Scalable Model Revision Control

X. Blanc
Université de Bordeaux / LaBRI
Xavier.Blanc@labri.fr

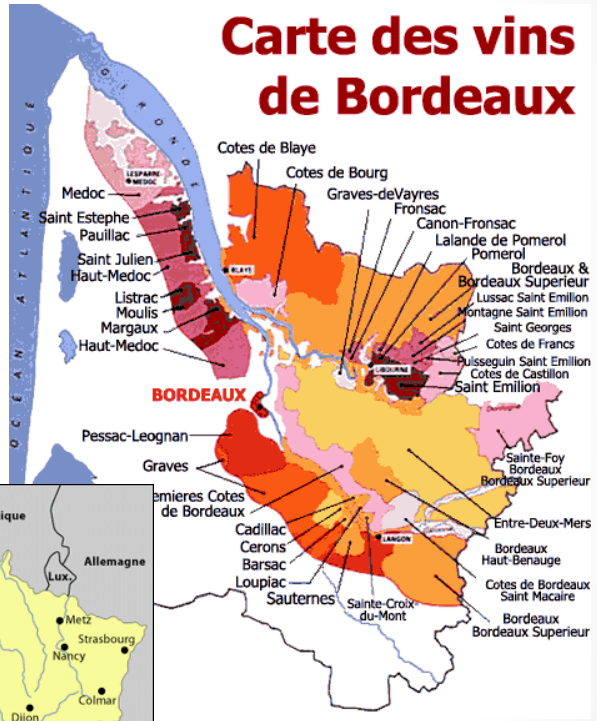
Outline

- SE@LaBRI
- Issues
- SCM key concepts
- Models
- How to deal with model fragmentation ?
  - Model Unit, Method Unit
  - Reuse Strategy
- Validation
- Conclusion

SOFTWARE ENGINEERING AT LABRI

LaBRI @ Bordeaux

- University Bordeaux 1
  - Science & Technology
  - 10000 students
  - 37 labs
- LaBRI
  - 130 researchers
  - 130 PhD Students



Software Engineering

- New research theme
  - September 2010 (5 months ago)
- Domain
  - Software Engineering / Design / Model Driven
  - Static Analysis / Consistency
  - Complex System (Internet)

- 2 researchers
  - Jean-Remy Falleri & Xavier Blanc
- 1 (+1) PhD Student
  - David Bruant => Web Client
  - Open Position =>

ISSUES

How to version a model ?

- What to version ?
  - A model, a model fragment ?
- How to deal with collaborative work ?
  - Copy-modify-merge
  - Lock
- What to copy / modify merge ? What to lock ?
  - A model (+ its links) ?
- What about diagrams / views ?
- What about model size ?

