INTRODUCING COREOBJECT

FOSDEM 2012

http://www.etoileos.com

Donald Norman

People will make errors, so make the system insensitive to them

Bruce Tognazzini

Effective interfaces are visually apparent and forgiving, instilling in their users a sense of control.

Effective interfaces do not concern the user with the inner workings of the system. Work is carefully and continuously saved, with full option for the user to undo any activity at any time.

Explorable UI

- From first principles of Interaction Design
  - Make Actions reversible
  - Always allow Undo
  - Always allow a way out
  - However, make it easier to stay in

Motivation

- You cannot lost data by mistake
- No wait cursor
- No modal dialogs
- No application management (no quit, no launch etc.)
- Almost all actions are undoable (all destructive actions are undoable)

Selective Undo

- Almost all actions are undoable and…
  - Undo is not restricted to the last action
    - but is allowed on each action individually
- Selective Undo is required to
  - resolve collaborative editing conflicts by reordering the actions
  - support multiple undo granularity (track)

Fake Undo

Users quickly formed the habit of emptying the trashcan as soon as the first document hit. This not only turned a single-step operation into a two-step operation (drag to the trash, then empty the trash), it negated the entire power of the trashcan, namely, undo.

⁂

Bruce Tognazzini

Object Resurrection

- No Trashcan… Just use selective Undo
  - Object can be deleted directly
    - but resurrected at any time (if not yet garbage collected)
- CoreObject is designed to make Undo pervasive, uniform, transparent and selective

You are too late?

- Apple has done it in Mac OS 10.7…
  - no selective undo or collaboration
  - burden and complexity shifted to the developer
  - no data sharing

Object Graph Diffing & Merging

Goals

- Selective Undo support
- Simpler implementation
- More robust
  - Catch more serialization logic issues
  - Object graph integrity checks
  - Prevent deterministic replay mistakes

Persistency

- New persistency approach based on
  - metamodel-driven serialization
  - object graph diffing
  - rather than message recording

Persistency

- At commit time
  - Object graph diff computed by reducing changes to primitive operations
- History is a revision sequence

Primitive Operations

- Property update
- Set add and remove
- Sequence insert and remove
    - Strings, Arrays etc. are sequences

⁂

Based on Difference and Union of Models paper

New Merging Model

- Object Graph Diff and Patch
- We don't use
  - Operational Transformations (OT)
  - Address Space Transformation (AST)
  - or some hybrid models (such as OT with tombstones)

Existing Merging Models

- OT are a proved model, but
  - slow (commutation)
  - complex (transpose)
- AST is simpler, but
  - requires the entire history in memory :-/

Undo in less than minute

- Nobody has ever built a revision control system based on OT or AST
- Merging 10000 operations can take minutes to hours with OT

Where is the challenge?

- Core Object is not just a collaboration system like Gobi, Google Wave etc.
- It's a revision control system
- A core object history could be as huge as
  - 500 000 commits
- Yet selective undo must be immediate

Framework Birdview

Low-level

Diffing and Merging

Store

High-level

Model Objects

Tracks

Editing

Querying

Collaboration

Track Model

Tracks

- Track is the central CoreObject construct
  - represents a revision sequence
    - in the store
    - in the high-level track model
  - includes a undo/redo pointer on the active revision

Store Tracks

- Commit Track used to represent
  - persistent root
  - branch
  - copy (cheap)
- Custom Track
  - user-built revision sequence

High-level Track Model

- Commit Track
- Custom Track
- History Track
  - A dynamically built revision sequence by aggregating interleaved revisions from multiple commit tracks

Undo Example

```
[calendar addAppointment: appt]; [ctx commit];

[calendar setName: @"Personal"]; [ctx commit];

COCommitTrack *track = [calendar commitTrack];

[track undo]; // Set the name back to nil

[track redo]; // Set the name back to Personal

// Remove the appointment (selective undo with new revision)

[track undoNode:  [[track currentNode] previousNode]];
```

