The Cha-Q Meta-Model: A Comprehensive, Change-Centric Software Representation

Coen De Roover, Christophe Scholliers, Angela Lozano, Viviane Jonckers
Javier Perez, Alessandro Murgia, Serge Demeyer
Cha-Q Meta-Model

→ to be shared by prototypes for analyzing, repeating, tracing changes

first interconnected representation of:

✓ **state & evolution** of the different entities of a software system
✓ each **change** to an entity that results in a new entity state
✓ system **snapshots** under control of a VCS

object-oriented (i.e., each concept and relation is represented by a class)

 driven by positive experience with FAMIX, RING/C/H, Cheops

memory-efficient tracking of states

 driven by poor scalability reported for Hismo and Syde
Cha-Q Meta Model: Overview & Inspiration

- **state & evolution**
  - Ring/H, FAMIX, Hismo

- **change**
  - ChEOPS, UniCase, Ring/C, Spyware, Syde

- **versioning**
  - Evolizer, STNA Cockpit
Change: Overview of Representation

- Applying a change results in a new entity state.
- Predecessor states are accessible.
- And might trigger derived changes to other entity states.
- To enable reverting, changes keep track of old state of modified or deleted entity.

PropertyChange hierarchy mirrors PropertyDescriptor hierarchy.

Insert an entity at a particular position in a list of entities.
**Change: Change Dependencies**

A change object $c_2$ depends upon another change object $c_1$ if applying $c_2$ without $c_1$ would violate invariants of the meta-model.

**Transactional Dependency:**
- Both are part of the same transaction which schedules $c_1$ before $c_2$.
- E.g., program transformations.

**Containment Dependency:**
- The subject of $c_1$ is the owner of the subject of $c_2$.
- E.g., owner of a field declaration is its declaring class.

**Type/Method/Variable Dependency:**
- $c_1$ creates a type/method/variable declaration referred to by $c_2$.
- E.g., type of a parameter declaration.
Evolution: Overview of Representation

- **Evolution**: enables tracking the evolution of an entity throughout its lifetime.
- **Snapshot**: represents the state of the system at a particular point in time, as seen by a particular developer.
- **EntityState**: accumulates the effect of one or more changes.
- **EntityIdentifiers**: each EntityState knows in which Snapshot to start looking up.
- **Subclasses**: represent system artefacts that are subject to change, of which the evolution is to be tracked.
- **Attributes and associations**: modeled after language-independent FAMIX3 meta-model.
- **PropertyDescriptor**: modeled after abstract grammar of JDT (Java) and OmniBrowser (Smalltalk).
- **EntityIdentifiers**: the values of an EntityState’s properties are EntityIdentifiers.
- **IdentifierToState**: enables tracking the evolution of an entity throughout its lifetime.
- **PropertyDescriptor**: describes the name, owner and value type of each property.
Evolution: Memory-Efficient State Tracking

copying an entity each time its state changes is expensive
for snapshot-centric Hismo model: 70min for full snapshot copy of 350MB
for change-centric Syde model: 3GB for SVN repo of 78MB

selective cloning is difficult to implement
all entities are transitively interconnected

Snapshot 2

performing a PropertyChange:
1/ clone EntityState
2/ update snapshot’s mapping from ID to clone
3/ update property value in clone

identifier-based technique used by Orion and Ring/H meta-models
Versioning & Issues: Overview of Representation

- A modification report is associated with each snapshot placed under version control.
- Revisions can be associated with a logical branch, identified by a site-specific BranchIdentifier.
- Multiple successors and predecessors accommodate physical branching and merging.
- Reports can mention and close reported issues.
- Issue-related information is subject to change, and hence represented by EntityState subclasses.
Implementation Highlight: Annotation Metadata

V1

```java
public class BreakStatement extends Statement {
    @EntityProperty(value = SimpleName.class)
    private EntityIdentifier label;

    public EntityIdentifier getLabel() {
        return label;
    }

    public void setLabel(EntityIdentifier label) {
        this.label = label;
    }
}
```

V2

```java
public class BreakStatement extends Statement {
    @EntityProperty(value = SimpleName.class)
    private EntityIdentifier<SimpleName> label;

    public EntityIdentifier<SimpleName> getLabel() {
        return label;
    }

    public void setLabel(EntityIdentifier<SimpleName> label) {
        this.label = label;
    }
}
```

Annotation for properties of which evolution is to be tracked.

More typesafe version, obtained through our own rewriting tool!
Implementation Highlight: Persistence

@EntityProperty + Neo4j + weak references

Diagram:
- Snapshot nodes
- Snapshot-specific identifier mappings
- Entity state and change nodes
Implementation Highlight: **Disk Footprint**

Evolution of Exapus project ([http://github.com/cderoove/Exapus](http://github.com/cderoove/Exapus))

- Single revision: 194149 nodes, 223979 properties, and 194147 relationships of 32 distinct types
- On average: 22.5 files per revision changed

![Graph showing disk footprint over revisions](image)

- Confirms advantages of state sharing previously observed in Orion and Ring/H meta-models
- But for graph-persisted rather than Smalltalk-persisted meta-models and for more fine-grained information (e.g., AST nodes)
Conclusions

defines the first interconnected representation of:

✓ **state & evolution** of the different entities of a software system
✓ each **change** to an entity that results in a new entity state
✓ system **snapshots** under control of a VCS

implementation highlights:

✓ memory-efficient
✓ familiar OO API for tool builders

http://soft.vub.ac.be/chaq/ for upcoming tools that share our meta-model!