# University of Alberta

Comparing XML Documents as Reference-aware Labeled Ordered Trees

by

Rimon A. E. Mikhaiel

The thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science


Fall 2011
Edmonton, Alberta

# Dedication

This thesis is dedicated to my lovely wife who has endlessly supported and encouraged me through the years of this research, and to my two blessed angels George and Daniel.

Also, I dedicate this thesis to my parents who have dreamed of seeing me holding a doctorate of philosophy.

--*Rimon Mikhaiel*

## Abstract

XML, the Extensible Markup Language, is the standard exchange format for modern Information Systems, Service Oriented Architecture (SOA) and the Semantic Web. Hence, comparing XML documents has become a necessary task for tracking and merging changes between versions of the same document, or for translating between documents referring to the same information but complying with different schemata or originating from different parties. In this scenario, given two documents, XML differencing is the process of finding an edit sequence, namely a sequence of exact and approximate matching, deletion, and insertion operations, which, if applied to the first document will result in the second. In practice, domain-specific differencing solutions are expensive to develop, and hard to reuse. Therefore, a generic differencing approach, able to serve various domains, would be both useful and cost-effective. This thesis presents *VTracker*, a generic XML differencing approach, which is capable of capturing domain knowledge and semantics through a configurable domain-specific cost function. VTracker views an XML document as an ordered labeled tree. Given two XML-document trees and a cost function VTracker calculates the tree-edit distance needed to transform one tree to the other. The first contribution of VTracker is an automatic method used to synthesize such a cost function based on the domain's XML Schema Definition (XSD). Second, VTracker considers the XML reference structure in addition to the natural XML containment structure. Third, VTracker implements an affine-cost policy that prefers edit operations applied to neighbors over dispersed elements. Finally, VTracker uses a set of simplicity heuristics to nominate the best edit script in case of multiple ones found with the same minimum cost. VTracker was applied to a variety of domains, namely OWL/RDF, WSDL, BPEL, UML/XMI, XHTML, and RNA secondary structure, where it performed competitively with, or even better than, state-of-the-art methods in each of these domains.

## Acknowledgement

This dissertation would not have been possible without the guidance and the help of several individuals who extended and contributed their valuable assistance in the preparation and completion of this study.

First of all, I owe my deepest gratitude to my supervisor, Professor Eleni Stroulia, whose support and guidance made this thesis possible. Her passion and encouragement have been my inspiration as I have overcome all the obstacles in the completion this research work.

My colleagues Marios Fokaefs, Nikolaos Tsantalis, and Natalia Negara whose constructive insights and feedbacks constituted valuable inputs towards the improvement of this research.

I would also like to thank my editor, Margaret Evans, who provided great help in reviewing the text of this thesis.

Last, but not least, my family, and the one above all of us, God the Almighty, who has given me strength and guided me through this journey. May His name be exalted, honored, and glorified.

--*Rimon Mikhaiel*

# Table of Contents

# List of Tables

# List of Figures

# List of Codes

# List of Symbols

| | |
|---|---|
| $T$ | An ordered labeled tree |
| $\|T\|$ | Size of tree $T$, and equals the index of the root of $T$. |
| $T[x_1..x_2]$ | A forest of nodes starting at node with index $x_1$ to node with index $x_2$. |
| $lm(x_i)$ | Index of left-most leaf child of node $x_i$ |
| $T[x_i]$ | Tree rooted by node $x_i$, which is equivalent to $T[lm(x_i)..x_i]$ |
| $l(x_i)$ | Label of node with index $x_i$, which is a symbol from an alphabet $\Sigma$. |
| $\lambda$ | A null node |
| $(x_i , y_i)$ | A matching edit operation between label of node $x_i$ to label of node $y_i$ |
| $(x_i , \lambda)$ | A deletion edit operation of node with index $x_i$ |
| $(\lambda, y_i)$ | An insertion edit operation of node with index $y_i$ |
| $\gamma(x_i , y_i)$ | Cost of matching label of node $x_i$ to label of node $y_i$ |
| $\gamma$ | Cost function |
| $\gamma'$ | Reference-aware Cost function |
| $\gamma(x_i , \lambda)$ | Cost of deleting node with index $x_i$ |
| $\gamma(\lambda, y_i)$ | Cost of inserting node with index $y_i$ |
| tdist | Tree-edit distance |
| fdist | Forest-edit distance |
| $context(x_i)$ | Set of nodes holding references to node $x_i$ |
| $\rho(x, y)$ | A similarity measure to determine whether label of node $x$ can substitute for label of node $y$. |

## List of Abbreviations

| | |
|---|---|
| AST | Abstract syntax trees |
| BPEL | Business Process Execution Language |
| BPM | Business Process Model |
| BULD | Bottom-Up Lazy-Down |
| CFG | Control- flow graphs |
| DOM | Document Object Model |
| DTD | Document Type Definition |
| ebXML | Electronic Business using XML |
| FMES | Fast Match Edit Script |
| H-mean | Harmonic mean |
| HTML | Hypertext Markup Language |
| IDE | Integrated Development Environment |
| LFG | Tight Fine-Grained (tree representation of RNA) |
| OAEI | Ontology Alignment Evaluation Initiative |
| OPF | Open Office Format |
| OWL-S | Web Ontology Language for Services |
| OWL-S | Web Ontology Language |
| RNA | Ribonucleic acid |
| SGML | Standard Generalized Markup Language |
| SML | Service Modeling Language |
| SOA | Service Oriented Architecture |
| SOAP | Simple Object Access Protocol |
| SPRC | Secondary and Primary RNA structure Comparison |
| TFG | Tight Fine-Grained (tree representation of RNA) |
| UDDI | Universal Description Discovery and Integration |
| UML | Unified Modeling Language |
| W3C | World Wide Web Consortium |

| | |
|---|---|
| WSDL | Web Service Description Language |
| WSFL | Web Services Flow Language |
| XID | Xylem Identifiers |
| XMI | XML Metadata Interchange |
| XML | Extensible Markup Language |
| XSD | XML Schema Definition |

# Chapter One Introduction

XML, the Extensible Markup Language, is the universal format for structured documents and data exchange on the World Wide Web. XML documents include embedded metadata that represent their logical and semantic structure and partially describe the behavior of computer programs that process them [147]. XML was conceived as a subset of Standard Generalized Markup Language (SGML), and was originally designed to facilitate the interoperability between SGML and Hypertext Markup Language (HTML) [19]. XML has now become the standard exchange format for modern Information Systems, and Lindholm states that "XML is the *lingua franca*[1] for information interchange, and will perhaps even surpass unstructured text someday" [89].

Many different types of data formats, specification languages, and interaction protocols are represented in XML. For example, XML is the *de facto* language for Service Oriented Architecture (SOA) technologies such as Universal Description Discovery and Integration (UDDI) [35], Simple Object Access Protocol (SOAP) [154], Web Service Description Language (WSDL) [155], Business Process Execution Language (BPEL) [36], Web Ontology Language for Services (OWL-S) [149], Electronic Business using XML (ebXML) [38] and Service Modeling Language (SML) [153]. XML is also the standard representation for Semantic Web technologies such as OWL [148] and RDF [151]. XML, nevertheless, has become the standard artifact data format in many other applications such as Open Office documents [131], SVG drawings [152], and XHTML documents [156], XSL, databases [140], Open Office Format (OPF) [24], and Open Office XML [113]. Additionally, XML is the standard exchange format for modeling metadata languages such as XML Metadata Interchange (XMI) [112].

---

[1] A language used for communication among people of different mother tongues

In each of these domains one encounters instances of differencing problems in the context of different activities. For example, XML document differencing [32][27][28][110][89] is very important for document management functions that include change detection and tracking, and version merging. A differencing problem sometimes is also called a comparison problem, or a matching problem. For example, in SOA, differencing is necessary for service discovery and for matching a requested service against a repository of advertised services, based on WSDL Matching [129][141][91][17], BPEL Matching [48][40], or OWL-S Matching [67]. Differencing is also necessary in SOA, for the purpose of automatic composition and integration of different services [82] [22], in addition to helping in the migration from one version to another, or from one service provider to another [23][53]. In the world of the Semantic Web, differencing plays a key role in the problem of ontology matching, which is essential for setting translation bases between vendors talking in terms of different ontologies [49][87][41][42][54][115]. Differencing is also a fundamental task in matching models such as Unified Modeling Language (UML). The latter is important for monitoring and tracking evolutions occurring to a certain model, or finding the proper mapping between elements of different models [161]. HTML differencing is necessary for automatic information extraction from the Web in order to be structured in an easy to process format [120][74][25][51][70] or even to automatically generate RSS feeds from sites of interest [90].

In most of the aforementioned application domains special-purpose methods have been developed to solve the differencing problem for these domains in particular, which is both expensive to build and hard to reuse. Other differencing methods, that rely on abstract syntactic representation and are not tied to a certain application domain, are usually incapable of capturing domain knowledge and semantics, and consequently are not able to produce results that are acceptable to subject-matter experts. ***The research problem then becomes the development of a general method for comparing XML documents for application to all of these domains, while at the same time, ensuring that the***

***method is aware of the domain-specific semantics, so that the reported differences correspond to domain benchmarks.***

This thesis presents VTracker, a generic XML differencing approach that is capable of capturing domain knowledge and semantics through a configurable domain-specific cost function. VTracker views an XML document as an ordered labeled tree. Given two trees and a cost function, therefore, VTracker calculates the tree-edit distance to transform one tree to another. VTracker is an extension of the Zhang-Shasha's tree-edit distance algorithm [166]. This thesis makes the following contributions to the state of the art.

With respect to the original algorithm for differencing ordered labeled trees, VTracker is innovative in two aspects. First, the original algorithm is (a) extended to consider an XML reference structure on top of the natural XML containment structure, (b) equipped with an affine-cost policy that promotes edit scripts that group edit operations in neighbors, and (c) associated with a set of heuristics for choosing the optimal edit script among multiple ones with the same cost. Second, with respect to the application of the ordered labeled tree-differencing paradigm to domain-specific differencing, VTracker develops a method for bootstrapping the algorithm in a domain. This is performed by automatically synthesizing a domain-aware cost function based on the underlying XML Schema Definition (XSD). VTracker was applied in five different domains: (a) OWL/RDF, (b) WSDL, BPEL, (c) UML/XMI, (d) XHTML, and (e) RNA Secondary Structure, and its performance is similar, or even better than, state-of-the-art methods in each of these domains.

The rest of the dissertation is structured as follows. Chapter Two presents various instances of the XML differencing problems, which constitute the motivation for this thesis. Chapter Three defines the requirements for a generic XML differencing approach, explains the original algorithm, and presents VTracker as a generic XML differencing method. Chapter Four explains the methodology of applying VTracker to a specific domain, and the necessary

configuration needed for VTracker to become domain-aware. Chapter Five presents the empirical evaluation results of how VTracker performed in various application domains. Finally, Chapter Six provides a discussion and the conclusion of the thesis.

