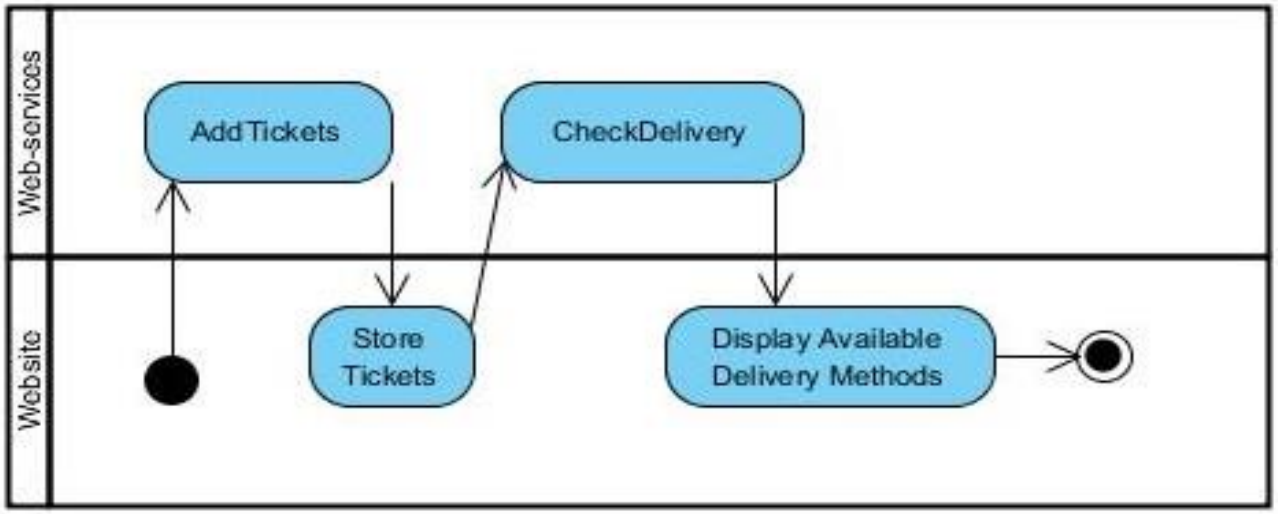
Business Filters

- 90% of enterprise users change business rules annually
- SOA applications use Web services input/output to process business rule
- Unfortunately the business rule will often lower availability by increasing the load and the latency
- Unfortunately the business rule will often lower consistency by running the rule in a separate transaction

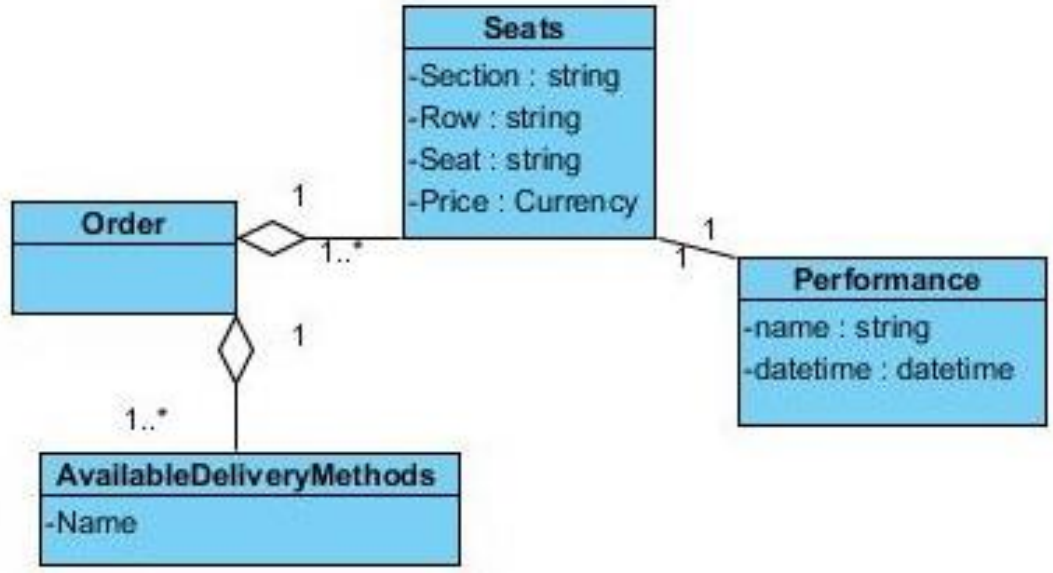
Highly Available, Consistent, Business Rule Filters, Internet Technology and Secured Transactions, 2014. ICITST 2014. International Conference for, Publication Year: 2014

A Customizable and Secure Software Architecture, *International Journal for Information Security Research* (IJISR), Volume 4, Issues 1/2, ISSN: 2042-4639, Accepted

Example Transaction

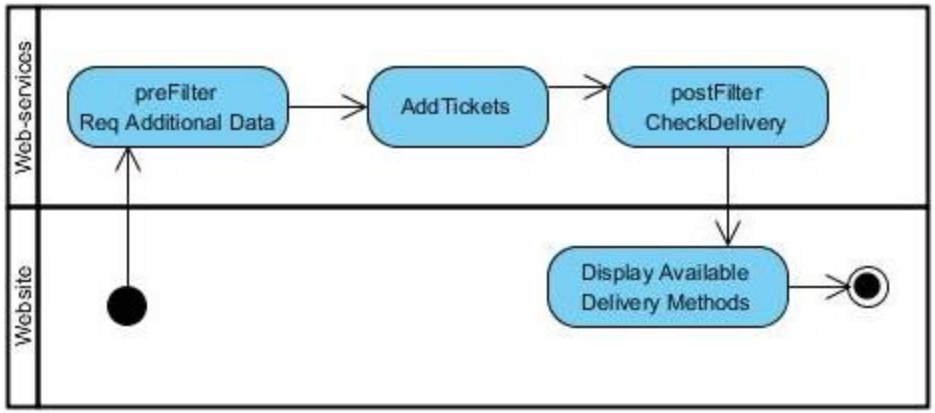


Original Response



Server-Side Pipe and Filter Architecture

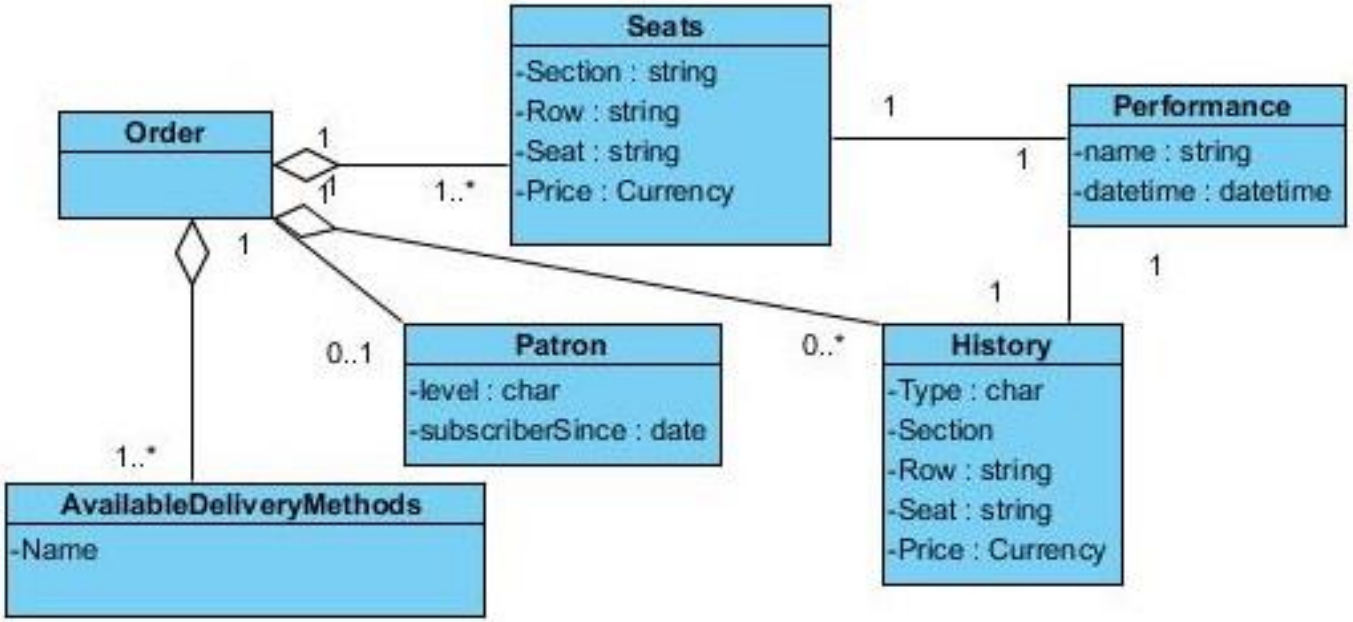
A Server-Side Pipe and Filter Architecture supports integration hooks
- Java EE supports this architecture with Servlet Filters
- Unfortunately filters request data in separate transactions from the main service



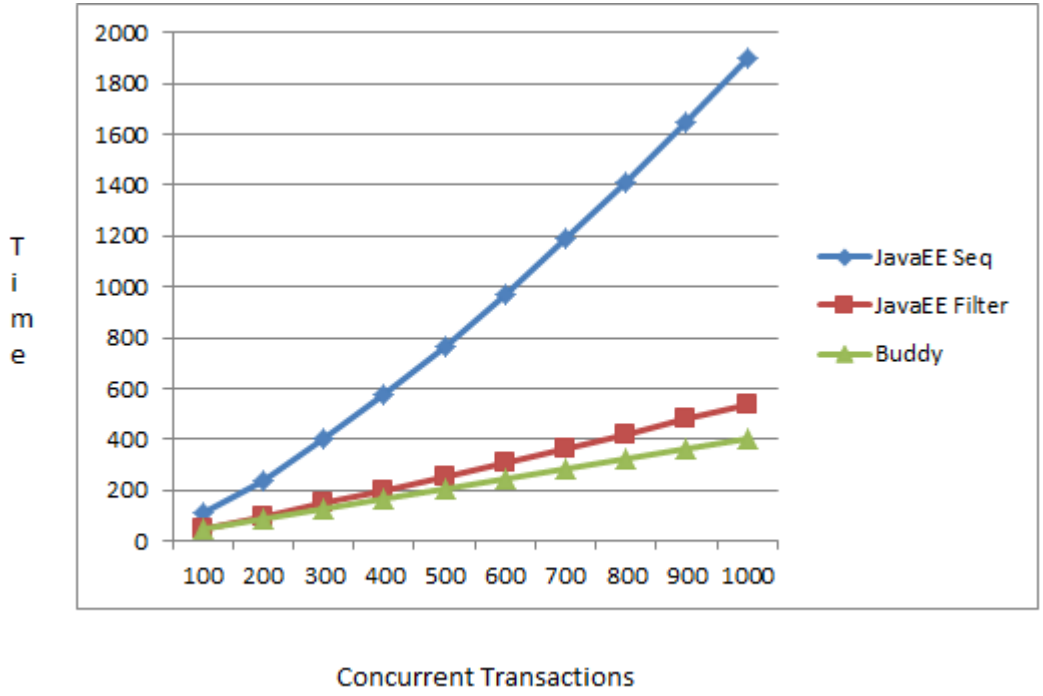
Buddy System Pipe & Filter

Our implementation allows UML additions from each filter. This results in a single UML to CRUD mapping

- Provides a single transaction for all filters and original service



Empirical Results



Outline of the Presentation

- Secure Data Engineering Lab @ College of Charleston
- Secure Software Development
- Overview of High Traffic SOA vs. 2-Tier Architectures
- Lost Updates Example
- Related Research
- Research
  - Buddy System
  - Capacity Constraints
  - Coarse Grained Services
  - Service Constraints
  - Business Filters
  - Long Running Transactions
  - Integrating Heterogeneous Systems
  - Autonomous Process Authentication
  - Modeling Vulnerable Application Partitions
- Questions