Track Demo

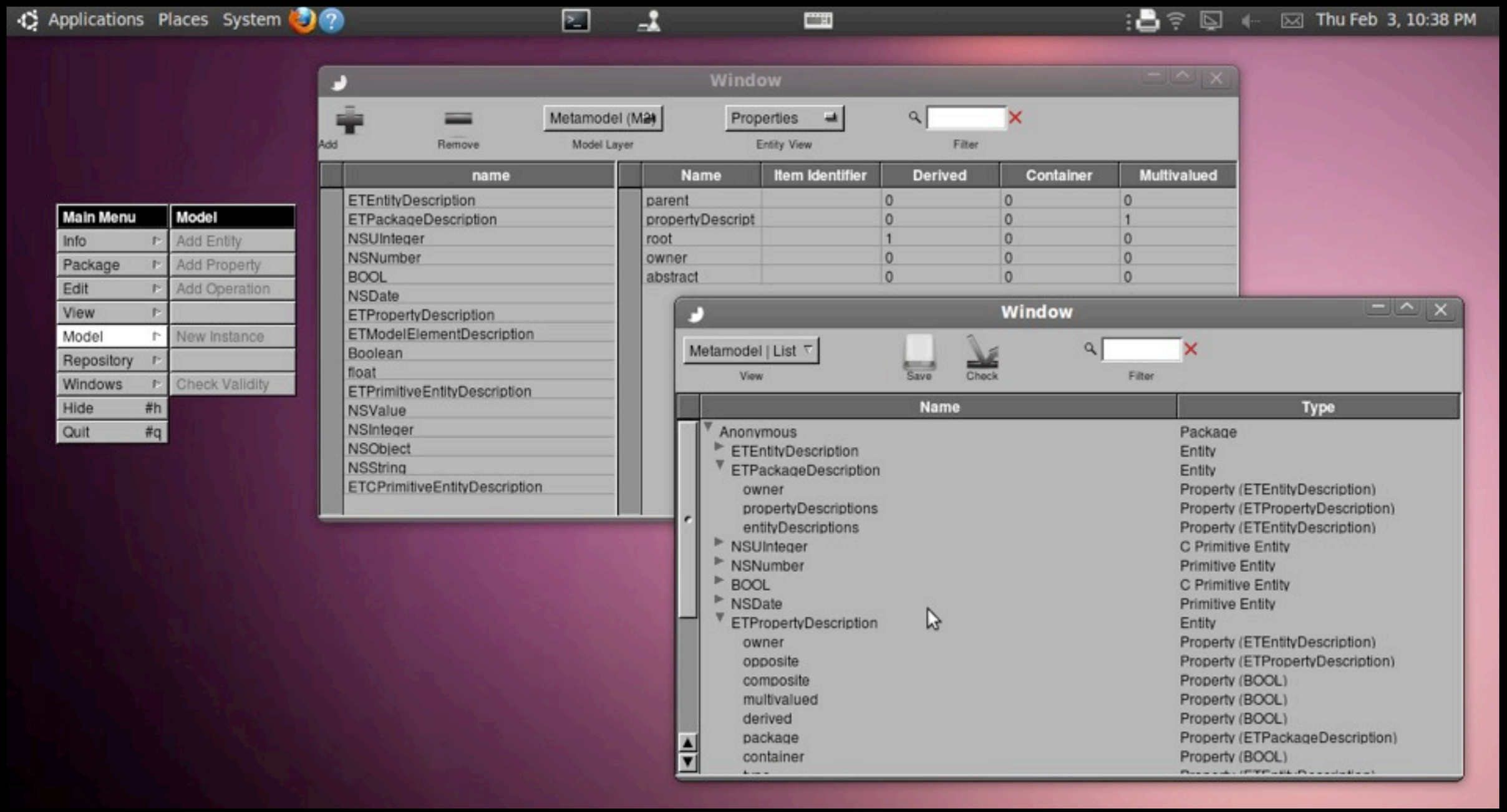
Metamodel

Metamodel

- FAME-derived Metamodel
  - EMOF (MOF subset) reduced to the max
  - e.g. Why bothers with cardinality when in practice nobody uses it
- Available in EtoileFoundation