SCM KEY CONCEPTS

SCM

- *Management support discipline* and *development support discipline*

- Use cases

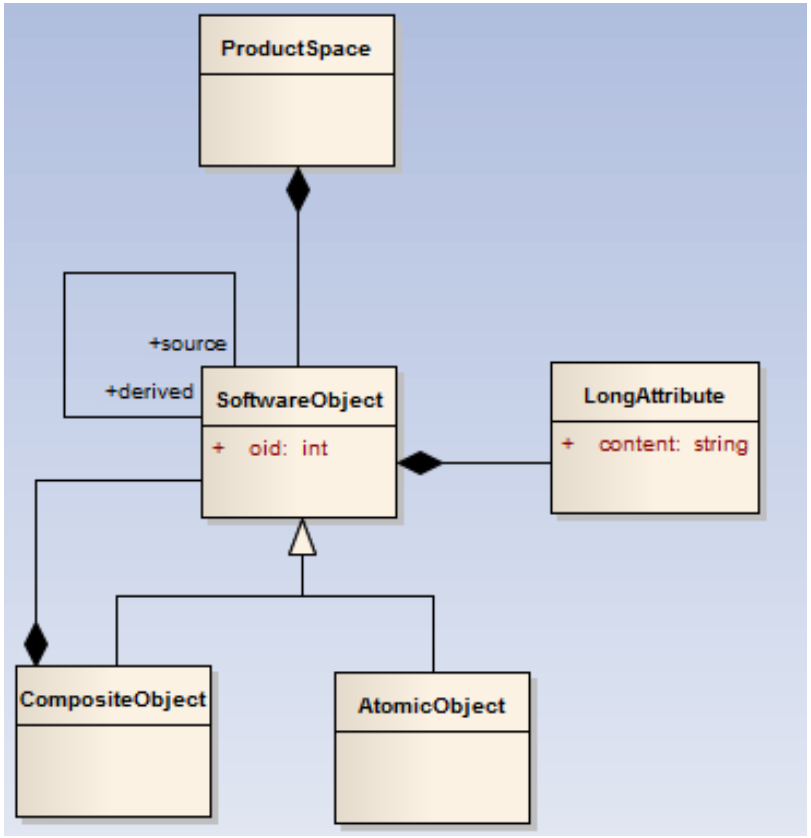
Management

1. Update a project
2. Look-up a particular version
3. Track the changes

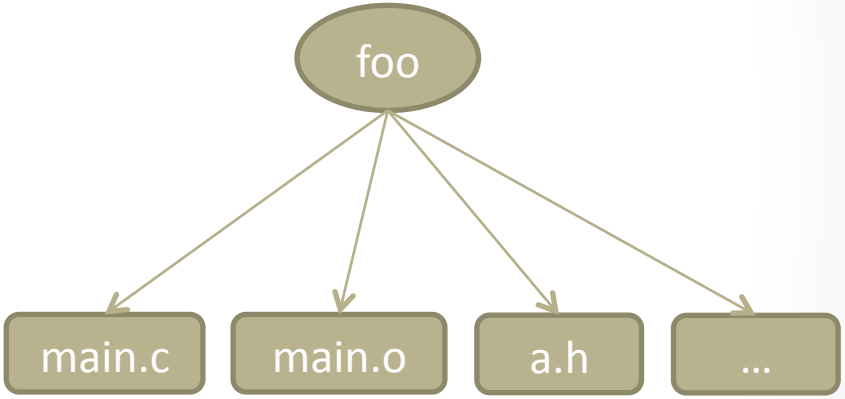
Development

1. Build a project
2. Share it
3. Commit & Update [Lock]

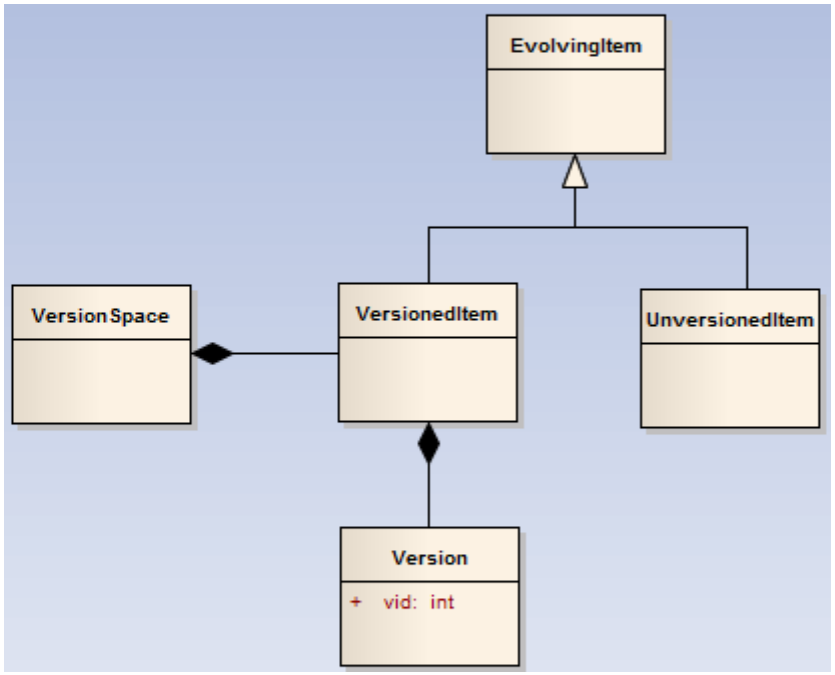
Product Space [Conradi98]



- How to represent a product space ?