Long Running, Consistent, Web Service Transactions

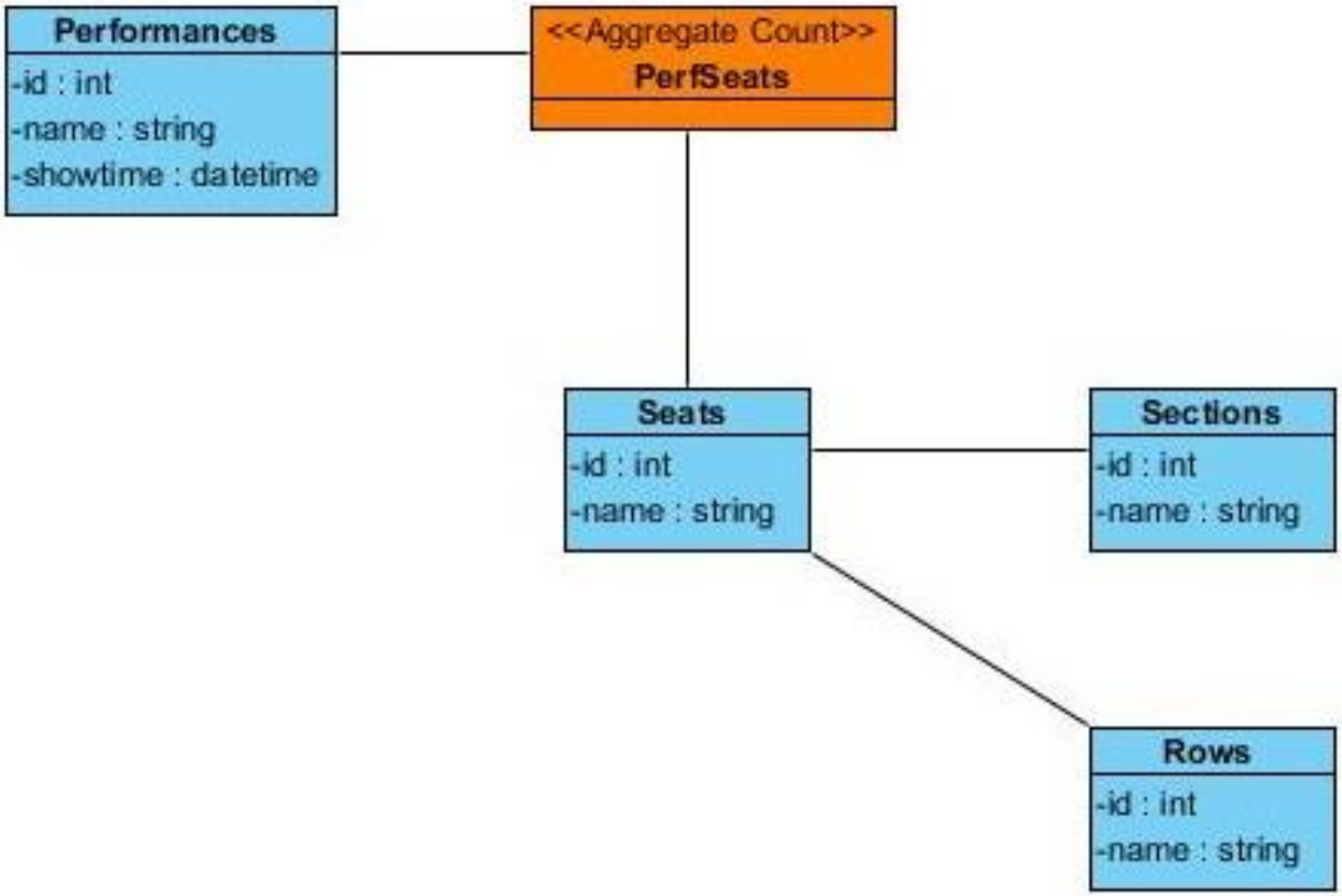
- Workflow transactions require several round trips
- Unfortunately they lower availability by holding locks longer
- Garcia-Molina defined sagas as a solution to maintain some of the atomic properties of ACID transactions when performing long running transactions.

Olmsted, Aspen. "Long Running, Consistent, Web Service Transactions." Proceedings of the 10th International Conference for Internet Technology and Secured Transactions (ICITST-2015). London: Institute of Electrical and Electronics Engineers (IEEE), 2015. 139 - 144.

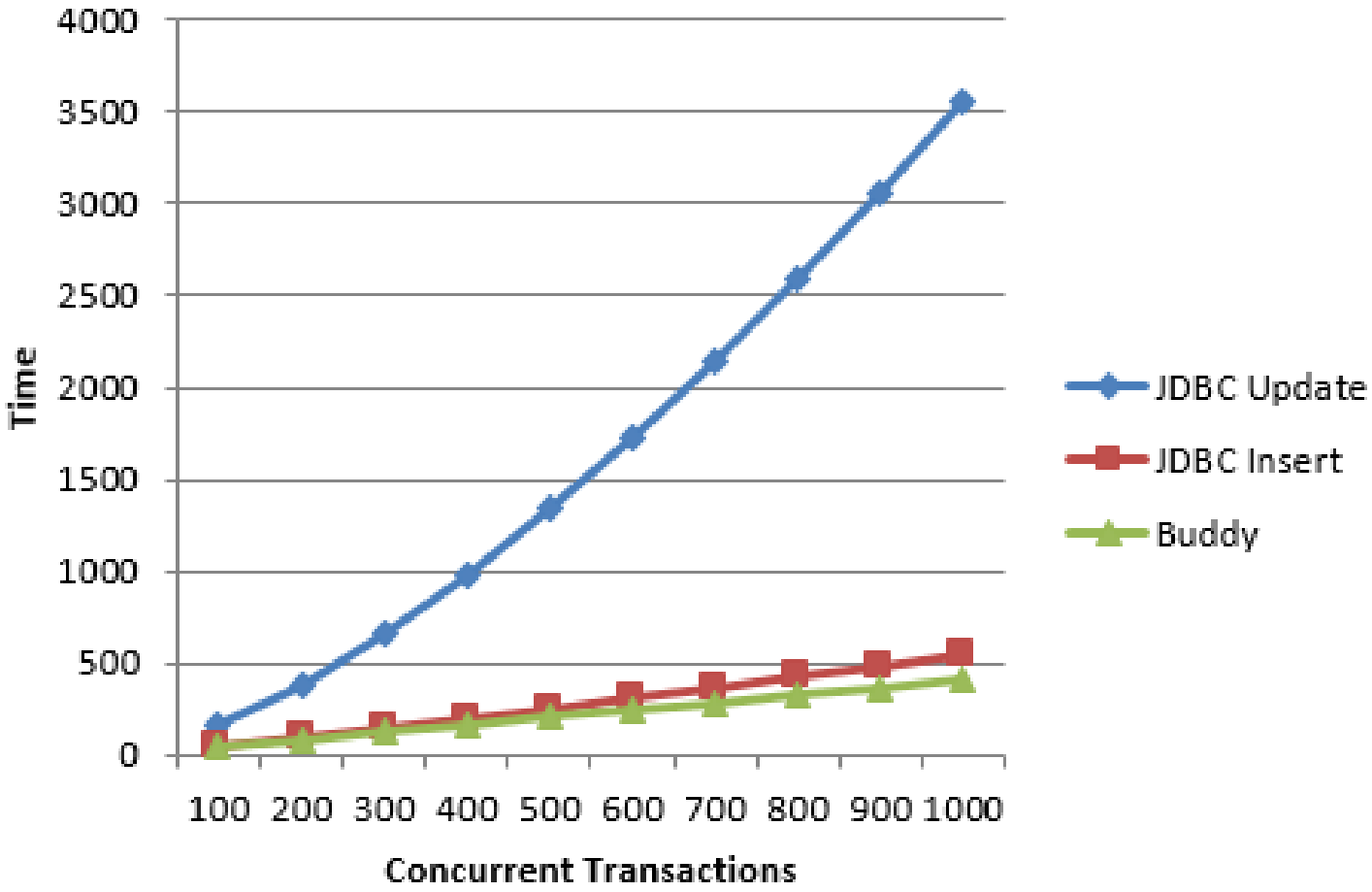
Compensators for CRUD

- Insert = Delete
- Delete = Insert
- Update ?

Model for Inserts



Empirical Results



Outline of the Presentation

- Secure Data Engineering Lab @ College of Charleston
- Secure Software Development
- Overview of High Traffic SOA vs. 2-Tier Architectures
- Lost Updates Example
- Related Research
- Research
  - Buddy System
  - Capacity Constraints
  - Coarse Grained Services
  - Service Constraints
  - Business Filters
  - Long Running Transactions
  - Integrating Heterogeneous Systems
  - Autonomous Process Authentication
  - Modeling Vulnerable Application Partitions
- Questions

Integrating Heterogeneous Systems

- Enterprise Functional Partitioning
- Transitioning Between Enterprise Systems

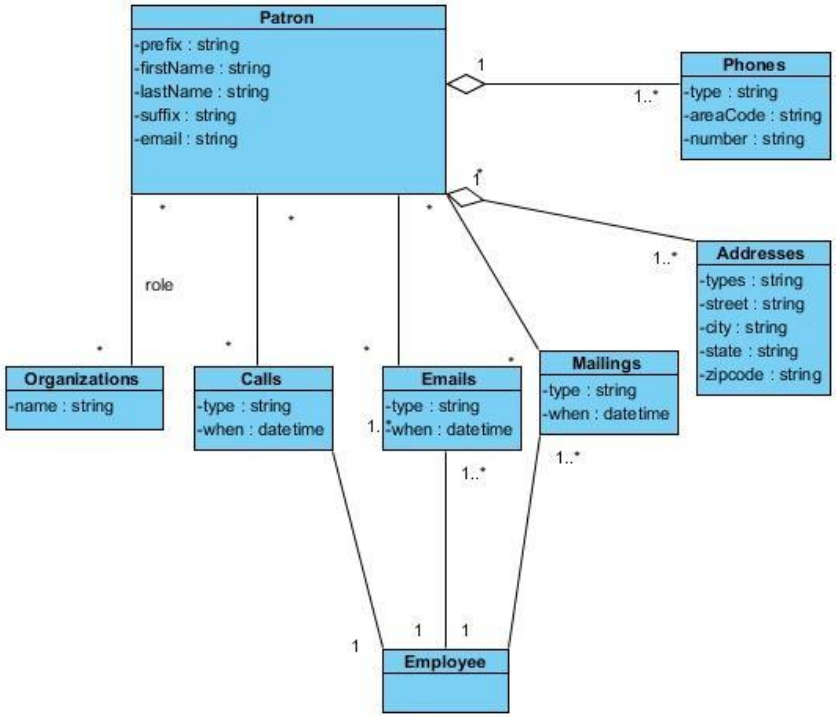
Olmsted, Aspen. Fresh, Atomic, Consistent and Durable (FACD) Data Integration Guarantees, *Software Engineering and Data Engineering, 2015 International Conference for* , Publication Year: 2015

Olmsted, Aspen. Continuous Data Integration Guarantees, *Software Engineering and Data Engineering, 2014 International Conference for,* Publication Year: 2014

Functional/Departmental Partitioning

- Enterprises value functional requirements over cost of departmental partitioning
- Survey of CIO Arts
  - Fund Raising Systems
    - CRM & ERP for Donations
  - Ticketing Systems
    - CRM & ERP for Tickets
  - Event Management Systems
    - CRM & ERP for rentals