# Chapter Two Background and Related Work

The problem of XML differencing has been studied in the context of many application domains. This chapter discusses instances of the differencing problem in a variety of domains, current state-of-the-art approaches to addressing the differencing problem, and their implications to this work. Differencing methods can be divided into two broad categories: general XML differencing and domain-specific differencing. The approaches in the former category aim to be so generic that they can compare any kind of XML document regardless of the underlying application domain. The approaches in the latter category are aware of the knowledge and semantics of the underlying domains, and are built to serve such domains in particular.

## 2.1 XML

XML differencing is defined as the process of finding proper mapping between elements of the two documents in order to detect changes, deletions, and insertions. The input consists of two XML documents, and optionally the Document Type Definitions (DTDs) or XSDs to which they conform. The output is an edit script that can transform one document into the other, in conjunction with a similarity measure between the two documents, called *edit-distance*.

XML-document differencing is necessary for version management functions such as change detection and tracking [144][6][109], version merging [32][27][28][110][89], indexing, and answering temporal queries [97]. Some applications have the luxury of recording the changes as they happen through the XML document editor, or an Integrated Development Environment (IDE), which is then utilized to produce the differencing results. However, a general XML differencing method should not rely on the assumption that editing and changes happen through a certain editing utility, or that the edit operations are consistently recorded as they happen.

XML differencing methods can be divided into two main categories based on whether they use a tree-to-tree correction model or not. It is essential to keep in mind that the approach proposed in this thesis is based on the tree-to-tree

correction paradigm. Therefore, the first category constitutes the more closely related work.

### *2.1.1 Non Tree-based Approaches*

The most basic XML differencing approach is simply to compare the textual content of these documents through a string-edit distance approach such as the UNIX *diff* command. Applying the diff command on the two documents, shown in Figure 2-1 (a), would report that one line was deleted and four new lines where inserted as shown in Figure 2-1 (b). However, Figure 2-1 (c) shows a more intuitive result that both documents have the same content with a new element "name" inserted. To deliver such a more "natural" comparison result one would have to recognize the internal tree structure of XML documents.

Inspired by string-edit distance, Lindholm et al. in Faxma and Faxma+ [89] transform an XML document into a sequence of events through a depth-first traversal, and then apply a sequence-based matching measure similar to a Levenshtein string-edit distance [86]. Calculating a string-edit distance is much cheaper than that of a tree-edit distance; however, representing an XML document as a sequence loses valuable details about the structural relationships between these elements such as the ancestor-child or sibling relationships.

In X-Diff [140], Wang et al. use X-Path queries to determine the similarities between nodes from different documents. Two elements are considered similar if they have the same X-Path signature. If one node is moved from its parent to another parent in the same level X-Diff will not recognize such a change. Additionally, X-Diff cannot detect changes that happen in the relative order of elements. In addition, if an internal node is deleted (or inserted) the entire sub-tree will be detected as deleted (or inserted) as well since its X-Path signature will be different.

```
<root>
 <person id ="1">John Smith</person>
</root>
```

```
<root>
 <person id ="1">
        <name>John
        Smith</name>
 </person>
</root>
```

**(a) Two XML documents**

```
2c2,5
<  <person id ="1">John Smith</person>
---
>  <person id ="1">
>        <name>John
>        Smith</name>
>  </person>
```

**(b) Results of Unix diff command**

```
-<root >
  -<person id="1">
    <name>John Smith</name>
  </person>
 </root>
```

**(c) Desired diff results**

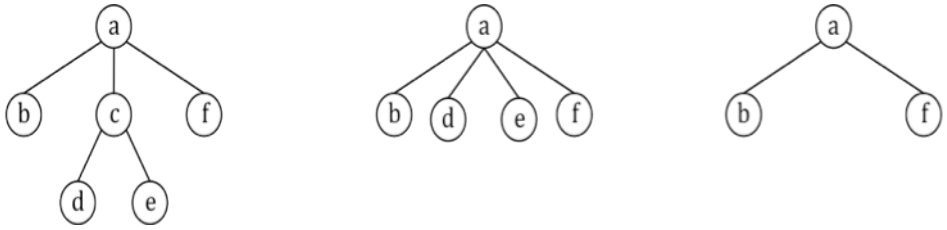
**Figure 2-1: An example of XML differencing using Unix *diff* versus an XML differencing mechanism**

### *2.1.2 Tree-based Approaches*

By nature an XML document can be represented as an ordered labeled tree in a very similar manner to a Document Object Model (DOM) representation of an XML document. In that sense, the XML differencing problem can be formulated as a tree-to-tree correction problem where the objective is to find the cheapest (i.e. most optimal) script of edit operations such as change, deletion, and insertion that transform one tree into the other.

The roots of the tree-to-tree correction problem can be traced back to 1977 and 1979 when Selkow [125] and Tai [134] published their work, respectively. In 1989, Zhang and Shasha published their tree-edit distance algorithm [166][168], which is based on Tai's model. Given two trees, tree-edit distance is the minimum

cost sequence of edit-operations that transforms one tree into the other. The set of possible edit operations in Tai's model is different from those in Selkow's model. On one hand, Tai's model allows fine-grained edit operations, namely to *change* label of a single node*, delete* an existing single node, or *insert* a new single node; on the other hand, Selkow's model allows coarse-grained edit operations, like *change* label of a single node, *delete* an entire sub-tree, or *insert* an entire sub-tree. Figure 2-2 illustrates the difference between the two models, where Selkow deals with entire sub-trees while Tai deals with single nodes. Selkow's model is appropriate for differencing applications where changes always happen to leaf nodes, or where a change to an internal (i.e. non-leaf) node implies a change to the entire sub-tree under this node. However, as it offers fine-grained edit operations, Tai's model allows for changes to happen anywhere in the tree without affecting the status of other nodes. In this way, Tai's model is more general than Selkow's. For the same reason, Tai's is more expensive as it requires comparing trees at the node level while Selkow's needs to compare trees at the sub-trees level. Therefore, XML differencing approaches based on tree-edit distance are recommended to use Tai's model as it offers fine-grained operations, and it allows changes to happen anywhere in an XML document not only at the leaves level.




**(a) Original Tree** **(b) Tai's model** **(c) Selkow's model**

**Figure 2-2: Selkow's Tree-to-Tree Editing model versus Tai's Tree-to-Tree Correction model with regard to deletion of a node c**

Generic XML-differencing approaches are the ones that are not tied to a certain application domain, and do not include heuristics, customizations, workarounds, or assumptions that are limited to a certain application domain. The following are the most relevant related works.

One of the most popular concepts in the literature of XML differencing is the work of Cobéna et al. named XyDiff [32]. This method is known for being efficient in terms of speed and memory space, and views an XML document as an ordered labeled tree. Intuitively, the algorithm starts by trying to detect large sub-trees left unchanged between the old and new versions. Then, the algorithm tries to match more nodes by considering ancestors and descendants of matched nodes and taking labels into consideration. The core idea of the XyDiff algorithm is to identify nodes using hash values called Xylem Identifiers (XIDs), and then to perform a greedy search for common sub-trees through an algorithm that is called Bottom-Up Lazy-Down (BULD) with complexity $O(n \log n)$. One limitation of XyDiff is that it is only efficient in comparing versions of the same document; otherwise it loses the advantage of skipping large unchanged trees in which case it must compare the two entire trees. Additionally, the concept of XID is not applicable to documents originated from different sources, since the document structure will be different, and consequently the elements will have different XID hash values. Finally, the XyDiff views XML documents as mere structured chunks of data. It does not consider the application semantics that might be captured within these structures.

Another important contribution, in the XML differencing literature, is the work of Chawathe et al., on Fast Match Edit Script (FMES) [27] and MH-DIFF [28]. FMES views an XML document as an ordered labeled tree. It aims at calculating an edit distance between two given trees. It starts with globally detected nodes that have a perfect or close match regardless of the structure or relative order. It then tries to make both trees structurally isomorphic by detecting deletion and insertion operations. The benefit of this approach is that it does not assume object-identifiers (e.g. XIDs in XyDiff), and that the cost function used to

measure matching between nodes is customizable to reflect the domain's knowledge and semantics. However, this method has the following drawbacks: (1) it cannot be made to detect structural changes since it matches nodes in a global manner without taking into consideration structural relationships between them; (2) consequently, it does not produce sound results in the case of documents with different structure, or those originated from different sources, and (3) there is no easy way to build such a domain-specific cost function. MH-Diff aims at detecting meaningful edit operations such as *copy* and *glue* in addition to standard change, deletion, and insertion. It arranges nodes from the first tree linearly against nodes from the second tree in a bipartite graph. In this graph, if an edge links a node $n$ in $T_1$ to node $m$ in $T_2$, then node $n$ is matched to node $m$. A node linked to special node "+" indicates that this node was inserted, and node linked to special node "-" indicates a deleted node. If a node in $T_1$ is linked to multiple nodes in $T_2$, then this node was copied multiple times. Similarly, multiple nodes in $T_1$ linked to a single node in $T_2$ indicate that these nodes were glued into that single node. In this way, the differencing problem is formulated as finding an edge cover in the induced graph. In spite of the advantages of MH-Differencing over FMES, both still suffer the same drawbacks. Both detect edit operations in a global manner while disregarding the locality of different sub-trees, and consequently the more structurally different are the two given trees, the more non-sense results both methods will produce. Finally, neither method is designed to compare documents originating from different sources or vendors.

A further contribution to the XML differencing techniques is the work of Nierman and Jagadish [110]. This work also views an XML document as an ordered labeled tree, and the objective is to calculate the tree-edit distance between two given trees. The new contribution of this model is that it combines operations of the Selkow's coarse-grained model and Tai's fine-grained model. This method reports five kinds of edit operations: *change* label of a single node, *delete* a single node, *insert* a single node, *delete a sub-tree*, and *insert a sub-tree*. This approach is better than previous methods in terms of generality since it will

compare structurally different documents as well as documents originated from different source. However, the efficiency of this method largely depends on the structural similarity of the two trees. In other words, if the two trees are structurally different, its actual complexity is significantly inferior to standard tree-edit distance algorithms (e.g. Zhang-Shasha's).

Another application of XML matching is answering twig queries where the objective is to find a small query tree inside a large XML document. An answer to a twig query is a list of sub-trees that match the small query tree in terms of their contents and structure. The approximate query answer is a list of ranked sub-trees that partially match the query tree content and structure. One method of solving this problem is the work of Vagenza et al. [137] that views both the query and the large XML structures as two directed acyclic graphs, and then it measures the structural similarities between the query and various sub-trees. Another interesting method to this problem is the work of Augsten et al. [14] that is based on pruning irrelevant sub-trees, and then applying a tree-edit distance on a small set of candidate sub-trees. This work uses a prefix ring buffer approach to perform a single scan in order to prune sub-trees that exceed a certain size threshold, or are contained in their relevant sub-trees.

