Lifecycle Management of Relational Records for External Auditing and Regulatory Compliance

Ahmed Ataullah
Frank Wm. Tompa
What is a business policy?

- A specification of what should (or should not) happen in the operations of a business.
  - Typically written in natural language

Policies manifest themselves in database systems as constraints or alerts

- Check constraints
- Transaction termination triggers
- Access control restrictions
Business Model

- Means of simplifying and summarizing a set of rules
  - Provides a high-level overview
  - Intermediate layer between natural language and implementation (code)
  - Logical or graphical

- Types of modeling:
  - Data modeling: ER, UML, ...
  - Process modeling: Workflow, Flow-chart, PERT charts, ...
  - Policy modeling: Hierarchical access control modeling, Object Constraint Language, ...
Database System Perspective

- Database reflects enterprise
  - Database instance ≡ overall ‘state of a business’
- Instance must comply with published policy
  - Database system must continuously monitor updates to ensure compliance.
  - Single compliance layer eliminates need to embed policy checking logic in every application
  - Significantly simplifies implementation of complex data constraints
Difficulties

- Each department has its own constraints
  - Shared database ⇒ conflicts can be detected beforehand
  - Many are complex (temporal, conditional, path oriented)
    - “A student can take CS446 only if she has passed CS330 and CS245 with a score of 75% or more”
    - Typically correspond to complex transaction termination triggers

- Many constraints derived from business policies
  - Scale ⇒ manageability a major problem for DB administrators and programmers
Why have existing models failed us?

- Complex (temporal, path) constraints difficult to model.
  - “A professor cannot teach CS448 and CS490 at the same time.”
  - “A professor can teach CS348 only once in two years.”
- No notion of policy-relevant object
- State of an “object” is static
  - Historical applications are “forgotten”
  - “Salary of an employee can only be increased three times in any one year period.”
- Inter-rule dependencies cannot be expressed
  - For example: “Rule 1 should only be enforced if Rule 2 was never violated in the past by a transaction”
Avoiding Hand-written Triggers

- Can we make the task of the programmer easier?
- Can we make the database level workflows more transparent to business level managers?
- Can we introduce policy-to-constraint transparency and manageability and offer compliance guarantees at the same time?
Step 1

Basic Framework: Objects

- Start with the object definitions
  - An object definition is *any relational view* over a fixed database schema.

- Object = tuple (row) in such a view
  - View can represent a complete business artifact
    - *All* data needed for policy making
  - Assume each row in view is uniquely identifiable
Step 2

Basic Framework: States

- State S: condition on objects in the view definition
  - Object x is in state S ⇔ its attributes satisfy S(x)
  - Example
    - EXPENSE_CLAIMDETAILS is user-defined view
    - “Claim objects” are rows in the view
    - Object O (tuple t) is in state P, the “paid state”, if the condition EXPENSE_CLAIMDETAILS.PAID = TRUE for t

- An object can be in multiple states at once.
Step 3: Convert business model into enforcement model

- Rectangles represent business states
- Transitions represent processes/actions
- Stick-figures represent agents
- Constraints implied by absence of transitions
  - E.g., Paid invoices are not deleted
States of an object are typically an interpretation of stages in business process.

- Including object creation and destruction

- All constraints should be made explicit.
Constraints on State Transitions

\[ \bullet A(x) \land \neg A(x) \Rightarrow B(x) \]

\[ \neg \bullet B(x) \land B(x) \Rightarrow \bullet A(x) \]

\[ \lozenge A(x) \Rightarrow \neg B(x) \]

- Define various types of constraint
  - Use past temporal logic
  - Create diagrammatic form

- Special interpretations for transitions to or from \( \Phi \)
- Convert to temporal logic

\[
\begin{align*}
\text{New}(x) & \Rightarrow \text{AwaitingApp}(x) \\
\neg \diamond \text{AwaitingPay}(x) \land \text{AwaitingPay}(x) & \Rightarrow \diamond \text{AwaitingApp}(x) \\
\neg \diamond \text{Paid}(x) \land \text{Paid}(x) & \Rightarrow \diamond \text{AwaitingPay}(x) \\
\text{Paid}(x) & \Rightarrow \text{Retain}(x)
\end{align*}
\]

