PyGK: The Python Graph Kernel

Marc Provost
McGill University
marc.provost@mail.mcgill.ca

February 18, 2005

Abstract

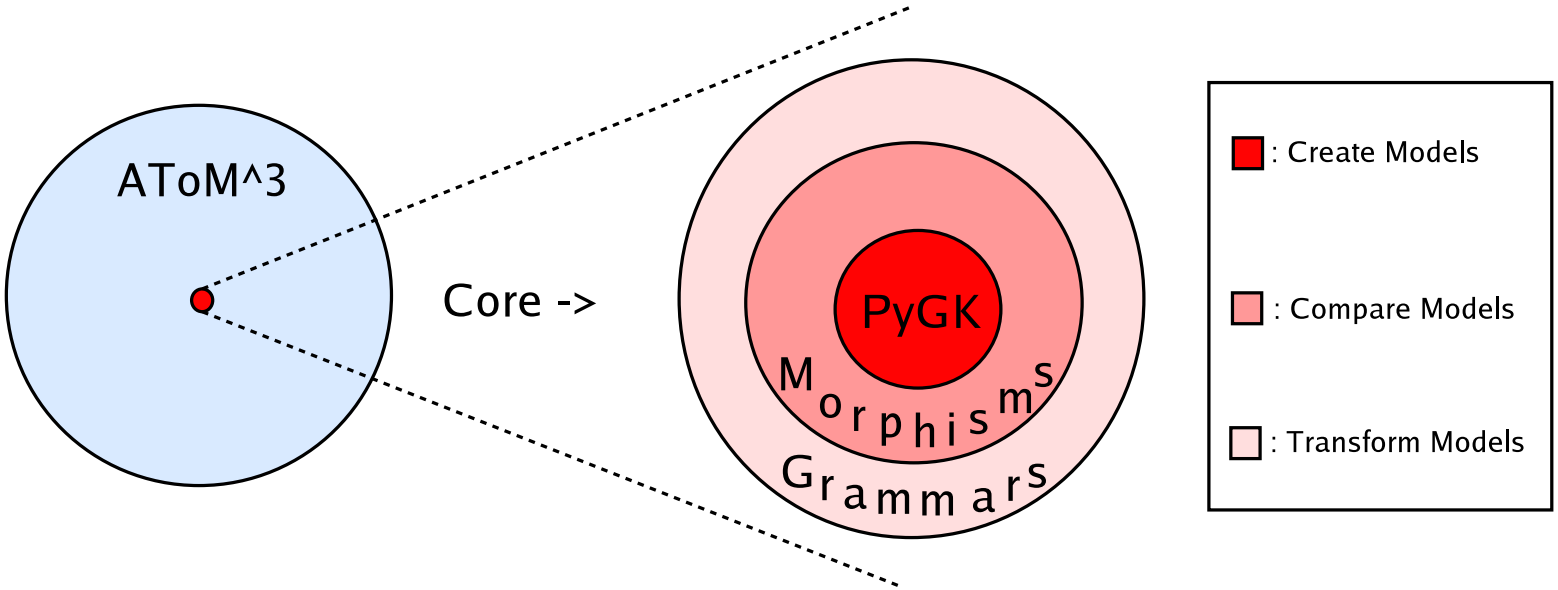
This presentation introduces PyGK, the Python Graph Kernel, which will be the core component of the next generation of AToM^3, the meta-modelling tool developed at McGill. PyGK is a package implementing *Labelled*, *Directed* and *Hierarchical* Graphs. We will go over its main features in details with several intuitive examples.

Presentation Overview

- Intro to AToM^3 Structure
- Functional Features
- Examples
- Non-Functional Features
- Performance Analysis
- Q & A
- What's next?

MSDL

AToM^3 Structure

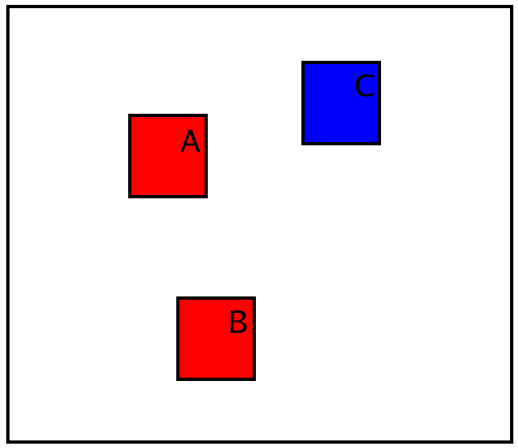


McGill

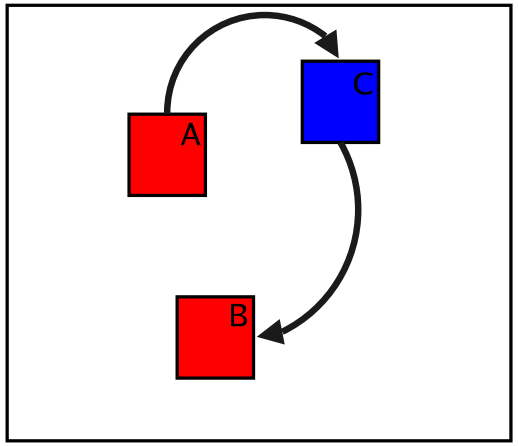
MSDL

Coarse Grained Functional Requirements

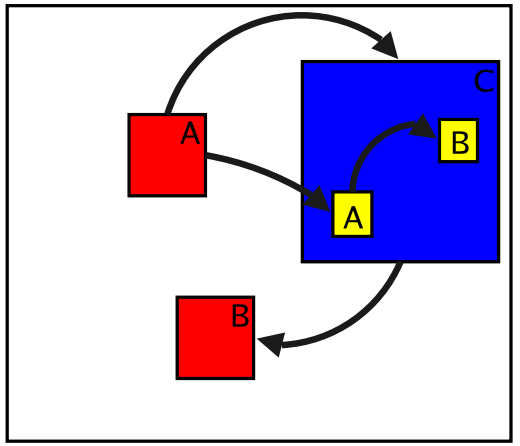
Labelled..