The literature of XML differencing is rich with many other approaches both similar to, and different from, the ones presented above. For example, Microsoft's "XML Diff and Patch" is based on the Zhang-Shasha algorithm; DiffMK, by Sun Micro Systems, is based on the Unix diff algorithm. DiffXML is based on FMES, and JXyDiff is a Java implementation of XyDiff. Mlýnková's [106] work and research combines the work of both Nierman [110] and XClust [85] together. Additionally, XClust X-Differencing is based on X-Diff and allows for some domain-specific customization. For additional details, the reader is recommended to review the surveys of Peters [117] and Cobéna [31].

## 2.2 Ontology

The World Wide Web Consortium (W3C) defines ontology as a set of "formalized vocabularies of terms, often covering a specific domain and shared by a community of users" [148]. Ontologies are important to formally describe a certain domain's terminologies, vocabularies, concepts, and relationships. An ontology description usually defines elements such as individuals (i.e. objects), classes, attributes, relationships, or restrictions on relationships. One important ontology-description language is Web Ontology Language (OWL) [148] that is designed to serve the needs of Semantic Web and Service Oriented Architecture [149].

Ontology matching is a task necessary for a variety of activities such as migration and bridging between various versions and evolutions of the same ontology, translation between different ontologies, discovery and composition of services, integration of software systems, and linking web-accessible data. In nearly every scenario where software components of different parties need to interact, it is necessary to translate between their underlying ontologies. The term "ontology matching" refers to the problem of identifying the proper semantic mapping between entities of different ontologies representing the same conceptual domains. The general technical problem driving the research around ontology matching is part of the overall Semantic-Web agenda, which envisions that the information available on the web will be annotated with semantic metadata in the form of ontology tags, and that heterogeneous information, provided by people and organizations will be integrated through mapping of their tag ontologies. As centralized coordination of the ontology-development process is unlikely, one can anticipate – and we are already witnessing – an explosion in the number of ontologies used today. Many of these ontologies describe similar (the same or overlapping) domains, but use different terminologies. To integrate data from such disparate ontologies one must recognize the semantic correspondences between their elements. Manual mapping of such correspondence is time-consuming, error prone, and clearly not possible on the web scale [30]. This is

why general, applicable across domains, automated methods for ontology mapping are necessary.

### *2.2.1 Related Work*

In principle, there are three categories of ontology-matching methods [30]. Some methods attempt to construct and maintain a global ontology based on several local ontologies, by describing and mapping the relationships between the elements of the local and global ontologies. Other methods focus on enabling interoperability in distributed environments and mediating between the distributed data in such environments by pair–mapping. Finally, a third family of methods is designed to map a set of overlapping ontologies through ontology merging and alignment.

The Ontology Alignment Evaluation Initiative (OAEI) organizes an international competition between Ontology Matching systems and frameworks. Every year OAEI appoints the top state-of-the-art approaches in that domain. According to the published results of OAEI 2010 "ASMOV and RiMOM are ahead, with as close follower, while SOBOM, GeRMeSMB and Ef2Match, respectively, had presented intermediary values of precision and recall" [50].

The ASMOV algorithm [49] iteratively calculates the similarity between entities for a pair of ontologies by analyzing four features: lexical elements (id, label, and comments), relational structure (ancestor-descendant hierarchy), internal structure (property restrictions for concepts; types, domains, and ranges for properties; data values for individuals), and extension (instances of classes and property values). The measures obtained by comparing these four features are combined into a single value using a weighted summation formula.

The RiMOM approach [87] uses three matching strategies. One is the name-based strategy which calculates the string-edit distance between the labels of two entities. The second is the metadata-based strategy that calculates the cosine distance between weighted feature vectors representing the words contained in the entity's label and comment. And the third is the instance-based

strategy that constructs another document for each entity consisting of the words in the instances related to that entity.

The AgreementMaker [42] comprises several matching algorithms, or matchers, that are either (1) concept-based matchers which are a combination of string matchers and a cosine distance matcher, or (2) structural matchers that make sure that if two nodes are similar, then their descendants should be also.

SOBOM [162] incorporates anchor generator matchers that use textual information such as label, id, and comments in addition to structural information such as number of super- and sub-concepts, the number of constraints. It then uses a structural matcher that uses anchors to induce the construction of similarity propagation graphs for sub-ontologies. Finally, it uses what it calls an R-matcher that matches the definitions based on the linguistics and semantics of relations.

GeRMeSMB [72] is composed of two modules, GeRMeSuite and SMB. GeRMeSuite is a generic matching framework that can match ontologies as well as schemas in other modeling languages. SMB is a 'meta' matching system that works on the similarity matrices produced by GeRMeSuite. It fine-tunes the clarity of the similarity values by improving 'good' values and decreasing 'bad' values.

In addition to the OAEI contestants there are other related works. For example, the work of Giunchiglia et al. [54] is based on the edit distance between matched ontologies in order to preserve relative structural relationships between matched elements. Another related work is Papavassiliou et al. [115] that aims at detecting meaningful changes between Ontology RDFs through a new set of meaningful edit operations.

In general, ontology matching systems consider the following features as an essential part of their approaches: (1) conceptual identity in terms of id, label, and comments, (2) structural identity in terms of inheritance and composition relationships, and (3) other relationships including dependency, association, and instantiation.

### *2.2.2 An Ontology as a Tree*

The scope of this thesis focuses on OWL as an example of ontology description language. According to W3C 2009 specification the primary exchange format for OWL2 is RDF/XML [148]. Code 2-1 shows a portion of an OWL Ontology described in RDF/XML syntax. This example is a part of the reference ontology specification used in the OAEI benchmark dataset. As shown in this example, the OWL ontology is composed of a set of classes, object properties, and individual objects. Each of those elements is then defined either in its own terms, or by referring to definitions of other elements.

As an XML document an OWL ontology specification can be represented as a tree. VTracker is based on a DOM model as a tree representation of an XML document where all XML elements such as classes, properties, relationships and restrictions are represented as tree nodes, and where element names are represented as tree labels and XML attributes are represented as node attributes. Metadata elements such as XML instructions and comments are not included in the tree model. For example, Figure 2-3 illustrates this idea by visualizing the ontology described in Code 2-1 as a tree. In this tree the ontology defines two classes named *Article and Part*, one object property named *author*, and one individual with id *a492378321*. Similarly, the *Article* class definition is composed of a *label*, a *comment*, and two inheritance relationships: the first is a restricted version of the *author* object property while the second is a normal sub-class relationship of *Part* class definition. In conjunction with the XML containment structure, this example illustrates another type of structure that is called the *reference structure*. In Figure 2-3 the solid lines denote containment relationships while the dotted arrows denote reference relationships. In this tree there are three reference relationships: an instantiation relationship between article *#a492378321* and the class definition of *Article*, and two association relationships

```
<rdf:RDF>
 <owl:Class rdf:ID="Article">
  <rdfs:label xml:lang="en">Article</rdfs:label>
  <rdfs:comment xml:lang="en">An article from a journal or
                       magazine.</rdfs:comment>
  <rdfs:subClassOf>
   <owl:Restriction>
    <owl:onProperty rdf:resource="#author"/>
    <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
    </owl:cardinality>
   </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="#Part"/>
  </owl:Class>

 <owl:ObjectProperty rdf:ID="author">
  <rdfs:subPropertyOf rdf:resource="#humanCreator"/>
  <rdfs:label xml:lang="en">author</rdfs:label>
  <rdfs:comment xml:lang="en">The list of the author(s) of a
                       work.</rdfs:comment>
 </owl:ObjectProperty>

 <owl:Class rdf:ID="Part">
  <rdfs:subClassOf rdf:resource="#Reference"/>
  <rdfs:label xml:lang="en">Part</rdfs:label>
  <rdfs:comment xml:lang="en">A part of something (either Book
                       or Proceedings).</rdfs:comment>
  <rdfs:subClassOf>
   <owl:Restriction>
    <owl:onProperty rdf:resource="#pages"/>
    <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1
    </owl:maxCardinality>
   </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
   <owl:Restriction>
    <owl:onProperty rdf:resource="#title"/>
    <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
    </owl:cardinality>
   </owl:Restriction>
  </rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#isPartOf"/>
   <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
   </owl:cardinality>
  </owl:Restriction>
 </owl:Class>
</rdf:RDF>
```

**Code 2-1: A sample Ontology OWL/RDF described in XML syntax**

linked to the definitions of the *Part* class and the *author* object property. The intent of such a reference structure is to allow ontological definitions to be reused within other definitions. This kind of hyperlinkage dramatically affects the semantics of an element's definition. Although the referenced element definition is not physically a part of the referring structure, it is definitely a part of its semantics. Therefore, when definitions of two elements are matched to each other it is not enough to only match the containment structure on both sides but also the referenced structures as well. In other words, a differencing approach should consider referenced structures as being a part of the referring structure.

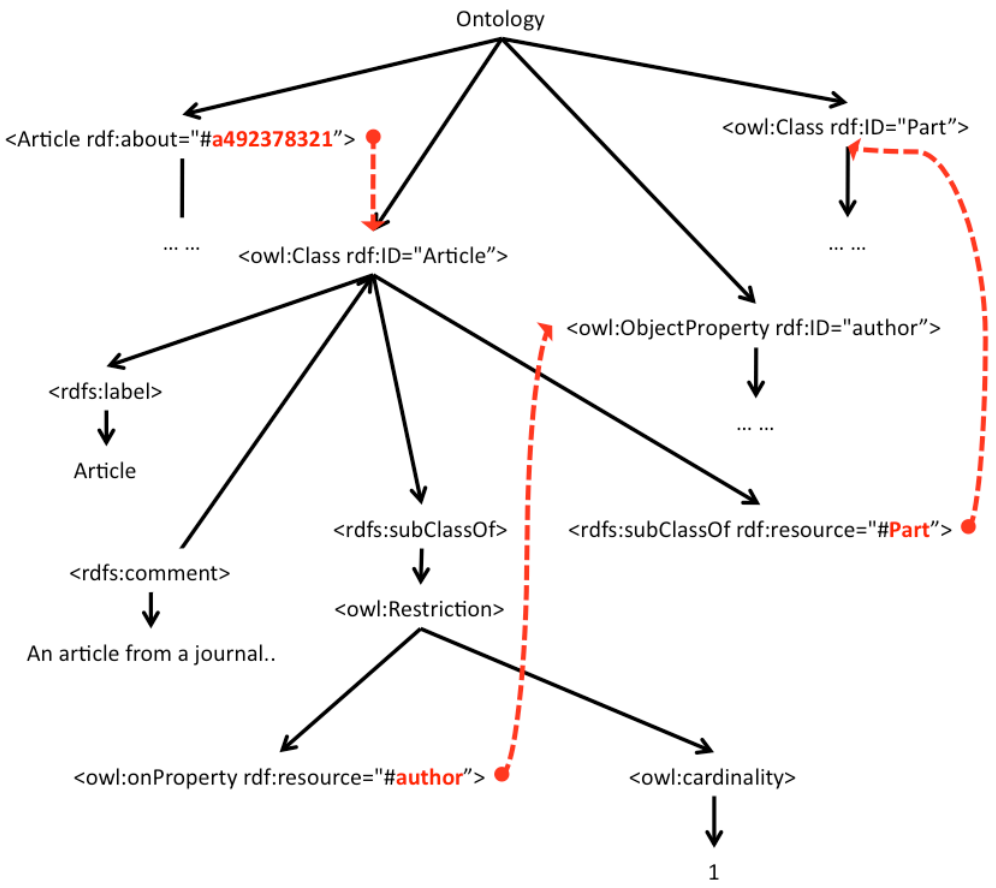



**Figure 2-3: A sample OWL/RDF ontology represented as a tree Structure including Ontology Classes, Properties, and Instances.**

