
Multi-User MetaCASE

CASE tool support for co-operative work in IS design

Steven Kelly

Introduction: Bridging the CASE Gap

Current IS design practice

Several designers, each with own module; interfaces, reuse, co-ordination

Methods non-standard, change in use, method introduction often fails

State of the art in CASE

CASE tools support a fixed, standard method: too rigid

Most CASE tools file-based, no true repository: → coarse granularity

Many single user, some multi-user CASE tools, but low permeability

Bridging the gap: multi-user metaCASE

Create, use and modify methods in the same environment

Repository-based: multiple users, fine granularity, high permeability

Multiple simultaneous users, both modellers and metamodellers

Need for MetaCASE

New domains benefit from new or adapted methods

Methods benefit from tool support

Especially important in domains with electronic implementation

E.g. document management for organisations

Metodi project: Method for document structure analysis with SGML

E.g. Internet service provider's information system

Brazilian ISP, multiple designers working on IS

Traditional domains benefit from customisable automation

Documentation generation to WWW with diagrams, links etc.

Distributed system design & generation to Java, IDL etc.

E.g. automating generation from models to final executable

German consultant: UML model → Java framework → application

Needs for Multi-User CASE Teamware

Most group design work is asynchronous

Mirrors familiar pattern of work before computerisation

Leverages strengths of modularisation, black box approach

Sharing should be as transparent as possible

Details of sharing should be available if required

Low frequency of updates reflecting others' work

Sufficient: because of modularisation

Familiar: designers are used to working with most recent paper version

Non-distracting: frequent updates on screen would make work restless

Meaningful unit of work: each transaction represents the implementation of one idea or change

ArtBASE Object Store for Smalltalk

Atomic transactions

Changes only visible to others after commit

User view of data consistent throughout transaction

Optimistic & pessimistic concurrency control

Automatic optimistic concurrency guarantees correctness

Pessimistic concurrency via locks guarantees work can be committed

Seamless integration with Smalltalk

No distinction between object in program and in database

No separate language for manipulating database objects

Classes treated as objects: class changes lazily updated to instances

The price: slow at start-up, but fast after objects cached

Automatic Locking Strategies

Remove burden of locking from user

Transparent sharing: can concentrate on work not tool

Guarantees safety of user's work: not reliant on him locking correctly

Infer intentions from actions, lock before execute

Fine granularity allows most actions to attempt to lock

User can override with shift key to signal he will only read

Minimal feedback, more on request

E.g. 'OK' button greyed in property dialog, 'Info' button shows who has lock

Menu items greyed, can't move objects in diagram, status bar shows lock holder

Don't grey out individual locked objects: makes diagram confusing

Locking Strategy for CASE Teamwork

Different types of data have different requirements

Objects, relationships, properties

Conceptual graphs and their representational diagrams

Projects, i.e. collections of related graphs

Metamodels

No data is ever read-locked

Read-write conflicts are not dangerous in CASE

Simply mean you are working on the basis of the latest released version

Fine granularity of locking

Lock conceptual graph and representational diagram separately

Lock objects and their properties separately from graphs

But lock all properties together: often semantically interrelated

Locking collections in CASE

How can multiple users add elements to same collection?

