Synchronization in Collaborative Applications

Jonathan Munson
University of North Carolina at Chapel Hill

What is a collaborative application?
- General: an application designed to assist people engaged in a common task
  - E-mail, video conferencing, database system, shared editor, the Web
- Specifically interested in collaborative applications that involve shared data
  - database system, shared editor, the Web

What is synchronization?
- Synchronization controls access to shared data to prevent users from interfering with each other.
- To illustrate, some scenarios involving collaborative applications

Synchronization
- Prevents users from editing same object simultaneously

Alternative policy: exclusive editing
- For safety, as plans get finalized, a user may wish to get exclusive access to an entire drawing

Planner—exclusive editing policy
- Users can specify that certain drawings cannot be modified concurrently, as a matter of policy.
Alternative application: room scheduler

- Synchronization should disallow concurrent insertions of reservations that conflict.

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>9:00 am</td>
<td>Budget meeting</td>
</tr>
<tr>
<td>9:30 am</td>
<td></td>
</tr>
<tr>
<td>10:00 am</td>
<td>Weekly review</td>
</tr>
<tr>
<td>10:30 am</td>
<td></td>
</tr>
<tr>
<td>11:00 am</td>
<td></td>
</tr>
</tbody>
</table>

Implications: (1) programmer-defined consistency criteria

- Notions of conflict vary from application to application
  - In drawing application, users can always insert objects concurrently
  - In room scheduler, only non-conflicting inserts are allowed.

Synchronization model

- Users submit operations in transactions
- Operations are validated w.r.t. concurrent operations
- Schedules (interleaved transactions)

Consistency requirements & criteria

- Consistency requirements:
  - specify the set of ideally allowable schedules.
  - “Users may concurrently add room reservations (that don’t overlap), but may not concurrently change the same reservation.”
- Consistency criteria:
  - specify the set of actually allowed schedules.
  - “Users must access the set of reservations one at a time.”
Optimal synchronization

- The synchronization system’s consistency criteria match the application’s consistency requirements.

Traditional criteria: **serializability**

- Concurrent transactions execute as if they were submitted one after the other.

Programmer-specified criteria

- Programmers should be able to specify the consistency criteria used for their application

User-specified consistency criteria

- Users cannot use low-level mechanisms (code, predicates)
- ...but can use high-level mechanisms (tables, buttons)

Mobile collaboration: cure in addition to prevention

- Users may not be at all times connected. Some may travel and modify the drawings when disconnected.

Implications: (2) late and early validation

- All users may be connected
- All users may be disconnected (mobile)
- Mixed
Early validation
- Operations are validated w.r.t. concurrent operations at time of submission
- Examples: locking, multiversion timestamp ordering

Late validation
- Operations are validated w.r.t. concurrent operations at commit time
- Examples: optimistic concurrency control, copy-modify-merge software development

Early validation vs. late validation

Early validation
- Per-operation overhead
- Conflict blocks only single operation
- For tight coupling, with safety

Late validation
- No per-operation overhead
- Conflict may cause loss of all operations
- For parallel development of alternatives

Early validation vs. late validation

Early validation
- Per-operation overhead
- Conflict blocks only single operation
- For tight coupling, with safety
- Not functional if network is down

Late validation
- No per-operation overhead
- Conflict may cause loss of all operations
- For parallel development of alternatives
- Functional when disconnected
Conclusion: need flexibility!

- Programmer-specified consistency criteria
- Early and late validation
- And many others...

A simple solution to flexibility:

- Programmers write the synchronization code themselves
  - Rover (late validation)
  - Prospero (early validation)

Our goal: a balance

Incremental Specification

- Support varying levels of “synchronization awareness” in applications
  - no awareness: inherit default policies
  - medium-level awareness: specify policy
  - high-level: implement policy
- Awareness can be incrementally increased
- Constraint: Co-exist with existing infrastructure

Incremental Specification

- Policy inheritance
- Policy specification
- Policy implementation
- Merge- and locking-aware types
- Merge matrices and lock tables
- Subclassing, new types

Scenarios and requirements

- Requirements varied from application to application
- Users added their own requirements
- Groups included both connected and disconnected users
- Specification was incremental
- Programmer-specified consistency criteria
- User-specified consistency criteria
- Early and late validation
- Incremental Specification
Three aspects of solution

- Merging
- Locking
- Integration (coexistence)

Co-existing with Existing Infrastructure

- Suite Collaboration System
  - Suite Merge Tool
    - Sharing of C Types
    - Assumes Connected Operation
    - Provides and Assumes C-based User-Interface Generator
  - Java + “MVC”

Suite Merge Tool

- Users work without coupling between their views, and then synchronize before committing
- Interactive selection of alternatives
- Merge policy selected at merge time, by users

Sync framework

- Uses asynchronous merging to serve disconnected, mobile users
- No predefined user interface, so may extend to applications with arbitrary user interfaces
- Basis in object-oriented programming language (Java), more extendible than Suite’s C basis—method overriding, user-defined abstract data types
- Fully automated merging.

