PyXDiff - Model Differences

How to specify and detect changes in graph-like models.

Ulf Schwekendiek

Modelling and Simulation Based Design 2008
Winter Term COMP 763B
McGill University, Montral, Canada
Student ID: 260225272
ulf.schwekendiek@gmail.com

1 Introduction

Detection computation of differences in information models are essential to many development and management practices. In most of the cases a differences is defines as a script explaining how to get from a source information to a target one. The GNU diff tool for instance is highly used in current version control systems such as CVS or SVN in order to compute such scripts. This tool is tailored for detecting plain text source code difference in a very efficient way. It uses the longest common subsequence algorithm in order to compute an upgrade actions such as add, delete and upgrade.

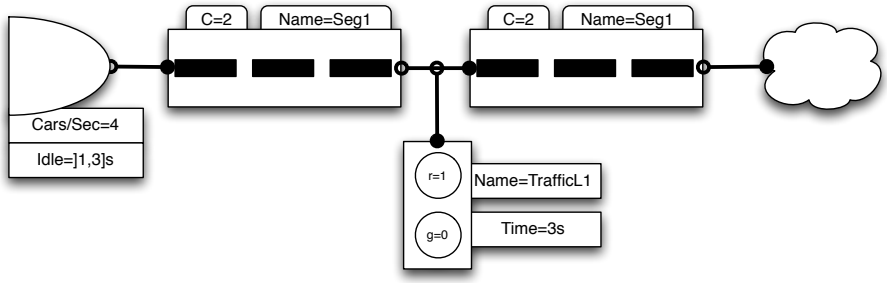

Fig. 1. Sample traffic net model.

Sadly this tool leads in Model-Driven Engineering (MDE) approaches to very unsatisfactory results. In this approach I use the possibility that every model can be mapped to its XML representation. The GNU diff tool can take now the flatten textual input and compare two models through their XML representation. However this leads to very bad results. There is another algorithm called X-Diff which improves this approach and should rather be used. This algorithm computes a nearly optimal update script in most of the cases as long as the changes between the source and target model are not that big. However when we are creating this XML based model information we generate them usually with some tools. This tools store in many cases a

lot of noise in the model's XML representation. For instance in a simple traffic net model like in Figure 1 not only the model relevant data is stored in XML. Last change time and position of each model element are also stored in many case tools. But if we compare Figure 1 and Figure 2 the only change really want to be aware of would be a change at the semantic level. Attributes are changing and a traffic light is gone. Therefore the meaning of this model changed. For a version control management system this information should not be relevant and hence be suppressed.

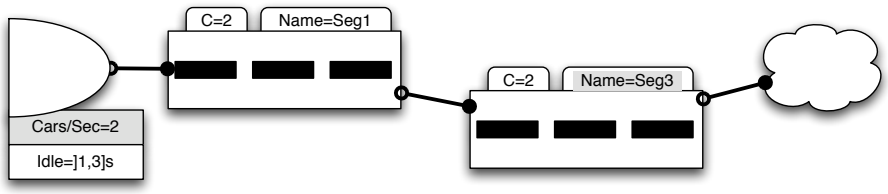

Fig. 2. Changed sample traffic net model from Figure 1. Only the grey colored attributes, and the absence of the traffic light are important.

In this approach I am documenting and testing my implementation of the X-Diff algorithm. A complete explanation of this algorithm can be found in the theory section. Since this API should help the improvement of the ATOM^3 tool, which is designed in Python, this implementation work was also done in python. The API is documented in the implementation section of this paper. It was also tested with several XML models, which is described in the analysis section. Furthermore this algorithm can be used in future for model difference computation, which I will discuss in the future work part.

2 Theory

2.1 XML

The Exensible Markup Language (XML) is basically used for creating customized markup languages. It allows its user to define their own elements and its primary purpose is to speed up the sharing of structured data across different systems. In most cases these information is shared over the internet. By adding semantic constraints even application languages can be implemented in XML, such as XHTML, RSS and Scalable Vector Graphics. It is a free open standart and recommended by the World Wide Web Consortium.

2.2 DOM trees

Any XML file can be represented as its Document Object Model (DOM) tree. Every node obect in this tree represents parts of the document contents. These trees are usually created by a parser or even manually. We are differentiating basically between three different kind of nodes that can occur in a DOM tree:

- **Element** nodes are non leaf nodes with one label, *name*
- **Text** nodes are leaf nodes with one label, *value*
- **Attribute** nodes are leaf nodes with two labels, *name* and *value*

The DOM specification defines that element- end text nodes are ordered and attribute nodes are not in a specific order. In most of the cases XML documents can be treated as unordered tree, since the order is not important and only the parent-child relationship should be of interest.

