# User Controlled Detection of Edit Sequences in Model Evolution

**Nuno Macedo** Jorge Mendes



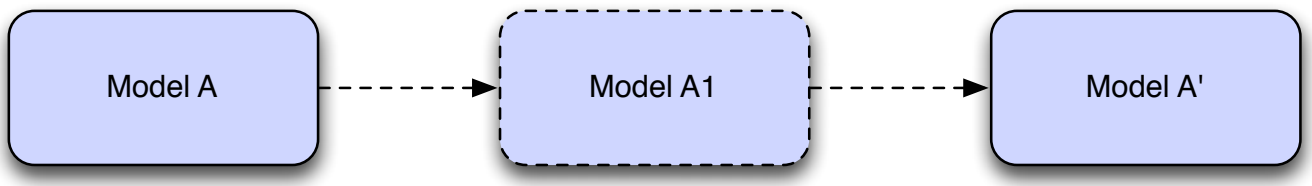
SSaaPP Workshop 2015
July 20
Braga, Portugal

# Introduction

- MDE is an highly dynamic development environment;
- Model *evolution* may impact related artifacts:
  - Evolution of models may affect other related models (model synchronization, bidirectional transformation, ...);
  - Evolution of meta-models may affect conforming artifacts (model co-evolution, transformation co-evolution, ...);
  - ...
- Cumbersome and error-prone task: should be supported by automated techniques;
- Being aware of the sequence of *edit operations* applied to the models is an important step.

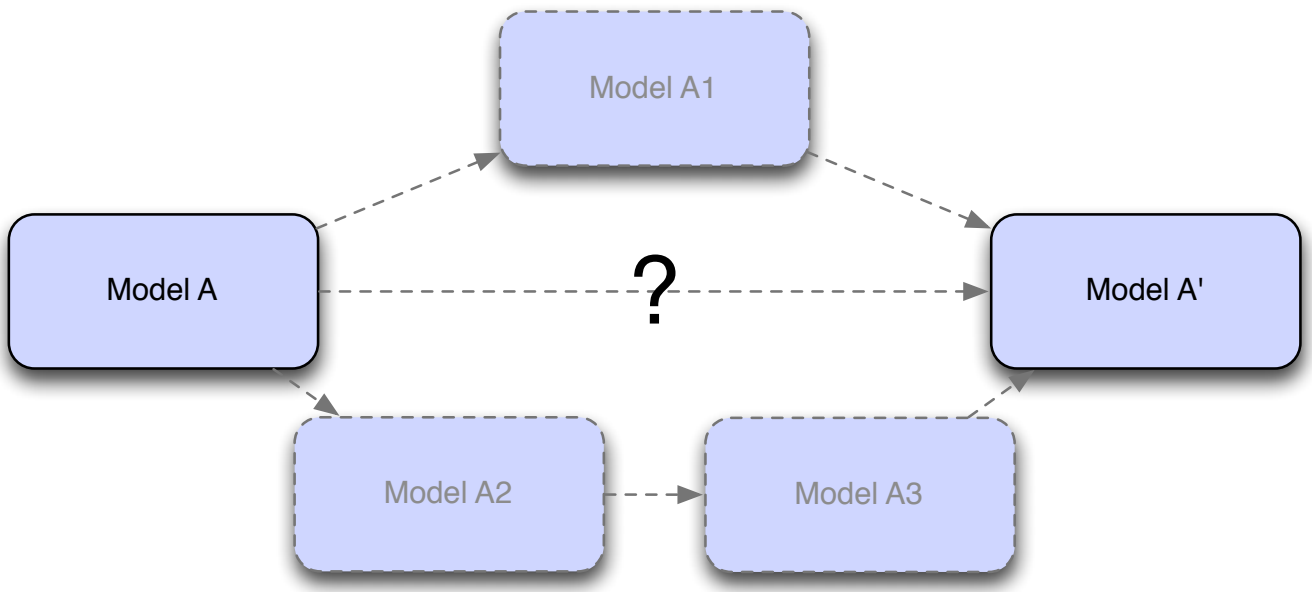
# Edit Sequence Detection

- Operation-based approach:
  - Records every edit operation performed by the developer;
  - Pro: *exact* information regarding the evolution of the model;
  - Con: requires a *dedicated* development environment to record editions.



# Edit Sequence Detection

- State-based approach:
  - Simply relies on the pre- and post-state of the model;
  - Pro: may be deployed over *standard* development environments;
  - Con: lack of exact operations may affect the *accuracy* of the technique.