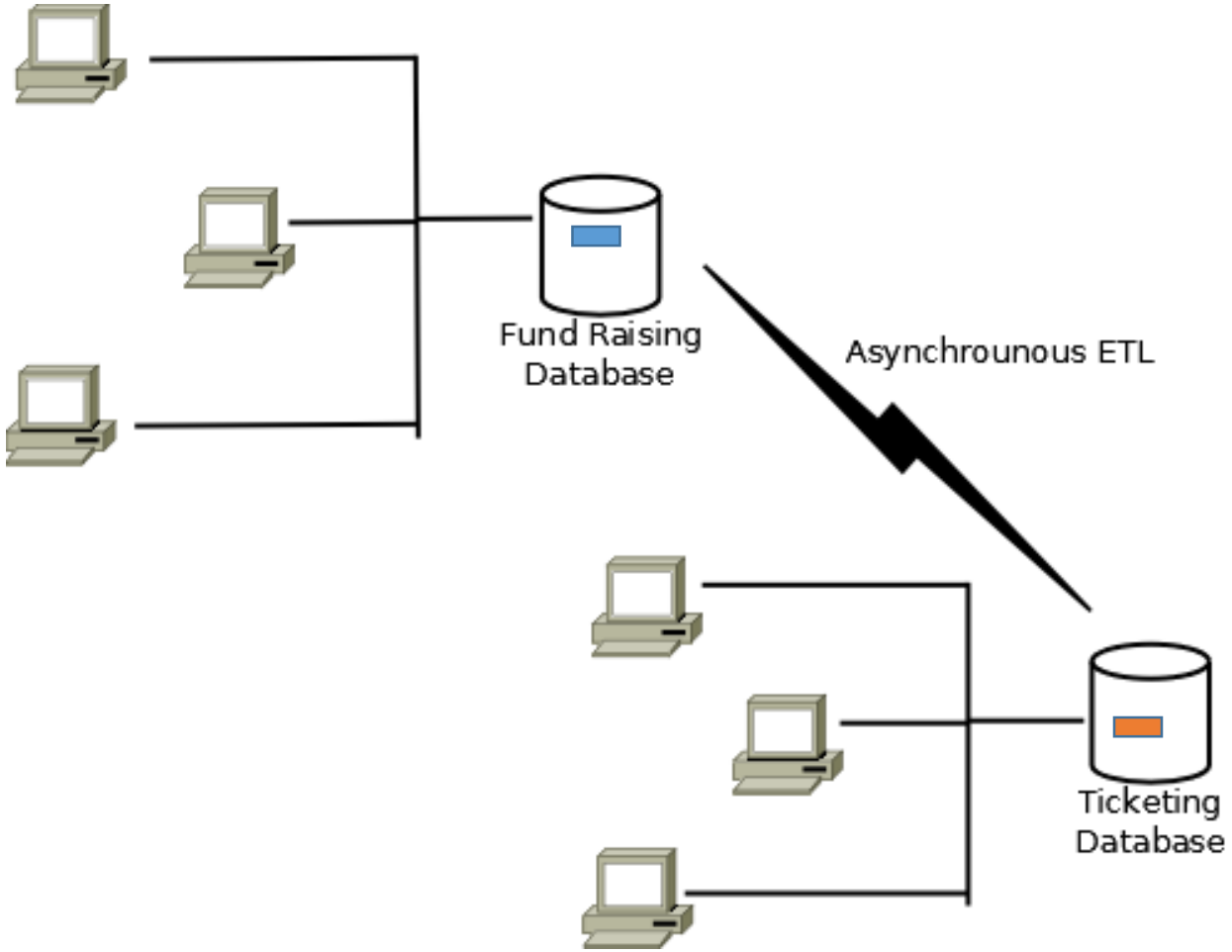
Example Class Diagram for Each Departmental CRM



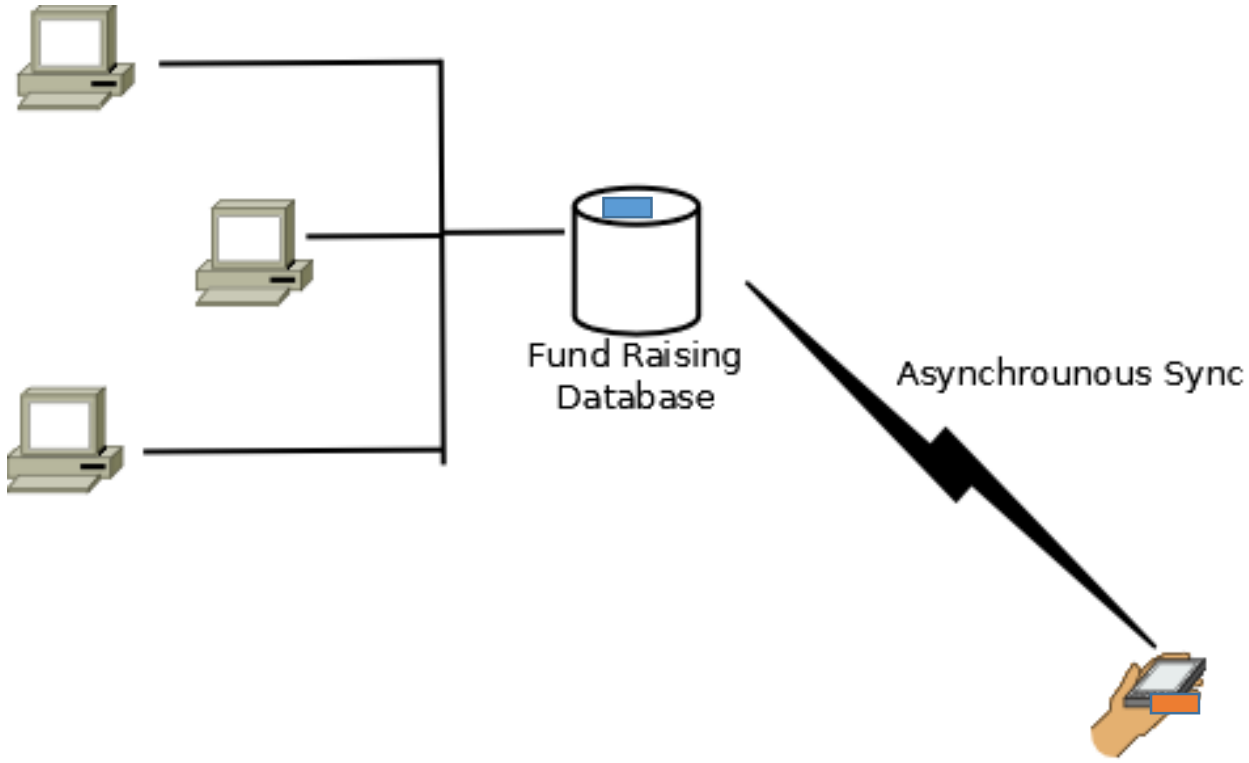
Duplicate Data

- System Partitioning by Functional Areas (Departments) can lead to duplicate data.
  - Biographical data
    - Sample organization data
      - CASS Standardized
      - Move Update Database
      - Name Search Web Services
    - Found 17% duplication caused by departmental partitioning

Continuous ETL



Mobile Sync Adaptors



Android Sync Adapter

```
/**
 * Handle the transfer of data between a server and an
 * app, using the Android sync adapter framework.
 */
public class SyncAdapter extends AbstractThreadedSyncAdapter {
    ...
    // Global variables
    // Define a variable to contain a content resolver instance
    ContentResolver mContentResolver;
    /**
     * Set up the sync adapter
     */
    public SyncAdapter(Context context, boolean autoInitialize) {
        super(context, autoInitialize);
        /*
         * If your app uses a content resolver, get an instance of it
         * from the incoming Context
         */
        mContentResolver = context.getContentResolver();
    }
    ...
    /**
     * Set up the sync adapter. This form of the
     * constructor maintains compatibility with Android 3.0
     * and later platform versions
     */
    public SyncAdapter(
            Context context,
            boolean autoInitialize,
            boolean allowParallelSyncs) {
        super(context, autoInitialize, allowParallelSyncs);
        /*
         * If your app uses a content resolver, get an instance of it
         * from the incoming Context
         */
        mContentResolver = context.getContentResolver();
        ...
    }
```

Sync Adapter Issues

- Developer overrides a method to handle conflict resolution
- Conflict only defined on the individual object level

Lost Update

- If the same object is updated between sync executions which version is kept.
- What about related records
  - Address updated in one system
  - Phone number updated in other

Homogenous Replication Related Work

- Improving Availability of Strict Replication
  - Snap Isolation Replication (Fekete, et al.)
- Improving Consistency of Lazy Replication
  - Frameworks with Lazy Consistency Guarantees (Breitbart & Korth)
- Hybrid Systems
  - Hybrid Majority Systems (Jajodia & Mutchler)
  - NoSQL Systems; Casandra, BigTable, etc.

Goals

- Develop Algorithms and Architecture that will
  - Guarantee Transaction Correctness for Disconnected Distributed Transactions Across Heterogeneous Systems

Problem Specification

- Leverage continuous ETL/Sync architecture (when connected sync changes, when disconnected allow changes)
- Add ACID transaction guarantees at snapshot isolation level
- Ensure data is fresh

Proposed Solution - CCETL

Continuous, Consistent, Extract, Translate and Load

- Form transactions for hierarchical data
- Pull hierarchy from design model
- Handle conflicts on the transaction level (not tuple)
- Guarantees one-copy serializability (At snapshot isolation level)

FACD

- F – Fresh – Data is kept up to date
- A – Atomic – All of a transaction is successful or none of the transaction is successful
- C – Consistent – The data in the database is correct before and after the transaction
- D – Durable – The effects of the transaction do not go away after the transaction is successful

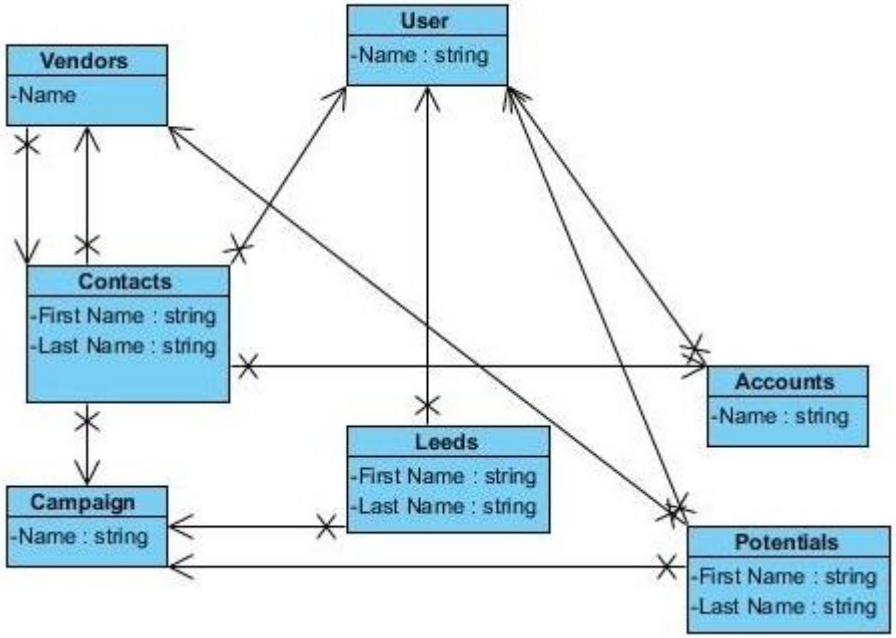
Transaction Formation – Option 1

- Intercept original transaction synchronously and send original transaction contents asynchronously
  - Requires Application Hook (Application Trigger)
    - Oracle Forms
    - JavaEE (Filters)

Transaction Formation – Option 2

- To reform a transaction asynchronously from the original transaction, we need a way to identify what data changed in the original transaction.
  - Homogenous Systems – database log
  - Heterogeneous Systems – Identify changes and reform transaction

Example System ZohoCRM



Changed Objects Web Service Call

Algorithm 1 REST Web-Service call to search for changed entity objects

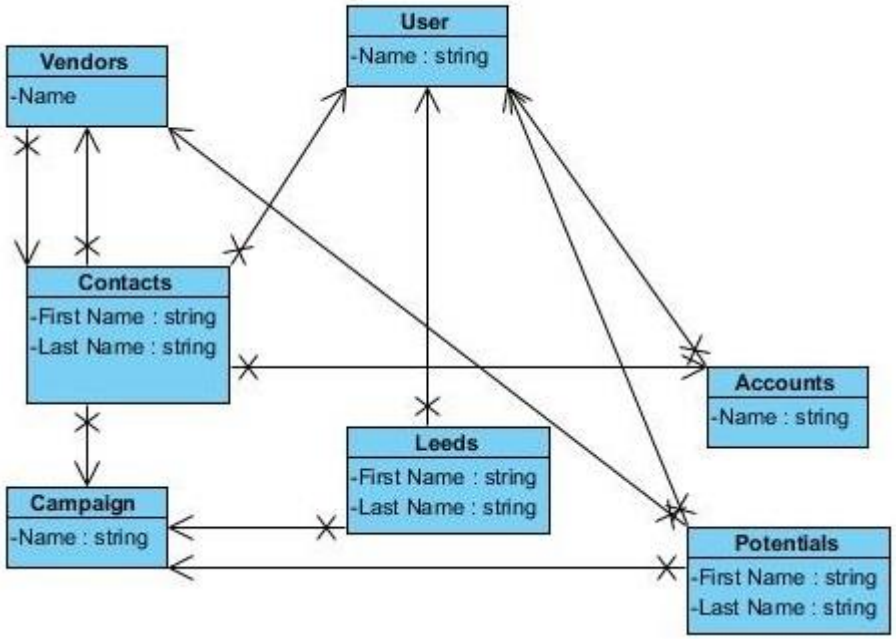
INPUT: date and time of last execution
OUTPUT: XML file

https://crm.zoho.com/crm/private/xml/Contacts/getSearchRecords?authtoken=Auth Token&scope=crmapi
&selectColumns=Contacts(First Name,Last Name,Email)&searchCondition=(lastModifiedTime|>|2010-04-21 11:09:23)

Transaction Identification

- To identify which records make-up a transaction, CCETL includes all associated records that were modified along with the parent record.
- This identification requires an ordering of the original UML diagram

UML Model (DCG)



Topological Sort

- Step 1
  - The first step uses Tarjan's algorithm to find cycles from individual nodes.
  - Once a cycle is found, an incoming edge is removed and the process continues until all cycles are removed.