In the X-Diff definition only unordered trees are in the main focus. Furthermore X-Diff considers two trees equal if they are identical except for the ordering of siblings (isomorphic).

2.3 Edit Scripts

A difference between two trees can be represented as a script, which transforms the source tree to the target tree. This script is a sequence of operations of this three types:

- **Add** - this operation adds to the tree another node to a specified position
- **Delete** - this operation deletes from the three a node
- **Update** - this operation changes the value of a label of a node

It is also not necessary to define any move operations that would transfer one node or a subtree to another. Since we are using an unordered tree this move operation are in nearly all cases not necessary because the order among siblings is not important. Other move operations can be replaced by delete and add commands.

2.4 Cost Model

Of course, there can be many different edit scripts for transforming one tree to another. Therefore we need to define a cost model in order to evaluate and optimize the X-Diff algorithm in order to yield always to nearly optimal results. The cost model X-Diff uses is a simple cost model.

Definition 1. *E is a valid edit script. $Cost(E) = n$, where $E = O_1 O_2 \dots O_n$ and O_i is a edit operation defined in the previous section.*

Based on this definition we can now define a minimum cost edit script:

Definition 2. *Given an edit script E, E is a "minimum cost edit script" iff $\forall$ edit scripts E' transforming one DOM tree to another, $Cost(E) \leq Cost(E')$.*

2.5 Signature

In order to find a matching between two DOM trees is it obviously that it is not a good idea to compare each node of the source tree with each node in the target tree because each node in XML has its own context. A wrong matching would harm this context and cause more not needed computation. It is not enough to prevent that just to compare the node types, we have to compare at the full path of each node to its root to see if the two nodes are worth to be compared. The signature is now defines as a first criterion for matching two nodes.

Definition 3. *x is a node in a DOM tree, $Signature(x) = /Name(root_{Tree})/.../Name(x_n/Name(x)/Type(x)$. If x is a text node we drop the $Name(x)$ from the Signature since a text node is nameless.*

In this approach a signature is simply computed by concatenating the names of the nodes in the path up to the root node. We can guarantee with that that X-Diff is only comparing nodes that are worth to be compared.

2.6 Matching

We want to compute for our algorithm a minimal cost matching set, which contains all matched nodes from the two trees in order to generate from that our minimal cost edit script.

Definition 4. *We have a set of node pairs (x, y). M is a called a matching from the source tree T_1 to the target tree T_2, iff:*

1. $(x, y) \in M, x \in T_1, y \in T_2, Signature(x) = Signature(y)$.
2. $\forall(x_1, y_1) \in M$ *and* $(x_2, y_2) \in M, x_1 = x_2$ *iff* $y_1 = y_2$.
3. M *is prefix closed.*

The proofs for that can be found in X-Diff, p.523.

The matching algorithm is defined in the X-Diff paper and calculating based on the previous definitions a minimum matching set containing pairs of nodes. A part of this algorithm uses another algorithm called X-Hash which is able to compute hashes for isomorphic trees. Using this algorithm the complexity of checking every node pair is reduced, since only different Xhashes are compared. The matching algorithm also uses dynamix programming to compute the distance between two trees ($Dist(T_1, T_2)$). It begins with computing the editing distance from the leaf node pairs and moves upward to the root pairs. When the editing distance between subtrees is computed the algorithm ses the minimum-cost maximum flow algorithm to find the minimum cost bipartite mapping. Here is should be pointed out again, that XDiff is just comparing nodes that have the same signature and not all different possibilities. This differentiates it from the NP-complete proven case.

3 Implementation: Module XDiff

Author: Ulf Schwekendiek

Contact: ulf.schwekendiek@gmail.com

License: GPLv3

To Do: add XHash in order to minimize search complexity.

3.1 Functions

collectLeafNodes(*root*)

This method collects all the leaf nodes.

Parameters

`root:` The root node from where the algorithm starts searching for leaf nodes.

(type=Node)

collectNodes(*root*, *list*)

This method collects the child nodes.

Parameters

`root:` The root node from where the algorithm starts searching for leaf nodes.