# Edit Sequence Detection

- Typical state-based approaches detect the differences between two models and derive sequences of atomic operations;
- May not reflect the actual edit sequence: limits their usability in succeeding tasks;
- Open challenges:
  - How are the acceptable edit operations defined?
  - How does the technique select which sequences are returned?
  - How may the user be able to control this selection?
- This (ongoing) work tries to address these issues with *model finding* techniques.

# Model Finding

- Model finding consists of searching model instances that satisfy certain constraints;
  - Successful approaches like Alloy/Kodkod rely on off-the-shelf SAT solvers;
- Our previous work on *target-oriented* model finding allows users to control the generation of solutions;
  - A *target* model is defined, which is approximated by the model finding procedure;
  - Definition of *weights* over different elements provides a finer control.

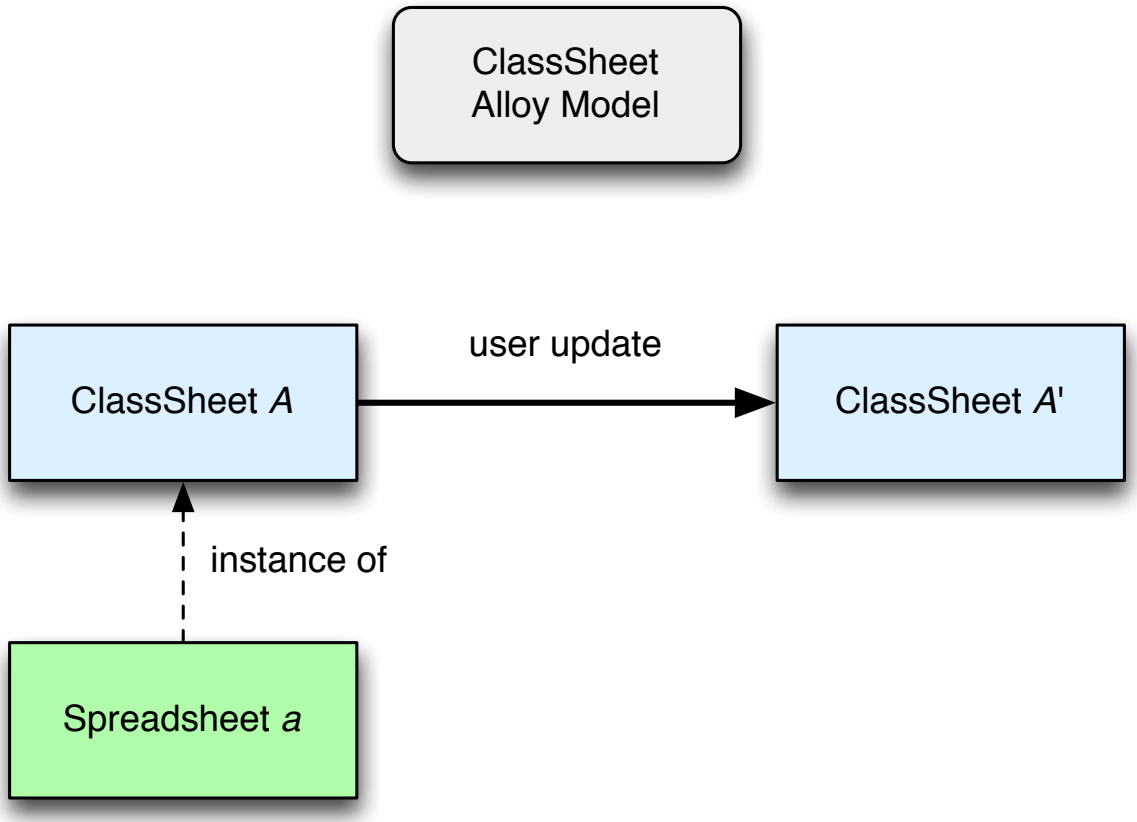
# Model Finding

- We propose to use model finding techniques (Alloy) to search for edit sequences;
- Returns *every* valid sequence by construction;
- By default, target is the empty sequence: *minimal* sequence lengths;
- With *weights* are assigned to operations, this can be controlled by the user;
- Requires the translation of the models and the acceptable operations into Alloy;
- Case study: ClassSheet/spreadsheet co-evolution.