Version Space [Conradi98]

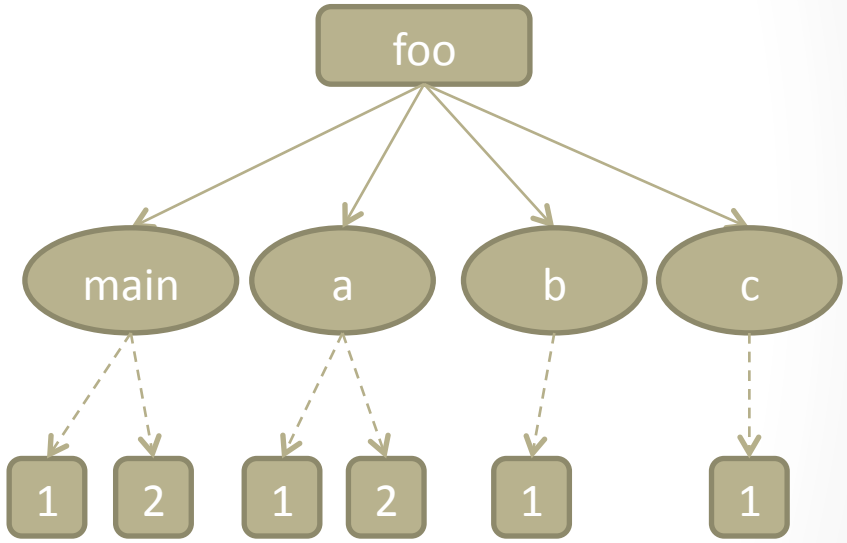


- A version v represents a state of an evolving item i.
- Versions of an item share a common properties called invariants.
- How to represent a version space ?
  - Tree, Graph
  - States / Changes

Interplay of product space and version space [Conradi98]

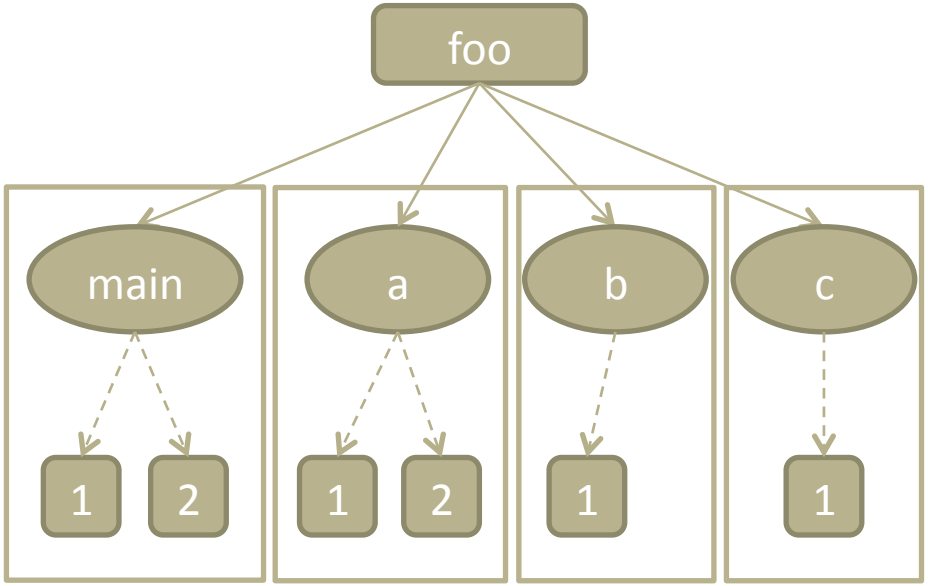
- A versioned object is simply represented by an OR node whose outgoing edges point to its versions.
- AND edges are used to represent both composition and dependency relationships.

- Product First



Granularity of versionning [Conradi98]

- At the external interface, software objects are the items subject to version control
  - Component versioning: only atomic objects are put under version space, modeled, for example, by a version graph.



Version Control Mechanism

- Pessimistic version control : parallel editing is prevented by locking
  - Items are locked
  - Consistency has to be preserved (links)
- Optimistic version control : parallel editing is supported thanks to a copy-modify-merge approach
  - Items are copied and modified
  - Items are merged in case of parallel modifications
  - Conflicts are identified and resolved

Software Merging [Mens01]

- Two-Way or Three-Way merging
  - A common base version (or not)
- Merging
  - Textual / Syntactic / Semantic
- State based or operation based
- Conflict
  - Detection
  - Resolution

SCM on Models

- What are the relationships between models and software objects?
- What are the versioned items?
  - What to version ?
- How to deal with collaborative work ?
  - What to commit / update ?
  - How to merge ?
  - What to lock ?