## 2.3 WSDL

Service discovery is an essential task in the process of developing service-oriented applications. In a typical service-discovery scenario the service requester has specific expectations about the candidate service. In general, there are three types of desiderata for a service: it has (a) to be *capable* of performing a certain task, i.e. maintain a shopping cart, (b) to expose a particular *interface*, i.e. provide view, add-product and remove-product, and (c) to *behave* in a certain manner, i.e. ignore any request for product removals if no product additions have been performed yet. Such expectations motivate and guide the developers' searches through web-services repositories, as they try to discover and select the service that best matches their needs. This thesis does not target the capability-matching problem since it is always done at the UDDI level, which is not a particularly challenging problem. However, this thesis focuses on WSDL matching as an example of interface matching, and on BPEL matching as an example of behavior and protocol matching problems

A WSDL specification is the description of a software component that includes a description of its interface, a description of where the actual implementation exists, and how it can be used [129][141]. W3C defines services:

> *"As collections of network endpoints, or ports. In WSDL the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions: messages, which are abstract descriptions of the data being exchanged, and port types, which are abstract collections of operations. The concrete protocol and data format specification for a particular port type constitutes a reusable binding. A port is defined by associating a network address with a reusable binding, and a collection of ports define a service".*

WSDL matching is the process of finding a proper mapping between elements of two specifications that maximizes the overall matching and

minimizes the number and cost of edit operations required to transform the first WSDL to the second one. An important application of WSDL matching is service discovery and matching. In general, when looking for a service, a developer has in mind both the signatures of the operations desired, and some behavioral scenarios in which the candidate service is expected to participate. WSDL matching is then responsible for mapping a desired set of desired operations against a set of provided ones, their inputs, outputs, data types, etc. Then the objective is to measure the distance between the desired service WSDL and the published ones, and to find the closest one to the desired interface The objective is first to find the best published service, and then to find the proper mapping between different elements of both interfaces such as data types, messages, operations, and ports.

Another application of WSDL matching is to increase service reusability by allowing a service to be consumed in multiple use-cases, not only the one it was designed for [102]. For example, consider the case of an *Album Catalog* service in Figure 2-4 against a consumer interested in a *Book Catalog* service Figure 2-5. In such cases, although the published service deals with different concepts than that of the service consumer, the consumer could still effectively use the service, if only a mapping between the divergent schema elements were found. For example, in this scenario, both services register items, search the catalog, get item details, and find other items from the same producer or publisher. It would, therefore, be desirable to discover the published service in response to such a request if no better match is available.

WSDL matching is also important for version migration where the objective is to precisely recognize the changes to the WSDL specification of a service interface, and (a) find a proper mapping between elements of the old interface and those of the new one, (b) characterize the changes in terms of their complexity and (c) semi-automatically develop adaptors for migrating clients from older interface versions to newer ones [53].

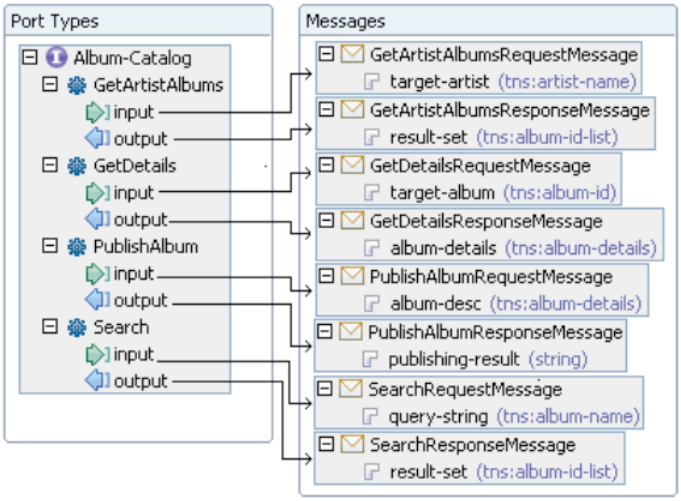



**Figure 2-4: A WSDL specification of a published online Album Web Service**

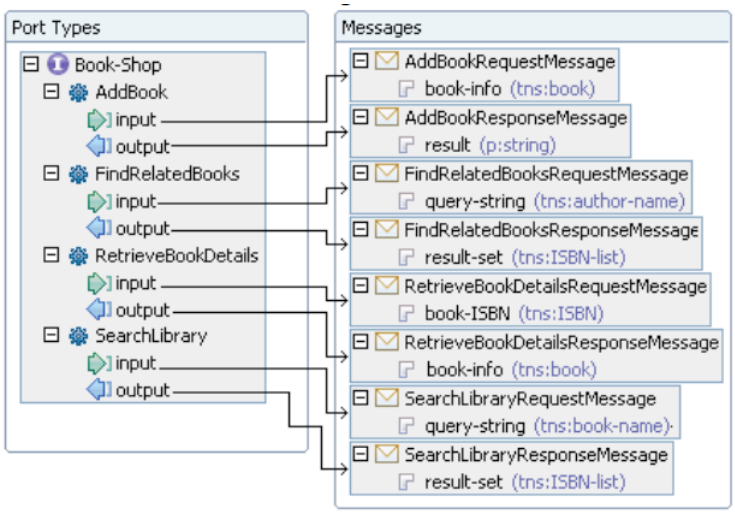



**Figure 2-5: A WSDL specification of a desired online Book catalog Web Service**

### *2.3.1 Related Work*

Interface matching is concerned with mapping the elements of a candidate published interface to the elements of the requested one. Usually, such mapping is based on signature matching between the published operations and the requested ones. It matches the input and output parameters in addition to pre- and post-

conditions [116]. For example, Wang and Stroulia [141] proposed a family of WSDL matching methods that consider both the identifier and structural similarity of data types and methods. Payne et al. [116] developed a DAML-S matching method, assuming a common ontology between the publisher and the requester, based on parameter matching using type subsumption and inheritance relationships. Syeda-Mahmood et al. [132] proposed an interface matching approach based on name similarity.

It is worth mentioning that most of the WSDL matching techniques suffer from two drawbacks: first, interface matching does not guarantee a successful interaction because such an interface usually does not specify the usage conditions of the operations involved. Hence, an improper usage of the published operations will lead to an interaction failure. Second, interface matching may easily become confused when services are not distinctive when the data types are simple, and when there is not much documentation. Both problems were addressed by the author in the context of examining usage protocols for service discovery through a mixed approach that incorporates both WSDL matching with BPEL matching in an integrated way [102].

### *2.3.2 A WSDL specification as a Tree*

A WSDL specification is an XML document by nature. A WSDL specification, therefore, can be easily represented as a partially ordered labeled tree. For example, the XML document shown in Code 2-2 describes the "evSoap" service that is represented as a tree in Figure 2-6. As shown in this Figure, a WSDL tree is composed of few main sub-trees: a set of data type definitions, a set of API messages signatures, a set of port types, a set of bindings, and finally a service specification sub-tree. Like ontology specifications, a WSDL specification largely depends on the concept of the reference-structure as an efficient way of reusing element definitions such as XML Schema definitions, messages, operations, etc. Intuitively, references are used to avoid duplicate definitions.

```
<definitions>
 <types>
  <s:schema>
   <s:element name="VerifyEmailResponse">
    <s:complexType>
     <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="VerifyEmailResult"
            type="s0:ReturnIndicator"/>
     </s:sequence>
    </s:complexType>
   </s:element>
   <s:element name="VerifyEmail">
    <s:complexType>
     <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="email" type="s:string"/>
     </s:sequence>
    </s:complexType>
   </s:element>
  </s:schema>
 </types>
 <message name="VerifyEmailSoapIn">
  <part name="parameters" element="s0:VerifyEmail"/>
 </message>
 <message name="VerifyEmailSoapOut">
  <part name="parameters" element="s0:VerifyEmailResponse"/>
 </message>
 <portType name="evSoap">
  <operation name="VerifyEmail">
   <input message="s0:VerifyEmailSoapIn"/>
   <output message="s0:VerifyEmailSoapOut"/>
  </operation>
 </portType>
 <binding name="evSoap" type="s0:evSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
        style="document"/>
   <operation name="VerifyEmail">
    <soap:operation soapAction="http://ws.cdyne.com/VerifyEmail"
            style="document"/>
    <input>
     <soap:body use="literal"/>
    </input>
    <output>
     <soap:body use="literal"/>
    </output>
   </operation>
 </binding>
 <service name="ev">
  <port name="evSoap" binding="s0:evSoap">
   <soap:address location="http://www.cdyne.com/emailverify/ev.asmx"/>
  </port>
 </service>
</definitions>
```

**Code 2-2: A service WSDL specification described in XML syntax**

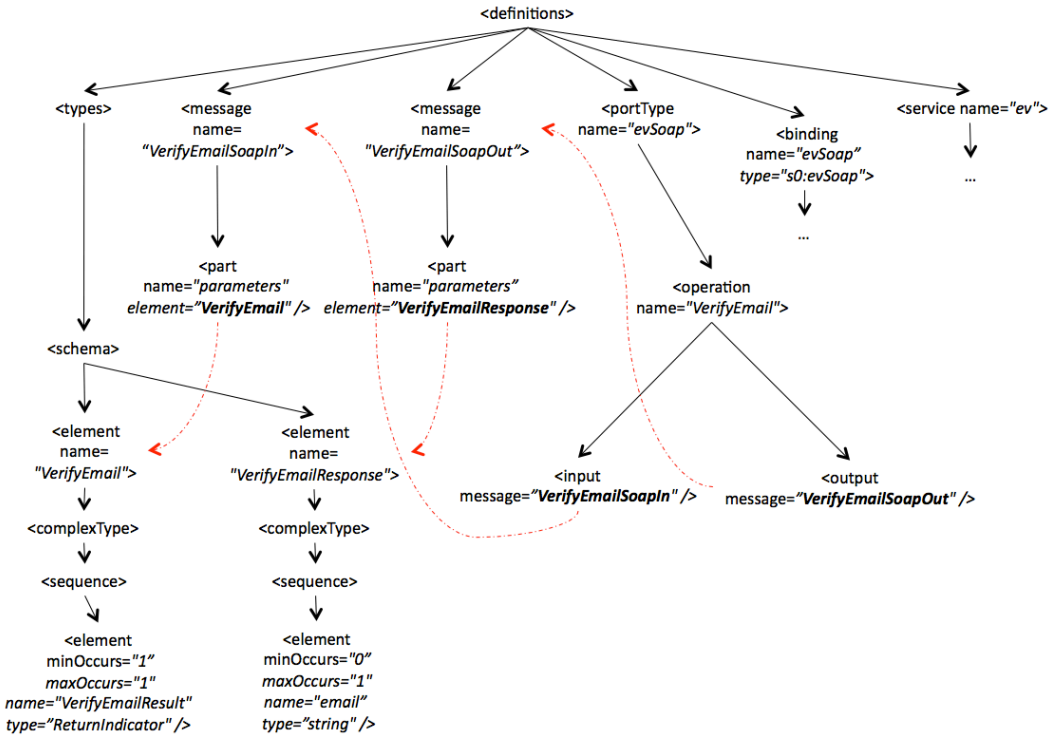



**Figure 2-6: A WSDL specification represented as a tree structure.**

## 2.4 BPEL

This thesis focuses on Business Process Execution Language (BPEL) as an example of workflow description language[2] [36][34]. *BPEL* is an XML-based language created for designing, composing, and executing web services. BEA Systems, IBM, and Microsoft, developed the BPEL specifications. It combines and replaces IBM's Web Services Flow Language (WSFL) and Microsoft's XLANG specification. BPEL cooperates with WSDL messages, XML Schema type definitions, and XPath data manipulation. BPEL can be used to describe either public *business protocols* that capture the exchange behavior of each of the parties involved in the protocols without revealing their internal behavior; or

[2] BPEL is also sometimes identified as BPELWS or BPEL4WS.

private *executable business processes* that model the actual behavior of each participant in business interactions that are private, but not publicly, visible. In essence, it provides the common core of process description elements, but can be extended to handle specific situations or concepts [36][34]. For example, Figure 2-7 shows a workflow that can possibly be attached to the WSDL interface of the online album described in Figure 2-4.