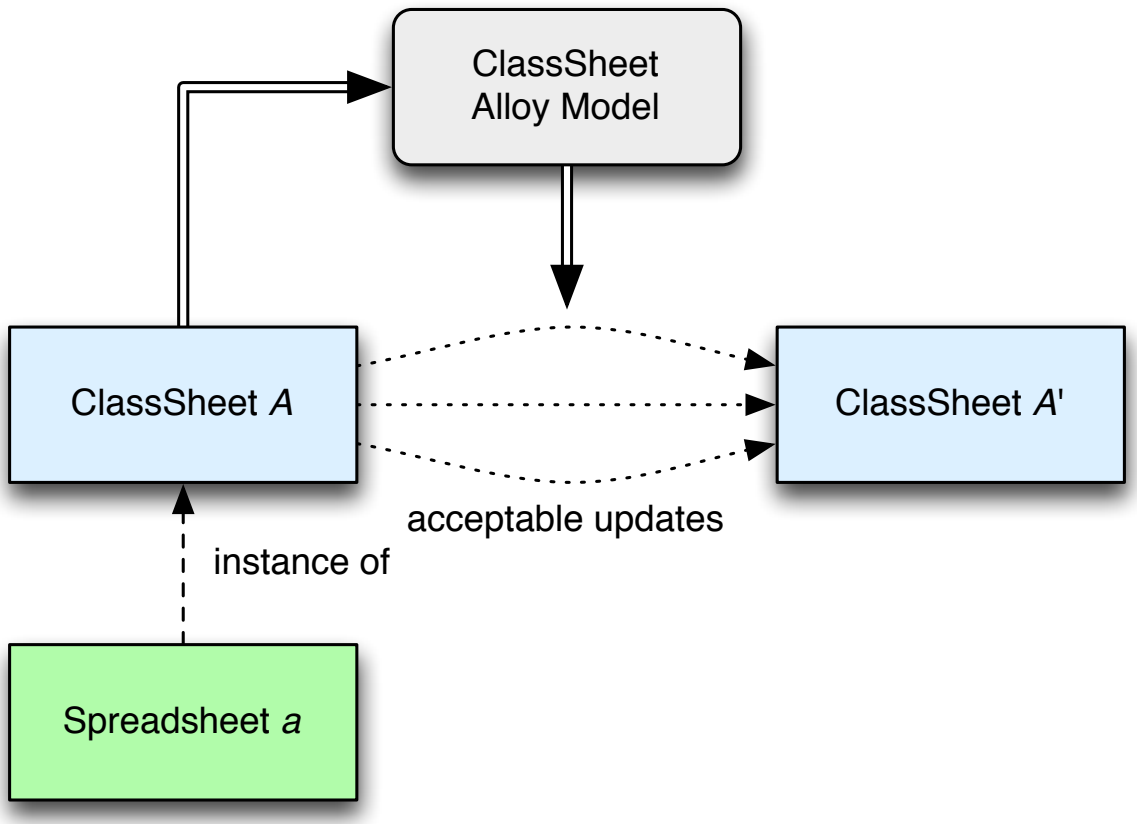
# Application: Spreadsheet Evolution

- Detecting the edit sequence between different versions of ClassSheet models allows the co-evolution of conforming spreadsheets;
- Spreadsheet systems are a paradigmatic example where recording the the user's actions is not feasible;
- Concrete application domain: generic atomic operations are not that helpful;
- E.g., the bounds of a class are not increased directly, but as rows/columns are inserted.