(type=Node)

`list:` A list in which the nodes are saved.

(type=set)

removeNodes(*nodeList*)

This method removes all the nodes given by *nodeList* from the DOM-tree.

Parameters

`nodeList:` A list of nodes that should be removed.

(type=list)

signature(*node*)

This method computes the signature of a node.

For more infomation see: **X-Diff: An Effective Change Detection Algorithm for XML-Documents** Y. Wang et al, University of Wisconsin, WI, USA in Proceedings of the 19th Internation Conference on Data Engineerin (ICDE'03) page. 523

Parameters

`node:` *(type=Node)*

dist(x, y)

This method computes the computes the distance of two nodes.

Parameters

`y:` *(type=Node)*

`x:` *(type=Node)*

isRoot(x)

This method returns if a node x is a root node.

Parameters

`x:` *(type=Node 1)*

findMatching(*t1*, *t2*)

This method calculates a minimum cost matching.

For more infomation see: **X-Diff: An Effective Change Detection Algorithm for XML-Documents** Y. Wang et al, University of Wisconsin, WI, USA in Proceedings of the 19th Internation Conference on Data Engineerin (ICDE'03) page. 525

Parameters

`t1:` root node from the first DOM-XML-tree

(type=Node)

`t2:` root node from the second DOM-XML-tree

(type=Node)

Return Value

The minimum cost matching

(type=set)

To Do: Filter out next-level subtrees that have equal XHash values.

editScript(*t1*, *t2*, *M_min*, *script*)

This method calculates the editScript.

For more infomation see: **X-Diff: An Effective Change Detection Algorithm for XML-Documents** Y. Wang et al, University of Wisconsin, WI, USA in Proceedings of the 19th Internation Conference on Data Engineerin (ICDE'03) page. 527

Parameters

`t1:` root node from the first DOM-XML-tree

(type=Node)

`t2:` root node from the second DOM-XML-tree

(type=Node)

`M_min:` The previous calculated minimum cost matching set

(type=set)

Return Value

The Edit script

(type=set)

3.2 Variables

Name	Description
BAD_BOUNDARYPOINTS_ERR	**Value:** `1`
DOMExceptionStrings	**Value:** {`1: 'Index error accessing NodeList or NamedNodeMap', 2: ...`
EventExceptionStrings	**Value:** {`0: 'Uninitialized type in Event object'`}
FT_EXCEPTION_BASE	**Value:** `1000`
FtExceptionStrings	**Value:** {`1001: 'XML parse error at line %d, column %d: %s'`}

continued on next page

Name	Description
INVALID_NODE_TYPE_ERR	**Value:** 2
RangeExceptionStrings	**Value:** {1: 'Invalid Boundary Points specified for Range', 2: 'In...
UNSPECIFIED_EVENT_TYPE_ERR	**Value:** 0
XML_PARSE_ERR	**Value:** 1001
implementation	**Value:** <xml.dom.html.HTMLDOMImplementation.HTMLDOMImplementation...

3.3 Class Action

object
└ **XDiff.Action**

This class represents an action for changing one xml node to another.

Methods

__**init**__(*self*, *type*, *source*=None, *target*=None)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Parameters

`type:` Can be Action.UPGRADE_TYPE, Action.INSERT_TYPE or Action.REMOVE_TYPE

`source:` If *type* is either Action.INSERT_TYPE or Action.REMOVE_TYPE this variable is is set.

(type=Node)

`target:` If the type is Action.UPGRADE_TYPE this variable is also set

(type=Node)

Overrides: object.__init__

__**str**__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from object

__delattr__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__()

Properties

Name	Description
Inherited from object	
__class__	

Class Variables

Name	Description
UPGRADE_TYPE	**Value:** 1
INSERT_TYPE	**Value:** 2
REMOVE_TYPE	**Value:** 3
STR	**Value:** {1: 'UPGRADE_TYPE', 2: 'INSERT_TYPE', 3: 'REMOVE_TYPE'}

3.4 Class XDiff

object
└ **XDiff.XDiff**

This class is a wrapper for the XDIFF methods. It takes two xml files and creates the editscript in *self.editscript*.