Directed..



Hierarchical..

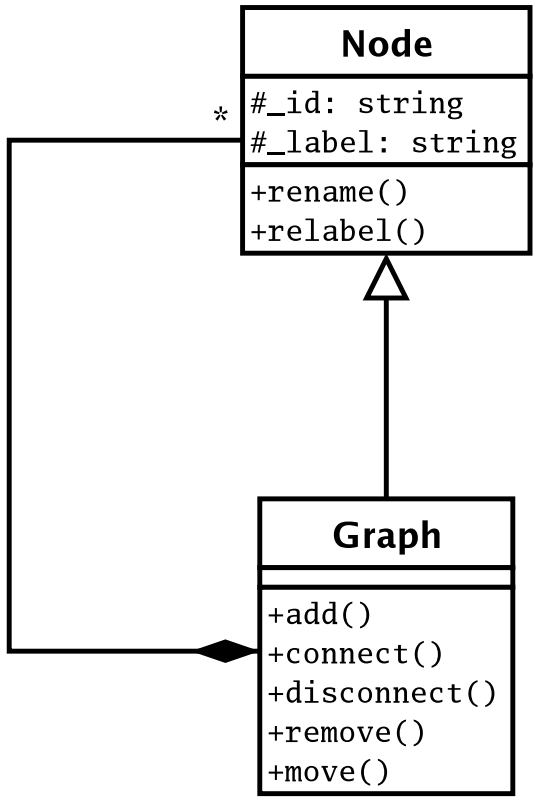


McGill

Fine Grained Functional Requirements

- Well-Defined set of operations
- High-level Iteration
- Simple Navigation language
- Primitive Types
- Import/Export to XML
- Undo/Redo
- Versioning

Well-Defined set of operations



MSDL

Well-Defined set of operations

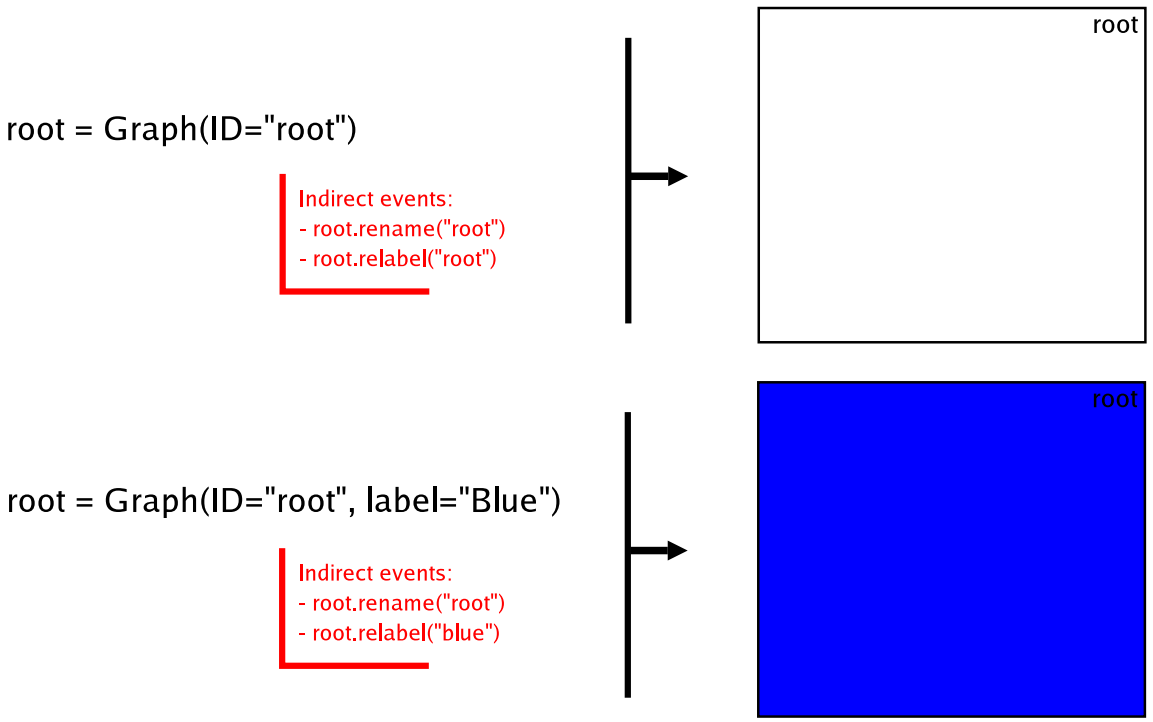


Figure 1: "CREATE", "RENAME", "RELABEL" Events

MSDL

Well-Defined set of operations

```
root = Graph(ID="root")
root.add(Node(ID="A", label="Red"))
root.add(Node(ID="B", label="Red"))
root.add(Graph(ID="C", label="Blue"))
root.add(Node(ID="B", label="Yellow", path="C"))
root.add(Node(ID="A", label="Yellow", path="C"))

# Alternate syntax
#root["C"] = Node(ID="B", label="Yellow")
#root["C"] = Node(ID="A", label="Yellow")
```