The task of comparing business process model (BPM) specifications to recognize similarities and differences is ubiquitous. It is necessary for discovering business processes that provide desired behaviors, for pinpointing changes between subsequent versions, and for verifying conformance of processes against desired protocols. Differencing business process models is a critical feature for integration development [77]. In a collaborative environment different team members concurrently manipulate shared process models that result in different versions of the same original process model. In this case, a process model integration technique is needed to smoothly merge all the versions together into an integrated process model. Basically, this technique is required to align unchanged model elements, and integrate all the changes accordingly. Differencing business process models is also useful for behavior matching in service-discovery [102]. In a typical service-discovery scenario, the service requester is looking for a service to complete a composite application and has specific expectations about the candidate service. In general, there are three types of desired outcomes for a service. It should deliver a certain function, expose a particular interface and behave in a desired manner. The first and second aspects, namely service functionality and interface, are usually checked through UDDI and WSDL matching techniques, while service behavior, the third aspect, needs a business process differencing technique.

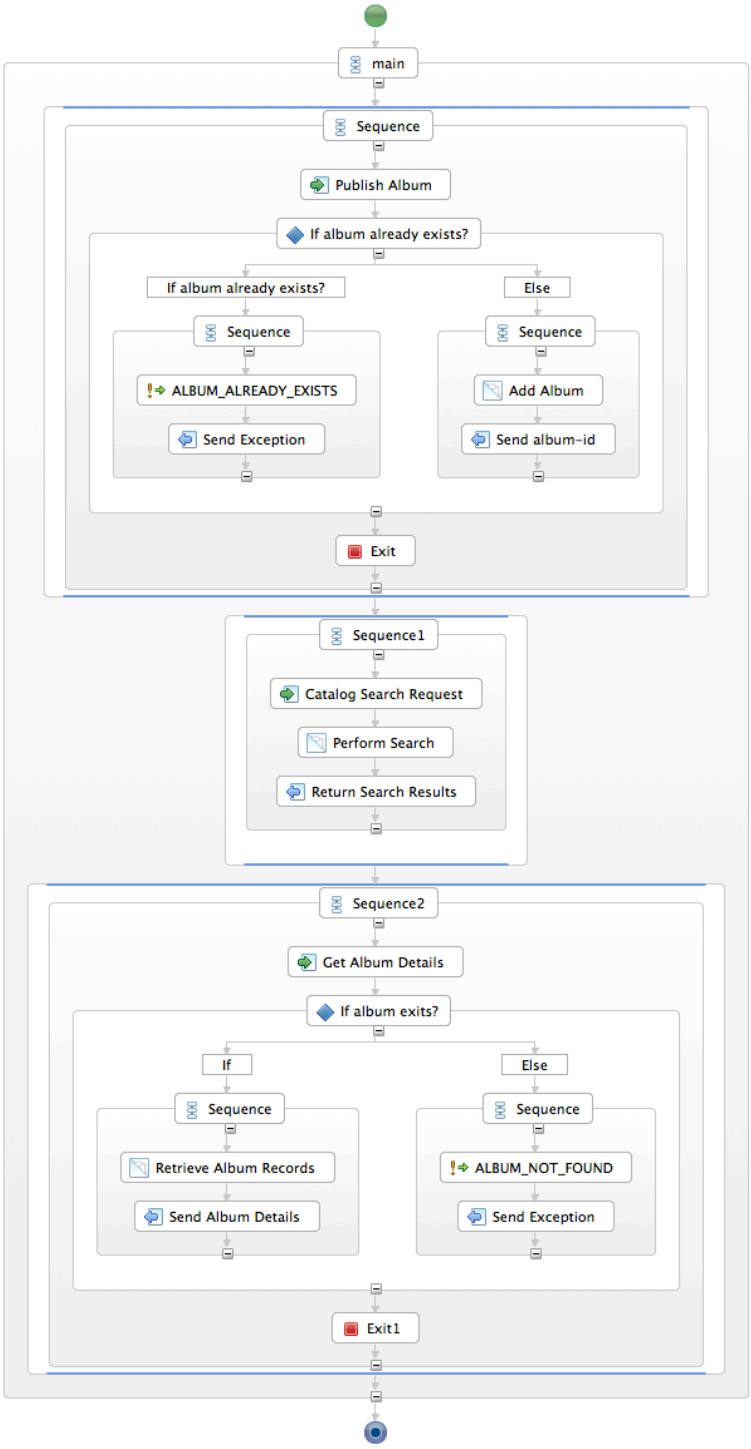



**Figure 2-7: A BPEL process workflow for an online-album service**

Beyond the above technical model-reasoning problems, aligning business process models is essential for process agility. In order to stay competitive companies must be able to adapt their business processes to the ever-changing market dynamics. However, such a dynamic adaptive market would leave the company with the big challenge of how to leverage available resources to satisfy new requests. A good alignment should utilize current portfolio while minimizing required not-yet available infrastructure. In both cases, the problem is how to match elements of one BPM to elements of another BPM. The objective is to maximize the similarity while minimizing the differences.

Furthermore, process-model differencing is useful for verifying conformance of processes against desired protocols. A typical business process model describes the details of a certain business process, and the interactions between this process and other processes, external entities, or human users. These interactions usually follow a certain communication protocol that should be respected at all times. Breaking any of these protocols is likely to cause a process failure. Therefore, any modification to the process model should be carefully checked against the interaction protocols to verify conformance. In this case, a process model differencing technique is to match a model interaction against a desired interaction protocol.

### *2.4.1 Related Work*

Most of the literature on process matching related work falls under the category of process-control model matching. Control matching means to match the control structure of the business model. There are three basic process control models: Petri-nets, Pi-Calculus, and Tree hierarchy. For example, Brockmans et al. [26] presents an approach for aligning the Petri-net models of two business processes [105], and presents an approach for aligning the pi-calculus formulations of two business processes. Additionally, there are approaches that compare business process models based on a given Finite State Automata [95], or Markov Decision Processes [45]. The common drawback with all of the above approaches is that

they only consider the control model of a process while ignoring the associated data-flow and message-flow model.

One of the earliest works on the BPEL matching problem is that done by Mikhaiel and Stroulia [102] which is based on representing a BPEL process as an ordered labeled tree, and then applying VTracker to it. Similarly, the work of Corrales et al. [40] represents a BPEL process as a graph, and then the BPEL differencing problem is formulated as a graph-edit distance problem. In 2007, Eshuis and Grefen [48] addressed the same problem. Their approach was based on representing a BPEL process as a tree. Then, two leaf nodes are compared based on their least common ancestor. The motivation behind the work of Eshuis and Grefen is to offer a BPEL matching approach that does not only consider the syntactic description of the process but also its semantic aspects. For example, they stated that the "drawback of such an approach is that different syntactic constructs typically mean the same. So two processes may not be matched even though they are equivalent." [48]. This concept will be discussed in more detail when presenting the concept of node similarity.

One last related work is WebSphere Integration Developer (WID)[3] that compares different versions of the same process model. The results of this tool are encouraging except that all changes have to happen through the WID IDE itself otherwise non-sense diff result will be produced. In other words, the tool keeps track of changes happening to the model, which is then used to compare different versions of the same. Consequently, this tool is not able to compare or integrate totally different models, or models originating from different sources.

### *2.4.2 A BPEL as a Tree*

Similar to WSDL, a BPEL specification is an XML document by nature. So, again, it is easy to represent it as an ordered labeled tree. To illustrate the idea, Code 2-3 shows the XML specifications of the BPEL workflow depicted in Figure 2-7. In this specification, it is identified that BPEL uses hyperlinks to refer

[3] http://www-01.ibm.com/software/integration/wid/

to operations, messages, partner links, and to link names. Figure 2-8 illustrates how to represent this example as an ordered labeled tree.

```
<bpel:process>
  <bpel:partnerLinks>
    <bpel:partnerLink name="client" partnerLinkType="tns:JobProcessing"
        myRole ="JobProcessingProvider" partnerRole="JobProcessingRequester"/>
    <bpel:partnerLink name="OnlineAlbum-link" partnerLinkType="ns1:OnlineAlbum"
        myRole="OnlineAlbumProvider">
    </bpel:partnerLink>
  </bpel:partnerLinks>
  <bpel:variables>
    <bpel:variable name="OnlineAlbum-linkResponse"
        messageType="ns1:PublishAlbumResponseMessage">
     </bpel:variable>
  </bpel:variables>
  <bpel:sequence name="main">
    <bpel:receive name="Publish Album" createInstance="no"
        partnerLink="OnlineAlbum-link" operation="PublishAlbum"
        portType="ns1:OnlineAlbum" variable="OnlineAlbum-linkResponse">
    </bpel:receive>
    <bpel:if name="If album already exists?">
      <bpel:sequence>
        <bpel:throw name="ALBUM_ALREADY_EXISTS"></bpel:throw>
        <bpel:reply name="Send Exception" partnerLink="OnlineAlbum-link"
           operation="PublishAlbum" portType="ns1:OnlineAlbum"
           variable="OnlineAlbum-linkResponse">
        </bpel:reply>
      </bpel:sequence>
      <bpel:else>
        <bpel:sequence>
          <bpel:opaqueActivity name="Add Album"></bpel:opaqueActivity>
          <bpel:reply name="Send album-id" partnerLink="OnlineAlbum-link"
             operation="PublishAlbum" portType="ns1:OnlineAlbum">
          </bpel:reply>
        </bpel:sequence>
      </bpel:else>
    </bpel:if>
    <bpel:exit name="Exit"></bpel:exit>
  </bpel:sequence>

</bpel:process>
```