Topological Sort

- Step 2
  - The second step uses a process of generalization by inserting mock objects into the inheritance tree.
  - The mock objects are inserted when there are identical inbound edges into a node.
  - The addition of the mock objects reduces the branches in the path of the UML graph.

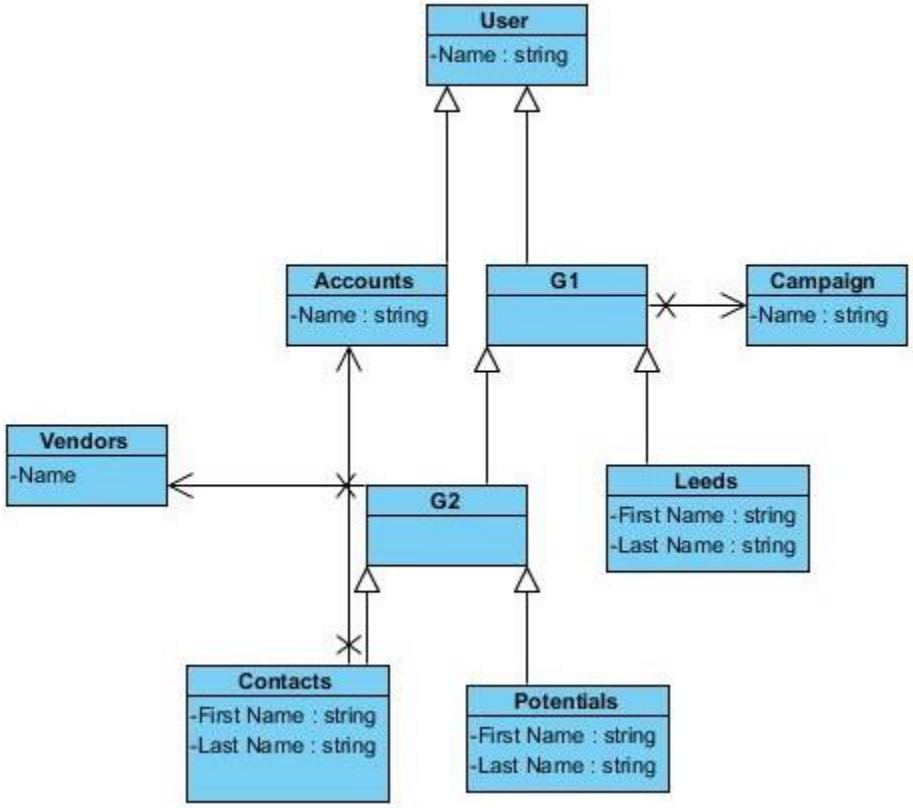
Topological Sort Algorithm

Algorithm 2 Sort and Remove Cycles in UML Graph

INPUT: XMI of UML graph
OUTPUT: XMI of new UML graph

```
/* remove Cycles in UML */
Loop through all objects
   Check for cycles using Tarjan's algorithm [4]
   While cycles exist that include node
     Remove edge that is part of cycle
     Repeat cycle check
 /* Sort */
Loop through all objects
  If number of incoming edges > 1
    Insert mock inheritance object
```

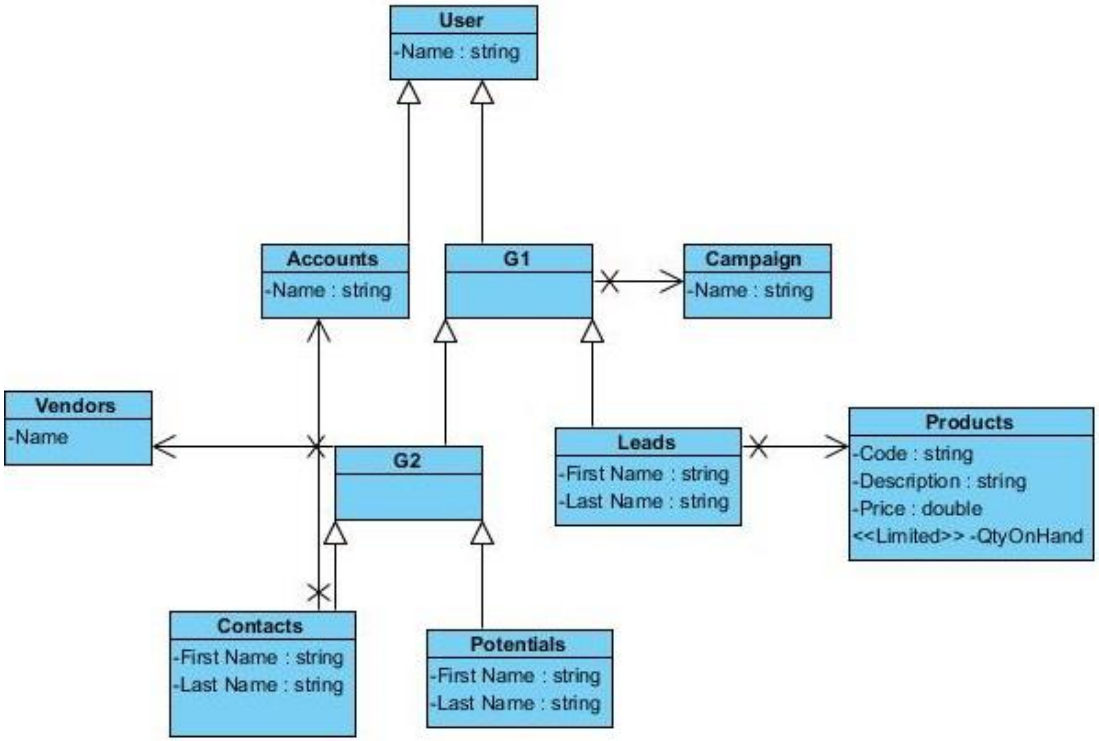
UML Model (DAG)



3 Types of Data

- Leaf Nodes – Lookup data
- Inner Nodes – Transactional data
- Limited Attributes – Stereotypes for synchronous data

UML Model (DAG) w/stereotype



Continuous Data Integration

- Algorithm picks up changed data
- Algorithm scheduled periodically
- Data is written to the transaction table to ensure that the integration does not pickup data that was previous written by the integration.
- Leaf Nodes Scheduled First

CDI – Data Table

Table 1 Example Transaction Table

Object	Key	Timestamp
Contacts	2500	2014-04-01 14:15:29
Contacts	4500	2014-04-01 14:15:29

Implementation

- Java
- ZohoCRM (Web Services)
- Tessitura ERP (SQL)
- Concurrent transactions (100-1000)
- Single Object vs Transaction

Empirical Results

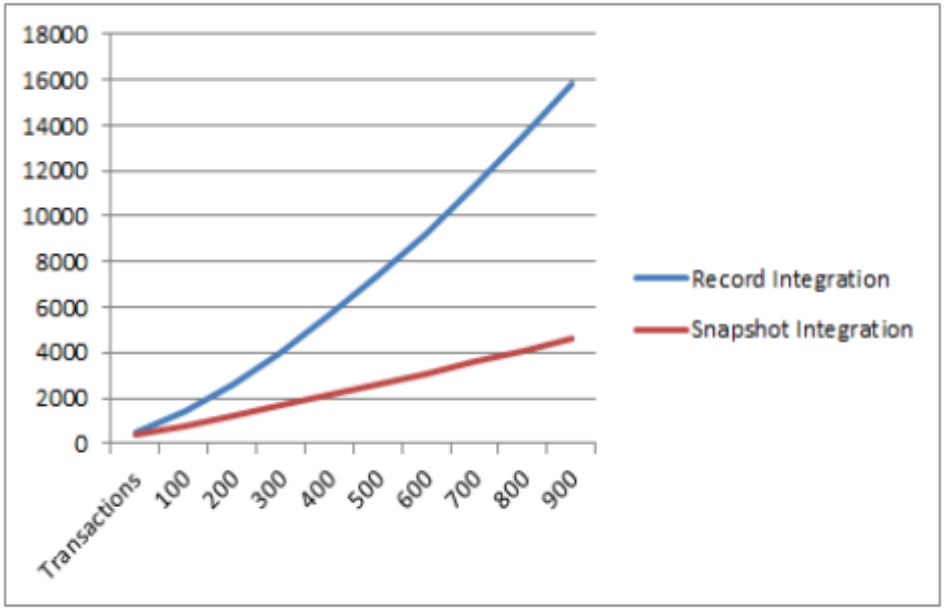


Figure 3 Throughput Record vs Snapshot Integration

Outline of the Presentation

- Secure Data Engineering Lab @ College of Charleston
- Secure Software Development
- Overview of High Traffic SOA vs. 2-Tier Architectures
- Lost Updates Example
- Related Research
- Research
  - Buddy System
  - Capacity Constraints
  - Coarse Grained Services
  - Service Constraints
  - Business Filters
  - Long Running Transactions
  - Integrating Heterogeneous Systems
  - Autonomous Process Authentication
  - Modeling Vulnerable Application Partitions
- Questions

Native Autonomous Process Authentication

Olmsted, Aspen. "Native Autonomous Process Authentication." Proceedings of World Congress on Internet Security 2016 (World-CIS 2016). London, UK: Institute of Electrical and Electronics Engineers (IEEE), 2016

Authentication

- Something you know
- Something you have
- Something about you
- Someplace you are

Autonomous Process - HTTP

- Apache – Runs under a Linux user (Something you know)
  - All websites share a single user
  - Credentials can not expire like normal users
- IIS – Integrated Security
  - Depends on security of client

OAuth 2.0 Workflows

- Web-Server (user authorizes web app)
- User-Agent (user authorizes fat client)
- JWT Bearer Token – Replay signed JSON token
- SAML Bearer – Replay sign SAML
- SAML Assertion – Federated single sign-on
- User and password

Enterprise System Security Audits

- Counterpoint – point of sale used by 1,000s of organizations
- Tessitura – ERP for Performing Arts and Museums (over 500 large organizations)

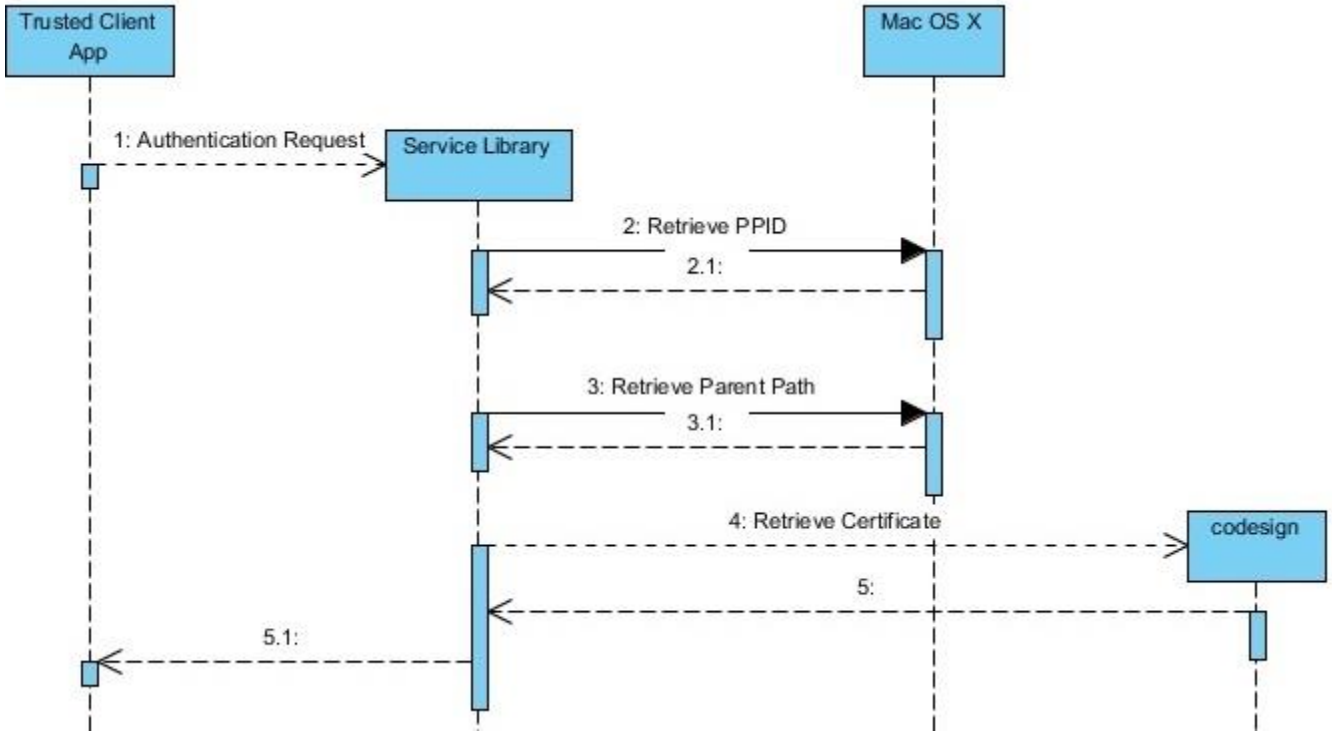
Goals

- Add “something about you” authentication for autonomous process
- Verify connecting process is not a malicious process

Solution

- Use OS X app certificate (not what they are designed for)
- Use codesign utility to validate the certificate from PPID

Tools/Process



Services Provided

- AddUser credentials
- AddApp credentials
- Create object
- Read object
- Update object
- Delete object
- Exec process

Outline of the Presentation

- Secure Data Engineering Lab @ College of Charleston
- Secure Software Development
- Overview of High Traffic SOA vs. 2-Tier Architectures
- Lost Updates Example
- Related Research
- Research
  - Buddy System
  - Capacity Constraints
  - Coarse Grained Services
  - Service Constraints
  - Business Filters
  - Long Running Transactions
  - Integrating Heterogeneous Systems
  - Autonomous Process Authentication
  - Modeling Vulnerable Application Partitions
- Questions

Modeling Cloud Apps For Partition Contingency

Olmsted, Aspen. "Modeling Cloud Applications for Partition Contingency." Proceedings of the 11th International Conference for Internet Technology and Secured Transactions (ICITST-2016). Barcelona: Institute of Electrical and Electronics Engineers (IEEE), 2016. 230-234.