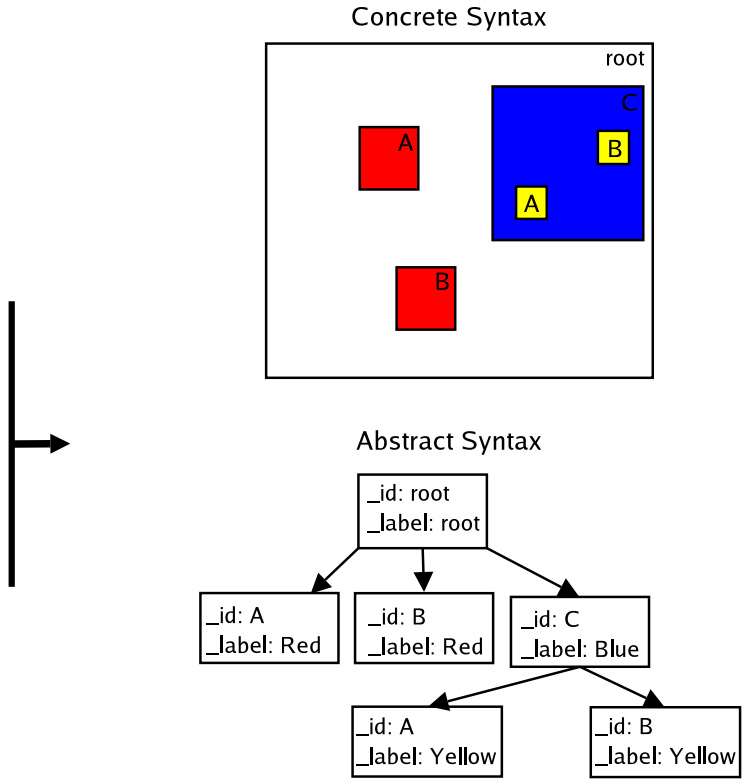


Figure 2: "ADD" Event

McGill

Well-Defined set of operations

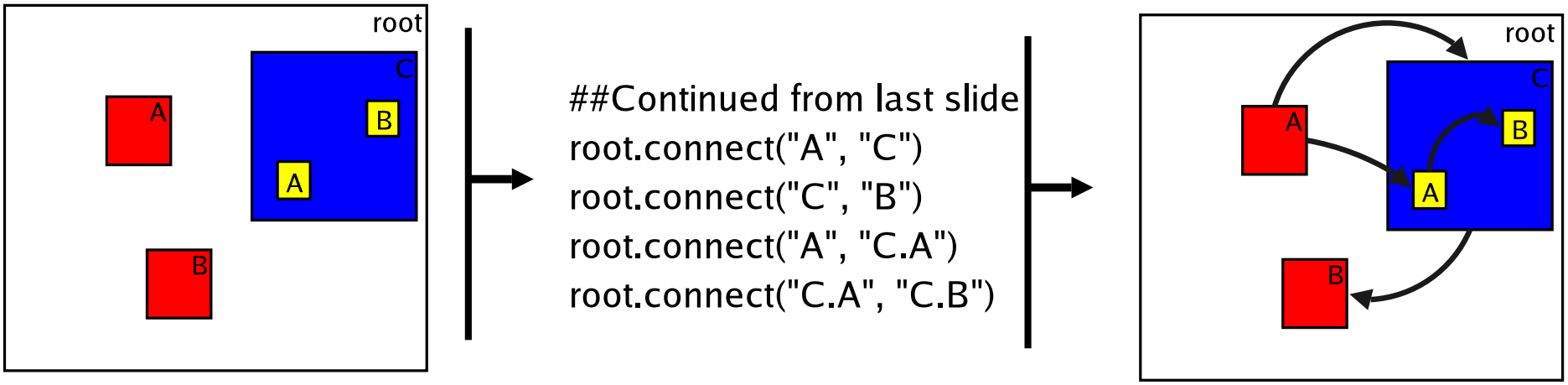


Figure 3: "CONNECT" Event

Well-Defined set of operations

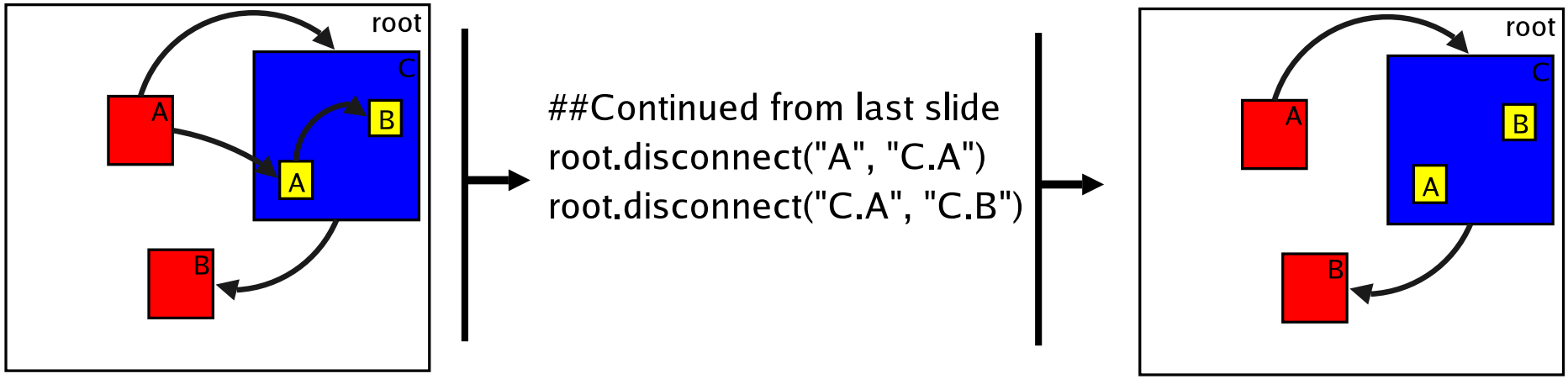


Figure 4: "DISCONNECT" Event

Well-Defined set of operations