**Code 2-3: A workflow BPEL specification described in XML syntax**

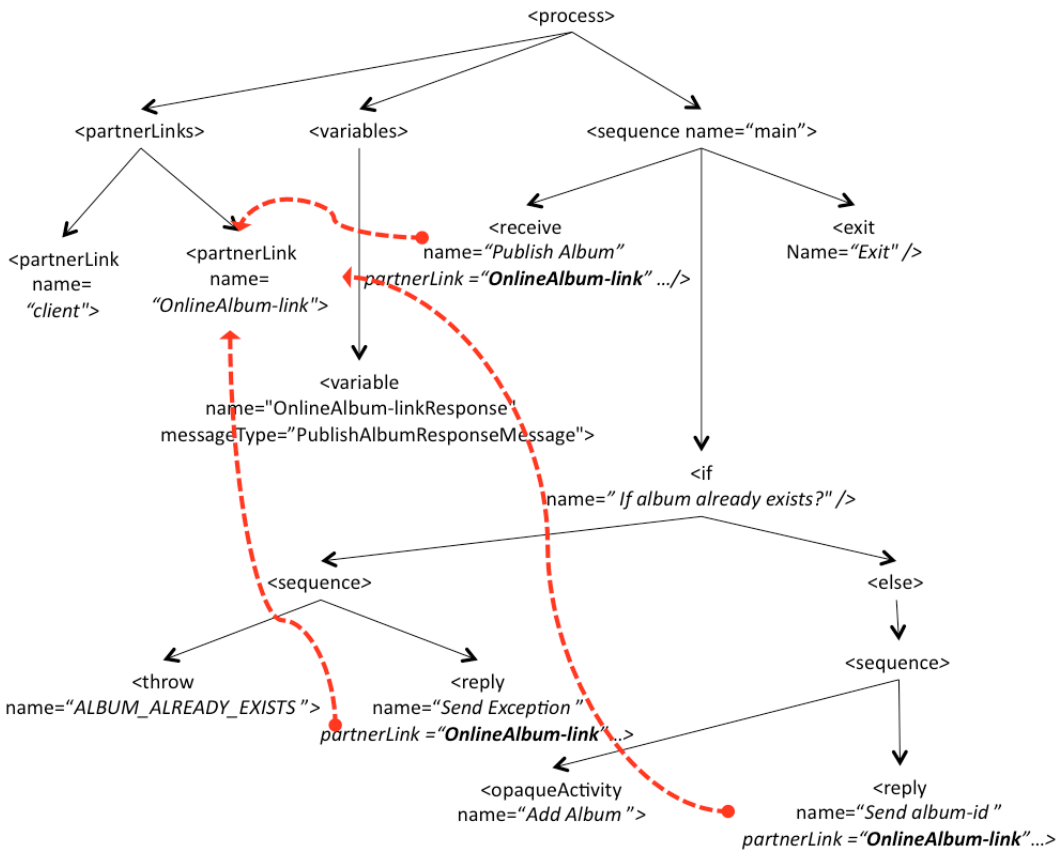



**Figure 2-8: A BPEL process workflow represented as a tree**

## 2.5 UML

Another core problem in software-evolution analysis is the detection of specific changes that occur between subsequent releases of a system. Consequently, it is necessary to analyze and understand the developmental steps that brought the system to its current state. Since structural changes are motivated by the need to improve the functionality and the quality of the software system, subsequent longitudinal analysis of the identified changes can lead to interesting insights on the change patterns as well as the rationale for the overall evolutionary history of a software system.

The drawback in UML differencing aims at finding design changes of long-living software systems. Given two object-oriented models the objective is to find the proper mapping between different elements like packages, classes,

interfaces, attributes, operations, parameters, etc. in addition to detecting minimum cost edit script that transforms one model into another.

### *2.5.1 Related Work*

A substantial body of research has focused on software differencing. Fluri et al. [52] suggested a tree-differencing algorithm that extracts fine-grained source-code changes between abstract syntax trees (AST). The algorithm is an extension of the Chawathe et al. [27] algorithm for change detection in hierarchical tree-like data structures. It also uses string similarity measures for leaves and tree-similarity measures for sub-trees.

Chevalier et al. [29] proposed a technique to detect similar structures in evolving C++ source codes that is also based on matching AST. The goal of the study was to visualize the evolution of the code clone structure and to indicate small to medium-scale changes, such as function and class-level refactoring code edits.

Apiwattanapong et al. [13] proposed a differencing algorithm CalcDiff that extends the existing Larski et al. [80] algorithm. CalcDiff compares two versions of an object-oriented program in order to identify and classify differences and similarities between them. Since traditional control-flow graphs (CFG) cannot model different object-oriented constructs such as dynamic binding, exception handling, synchronization, and reflection, the authors introduced an extended graph representation of a traditional CFG (ECFG). The new representation enables the comparison of object-oriented features of general object-oriented languages. Using this graph, the algorithm identifies behavioral changes resulting from structural changes, and relates them to the point of the code where this different behavior occurs. CalcDiff first performs matching on the class level, then on the method level, and finally on the node level using a hammock comparison algorithm on an extended ECFG.

Xing and Stroulia [161] suggested a differencing algorithm UMLDiff, which is designed to automatically identify structural changes between two

software logical models [76]. It outputs a set of change facts describing the differences between the two models. UMLDiff was implemented in the context of the JDEvAn tool, an Eclipse plug-in.

### *2.5.2 A UML model as a Tree*

One of the most common formats to exchange UML model specification is XML Metadata Interchange (XMI). XMI is a model driven XML Integration framework for defining, interchanging, manipulating and integrating XML data and objects. XMI-based standards are in use for integrating tools, repositories, applications and data warehouses. In this way a UML model can be represented as an XML document which in turn can be represented as an ordered labeled tree. UML/XMI tree is composed of a set structure of *packaged elements* nodes that describe the *package* and the *class* hierarchies. A *class* tree structure is composed of a set of *owned attributes* nodes and *owned operations* nodes that represent the attributes and methods, respectively.

For example, Code 2-5 shows UML/XMI representation of the simple Java class shown in Code 2-4, which was generated using Jar2UML[4]. Figure 2-9 illustrates how to represent a UML/XMI specification as an ordered label tree. In this way the problem of UML differencing can be formulated as a tree-edit distance problem between the tree structures corresponding to their XMI representation. One more observation is that UML/XMI largely relies on the XML reference structure to reuse definitions of some elements. For example, in this figure, there are hyperlinks between type attributes with value "java.lang.int" to the real definition of the "int" under the package "java.lang". Therefore, the reference structure is essential in the UML differencing problem.

[4] http://soft.vub.ac.be/soft/research/mdd/jar2uml

```
package com.foo;
public class Operation {
	int op1, op2;
	public Operation(int op1, int op2){
		this.op1 = op1;
		this.op2 = op2;
	}
	public int getOp1() {
		return op1;
	}
	public int getOp2() {
		return op2;
	}
}
```

**Code 2-4: Simple Java class**

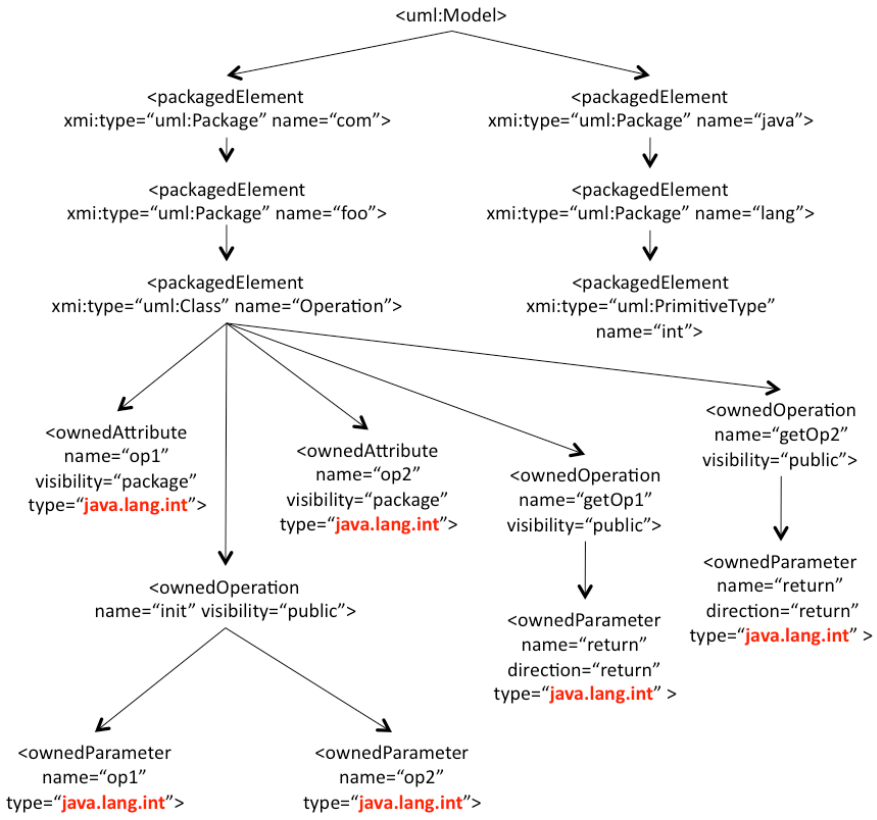



**Figure 2-9: A UML/XMI sample represented as a tree structure**

```
<uml:Model xmi:version="2.1"
           xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
           xmlns:uml="http://www.eclipse.org/uml2/3.0.0/UML"
           xmi:id="_0GWAUKaxEeCv2JHI4_R6HA" name="foo">
<packagedElement xmi:type="uml:Package"
    xmi:id="_0NB3IKaxEeCv2JHI4_R6HA" name="com">
  <packagedElement xmi:type="uml:Package"
      xmi:id="_0NB3IaaxEeCv2JHI4_R6HA" name="foo">
    <packagedElement xmi:type="uml:Class"
        xmi:id="_0NB3IqaxEeCv2JHI4_R6HA" name="Operation">
      <generalization xmi:id="_0NB3I6axEeCv2JHI4_R6HA"
          general="_0M2Q8qaxEeCv2JHI4_R6HA"/>
      <ownedOperation xmi:id="_0SC6A6axEeCv2JHI4_R6HA"
          name="&lt;init>" visibility="public">
        <ownedParameter xmi:id="_0SDhEKaxEeCv2JHI4_R6HA"
            name="op1" type="_0NBQEKaxEeCv2JHI4_R6HA"/>
        <ownedParameter xmi:id="_0SDhEaaxEeCv2JHI4_R6HA"
            name="op2" type="_0NBQEKaxEeCv2JHI4_R6HA"/>
      </ownedOperation>
      <ownedOperation xmi:id="_0SDhEqaxEeCv2JHI4_R6HA"
          name="getOp1" visibility="public">
        <ownedParameter xmi:id="_0SDhE6axEeCv2JHI4_R6HA"
            name="return" type="_0NBQEKaxEeCv2JHI4_R6HA" direction="return"/>
      </ownedOperation>
      <ownedOperation xmi:id="_0SDhFKaxEeCv2JHI4_R6HA"
          name="setOp1" visibility="public">
        <ownedParameter xmi:id="_0SEIIKaxEeCv2JHI4_R6HA"
            name="op1" type="_0NBQEKaxEeCv2JHI4_R6HA"/>
      </ownedOperation>
      <ownedOperation xmi:id="_0SEIIaaxEeCv2JHI4_R6HA"
          name="getOp2" visibility="public">
        <ownedParameter xmi:id="_0SEIIqaxEeCv2JHI4_R6HA"
            name="return" type="_0NBQEKaxEeCv2JHI4_R6HA" direction="return"/>
      </ownedOperation>
      <ownedOperation xmi:id="_0SEII6axEeCv2JHI4_R6HA"
          name="setOp2" visibility="public">
        <ownedParameter xmi:id="_0SEIJKaxEeCv2JHI4_R6HA"
            name="op2" type="_0NBQEKaxEeCv2JHI4_R6HA"/>
      </ownedOperation>
      <ownedOperation xmi:id="_0SEvMKaxEeCv2JHI4_R6HA"
          name="toString" visibility="public">
        <ownedParameter xmi:id="_0SEvMaaxEeCv2JHI4_R6HA"
            name="return" type="_0M_a4KaxEeCv2JHI4_R6HA" direction="return"/>
      </ownedOperation>
    </packagedElement>
  </packagedElement>
</packagedElement>
</uml:Model>
```