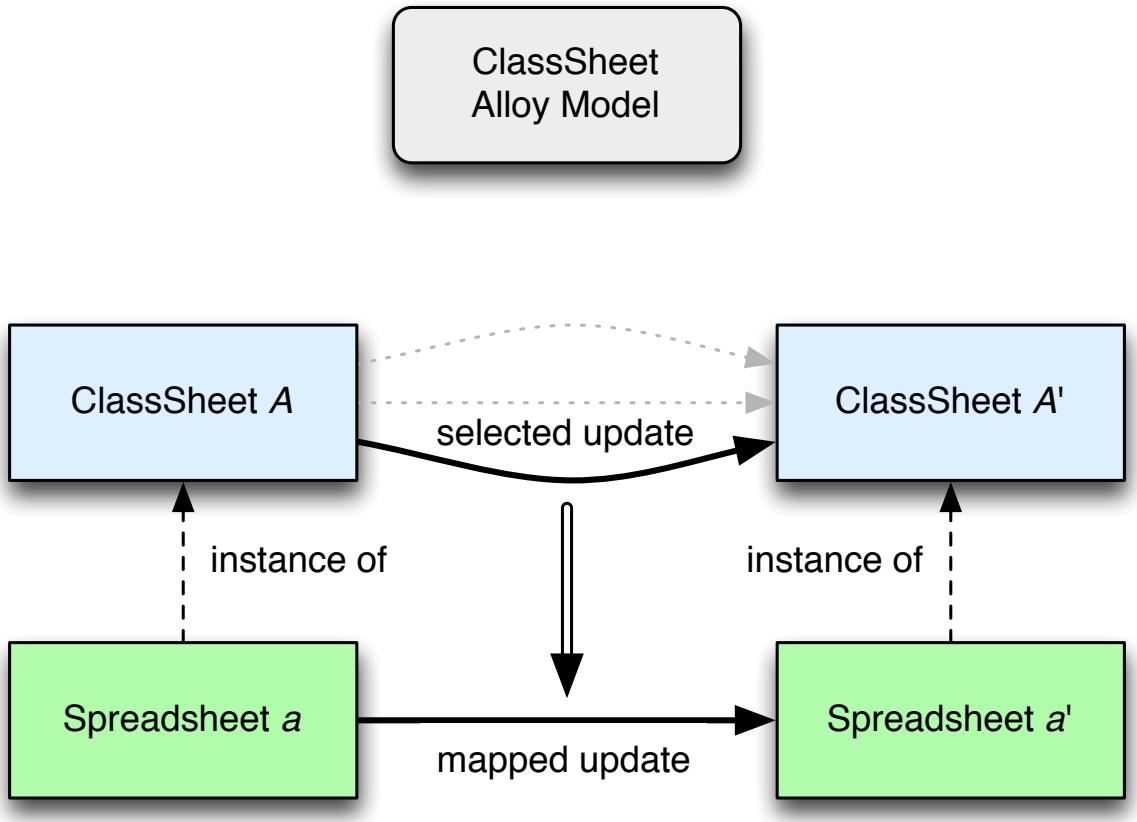
# Overview


ClassSheet
Alloy Model
ClassSheet A
user update
ClassSheet A'
instance of
Spreadsheet a

# Overview


ClassSheet
Alloy Model
ClassSheet A
ClassSheet A'
acceptable updates
instance of
Spreadsheet a

# Overview


ClassSheet
Alloy Model
ClassSheet A
ClassSheet A'
selected update
instance of
instance of
Spreadsheet a
Spreadsheet a'
mapped update

# Current State

- Alloy implementation of ClassSheet models;
- Defined operations: `addColumn`, `removeColumn`, `setCell`, `addClass`, ...
- Calculates *every minimal edit sequence* that produces the evolved ClassSheet;
- Edit operations are then mapped into instance operations to co-evolve the spreadsheet.

# Example

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | **Number** | **Name** | **Exam1** | **Exam2** | **Grade** |
| 2 | num=0 | name="" | exam1=0 | exame2=0 | grade=0 |
| 3 | ... | ... | ... | ... | ... |
| 4 | | | | | |

delColumn 2; setCell (1,3) "Exam"; addColumn After 4; setCell (1,5) "Avg"
...
delColumn 3; setCell (1,3) "Exam"; addColumn After 4; setCell (1,5) "Avg"

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | **Number** | **Name** | **Exam** | **Grade** | **Avg** |
| 2 | num=0 | name="" | exam=0 | grade=0 | avg=0 |
| 3 | ... | ... | ... | ... | ... |
| 4 | | | | | |

# Example

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | Number | Name | Exam1 | Exam2 | Grade |
| 2 | num=0 | name="" | exam1=0 | exame2=0 | grade=0 |
| 3 | ... | ... | ... | ... | ... |
| 4 | | | | | |

delColumn 2; setCell (1,3) "Exam"; addColumn After 4; setCell (1,5) "Avg"
...
delColumn 3; setCell (1,3) "Exam"; addColumn After 4; setCell (1,5) "Avg"

moveColumn 3; setCell (1,3) "Exam"; setCell "(1,5) Avg"

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | Number | Name | Exam | Grade | Avg |
| 2 | num=0 | name="" | exam=0 | grade=0 | avg=0 |
| 3 | ... | ... | ... | ... | ... |
| 4 | | | | | |

# Planned Work

- We have previously successfully applied model finders to *consistency management* in MDE;
- Meta-models with constraints, as well as conforming models, are translated to Alloy;
- Model finder used to search for *closest* consistent models;
- Seamless integration: Eclipse plugin over the EMF architecture.

# Planned Work

- We expect to adapt such techniques to the detection of edit sequences in standard MDE evolution;
- Generic technique:
  - Seamless integration in the MDE development process;
  - Built over the EMF framework (Ecore meta-models, XMI models);
  - Operations defined as pre- and post-conditions (in OCL);
  - Weights over different operations allow finer control;
- Artifacts, as well as the operations, are translated into Alloy to search for minimal edit sequences according to defined operations.

# Conclusions

- A generic MDE framework for the detection of customized edit sequences to address model evolution.
- Spreadsheet evolution is a paradigmatic example: state-based with context-specific operations.

- Scalability is the major problem: not that crucial since this is to be run sporadically?
- Overwhelming number of solutions: will weights suffice to control the selection?
- How should the edit sequences be processed so that they can be consumed in succeeding tasks?