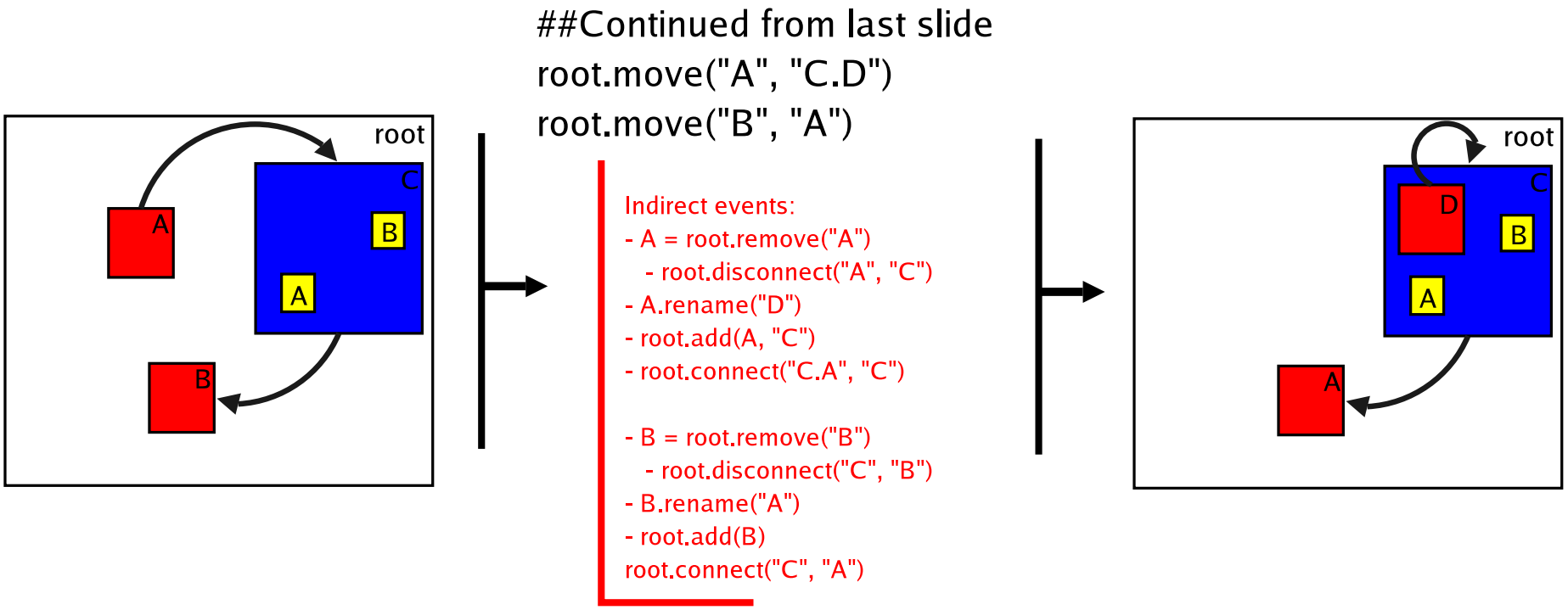


Figure 5: "MOVE" Event

Well-Defined set of operations

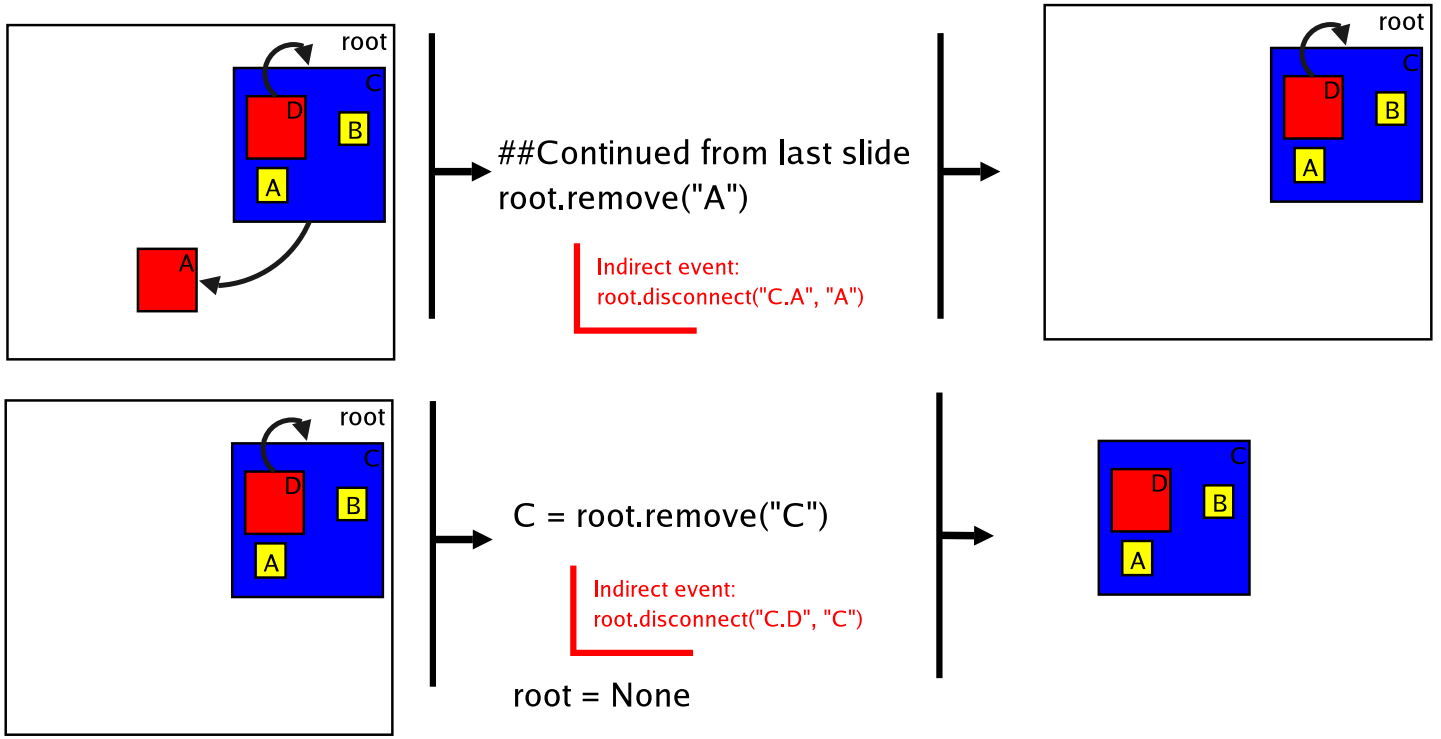
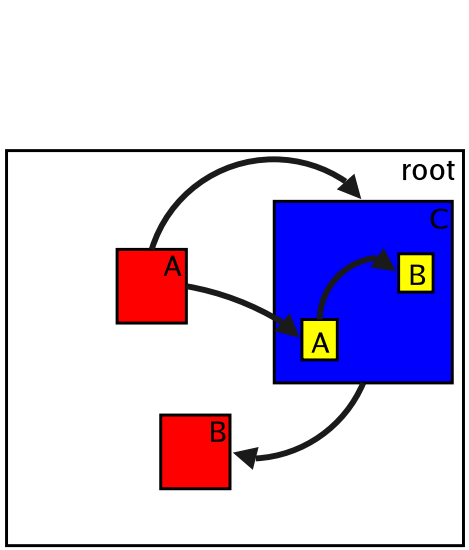


Figure 6: "REMOVE" Event

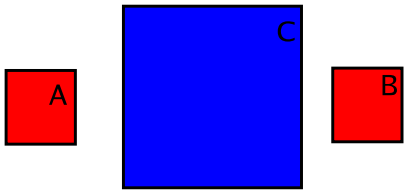
MSDL

High Level Iteration

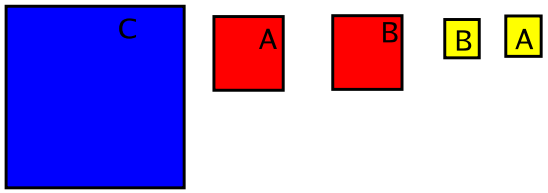
IMPORTANT: The elements inside a graph are NOT ordered. They are iterated in an undefined order.
Note: This could easily be changed if needed in the future.



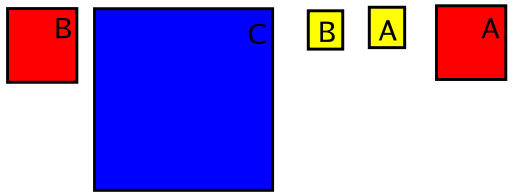
for elem in root.iterate()



for elem in root.iterateAll(traversal="BreathFirst")