**Code 2-5: A UML/XMI representation of the sample Java class**

## 2.6 XHTML

HTML differencing is an interesting problem with a variety of useful applications. It is relevant to web-site maintenance where a manager might wish to periodically

review the changes made by the various web-site users in order to approve their publication. It is also useful to recurring web-site visitors who may want to quickly assess whether or not an interesting change has been made to their page of interest. It is essential for web-content warehouses where documents are periodically collected by crawlers; upon receiving new versions of an existing document, the warehouse manager may want to track the changes that occurred since the last received version. Finally, it is a necessary step for automatic web wrapping [120][164][70] where document comparison is used to automatically extract data from the web. Figure 2-10 depicts a simple XHTML differencing example illustrating four kinds of edit operations.

### *2.6.1 XHTML as a Tree*

An XHTML document can easily be described as an ordered labeled tree where the root of the tree is the HTML tag that is composed of two main sub-trees: head and body. As shown in Figure 2-11 XHTML elements of both sections become nodes in the tree representation where elements are represented as nodes, element names are labels of the nodes, and attributes of elements become attributes of nodes. Instructional elements such as comments, scripts, styles, etc are ignored in the tree representation, as they do not affect the containment structure of an XHTML document. Unlike previous kinds of XML documents, XHTML does not have a reference structure *per se*. In HTML, hyperlinks main objective is to be used by a real user to navigate from one location to another, rather than being used by a processing program to reference other elements in the same document.


**(a) Page1** **(b) Page2**


**(a) Desired results**

**Figure 2-10: A XHTML differencing example**

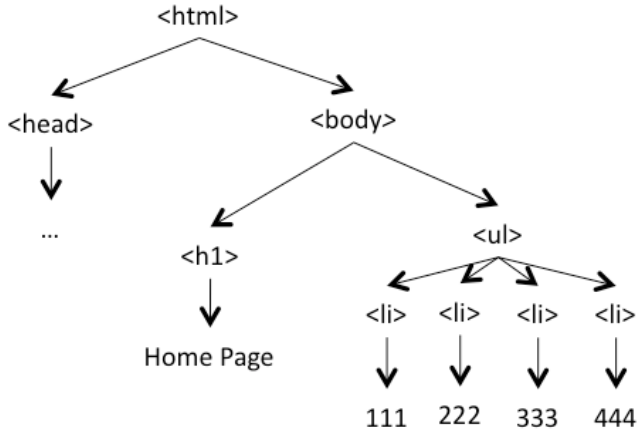



**Figure 2-11: An XHTML document represented as an ordered labeled tree**

## 2.7 RNA Secondary Structure Comparison

Ribonucleic acid (RNA) molecules are involved in many important biological processes. Some, such as mRNA, carry genetic information; others, such as tRNA, rRNA, and the recently discovered microRNA, are directly responsible for the accomplishment of distinct functions. Biology research has shown that specific organism processes and functions can be attributed to particular secondary structures of RNA molecules [94]. As the organisms evolve their RNA structure changes and their processes and functions change correspondingly. Consequently, the processes and functions of newly discovered organisms can be inferred based on the processes and functions of known organisms to which they are related in evolutionary terms, and with which they share corresponding RNA secondary structures. Thus, a precise and efficient RNA secondary structure comparison is essential for providing useful hints on possible RNA molecule functions, as well as their phylogenetic relationships. The primary structure of an RNA molecule is a sequence of nucleotides (bases) over the alphabet {A, C, G, U}. Its secondary structure is a folding of its primary structure and is formally specified as a set of base pairs that form bonds between A-U, C-G, and G-U bases [94].

### *2.7.1 Related Work*

Many approaches have been proposed for RNA alignment and comparison, and, in general, they can be categorized against two dimensions. The first dimension concerns the RNA features considered by the approach (e.g. primary structure, secondary structure, or both primary and secondary structure), and the second concerns the model used to represent RNA (e.g. string based or tree structure).

Some approaches adopt string-based representations of the RNA primary structure only. In these cases, RNA molecules are represented as strings and standard string-alignment algorithms are applied [7][8]. However, because they ignore the associated secondary structure these approaches match RNA bases

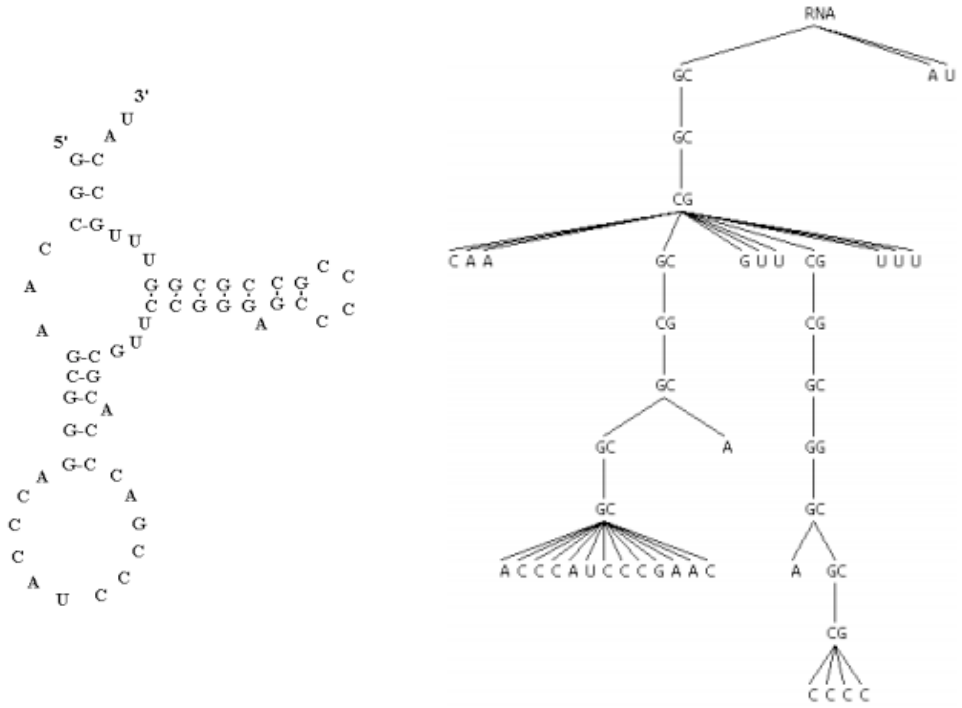
regardless of the molecular interactions in which they are involved (i.e. the loops they constitute) leading to biologically implausible alignment solutions [16].

Yet other approaches consider the RNA secondary structure only which is abstractly represented as a structure of loops. Such representations capture the topological skeleton of RNA molecules regardless of their underlying primary structure, and are useful for measuring the high-level structural similarity between molecules, but the measurements are not precise. Shapiro's work (1988) exemplifies these approaches in that it represents a secondary structure as a tree where each node represents a loop (e.g. Internal, Hairpin, Bulge, or Multi loop).

| | **String/Arc representation** | **Tree-based representation** |
|---|---|---|
| **Primary structure only** | Bafna, 1995; | |
| **Secondary structure only** | El-Mabrouk et at, 2002. | ***Coarse-grained***<br>Jin et at, 2005;<br>Shapiro and Zhang, 1990; Le et al., 1989;<br>Shapiro 1988. |
| **Primary and secondary structure** | Jiang et al., 2002;<br>Lin et al., 2001;<br>Collins et al., 2001;<br>Wang and Zhang, 2001, 2005;<br>Corpet and Michot 1994. | ***Fine-grained***<br>- *Tight Fine-grained:*<br>Herrbach et al., 2006;<br>Liu et al., 2005;<br>Zhang, 1998.<br>- *Loose Fine-grained:*<br>Höchsmann, 2005 |

### *2.7.2 RNA Secondary Structure Comparison as a Tree*

Figure 2-12 illustrates how Mikhaiel and Stroulia [103] represented an RNA molecule as a Tight Fine-Grained (TFG) tree structure. In this representation, a loop is represented by: (1) one node representing the opening base pair of this loop, and (2) a set of nodes each representing other single bases this loop. This representation captures both the primary and secondary structures leading to a better comparison quality.




**(a) An RNA Secondary Structure** **(b) RNATree representation**

**Figure 2-12: An RNA Structure represented as an ordered labeled tree**

Another way to represent RNA Secondary Structure is called Loose Fine-Grained (LFG) tree structure introduced by RNA Forester [59][60][61]. As shown in Figure 2-13, in this representation stem loops are decomposed into a joint node and two separate nucleotide (base) nodes. In LFG, each element is represented as a single node while each bond is represented as a joint node. The main difference between LFG and TFG is that LFG allows fine-grained edit operations. For example, in TFG an edit operation may include replacing, deleting, or inserting a pair of elements while LFG allows to report edit operations like breaking the joint between a pair of elements, or deleting/inserting a single node.

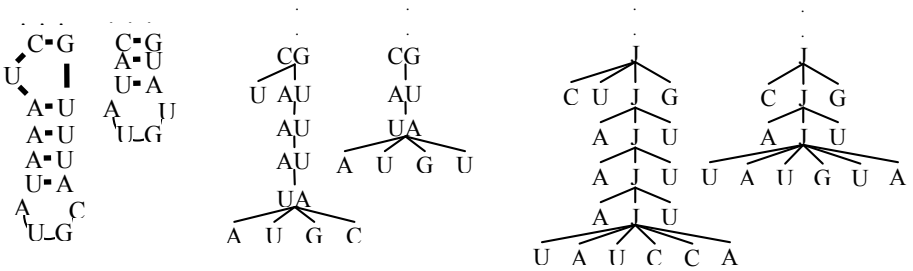


**Figure 2-13: LFG versus TFG RNA tree structures**

# Chapter Three VTracker: A Generic XML-Differencing Method

VTracker is a generic solution to the XML differencing problem. VTracker is applicable to various domains, and is able to compete with, and replace, domain-specific differencing approaches. This chapter presents the requirements of a generic XML differencing approach, the original Zhang-Shasha algorithm, and then the details of VTracker by explaining how it meets these requirements.