MODELS

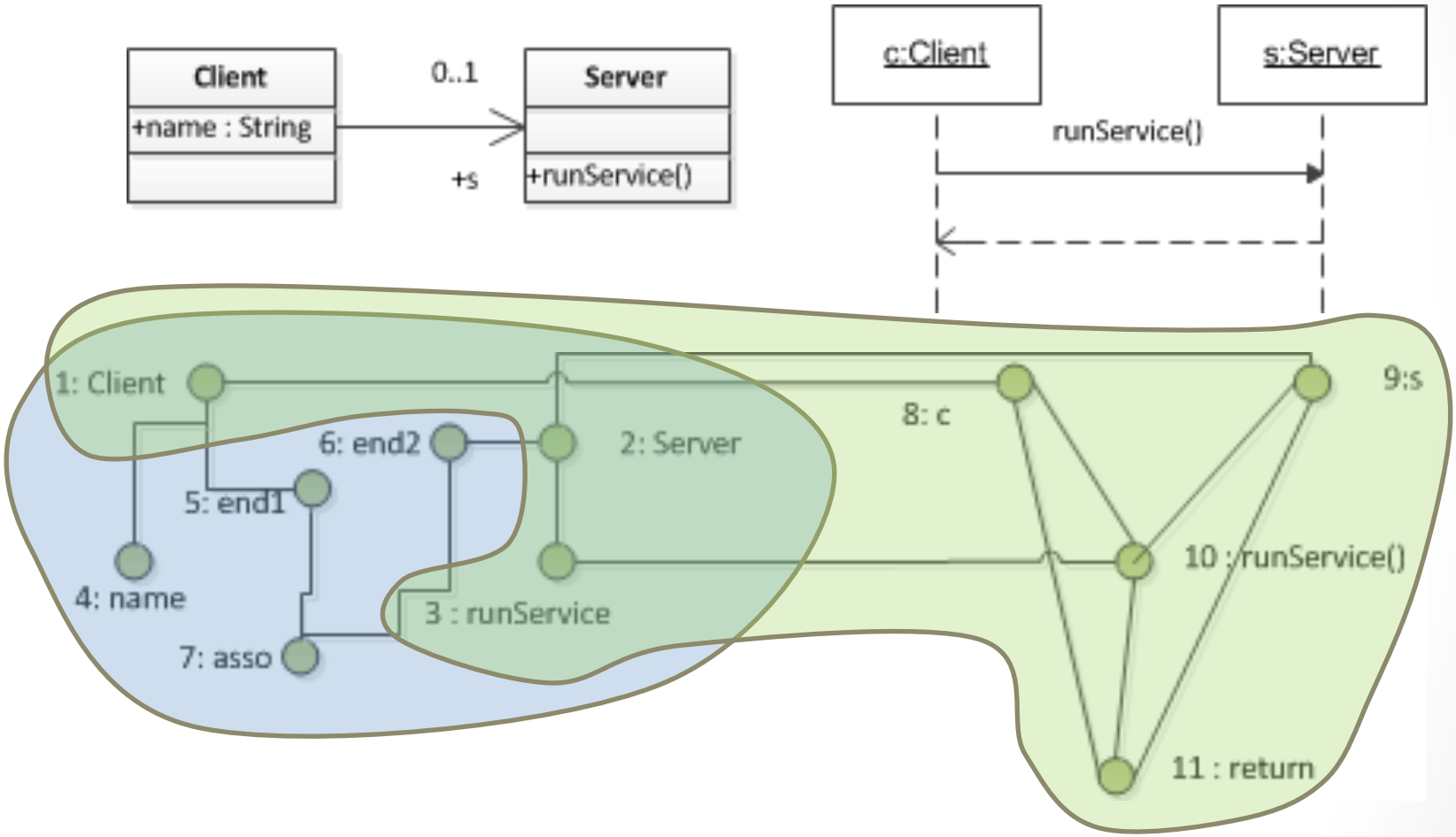
Models

- Principles [OMG01][Mellor03][Selic03][Schmidt06]
  - Software artifact => Model
  - Software => described by on global model (N models)

- A model is composed of elements (Typed Graph) [Spanoudakis01]

```
//Model in Alloy
sig Model {
        me: set ME,
        metaclass: me -> one MC,
        valueP: me -> P -> V,
        valueR: me -> R -> me,
}
```

Fragments and views (1/2)



Fragments and Views (2/2)

- A fragment is included in a model
  - Some of the model elements
  - Some of the values
  - Some of the references
- A view is a graphical representation of a fragment
  - No additional element

=> What are the fragments and views that should be versioned ?

HOW TO DEAL WITH MODEL FRAGMENTATION ?