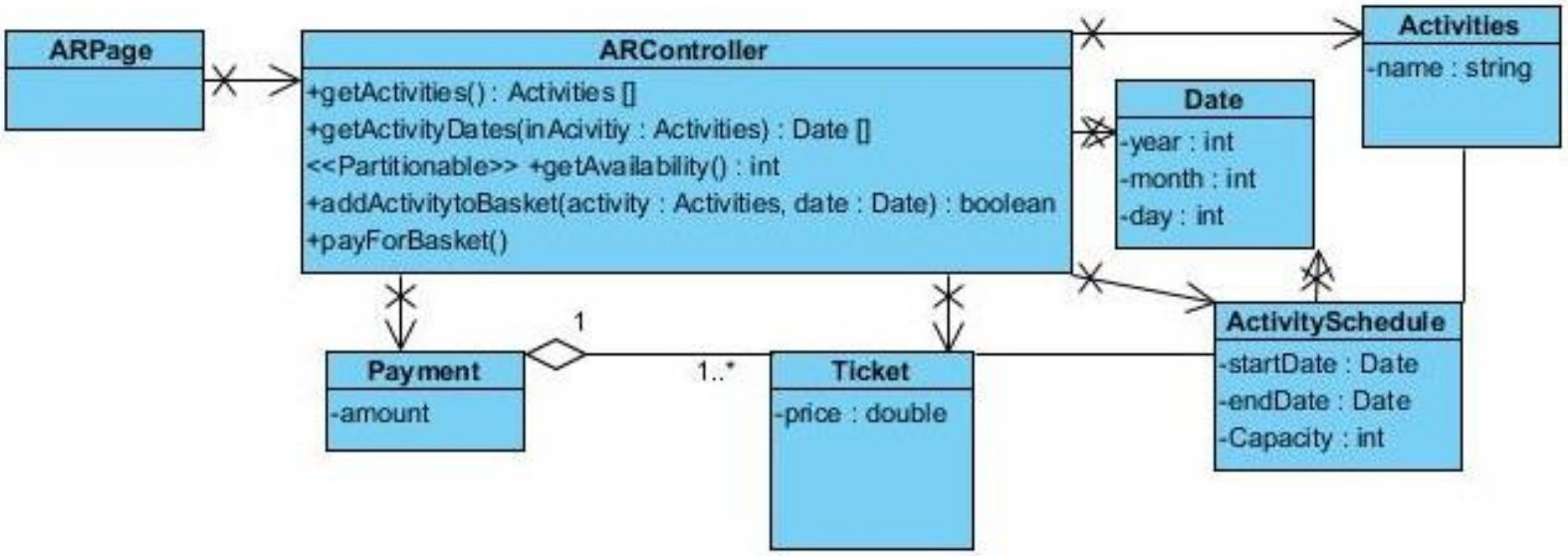
Cloud Computing

- IAAS – Infrastructure as a service
- SAAS – Software as a service
- PAAS – Platform as a service

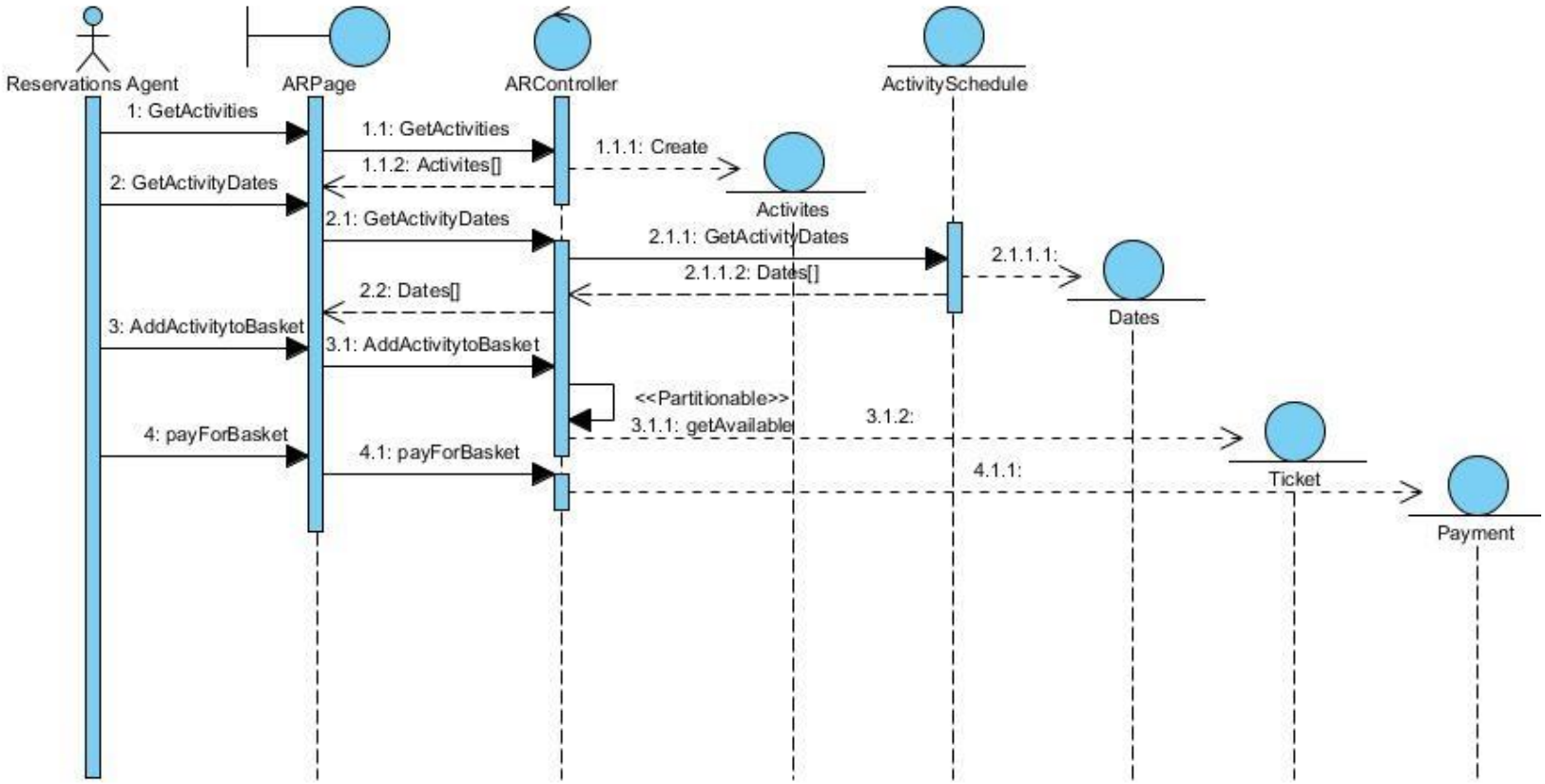
Motivating Example

- Gettysburg National Battlefield
  - Students building reservation systems in cloud
  - Using Salesforce PAAS

Subset of UML Class Diagram



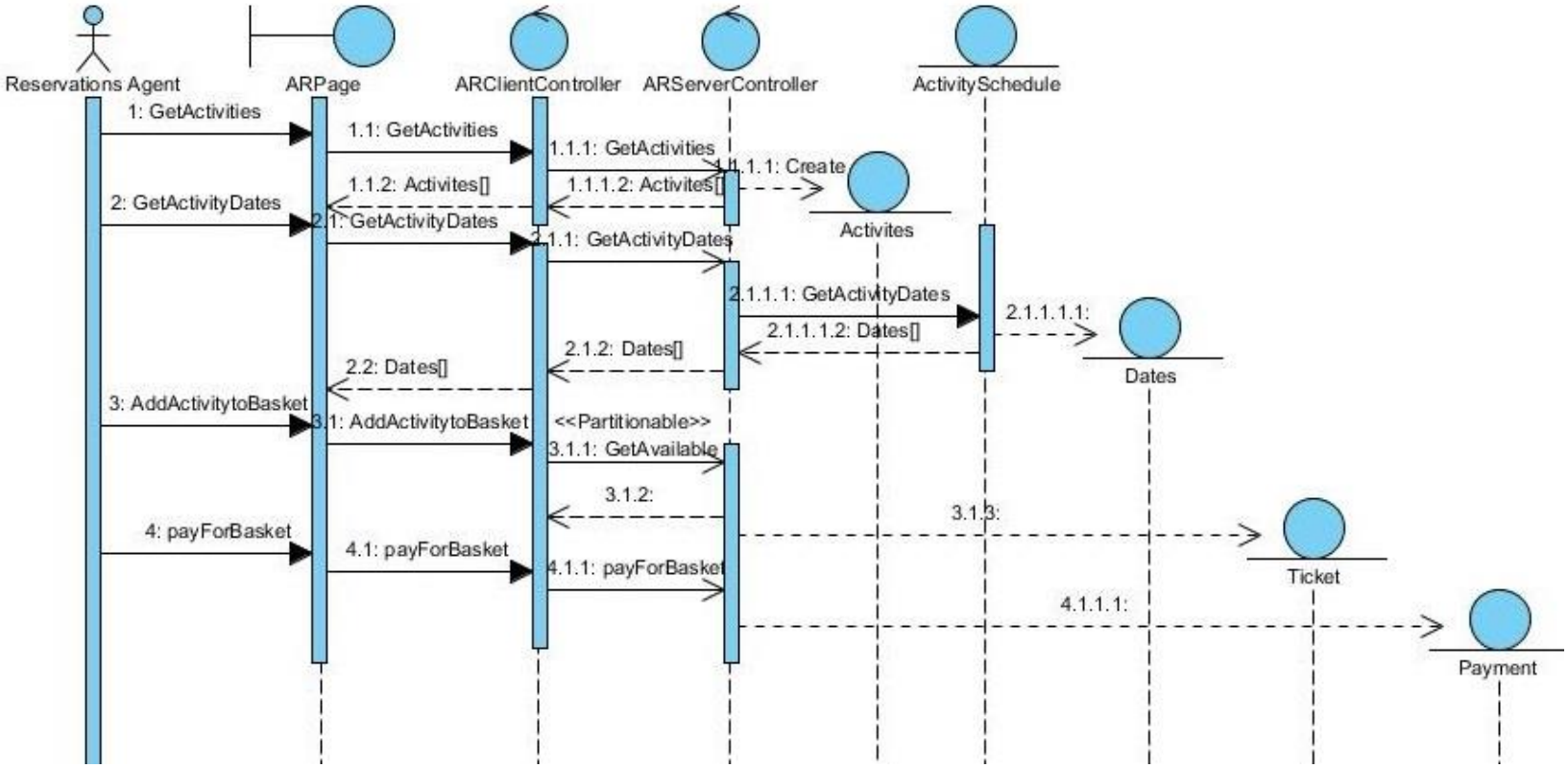
Sequence Diagram



Partition Tolerant Modeling

- Route all logic through a client-side controller
- All messages from the ARPage now flow through the ARClientController object.
- Messages are passed asynchronously to the controller in the cloud.
- If a response is not received then, a previous result will be used from the client cache.