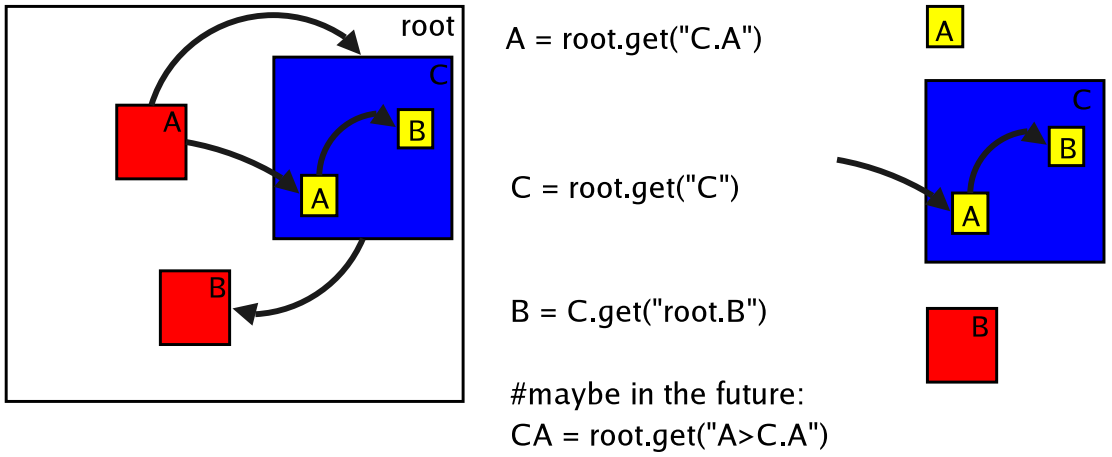


for elem in root.iterateAll(traversal="DepthFirst")



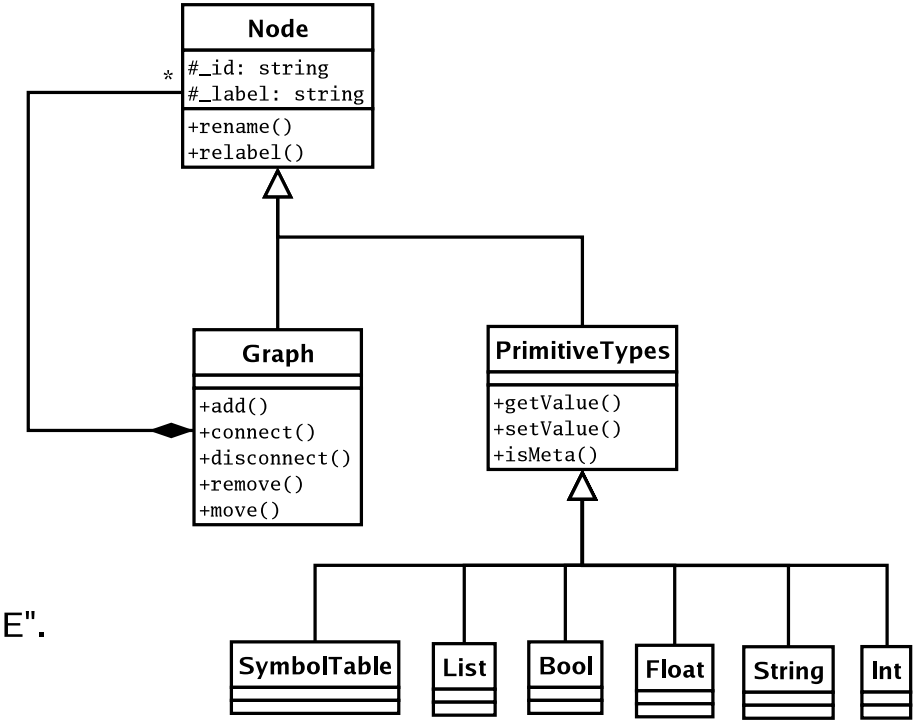
Simple Navigation Language

- Every operation acting on a graph takes one or multiple *path* arguments. A path is used to locate the elements concerned by the operation. A path is simply a string "X.Y.Z" navigating through the graph hierarchy.



Primitive Types

A node contains only an id and a label. In order to support complex models, the graph kernel supports generalized nodes containing values. A graph will use them as if they were generic nodes. At some point in the future, AToM3 will understand the meaning of an "Int" node in a given model. Right now, the kernel itself cannot tell the difference. Note that a new event is added for the primitive types: "SETVALUE".

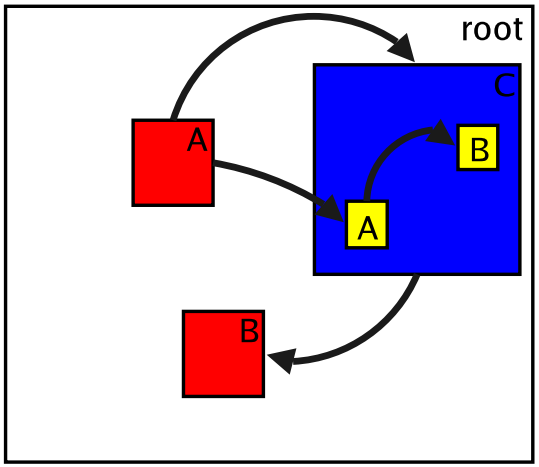


Import/Export to XML

- A simple XML graph language (AGL) was designed to export graphs to stable storage.
- To ease reuse, one AGL file is generated for each children graph contained in a the exported graph.
- Now very simple to use..