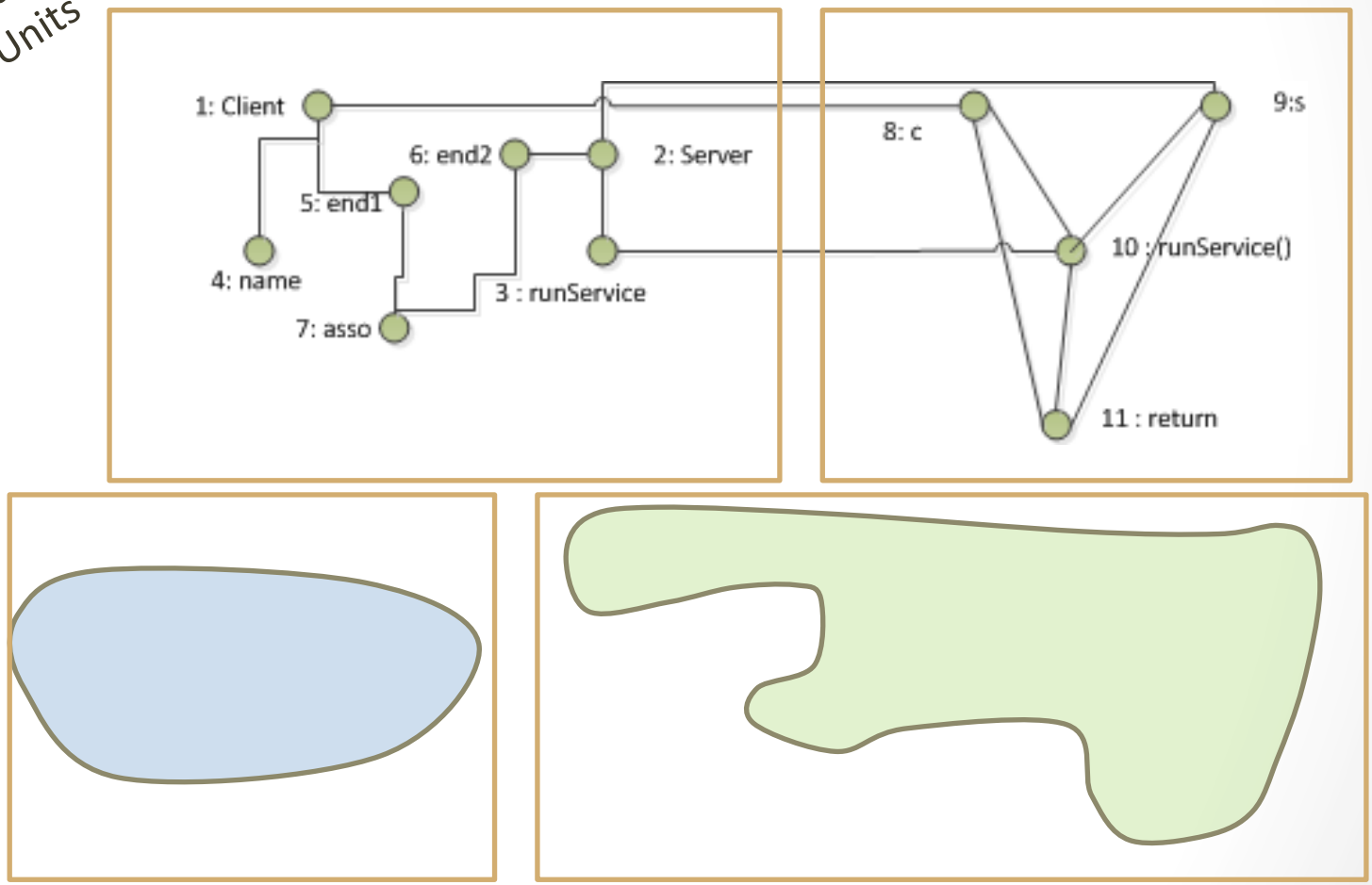
Reuse Unit and Method Unit

- Key Principles
  - The model Elements should be partitioned, each partition is a versioned item
  - The view elements just reference model elements. A view is composed of view elements. A view is a versioned item.
- Two concepts
  - Reuse Unit (RU) : a versioned item that contains a partition of model element
  - Method Unit (MU) : a versioned item that contains one view

=> Modifying a model element has an impact only on its containing RU and on its referencing MU

Example

2 Reuse Units
2 Method Units



Sub-problems

- How to fragment models and views in RU and MU ?
  - How to assign model elements into RU ?
  - How to assign view elements into MU ?
  - How to organize RU and MU ?