New Sequence Diagram



Partitionable Methods

- Stereotype methods that need an allotment
- The stereotype “partitionable” tells the server to do this allotment in the UML design model.
- This stereotype allows the client to continue selling but not oversell available resources in the case of a network partition.

Scarce Recourses

- The Guides at Gettysburg were a limited resource so allocation did not work.
- Implemented a pier-to-pier solution where server told client who held “allocations” of scarce recourses.

Generalizing Solution

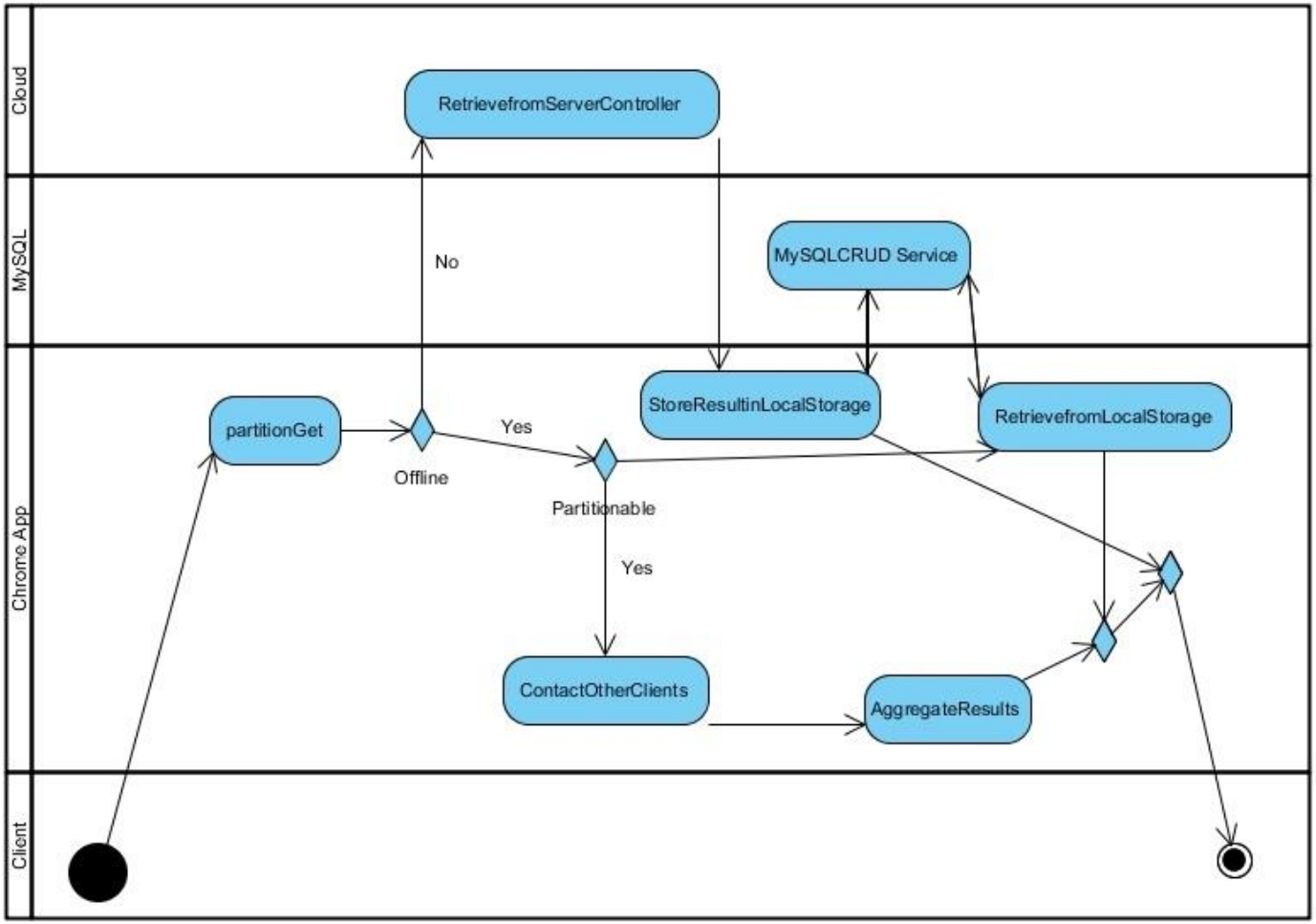
- Aura Framework
  - model-view-controller-controller pattern.
  - both a client side controller and a server side controller.
  - Salesforce uses the Aura framework for their Lightening Component architecture.

- We developed Chrome Extension, App and Javascript library

Chrome App/Extension and Javascript Library

- Google Chrome apps have greater permissions than traditional browser applications.
- These higher permissions allow the chrome app to provide the local network services our algorithm needs.
- The JavaScript code in the library sends local messages from the sandbox provided by the Chrome browser to our Chrome app.

Activity Diagram



Chrome App/Extension and Javascript Library

- partitionGet
- partitionSave
- dataGet
- dataSave

Conclusions/Future Work

- Successfully modeled/implemented componentized partition tolerant architecture
- Google chrome app runs an asynchronous job to see if it can go back online
  - Inserts and deletes are sent to server

- Future work needs to either handle updates or stress the insert only modeling
- Future work needs to generate the code based on the XMI

Modeling Non-Functional Requirements

Devata, Santoshi, and Aspen Olmsted. "Modeling Non-Functional Requirements in Distributed Application Software Engineering." Proceedings of The Seventh International Conference on Cloud Computing, GRIDs, and Virtualization (Cloud Computing 2016). Rome: International Academy, Research, and Industry Association (IARIA), 2016. 47-50.

Olmsted, Aspen. "Secure Software Development through Non-Functional Requirements Modeling." Proceedings of the International Conference on Information Society (I-Society 2016). Dublin, 2016. 22-27.

Introduction/Background

- Functional Requirements – Things a system must do
- Non-Functional Requirements – Things that must be true in the system

Introduction/Background

- Software architects model functional requirements using UML diagrams
  - Functional Model – Use Case
  - Structural Model – Class Diagrams, OCL
  - Dynamic Model – Activity & Sequence Diagrams
- No standard modeling notation to represent Non-functional requirements

Non-functional Requirements Represent Challenge with Cloud/Distributed Software Engineering

- Concurrency – HealthCare signup database
- Security – App partitions cross domain boundaries
- Latency – Network outside of domain boundaries
- Shared Resources – Locking resources consumed by use outside domain boundaries

Motivating Example – Ticketing Application

- Sell limited supply of tickets for specific events
- Inventory locks on seat or section level

Goal

- Use standard design tools to model non-functional requirements
  - UML
  - OCL
- Export XMI
- Read XMI and generate code to enforce non-functional requirement

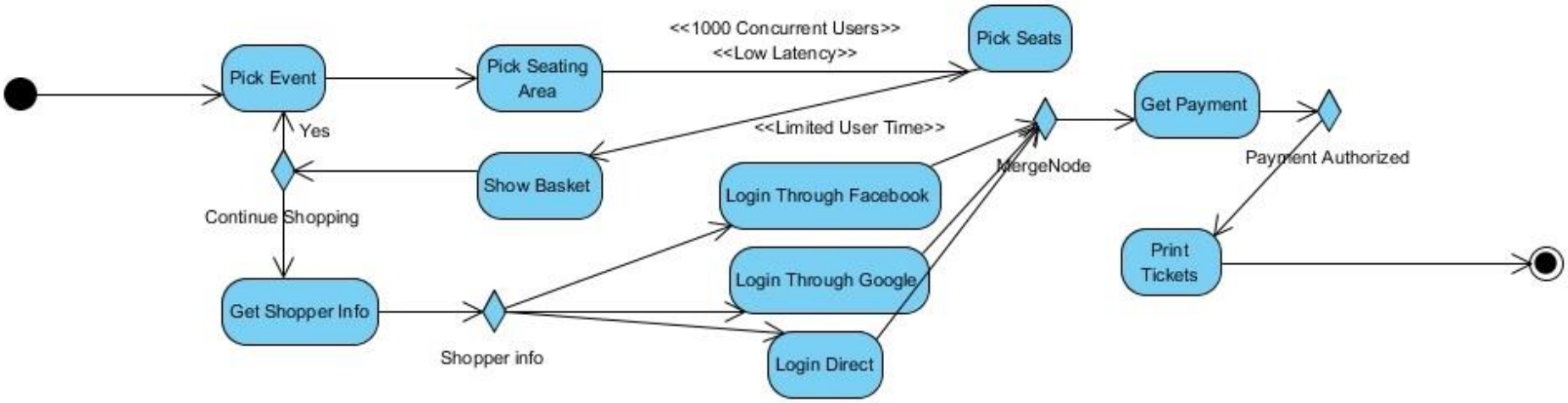
Tools

- Model data in UML with stereotypes for non-functional requirements
- Export to XMI format
- Run Algorithms on XML to produce application code

3 Stereotypes

- Concurrent Users (Handle a number of users, rest to queue)
- Low Latency (Server must respond in certain amount of time)
- User Response Time (User must respond in certain amount of time

Activity Diagram for Ticketing Workflow w/Stereotypes



• Concurrency Algorithm

INPUT: XML of request, timeout
OUTPUT: XML of data entered or XML with error

```
Check if anythreads in pool
If no threads in pool
   Set timer to fire every second
   Set timeExpired = 0
   Do
      Check if thread available in the pool
   While timeExpired < timeout or thread received
If not thread received
   Set response to timeout error
ELSE
    Execute request in thread
End if
Return response
```

- System can only handle x number of users picking seats at the same time.

- Low Latency Algorithm

INPUT: XML of Send to Server, timeout
OUTPUT: XML of response with server

```
Send request to server
Set timer to fire every second
Set timeExpired = 0
Do
    Check if response is received
While timeExpired < timeout or response received
If not response received
   Set response to time expiration error
   Set response to timeout error
    Notify server of timeout
End if
Return response
```

- System must respond to user in a maximum amount of time.

• User Response Time Algorithm

INPUT: XML of form to display, timeout
OUTPUT: XML of data entered or XML with error

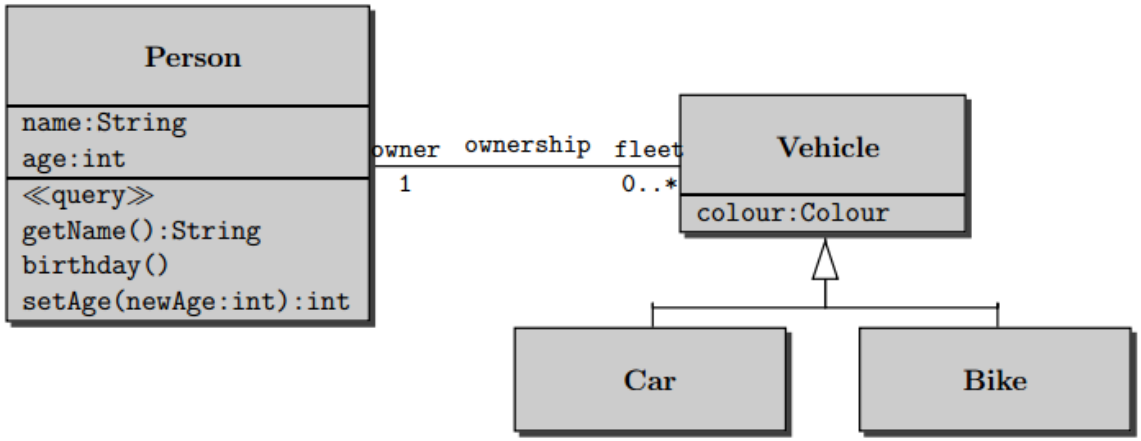
```
Show form to user
Set timer to fire every second
Set timeExpired = 0
Do
    Check if response is received
While timeExpired < timeout or response received
If not response received
   Notify user of time expiration
   Set response to timeout error
End if
Return response
```

- User must complete request in a maximum amount of time.