MSDL

Import/Export to XML

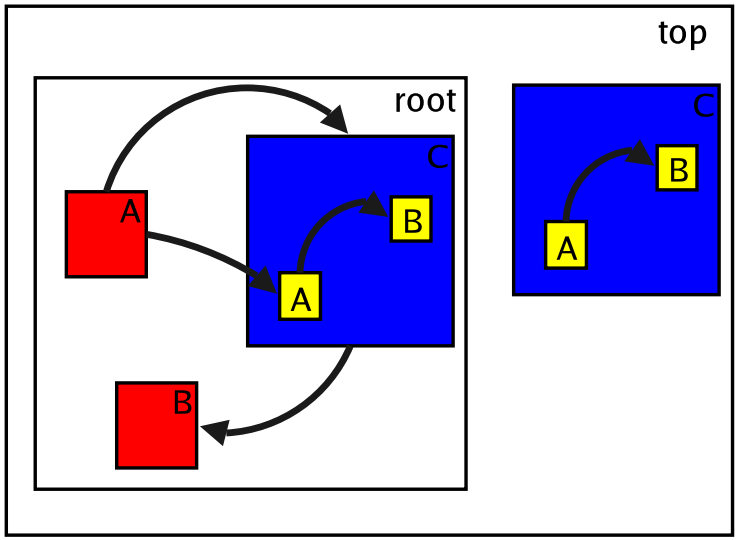



```
gen = AGLGenerator(root, directory="./")
gen.genCode()

loader = AGLLoader(directory="./")
root = AGLLoader.load("root")

C = AGLLoader.load("root.C")

top = Graph(ID="top")
top.add(C)
top.add(root)
```



Undo / Redo

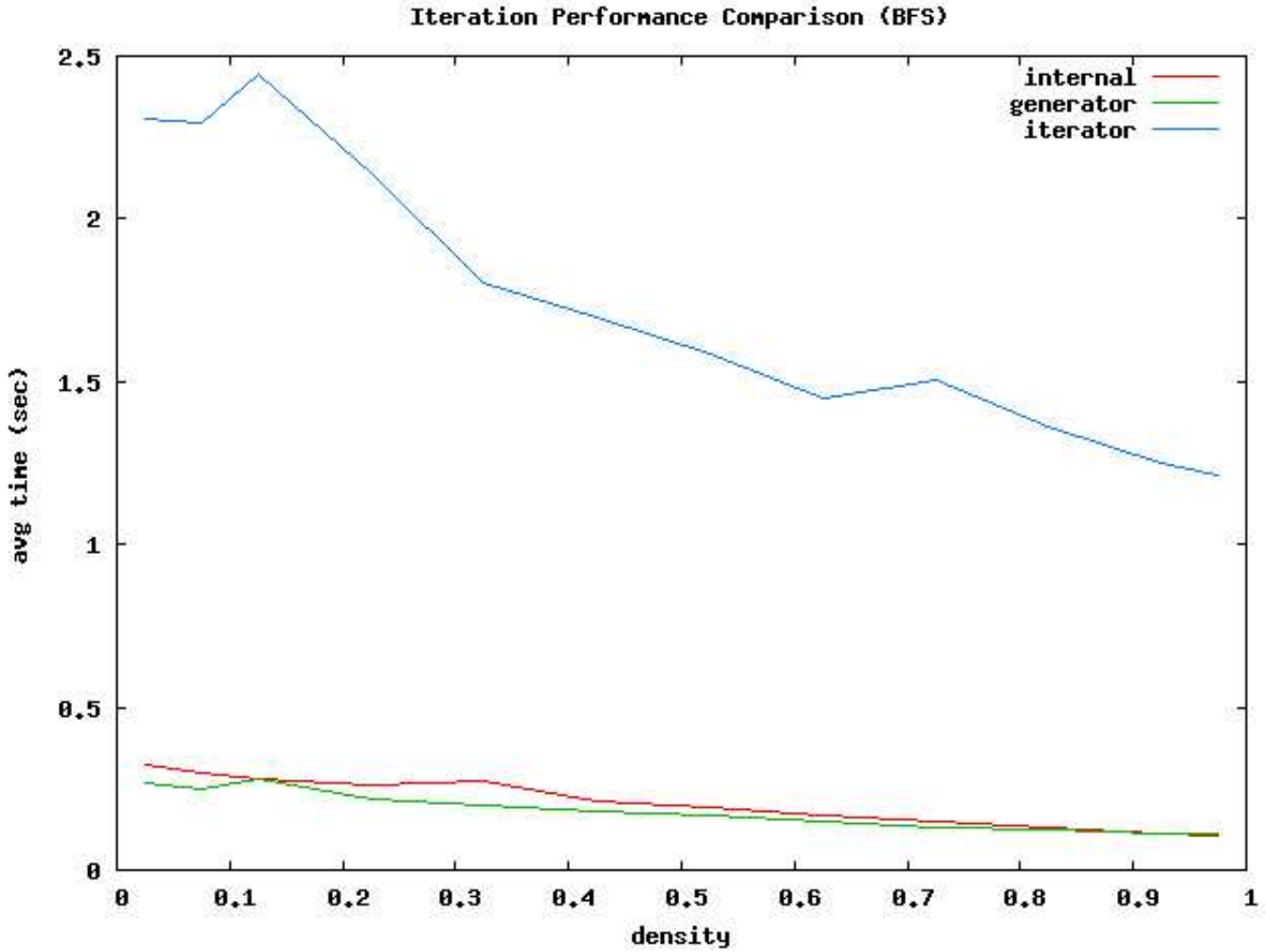
- Still in development :)
- Is based on the notion of events presented previously
- When executed, an event will be pushed on a stack with the necessary information to perform the inverse operation.
- Each Graph will remember the performed operations that concerns it.
- I am now explaining this on the board :)

