September 4

Paper Presentations

- Assume everyone has read the papers
- Don't just present the content
  - especially not section-by-section
- Provide
  - analysis
  - commentary
  - criticism
  - suggestions
- Connect the ideas together
- *Suggest projects*

Orthogonal Persistent Object Systems
Atkinson & Morrison

Concurrency – the Fly in the Ointment?
Blackburn & Zigman

William Cook

September 4, 2003

Tennent's Principles

Correspondence

int x = 3;	void foo(int x) {…} foo(3)
typedef T int;	template <typenameT> void foo(T x)… foo<int>(3)
SQL is lacking in abstraction constructs (e.g parameter passing)	

Abstraction

statement	function
exception	??? (C++ stack-based objects help)

Data Type Completeness

Any type can be member of a struct, union.
SQL: Should a table be allowed as a value in a table?

Persistence Design Rules

- Parsimony
  - match the application domain
  - absence of extra features ensures simplicity
- Orthogonality
  - composed of atoms
  - combined to form appropriate abstractions
- Computationally complete
  - no need for anything else
  - prevents disharmony
- Put them together…
  - "combination of principles yields integrated persistent systems"

Contrast With

- Compression
  - Meaning depends upon context… gives richer meanings (e.g. poetry)
- Expressiveness
  - Almost all languages are computational complete in theory (including C++ templates)
  - No good measure of language expressiveness
  - What is difference?
    - Computation: availability of effects
    - Expressiveness: availability of notation

Principles

- Orthogonal Persistence
  - Persistence Independence
    - Programs look the same whether they manipulate short-term or long-term data
  - Data Type Orthogonality
    - All data objects are allowed full range of persistence. No special cases for lifetime
  - Persistence Identification
    - The mechanism for identifying persistent objects is not related to the type system

Persistence Independence

- Definition
  - Programs look the same whether they manipulate short-term or long-term data
- Example
  - Must work the same when person, company are persistent or not
  - for person in company.employees()
    print person.name
  - function Certify(x : Company)
    - x.certified = true
    - x.date = today()

Persistence Independence

- Definition
  - Programs look the same whether they manipulate short-term or long-term data
- Performance
  - "sorting an array of objects"
  - Algorithm cannot know about paging, etc
  - To succeed, a languages must have a simple performance model
    [Worse is Better, R. Gabriel]
  - Note: sorting is a primitive in SQL!

Persistence Independence

- Definition
  - Programs look the same whether they manipulate short-term or long-term data
- Concurrency & Transactions
  - Program must indicate transaction boundaries when dealing with persistent data
  - If data can be concurrently accessed, programmer must deal with transaction abort
  - *More later*

Data Type Orthogonality

- Definition
  - All data objects are allowed full range of persistence
- Question
  - This is true in most systems
    - anything can be written out
  - This is important when persistence is automatic
  - Extreme view would require persistence for
    - Processes, threads, TCP connections
    - Transactions, Iterators, Windows

Persistence Identification

- Definition
  - The mechanism for identifying persistent objects is not related to the type system
- Key question:
  - How are persistent objects identified?
    - In this case, by *reachability*, garbage collection
- On the other hand:
  - How do you identify what *not to persist*
  - Accumulation of garbage…

Garbage Retention

- Non-persistent parts of persistent objects
  - Caches
  - Bookkeeping for specific algorithms
  - Callbacks
- These are all memory leaks
- Use *weak pointers*?
  - Requires programmer to identify boundaries of persistence
- How is it done in databases?
  - Explicit delete; *cascading* delete
  - Consistency rules (foreign key constraints)

Contents

- Principles
- Integration Concepts
  - Type Systems
  - Binding
  - Concurrency
- Technology to support Persistence
  - Reachability
  - LinguisticReflection
- Extensions

Data Models & Types

Databases	Programming Languages
data model's	type systems
schema	type extension
database	variable
database extent	value

Interesting Points

- Store contains behavior and state
  - Advocates putting code (methods) into the database along with everything else
- Heavy focus on type systems
  - Polymorphism, but also "Any" type
  - Compare to SQL
    - Very basic user-visible types
    - Complexity of typing join operator is invisible
- Hyper-code
  - Similar to
    - Intentional Software – C. Simonyi
    - Syntax-directed editing
  - Are there any benefits?

"Extensions"

- Conceptual Modeling
- Bulk Types, Queries, Optimization
- Transactions and Concurrency
- Scalable Systems
- Evolution
- Integration with other domains
  - If these are the extensions, what do we get in the base?