Challenges with Stereotype Solution

- No Stereotype Categorization
- Enterprise Size Project Would Have Thousands of Stereotypes

Object Constraint Language (OCL)



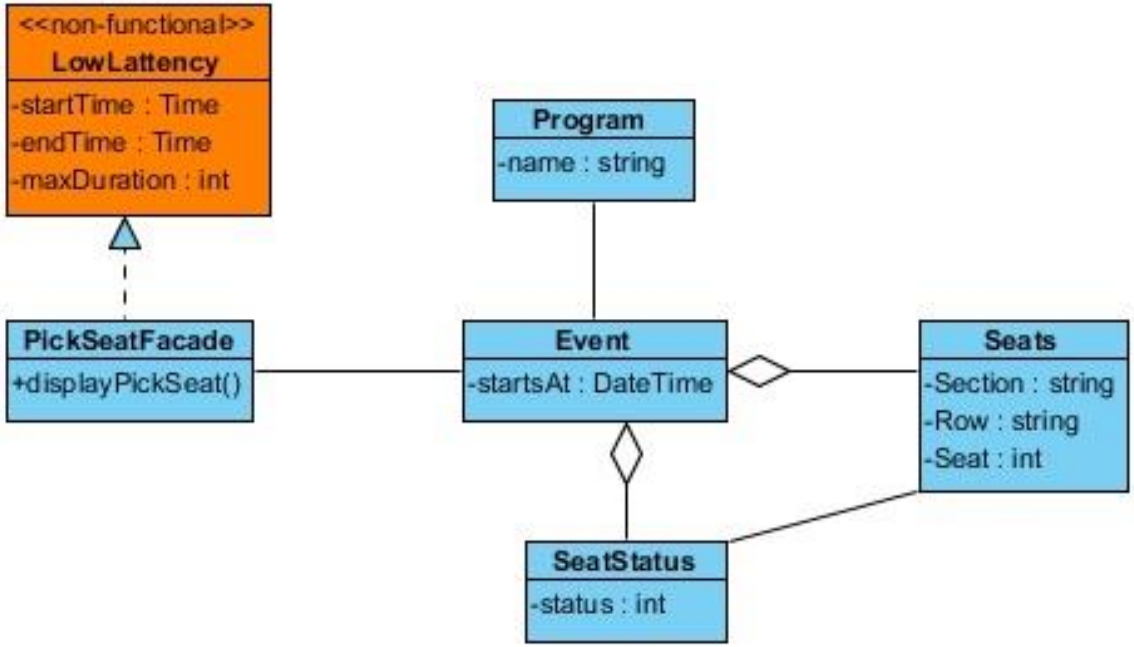
"A vehicle owner must be at least 18 years old":

context Vehicle
inv: self. owner. age $>=$ 18

Mock Objects

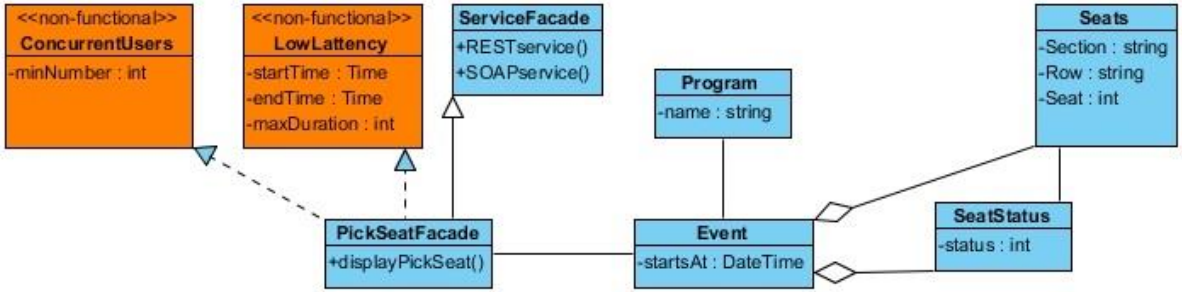
- Insert Mock Objects to Represent NFR
- Use Inheritance to Specify OCL Constraint on Inherited Attributes

Object Constraint Language (OCL)



Context PickSeatFacade::displayPickSeat **Post:**
self.startTime – self.endTime <= 60

Specification Multiple Inheritance



Inheritance is only required during code generation, allowing support in languages that do not support multiple inheritance syntactically

Code Generation From UML & OCL

1.) Import Profile
2.) Apply Stereotypes
3.) Insert Mock Objects
4.) Generate Java Class Stub from UML
5.) Export Model to XMI
6.) Parse XML to find OCL constraints and insert Java stub to guarantee constraint
7.) Remove generated mock objects

Modeling for Anonymous Cloud Data

- Tuple based Modeling to increase availability

Olmsted, Aspen, and Gayathri Santhanakrishnan. "Cloud Data Denormalization of Anonymous Transactions." Proceedings of The Seventh International Conference on Cloud Computing, GRIDs, and Virtualization (Cloud Computing 2016). Rome: International Academy, Research, and Industry Association (IARIA), 2016. 42-46.

Motivating Example – Gettysburg Foundation

- Grant for 5 undergraduate students to build cloud solution for ticketing system
- Reservations for Cycloroma, Film & Museum, Historical Buildings and 3rd Party Guides for Tours
- Self-Service sales for same activities
- Front-desk walkup sales

Old System – 2 Tier Windows Client/SQL Server

- Over 300 tables in the database
- Over 50 GB data for 10 years worth of history
- Requires expensive annual maintenance ($30,000/year), internal IT support and very complicated for end user reporting.

Solution – Custom Application on force.com

- As a charity they receive over 80% off on subscription cost
- 10 free enterprise users
- PaaS (Platform as a Service)
- Loads of training resources available

Challenges

- Only 1GB of data included in subscription
- Can subscribe to more data ($1,000/year/GB)

Measuring Tuples

- Force.com says “you get 1gb of data”, but really every tuple is 2kb. So you get 500,000 tuples before you need to pay more.
- Several other cloud platforms use tuples as the measure of data. Another example is Zoho.com

Goal

- Reduce tuples required in data model
- Maintain same reporting and auditing capability
- Make model as simple as possible for end user querying

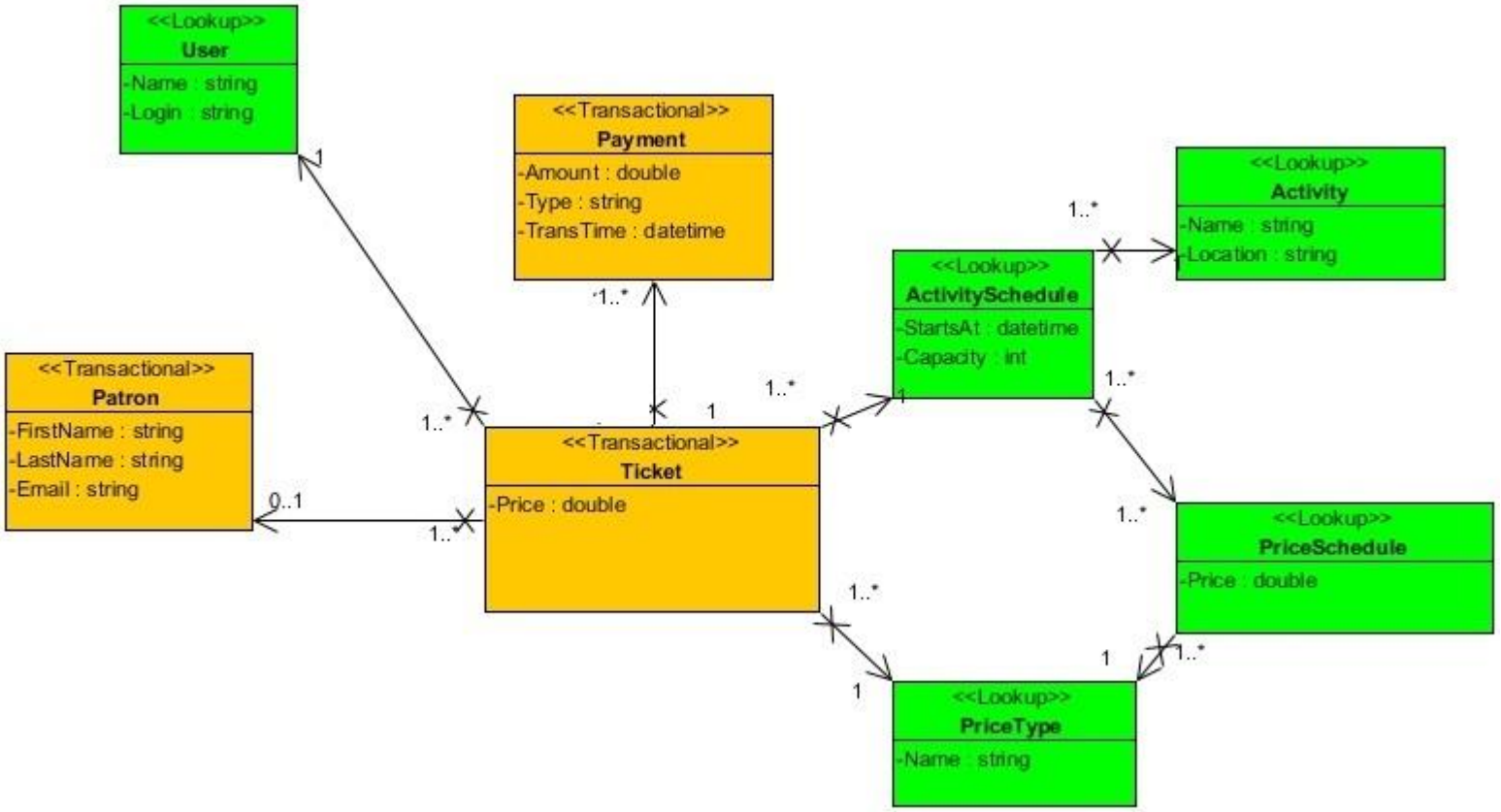
Tools/Process

- Model normalized data in UML with stereotypes for transactional and lookup data
- Export to XMI format
- Run Algorithms on XML to produce de-normalized ER

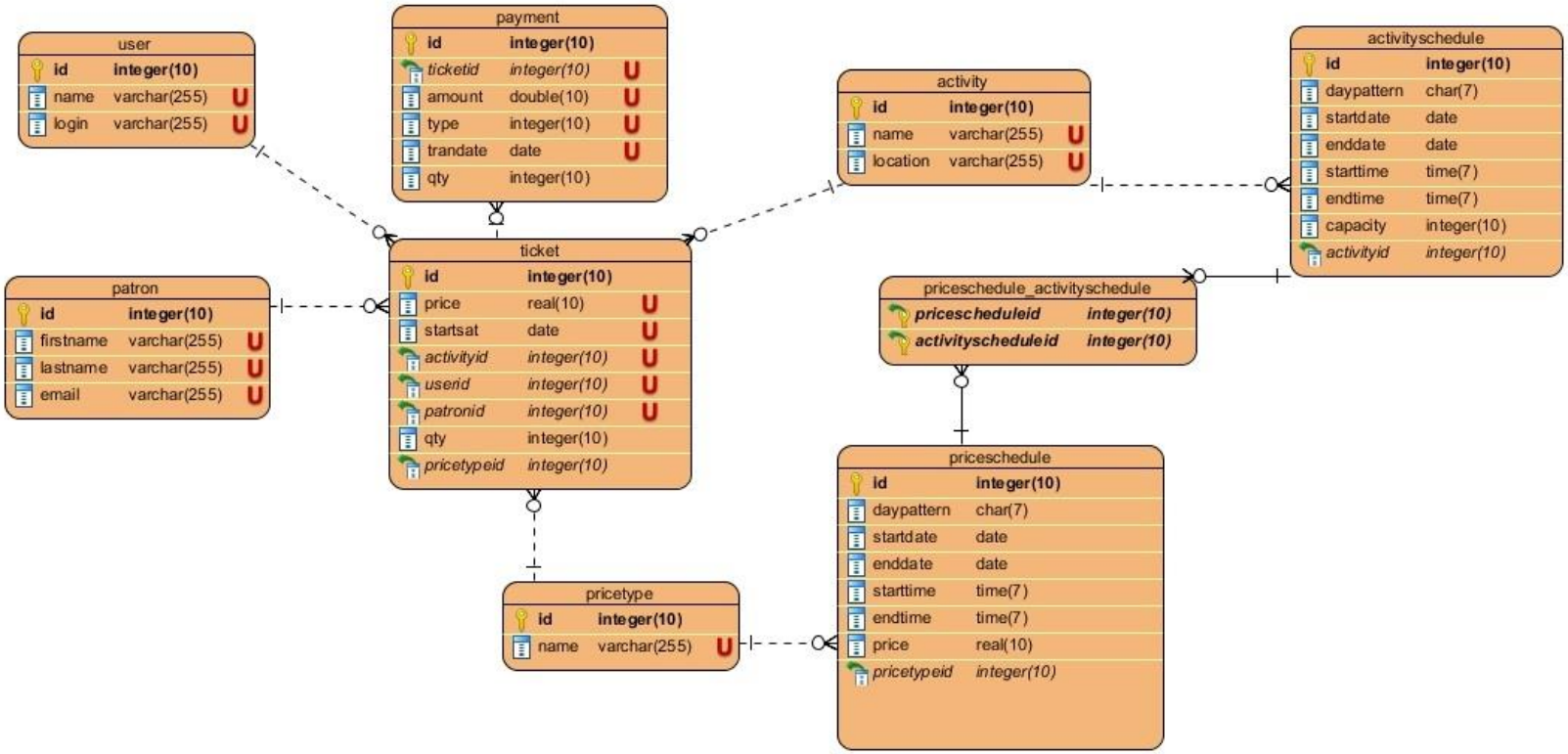
Database Normalization

- Set of steps taken to modify a database to
  - Free database from modification anomalies
  - Minimize redesign required to support functional changes
- Technically outputted schema is in 6th normal form
- Algorithm replaces natural keys with surrogate keys (often requiring redesign to support changes).
- Redesign should be done on normalized model