Synchronous Vs Asynchronous Merging

Connected Merging in Suite

- Assumes connected system
- Users must “meet” to merge
Disconnected Merging in Sync

Client
- User Interface
- object methods
- Sync Replication Client

Server
- External Resources
- Replicated Object
- Sync Replication Server

Other clients

Merge process—overview
- Create change sets
- Merge them
- Apply results

Change(X→X)

Merge Algorithm

Change(X→X')

Merge Algorithm

Change(X→Y)

Type-Based Merge Algorithm
- Mergeable versions of
  - Record
  - Sequence
  - Dictionary
  - Integer, Float, String
- Shared data defined by hierarchical composition of these types
Composition of a drawing

Merge matrix—overview

- Table that specifies outcome of one merge decision
- Entries of table specified by programmers and users

Merge matrix—general form

- Rows represent one change set, columns represent other change set.
- Matrix entries (merge actions) specify how conflicts between operations are resolved.

<table>
<thead>
<tr>
<th>Type</th>
<th>Op 1</th>
<th>Op 2</th>
<th>Op 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Op 1</td>
<td>Users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Op 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Op 3</td>
<td>Take both</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Merge action—general form

- General form of matrix action is a function $O_1, O_2 \rightarrow A_1, A_2$

<table>
<thead>
<tr>
<th>Name</th>
<th>Action</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>AcceptBoth</td>
<td>$&lt;OP_C, OP_R&gt;$</td>
<td>No conflict</td>
</tr>
<tr>
<td>SameChanges</td>
<td>$\emptyset, \emptyset$</td>
<td>Identical operations</td>
</tr>
<tr>
<td>AcceptRow</td>
<td>$\emptyset, OP_R$</td>
<td>Row operation only</td>
</tr>
<tr>
<td>AcceptColumn</td>
<td>$OP_C, \emptyset$</td>
<td>Column operation only</td>
</tr>
<tr>
<td>Resolve</td>
<td>$\text{resolve}(OP_C, OP_R)$</td>
<td>Externally resolution</td>
</tr>
<tr>
<td>Merge</td>
<td>$\text{merge}(OP_C, OP_R)$</td>
<td>Merge at lower level</td>
</tr>
</tbody>
</table>

Merge matrix—Atomic, Record

<table>
<thead>
<tr>
<th>Atomic</th>
<th>Modify</th>
<th>Null</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modify</td>
<td>Resolve</td>
<td>Row</td>
</tr>
<tr>
<td>Null</td>
<td>Column</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Record</th>
<th>Modify(field)</th>
<th>Null</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modify(field)</td>
<td>Merge</td>
<td>AcceptRow</td>
</tr>
<tr>
<td>Null</td>
<td>AcceptColumn</td>
<td></td>
</tr>
</tbody>
</table>
Merge matrix—Sequence

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Insert(#)</th>
<th>Delete(#)</th>
<th>Modify(#)</th>
<th>Null</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert(#)</td>
<td>Both</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delete(#)</td>
<td>Same</td>
<td>Column</td>
<td>Row</td>
<td></td>
</tr>
<tr>
<td>Modify(#)</td>
<td>Row</td>
<td>Merge</td>
<td>Row</td>
<td></td>
</tr>
<tr>
<td>Null</td>
<td>Column</td>
<td>Column</td>
<td>Column</td>
<td></td>
</tr>
</tbody>
</table>

Multiple merge policies

Consolidation

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Insert(#)</th>
<th>Delete(#)</th>
<th>Modify(#)</th>
<th>Null</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert(#)</td>
<td>Both</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delete(#)</td>
<td>Same</td>
<td>Column</td>
<td>Row</td>
<td></td>
</tr>
<tr>
<td>Modify(#)</td>
<td>Row</td>
<td>Merge</td>
<td>Row</td>
<td></td>
</tr>
<tr>
<td>Null</td>
<td>Column</td>
<td>Column</td>
<td>Column</td>
<td></td>
</tr>
</tbody>
</table>

Reconciliation

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Insert(#)</th>
<th>Delete(#)</th>
<th>Modify(#)</th>
<th>Null</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert(#)</td>
<td>Users</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delete(#)</td>
<td>Users</td>
<td>Users</td>
<td>Users</td>
<td>Users</td>
</tr>
<tr>
<td>Modify(#)</td>
<td>Users</td>
<td>Merge</td>
<td>Row</td>
<td></td>
</tr>
<tr>
<td>Null</td>
<td>Column</td>
<td>Users</td>
<td>Column</td>
<td></td>
</tr>
</tbody>
</table>

Change sets

- Not linear logs
- Structured, mirroring structure of data
- more efficient access
- automatic compaction
- computed as changes are made
- operations call setChanged()

Planner Application

Change sets

Temporary wall here
Pair up corresponding changes from change sets.