‘Simultaneously’, i.e. in overlapping transactions

Locking the collection prevents others adding for too long

Traditional answer: B-trees

Implement collection as tree structure

Value added acts as key at each branch, says which branch to follow

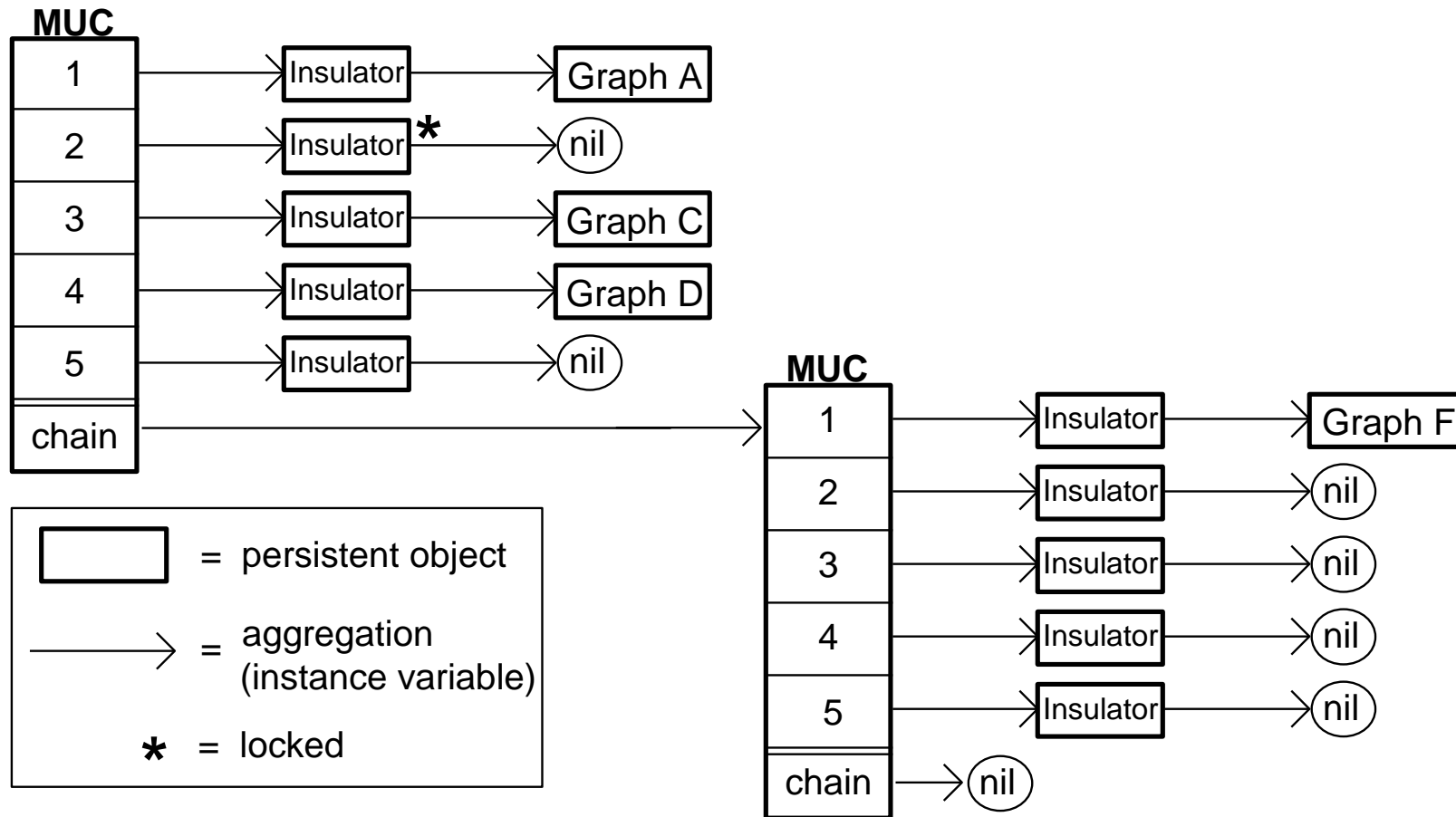
Only need to lock lowest branch on add: nothing else changes

B-tree problems in CASE for collection of graphs

No good key: only immutable part of graph is its OID, but this is sequentially assigned → poorest B-tree performance

B-trees give good permeability only when large, but graph creation frequency is highest at start of project, when collection is tiny

Solution: MultiUserColl



Locks for Metamodelling

Significantly different from modelling

Implementation level: instances update to reflect type changes

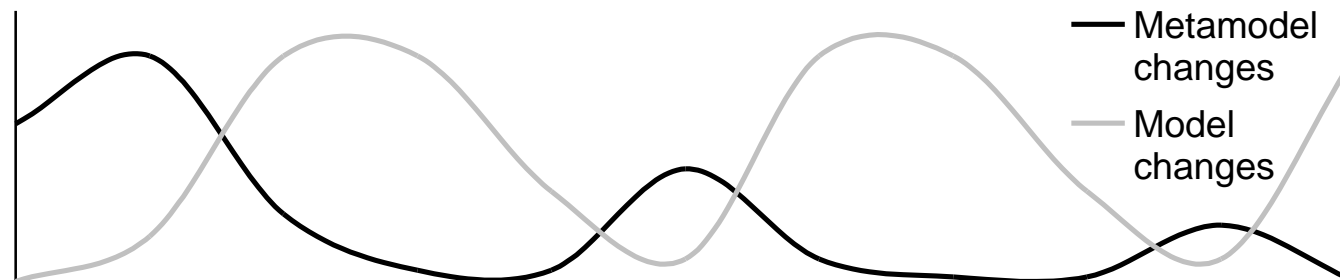
User level: method changes while working may be unsettling