- Test these conditions in the database
States = \{\text{Awaiting Approval}, \\
\text{Awaiting Payment}, \\
\text{Paid}\}

State space = \{(0,0,0),(0,0,1),(0,1,0),\ldots,(1,1,1)\}

Some configurations are not satisfiable
- **A complete, temporally ordered** list of state configurations for an object

<table>
<thead>
<tr>
<th>Time</th>
<th>Awaiting Approval</th>
<th>Awaiting Payment</th>
<th>Paid</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t3</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>t4</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Implementation: Tracking the State Configuration History

<table>
<thead>
<tr>
<th>Time</th>
<th>Awaiting Approval</th>
<th>Awaiting Payment</th>
<th>Paid</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t3</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>t_new</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- Every time an object is modified, look back at the history and answer the query related to each constraint!
Allow Multiple Constraint Diagrams

- Model business processes as constraint diagrams

- Convert to temporal logic implemented as triggers

\[- \exists x \in R_1 : \Diamond \text{Paid}(x) \land \text{UnderReview}(x)\]

\[- \exists x \in R_1 : \neg \Diamond \text{AwaitingPayment}(x) \land \text{AwaitingPayment}(x) \land\]

\[- \Diamond \text{UnderReview}(x)\]
Benefits: Separation of Duties

- Instead of a single large complex workflow

- Each department, person, or stakeholder can independently establish state oriented path restrictions
Multi-user lifecycles

- Each colour represents states of interest for a different party
Allow Complex Path Constraints

- E.g., an object should never reach state C if it has previously transitioned from A to B

\begin{align*}
\text{Constraint 1} & : \text{B}(r) \land \neg \Box \text{B}(r) \Rightarrow \Diamond \text{A}(r) \\
\text{Constraint 2} & : \Diamond 1(x) \Rightarrow \neg \text{C}(x) \\
& \quad \Diamond (\text{B}(r) \land \neg \Box \text{B}(r) \Rightarrow \Diamond \text{A}(r)) \Rightarrow \neg \text{C}(x)
\end{align*}
Policy designer only needs to list the “states of interest”
- By specifying the conditions that the object must satisfy to be in those states
- Each policy designer in the organization can list his/her own states of interest

Policy restricts paths that an object can (or should) traverse
- Constraint diagram (the model)
  - Some paths must always be taken, some must never be traversed, and others can be conditionally traversed
- Need flexibility in specifying constraints
Define business records as database views

Invoice = (INV_ID, CREATE_DATE, PAID, AMOUNT, PAID_DATE, APPROVED)

- PAST-TL constraint is a restriction on current state of a view given its history

Augment DB audit trail with state vector

\[ x_t = (\text{timestamp}, a_1, a_2, \ldots, a_n, \text{user}, \text{user\_group}, \text{purpose}, \text{application\_context}, \text{transaction\_type}, \text{txn\_starttime}, \ldots, s_1, s_2, s_3, \ldots, s_k) \]

- Check all constraints on newly entered states
Extending the model

- Define additional transition constraint types
- Include temporal access control
  - Use transactional meta data
  - E.g., claim can only be approved by someone in the “management” user group and can only be paid by someone in the “finance” user group

<table>
<thead>
<tr>
<th>Time</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>User_Group = Management</th>
<th>User_Group = Finance</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>t_new</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Performance Analysis: Storage Overhead

- Additional storage space required to store the current (extended) state configuration
- 1 bit for each state
  - 256 states = 32 bytes or characters
- For business scenarios where large text fields are logged, the additional overhead will be minimal
Performance Analysis: Computational Costs

- CPU time spent computing the (extended) state configuration history
  - One check per state to establish membership
  - Proper nesting of these checks can minimize computation costs
- Cost is insignificant compared to writing audit entry onto disk
Conclusions and Future Work

- Bridge between business policy manager and database
  - Constraint diagram makes policy statements explicit
  - Emphasizes data but models processes and user interactions
  - Allows multiple diagrams and multiple concurrent states
  - Equivalent to past temporal logic
  - Implementable as database triggers on meaningful views
  - Experiments show that extra space and time to check constraints is insignificant overhead on auditing.

- Future work
  - Translating NLP policies into constraint diagrams
  - Extracting workflows from database transaction logs