=> Reuse Strategy

Reuse Strategy

- The assignments of model elements and view elements to RU and MU are statically defined by a reuse strategy.
- A strategy defines two assignment functions
  - ME -> RU
  - VE -> MU

- In order to organize RU and MU, we define the concept of Product Unit (PU) that groups RU and MU in a hierarchy (directory like).

Examples of Strategy

- AllInOne strategy
  - All model elements are assigned to the same RU
  - All view elements are assigned to the same RU
  - The RU is organized in one PU
    - ⇒ XMI strategy with IBM Software Modeler
- AllInAll strategy
  - All model elements are assigned to their own RU
  - A MU is created for each view. The MU contains all the view elements of the view.
  - The RU and MU are organized in one PU
- SequenceAndClass strategy (UML)
  - For each class, a dedicated RU is created
    - All operations and properties of a class are assigned to the RU of their class
  - For each sequence, a dedicated RU is created
    - All lifelines of a sequence are assigned to the RU of the sequence
  - For each diagram, a dedicated MU is created. The MU contains the corresponding view elements that reference model elements in RU.

Use Case

- Update
  - The developer choose the elements he wants to update
    - He may choose a root elements and all the contained elements will be selected
  - The corresponding RU and MU (the ones that contain the chosen elements) are identified
  - The corresponding RU and MU are updated to the last version
- Commit
  - The developer choose the elements he wants to commit
  - The corresponding RU and MU are identified
  - The corresponding RU and MU are committed
- Lock
  - …

=> Side effect = considered elements >> chosen ones

Scalability

- A strategy is scalable if its maximum number of model elements in a RU (MAS) is a constant.
  - Motivations:
    - Size of the overhead (considered elements should be equal to the chosen ones)
    - Complexity of the diff / merge (the more elements, the more complex)
- *A strategy is scalable if, considering a classical sequence of changes, the maximum number of exchanged messages (MEM) between the workspace and the repository is a constant.*
  - *Motivations:*
    - *Transaction of commit (if the model elements to commit belong to more than one RU).*

Efficiency

- A strategy should not need too many RU
  - (P1) *let s1, s2 be two strategies, s1 is more pragmatically scalable than s2 if and only if* Card$(RU)_{s1}$ < Card$(RU)_{s2}$
- A strategy should not imply the modification of too many RU in response to classical changes done by the developers.
  - (P2) *let s1, s2 be two strategies, s1 is more pragmatically scalable than s2 if and only if* Card$(RU(AfActions))_{s1}$ < Card$(RU(AfActions))_{s2}$

VALIDATION

Prototype

- A CASE tool proxy that implements a reuse strategy and that interacts with SVN

- A sample scenario
  1. Init SVN
  2. Select a strategy
  3. Build a model
     - A class diagram and a sequence diagram
  4. Commit the model
  5. Move an operation
  6. Commit again

Numbers

Strategy	SequenceAndClass	AllInAll	AllInOne
Exp1	14863	23063	6940
Exp2	14676	22889	6979
Exp3	14817	23083	6988
Exp4	15298	23003	6449
Exp5	14793	23197	6381
Average (ms)	14889,4	23047	6747,4
Ratio (/smaller one)	2,20668702	3,41568604	1
Nb of units	4	7	1

Many more to do

- Real models
- Real strategy
- Real scenario
- Real Merge
  - The state of the art is very rich on that topic
- Empirical Study

CONCLUSION

Related Work

- A lot for model merge
  - [Alanen03][Ohst03][Sriplakich06] (Treude07][Gerth10] [Könemann10]
- Not so much (found) for model fragmentation
  - Model Slicing => not the same problem
  - [Nguyen05][Nguyen06] => not for huge models

Synthesis

- What are the relationships between models and software objects?
  - RU and MU are the software objects
  - They fragment model elements and view elements
- What are the versioned items ?
  - RU and MU are the versioned items
- How to create them ?
  - Thanks to a predefined reuse strategy
- How to deal with collaborative work ?
  - What to commit / update ?
    - The identified RU and MU
  - How to merge ?
    - Merge algorithm on RU and MU ? => Further work
  - What to lock ?
    - The identified RU and MU

Further Work

- Formal definition for strategy
- Validation of our comparison criteria (scalability and efficiency)
- Experiment with developers of models
- Bridge UML strategy with SVN & GIT