Methods

__**init**__(*self*, *xmlfile1*, *xmlfile2*)
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
Overrides: object.__init__ extit(inherited documentation)

Inherited from object

__delattr__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __str__()

Properties

Name	Description
Inherited from object	
__class__	

Instance Variables

Name	Description
editscript	The computed editscript

4 Analysis

In order to analyze the implementation of the algorithm I created two different model. A simple bookstore and a traffic-net model. The bookstore has the following two versions:

```
Version A:
<BookStore>
  <Book id='1'>
    <isbn>1234</isbn>
    <title/>
```

```
  </Book>
  <book/>
</BookStore>

Version B:
<BookStore>
  <Book id='1'>
    <name>ABCD</name>
    <isbn>1234</isbn>
    <title/>
  </Book>
  <book>
    <name/>
  </book>
</BookStore>
```

I applied PyXDiff on this two different versions and got as the result this:

```
INSERT_TYPE, source='<Element Node at f9b760: Name='name' with 0 attributes and 1 childr
(sig:1nameBookBookStore), target='None' (sig:None)

REMOVE_TYPE, source='<Element Node at f99328: Name='book' with 0 attributes and 0 childr
(sig:1bookBookStore), target='None' (sig:None)

INSERT_TYPE, source='<Element Node at f9d5f8: Name='book' with 0 attributes and 1 childr
(sig:1bookBookStore), target='None' (sig:None)
```

As a second example I had a traffic net model:

```
Model 1:
<Trafficnet>
  <Source>
    <id>1</id>
    <name>source node</name>
    <script>SCRIPT1</script>
  </Source>
  <Segment>
    <id>2</id>
    <capacity>2</capacity>
    <name>Segment 1</name>
  </Segment>
  <Sink>
    <id>3</id>
    <name>sink node</name>
    <script>SCRIPT</script>
  </Sink>
```

```
  <Car>
    <id>C1</id>
    <velocity>90</velocity>
    <onSegment>2</onSegment>
  </Car>
  <Connection>
    <source>1</source>
    <target>2</target>
  </Connection>
  <Connection>
    <source>2</source>
    <target>3</target>
  </Connection>
</Trafficnet>

Model2:
<Trafficnet>
  <Source>
    <id>1</id>
    <name>source node</name>
    <script>SCRIPT1</script>
  </Source>
  <Segment>
    <id>2</id>
    <capacity>2</capacity>
    <name>new Segment 1</name>
  </Segment>
  <Sink>
    <id>3</id>
    <name>sink node</name>
    <script>SCRIPT</script>
  </Sink>
  <Car>
    <id>C1</id>
    <velocity>90</velocity>
    <onSegment>2</onSegment>
  </Car>
  <Connection>
    <source>1</source>
    <target>2</target>
  </Connection>
  <Connection>
    <source>2</source>
    <target>3</target>
```

```
  </Connection>
</Trafficnet>
```

The result was the following:

```
UPGRADE_TYPE, source='<Text Node at f8d300: u'1'>'
(sig:3sourceConnectionTrafficnet), target='<Text Node at fb74e0: u'2'>'
(sig:3sourceConnectionTrafficnet)

UPGRADE_TYPE, source='<Text Node at f8b3a0: u'2'>'
(sig:3targetConnectionTrafficnet), target='<Text Node at fb7d78: u'3'>'
(sig:3targetConnectionTrafficnet)

UPGRADE_TYPE, source='<Text Node at f8feb8: u'Segment 1'>'
(sig:3nameSegmentTrafficnet), target='<Text Node at fae878: u'new Segment 1'>'
(sig:3nameSegmentTrafficnet)

UPGRADE_TYPE, source='<Text Node at fa7738: u'3'>'
(sig:3targetConnectionTrafficnet), target='<Text Node at fb5f08: u'2'>'
(sig:3targetConnectionTrafficnet)

UPGRADE_TYPE, source='<Text Node at f8b9b8: u'2'>'
(sig:3sourceConnectionTrafficnet), target='<Text Node at fb5670: u'1'>'
(sig:3sourceConnectionTrafficnet)
```

Furthermore in order to verify the correctness of the implementation I implemented another method called $patch(tree, changeSet)$ which applies the calculated edit script to the tree.

4.1 Future Work

As I already mentioned in the previous chapters I was not able to find a proper implementation of XHash in Python. To speedup the algorithm the paper proposes the use of XHash, which calculates a hash value for a node in an isomorphic tree. I was not able to find such algorithm yet, so this needs to be also integrated in PyXDiff. Since PyXDiff should be used in order to detect differences in models it has to be more refined. Maybe it makes sense to not compute difference on models itself and rather map the model to its semantic domain and try to differentiate there.