For each pair:
- Look up action in merge matrix.
- If action is “merge” call merge procedure recursively on changed structure, else perform indicated action.

<table>
<thead>
<tr>
<th>Record</th>
<th>Modify(field)</th>
<th>Null</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modify(field)</td>
<td>Merge</td>
<td>AcceptRow</td>
</tr>
<tr>
<td>Null</td>
<td>AcceptColumn</td>
<td></td>
</tr>
</tbody>
</table>
**Merge algorithm**

**Synchronization in Sync**

1. `synchronize()`
2. `C_1 = getChange()`
3. `synchronizeRequest(C_1)`
4. `C_0 = changes since last synchronization`
5. `<S,C> = mergeChanges(C_0, C_1)`
6. `applyChange(S)`
7. `(3's reply)`
8. `applyChange(C)`

**Sync classes**

<table>
<thead>
<tr>
<th>Sync class</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replicated</td>
<td>Abstract class inherited by all replicated classes</td>
</tr>
<tr>
<td>ReplicatedRecord</td>
<td>Abstract class for programmer-defined classes with replicated fields</td>
</tr>
<tr>
<td>ReplicatedAtomic</td>
<td>For programmer-defined classes without replicated fields</td>
</tr>
<tr>
<td>ReplicatedSequence</td>
<td>Implements subset of java.util.Vector</td>
</tr>
<tr>
<td>ReplicatedDictionary</td>
<td>Implements java.util.Dictionary, subset of java.util.HashMap</td>
</tr>
</tbody>
</table>

**Large policy space**

- Merge matrices defined at each structural level of shared data
- Structure × merge matrices = policy space

**Subclassing, new types**

- define new merge action
- create new merge matrix
- define new change types
- override getChanges
- override applyChanges
## Solution components
- Policy inheritance
- Policy specification
- Policy implementation
- Early and late validation
- Merge- and locking-aware types
- Merge matrices and lock tables
- Subclassing, new types
- Merge and lock algorithms, systems

## Sync applications—bibliographic database
- Maintains collection of bibliographic references
- Insertions with same key should be accepted
- Used to demonstrate Bayou and Prospero systems
- Implemented with Dictionary, special merge action

## Sync applications—conference room scheduler
- Used for reserving conference rooms remotely
- Automatically blocks conflicting reservations
- Used to demonstrate Bayou and Rover systems
- Implemented with Dictionary

## Sync applications—Issue-Based Information System (IBIS)
- Synchronization system must prevent one user from deleting an item another is responding to
- Used to demonstrate flexibility of NT/PV model
- Implemented with Record and Sequence

## Experience
- Several applications to demonstrate flexibility and ease of specification of Suite and Sync

<table>
<thead>
<tr>
<th>Application</th>
<th>Source</th>
<th>Lines of code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yacc editor</td>
<td>Myself</td>
<td>0</td>
</tr>
<tr>
<td>Planner</td>
<td>Myself</td>
<td>2</td>
</tr>
<tr>
<td>Bibliographic database</td>
<td>Bayou, Prospero</td>
<td>13</td>
</tr>
<tr>
<td>Room scheduler</td>
<td>Bayou, Rover</td>
<td>1</td>
</tr>
<tr>
<td>gIBIS</td>
<td>NT/PV</td>
<td>0</td>
</tr>
</tbody>
</table>

- 2000+ lines of code in Sync
**Conclusion**

- Developed a new model that fulfills more of the synchronization requirements than existing systems.
- Model provides balance between flexibility and ease of specification
- Model used to implement five diverse applications
- Can simulate almost all other systems

**Future work**

- More flexibility for merge matrix
- Unify merge matrices and lock tables
- Partial-object replication
- Implement for WWWNG

**More flexibility for merge matrix**

- Current form integrates conflict definition and conflict resolution
- Restricts range of policies

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Insert(¥)</th>
<th>Delete(¥)</th>
<th>Modify(¥)</th>
<th>Null</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert(¥)</td>
<td>Both</td>
<td></td>
<td></td>
<td>Row</td>
</tr>
<tr>
<td>Delete(¥)</td>
<td>Same</td>
<td>Column</td>
<td>Row</td>
<td></td>
</tr>
<tr>
<td>Modify(¥)</td>
<td>Row</td>
<td>Merge</td>
<td>Row</td>
<td></td>
</tr>
<tr>
<td>Null</td>
<td>Column</td>
<td>Column</td>
<td>Column</td>
<td></td>
</tr>
</tbody>
</table>

**Future work**

- More flexibility for merge matrix
- Unify merge matrices and lock tables
- Partial-object replication
- Implement for WWWNG

**Partial-object replication**

- Replicated Application data
- Partial replication

**Future work**

- More flexibility for merge matrix
- Unify merge matrices and lock tables
- More flexible merge protocol, to utilize one-way messaging services
- Partial replication
- Implement for WWWNG