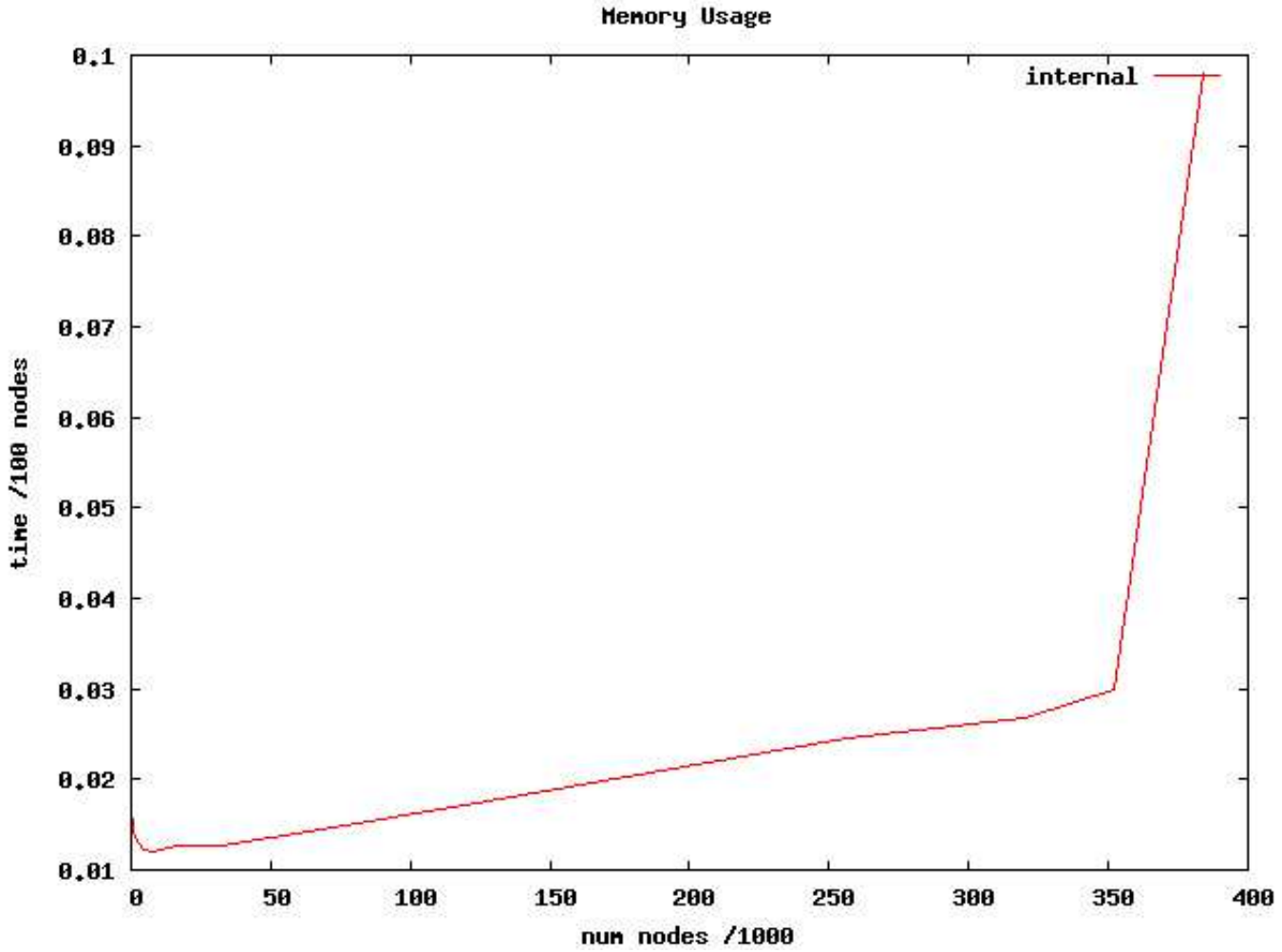
Versioning

- Generalizes Undo/Redo
- Basically, a version groups a set of events that were performed on a model.
- A user can modify a model and at any point in time define a particular state to be a version X. At this point, a new Undo/Redo stack is initialized.
- When exporting the models to stable storage, all the Undo/Redo information for each version is also saved.
- When importing a model, a particular version could be imported, or even more than one version.

Non-Functionnal Features

- Fast, but consumes a lot of memory. (Partly due to python)
- Simple Design, minimal
- Easy to use (I Hope!)
- Heavily tested (I am still creating new test cases)
- Optimized for meta-modelling:
  - Hashtables were used in combination with lists: fast element retrieval, fast iteration. Good for simulators, code generators.


Iteration Performance Comparison (BFS)
internal
generator
iterator
avg time (sec)
density
2.5
2
1.5
1
0.5
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Memory Usage
internal
time /100 nodes
num nodes /1000
0.1
0.09
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
50
100
150
200
250
300
350
400

Q & A

- Why no labelled edges?
- Dangling edges?
- Why no hyper edges?
- What about ports?
- What about cyclic hierarchy?
- Any other questions?

What's Next

- Undo, Redo, Versioning
- Tests, Tests, Tests..
- Higraph morphisms (already have graph morphisms)
- Higraph Transformation!