Minimalistic

- Property, Entity and Package descriptions
- Property role stereotypes
  - number, relationship, option list etc.
- Model and Metamodel Repository
- Self-described
  - Metamodel can evolve at runtime



Model Builder

Editing a package & browsing a repository

Concurrency Control & Collaboration

Concurrency Control

- Two distributed notifications
  - on commit
  - on current track node change
- Transparently handled
  - Tracks are updated
  - Affected root objects are reloaded

Concurrent Edits

- For a single user
  - no conflicts should arise
- For multiple users editing the same root object
  - merging using selective undo can be required (not yet implemented)

For a single user, concurrent editing is sequential. Multiple applications editing the same document are not all active at the same time, but only once at a time.

How does the code look?

Class Example (1)

```
// Class declaration is optional unlike the metamodel
@interface Calendar : COObject {
    NSMutableArray *appointments;
    NSDate *today;
}
- (NSArray *)appointments;
- (NSDate *)today;
@end
```

Metamodel Example (2)

```
@implementation Calendar

+ (ETEntityDescription*)makeEntityDescription {

    ETEntityDescription *desc = [self newBasicEntityDescription];

    if ([[desc name] isEqual: [Calendar className]] == NO)

        return desc;

    // Property description declarations

    return desc;

}
```

Metamodel Example (3)

```
id today = [ETPropertyDescription descriptionWithName: @“today” type: (id)@“NSDate”]];

id appointments = [ETPropertyDescription descriptionWithName: @“appointments” type: (id)@“Appointment”];
[appointments setMultivalued:YES];
[appointments setOrdered:YES];
// Finally add the property descriptions to the entity
[desc setPropertyDescriptions: A(appointments, today)];
[[[desc propertyDescriptions] mappedCollection] setPersistent:YES];
```

Persistency Example (4)

```
COEditingContext *ctx =
    [[COEditingContext alloc] initWithURL: testStoreURL];
// Create a new root object of type Calendar
Calendar *calendar =
    [ctx insertObjectWithEntityName: @"Anonymous.Calendar"];
// Commit the changes
[ctx commit];
```

Persistency Example (5)

```
// Create a new Appointment object and attach it to the context
Appointment *appt =
        [[Appointment alloc] initWithStartDate: [NSDate date]
            endDate: endDate];
    [appt becomePersistentInContext: ctx rootObject: calendar];
    [calendar addObject: appt forProperty: @"appointments"];
    // Commit the changes
    [ctx commit];
```

Object Model & Organization

Built-in Object Model

- Object
  - UI Object (in EtoileUI)
- Collection
  - Container (weak aka parent/children relationship is a composite)
  - Group (strong)

Dedicated Collections

- Group
  - Smart Group (dynamically provided content)
  - Tag and Tag Group
- Container
  - Library (photo, music, tag etc.)

Tagging Demo

Searching

- Query based on NSPredicate
- In Store
  - Full-text query
- In Memory
  - Faster against a loaded root object subset

What’s next?

Future Work

- Schema Versioning
- Fallback Classes (Soup Access on Newton)
- Collaboration
- Import/Export (e.g. Filesystem Bridges)
- Improved Tagging Model

Future Work

- More Integrity Checking
- Garbage Collection (e.g. History Compaction)
- Performance
- Delta compression

CoreObject Team

Eric Wasylishen

Christopher Armstrong

Quentin Mathé

http://www.etoileos.com

References

- Difference and Union of Models - Marcus Alanen and Ivan Porres
- Design Rules Based on Analysis of Human Error - Donald A. Norman
- First Principles of Interaction Design - Bruce Tognazzini