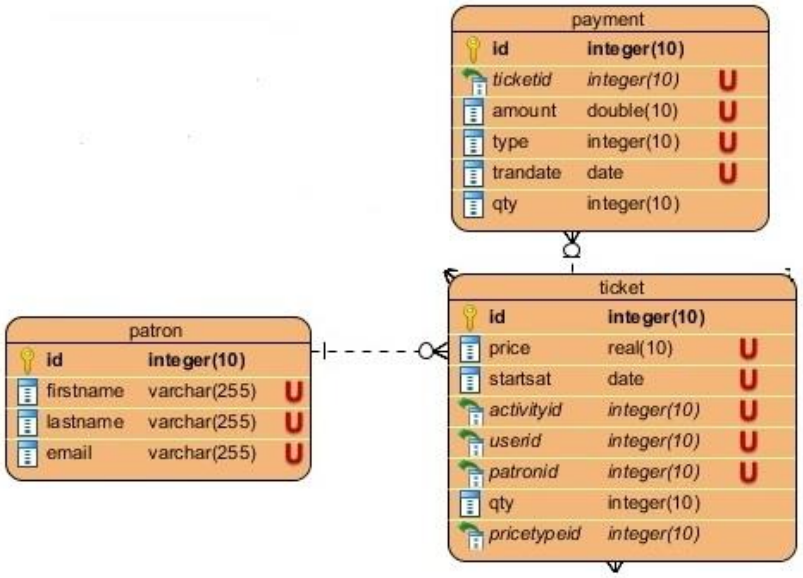
Normalized Data Model



De-Normalized Data Model

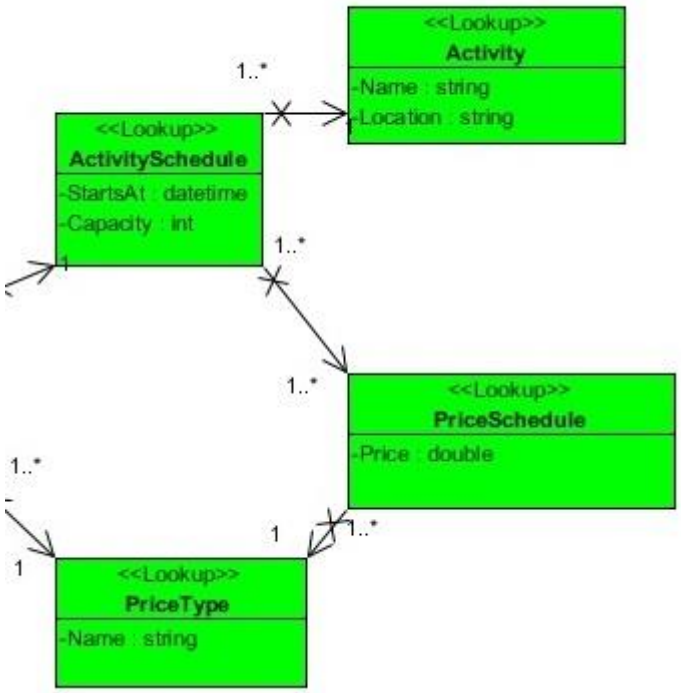


- Anonymous Patron Aggregation



- Large Percentage of transactions walkup
- Force.com provides a journal for audit type reports
- Transactional stereotype objects can be aggregated on *Unique* fields
- See paper for complete algorithm

- Swap Leaf Lookup Tables and Convert to Business Rules



- Large number of tuples used to store temporal intersections of lookup values
- Swap the leafs and turn into business rule pattern tables
- See paper for complete algorithm

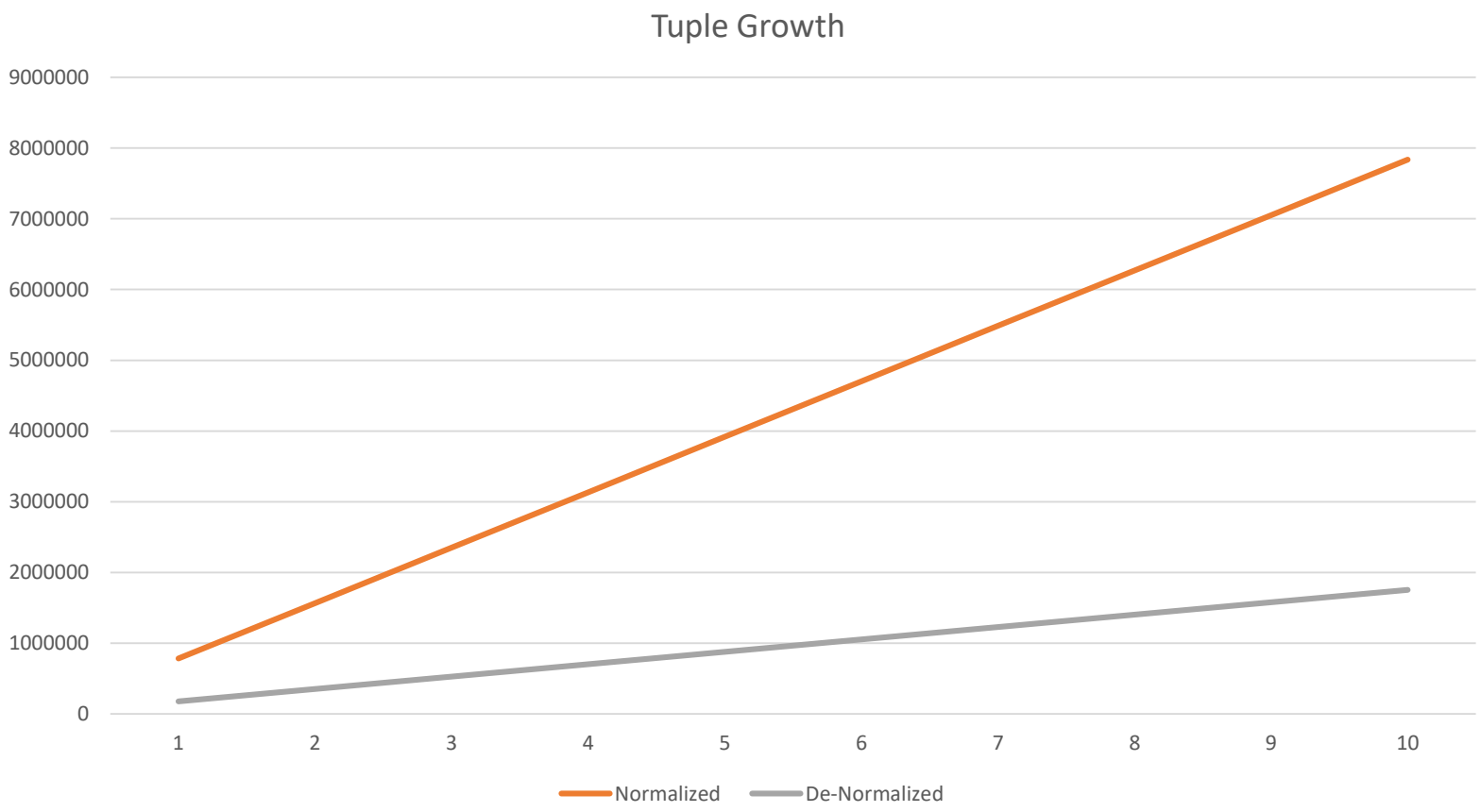
Journal to instance data

- Provided a cloud side function for reporting that takes object and date/time range and creates instance data for end user reporting

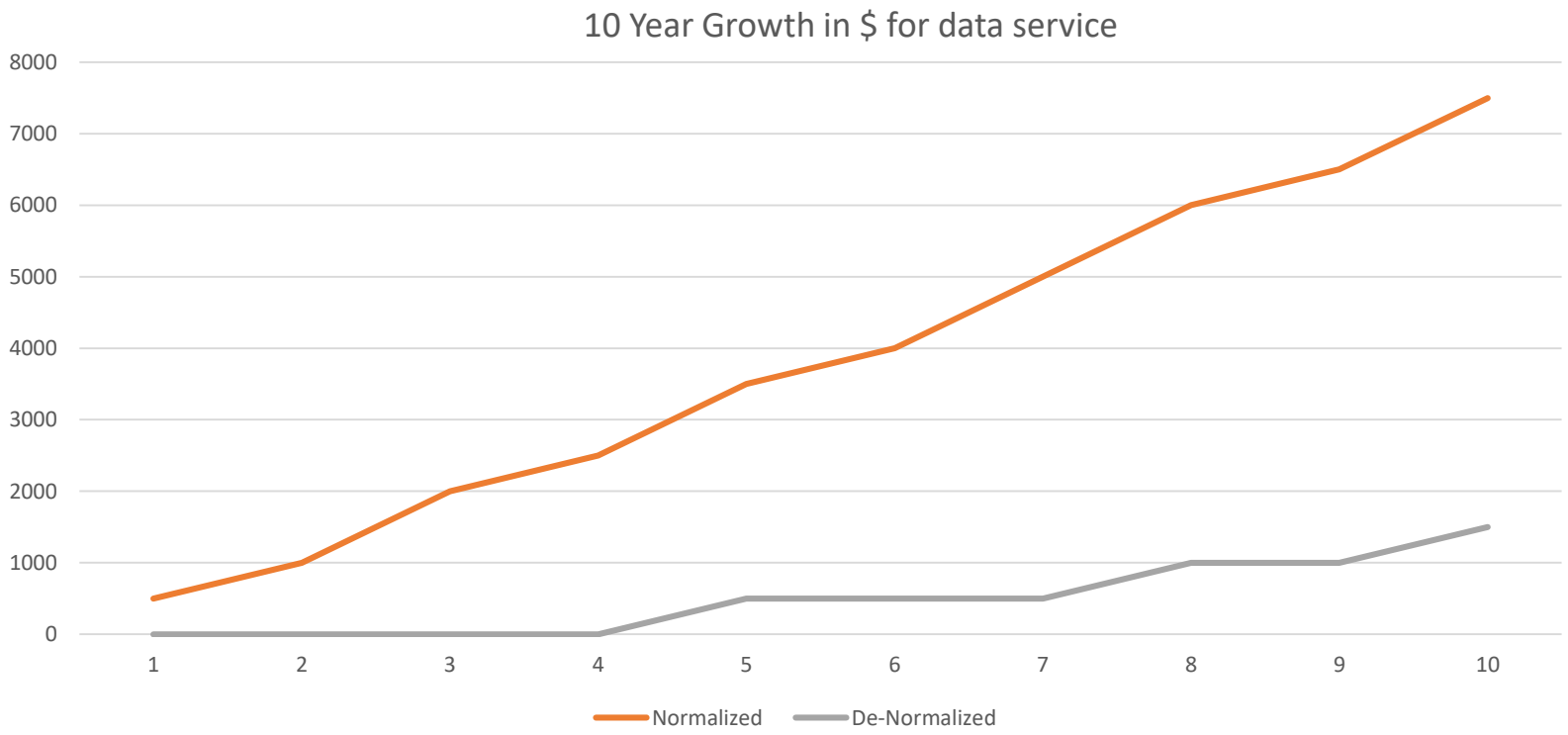
Results for 1 Year of Transaction Data

Table	*Normalized Tuples*	*Denormalized Tuples*
user	*31*	*31*
patron	*17,610*	*17,610*
ticket	*738,981*	*157,780*
activity schedule	*26,697*	*30*
price schedule	*220*	*24*
activity	*17*	*17*
Total	*783,556*	*175,492*

Tuple Growth



Cost Growth


10 Year Growth in $ for data service
8000
7000
6000
5000
4000
3000
2000
1000
0
1
2
3
4
5
6
7
8
9
10
Normalized
De-Normalized

Questions?