Conceptual Modeling

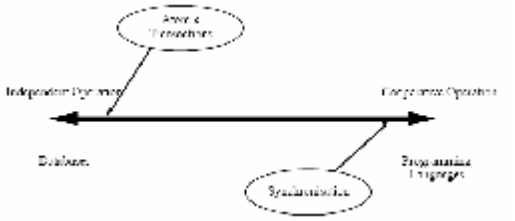
- Data Modeling
  - Type systems versus Entity-Relational Model
- Need for "*mappings*"
  - Is persistent system not computationally complete (or representationally complete)
- Raises question:
  - …benefits to be gained by generated code from many high-level notations into a common orthogonally persistent system

Bulk Types, Queries, Optimization

- Good citations
  - We will be reading some of the papers they mention in a few days
- Import this work directly
  - Will there be any unforeseen interactions?
  - As far as I know, this is still untested.

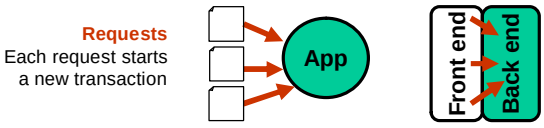
Transactions and Concurrency

- Concurrency – the fly in the ointment
  - has very good discussion of issues



Transactions and Concurrency

- Where do transactions come from?
  - If everything is inside a transaction, who can create a transaction?
    - Need something external that makes requests which are processed transactionally.
  - What are natural transaction boundaries?
    - In web applications?
    - In GUI applications?
    - In a file server?



Transactions and Concurrency

- A proposal
  - Chain-and-spawn method:
    - Transaction ends and another one starts
    - AAAAA,BBBBBB,CCCCCC,DDDDDD
  - Imposes structure on implementation
    - So perhaps not really orthogonal
  - Not clear how this works in multi-threaded application

Transactions and Concurrency

- Two transactions accessing same *object*
  - How do you know what operation they will perform?
  - Both read, or will either of them write?
- Approaches to Isolation:
  - 1) Copying
    - Each transaction have a copy, in case one writes
    - What if both update their copy of the object?
    - How will the resulting changed be merged?
  - 2) Locking
    - Only one transaction at a time can access the object
  - 3) Distinguishing reads/write methods
    - Difficult to do for general OOP

Evolution

- Mentioned several times
  - Doesn't really present a convincing explanation of how maintenance works
  - Reflection is only a small part of solution
  - Suggestion of need for "change absorbers" is interesting
  - "If you get a bunch of programmers together, they will usually start talking about source control"

Evolution

- Software release process
  - R1: Release 1.0
  - R1x: Customized version
  - S: Configured running system based on R1x
  - R2: New Release 2.0
  - Problem: Merge R2 into S
  - Test result in staging environment
  - Deploy to production
  - (repeat as quickly as possible!)
- Closer to reality…

Evolution

- Deployment to Production
  - Developer does not have copy of real system
  - New behavior must be *staged*
    - tried out with copy of live data…
    - but configured so to not affect environment!
  - Updates must be applied to running system?
- Merge affects:
  - behavior, data, configuration, structure
  - each part has a different *master/owner*
  - ensure that changes are only made to master of each definition, then propagate change consistently everywhere

My Questions

- Scalability via Distribution
- Performance of Incremental Loading
- Multiple Programs

Scalability via Distribution

- Load-balancing Multiple machines
  - Load is distributed across machines
    - Scalability
    - Availability
  - Use of shared resources must be controlled
- Problems
  - Cache coherency
    - ensuring that changes on multiple machines are consistent
  - Locking
    - Distributed transactions

Performance Issues

- Incremental Loading: High latency
  - Objects are loaded on demand
    - p = root.FindPerson("william")
    - d = p.getDepartment()
    - m = d.getManager()
    - for p in m.getProjects()
      - write p.getName()
  - Each object is loaded individually
  - Each project is loaded individually as well!

Performance Issues II

- Made worse if
  - machines are load-balanced
  - data set size > cache size
  - transactions require copies
- Can you optimize if you know total set of objects to be loaded?

Multiple Programs

- How do you write multiple programs that work on same data?
  - Information is used in different ways
    - Object model may be different
  - Create multiple views of data
    - For analysis
    - For updates
    - For derivation/transformation
  - I must be missing something, but I just don't get it

Vision

- Integrating different languages & subsystems makes programming harder
  - Eliminate disparate sub-systems
  - Programs, Operating Systems, Database, User Interface Management System
- Define *single unifying model*
  - Persistent Object Systems
  - "One coherent design" or "Closed world"
- Is specialization always bad?
  - What about "best of breed"
  - Specialized solutions for different program aspects may be better than generic model

Claimed Benefits

- Improving *programming productivity* from simpler semantics
- *Avoiding ad hoc arrangements* for data translation and long-term data storage
- Providing *protection mechanisms* over the whole environment
- Supporting *incremental evolution*
- Automatically preserving referential integrity over the *entire computational* environment for the whole life-time of a PAS

Other Claims

- Constructing Persistent systems is made considerable easier when the whole computational environment is persistent.
- Statically check program have better documentation properties and better cost properties throughout the life cycle of programs.
- Many claims in paper are not supported

Project ideas

- Use Landin's language principles to design an incremental improvement to SQL
- Conduct and experiment using PJama to evaluate evolution, performance…

Discussion