Organisation level: method represents a standard, strictly controlled

Analogy with traditional database schema changes

Rare event, all users logged out, long down-time

Metamodel change intensity varies over lifecycle



Locks for Method Engineering

Scenario: multiple metamodelers build and test a method

E.g. consultants, large organisation, method developers, researchers

Multiple simultaneous schema updaters

Multiple simultaneous modellers (metamodelers build test models)

Great difference from traditional ‘schema changes’

Clearly, one automatic strategy is not enough for metamodel locks

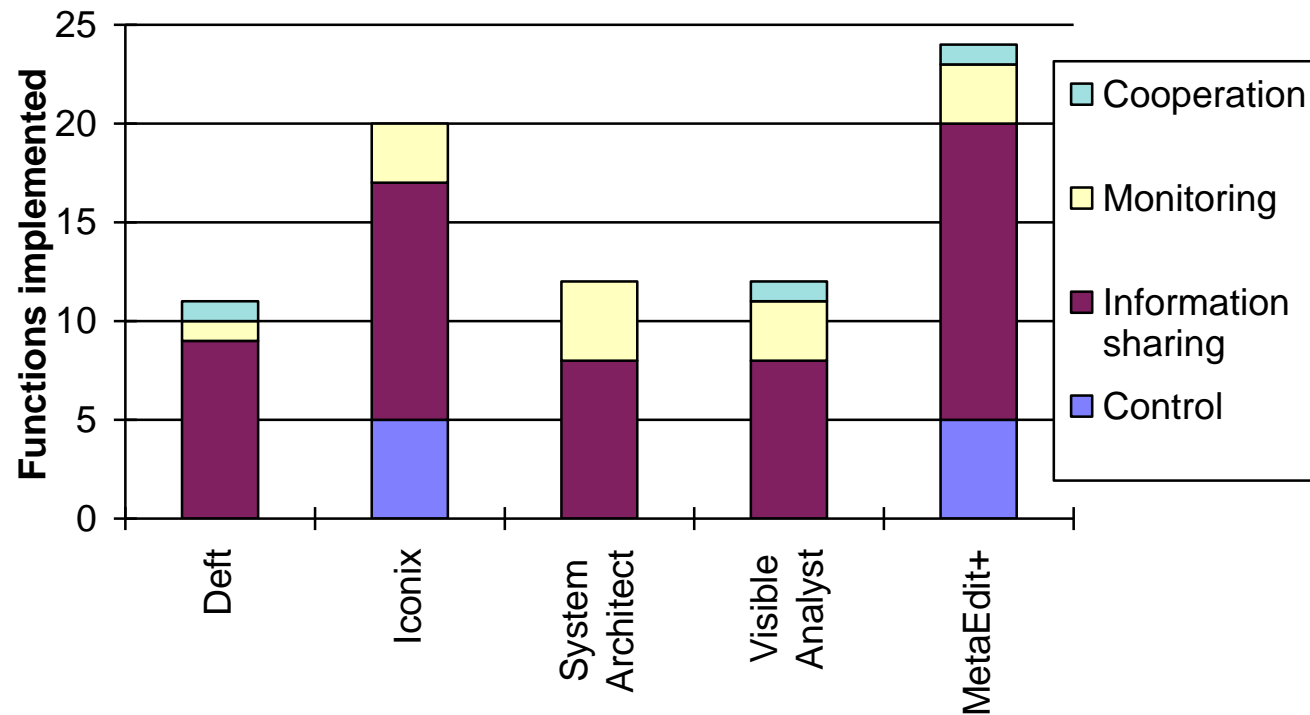
Implement several strategies, allow sysadmin to choose

1. Exclusive: one single metamodeler, no modellers
2. Single: one single metamodeler, any number of modellers
3. Project: several metamodelers (one per project), many modellers

Theoretically possible to lock individual types

But more permeability than 3. scarcely needed at this stage

Evaluation and Comparison



MetaEdit+ fares well, even though others are just CASE

Conclusions and Further Work

First multi-user metaCASE environment

Supports metamodelling alongside modelling

Supports multiple simultaneous metamodellers

Repository allows reuse, high permeability, fine granularity

Fully automatic locking: transparent to users

MultiUserColl fits CASE-specific requirements for collection data structure

Need testing with larger numbers of users

ArtBASE has been tested with 100's; MetaEdit+ only with 10 so far

Work begun on 'light' MetaEdit+ client via WWW

Allows read-only access to models & metamodels for many users

Implemented as a single MetaEdit+ client answering WWW requests