## 3.1 Requirements of Generic XML differencing

As discussed in the previous chapter the problem of XML differencing has been studied from many perspectives. Many researchers have designed domain-specific XML differencing approaches that are intended to serve some domains in particular. Other researchers looked broadly and tried to design generic approaches that were supposed to fit in many domains. However, many of the so-called, generic approaches suffer critical limitations that prevent them from being practically generic. This section lays out a set of guidelines for any XML differencing approach that aims at being generic.

**(1) Be domain independent**

The first obvious requirement of a generic XML differencing approach is to avoid being tied to, or constrained by, a specific application domain. Also, it should not include implicit semantics or knowledge of some specific domains in particular. It should be able to serve multiple domains, and be capable of capturing domain specific knowledge and semantics through an easy to develop customization technique.

**(2) Produce meaningful minimal edit script**

Another requirement of an XML-differencing approach is to produce the shortest possible edit script that can transform a given XML document to another one. A differencing method should avoid reporting unnecessary deletions or insertions. For example, if the root of a sub-tree is deleted (or inserted), the entire sub-tree should not necessarily be deleted (or inserted). In this sense, Selkow's tree-to-tree correction model is not the best in terms of delta size since it offers three kinds of

edit operations: change node label, delete sub-tree, or insert sub-tree. Hence, tree-edit distance methods offering fine-grained edit operations are more efficient in terms of the produced delta.

Another requirement on the edit script is to be meaningful. A produced delta should be sound, reasonable, and acceptable by subject-matter experts; otherwise, the results are not useful and do not reflect any meaning to the user. Furthermore, if a meaningful script is not the most minimal script, being meaningful suppresses the minimalistic requirement.

**(3) Consider the hierarchical data structure represented in XML**

By nature, XML has a tree like structure. An XML-differencing approach should be aware of this structure and should not consider an XML as a flat file. Mapping results, therefore, should obey the structural relationships between mapped elements. The Zhang-Shasha algorithm formalized this aspect in the following rules. Consider nodes $i_1$ and $i_2$ belong to $T_1$ while $j_1$ and $j_2$ are nodes in $T_2$, then in order to map node $i_1$ to node $j_1$, while node $i_2$ is mapped to node $j_2$ the following conditions should apply [166]:

I. $i_1 = i_2$ iff $j_1 = j_2$; one-to-one mapping where each node cannot be involved into more than one edit operation. This condition will not be valid in methods that allow split, copy, and glue-edit operations.

II. $i_1$ is on the left of $i_2$ iff $j_1$ is on the left of $j_2$ ; preserving siblings order.

III. $i_1$ is an ancestor of $i_2$ iff $j_1$ is an ancestor of $j_2$ ; preserving ancestor-child order.

**(4) Allow edit operations anywhere**

In XML documents edits may happen anywhere in the document, to a leaf node or to an internal node. For example, edits may include changing the value of a certain leaf node, or restructuring the sub-tree of another node. A differencing method should, therefore, be able to detect changes that may happen anywhere in

the two given XML documents, and not to be limited to certain type of changes. Additionally, a method's performance should be consistent in detecting different types of changes.

**(5) Identify elements based on all available information**

XML elements should be identified by value, attributes, content, context, structure, and references. They should not only be identified by attributes or hash values that were preset in an earlier version comparison, or only with attributes considered keys in that domain. For instance, sentences and paragraphs in structured-text documents do not come with an identifier [27]. Also, some elements may have different domain identifier values but have the same content structure. For example consider a case where an element $x$ on one side is being matched against two elements $y_1$ and $y_2$ on the other side, and where $y_1$ has the same identifier value as $x$ but with different content while $y_2$ has the same contents but with a different identifier value. Then, the question becomes, what identifies an element: identifier values, or content and structure? Either answer to this question is right as long as it considers identifier values, contents, structure, context, and other attributes.

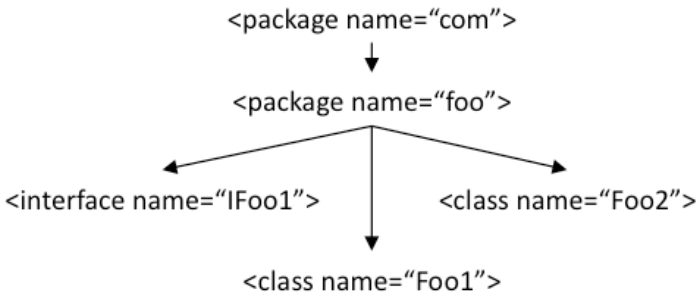
**(6) Do not assume prior change-tracking log**

A differencing method should assume that the two given documents were independently developed and edited by different parties and through different technologies. A generic XML differencing method should not assume that edits and changes are tracked by the editor utility. An XML differencing method should not rely on the fact that changes always happen through a particular tool, whose job is to keep track of, and record, changes happening to a certain XML document. Otherwise, it is not differencing but rather a method to report changes that were previously recorded.

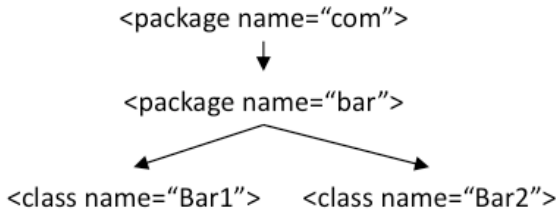
**(7) Be as efficient as domain-specific differencing techniques**

One main reason motivating the development of domain-specific methods is the inefficiency of generic approaches. Generic methods usually perform extra steps

that are neither necessary nor justifiable at least from the point of view of some domains. For example, generic methods will have to consider all possible combinations when matching sub-trees against each other. In some domains, such behavior is considered a waste of time, since according to the domain knowledge some nodes are impossible to be mapped to some other nodes so why should it try to match them. For example, Figure 3-1 shows two java object-oriented hierarchies, where it is not acceptable, by any means, to match a package node neither to a class node nor to an interface node while a class node might be matched to either an interface or a class node. Unlike generic approaches a domain-aware method is more efficient since it will try to match a package sub-trees only to a package sub-tree, and similarly, a class sub-trees against only a class or an interface sub-trees. Therefore, a generic method to compete with domain-specific methods should be intelligent enough to skip, and avoid any unnecessary comparison steps.




**(a) An object-oriented Model version 1**




**(b) An object-oriented Model version 2**

**Figure 3-1: Two object-oriented samples represented as tree structures**

**(8) Consider XML reference structure in the differencing process**

The reference model is a special feature in XML that aims at increasing the reusability of some element definitions by referring to them from other elements. And usually that is either to apply some normalization mechanisms, or to implement certain relationships such as association, specialization, or instantiation. Chapter Two presents many domain applications that largely rely on the XML reference structure, and therefore the reference structure should play a role in an XML differencing process. To illustrate the idea, Figure 3-2 shows two different structures to represent the same Student Enrolment database. The first structure does not use hyperlinks (i.e. references) while the second does. In the first structure, a course definition is repeated every time it is mentioned while in the second structure; the course is defined once and is then referenced when needed.

```
<?xml version="1.0" encoding="iso-8859-1"?>
-<school >
  -<students >
    -<student id="S00123">
      <name >John Smith</name>
      <email >john.smith</email>
      -<enrollment >
        -<course id="CMPUT101">
          <name >Introduction to Computing</name>
          <location >Room 309</location>
          -<instructor id="I00289">
            <name >David Jordan </name>
            <room >315</room>
          </instructor>
        </course>
        -<course id="CMPUT114">
          <name >Programming with Data
Structures</name>
          <location >Room 310</location>
          -<instructor id="I00305">
            <name >Mike McDonald</name>
            <room >410</room>
          </instructor>
        </course>
      </enrollment>
    </student>
    -<student id="S00403">
      <name >Wayne Jackman</name>
      <email >wayne.jackman</email>
      -<enrollment >
        -<course id="CMPUT101">
          <name >Introduction to Computing</name>
          <location >Room 309</location>
          -<instructor id="I00289">
            <name >David Jordan </name>
            <room >315</room>
          </instructor>
        </course>
        -<course id="CMPUT114">
          <name >Programming with Data
Structures</name>
          <location >Room 310</location>
          -<instructor id="I00305">
            <name >Mike McDonald</name>
            <room >410</room>
          </instructor>
        </course>
      </enrollment>
    </student>
```

**(a) An XML sample with no reference model**

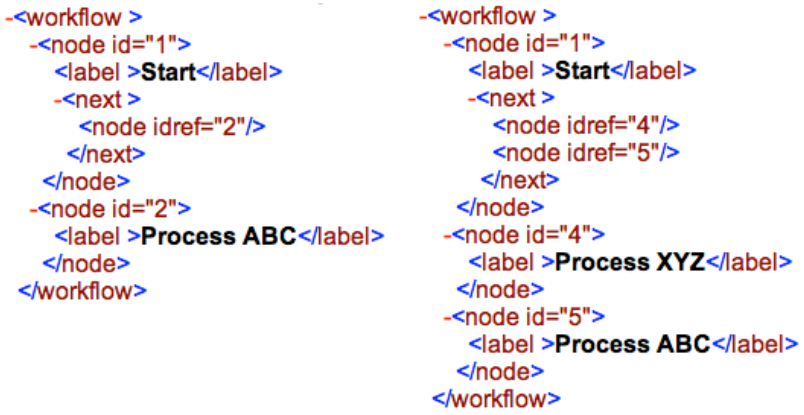
```
<?xml version="1.0" encoding="iso-8859-1"?>
-<school >
  -<students >
    -<student id="S00123">
      <name >John Smith</name>
      <email >john.smith</email>
      -<enrollment >
        <course idref="CMPUT101"/>
        <course idref="CMPUT114"/>
      </enrollment>
    </student>
    -<student id="S00403">
      <name >Wayne Jackman</name>
      <email >wayne.jackman</email>
      -<enrollment >
        <course idref="CMPUT101"/>
        <course idref="CMPUT114"/>
      </enrollment>
    </student>
  </students>
  -<courses >
    -<course id="CMPUT101">
      <name >Introduction to Computing</name>
      <location >Room 309</location>
      -<instructor id="I00289">
        <name >David Jordan </name>
        <room >315</room>
      </instructor>
    </course>
    -<course id="CMPUT114">
      <name >Programming with Data
Structures</name>
      <location >Room 310</location>
      -<instructor id="I00305">
        <name >Mike McDonald</name>
        <room >410</room>
      </instructor>
    </course>
```

**(b) An XML sample with a reference model**

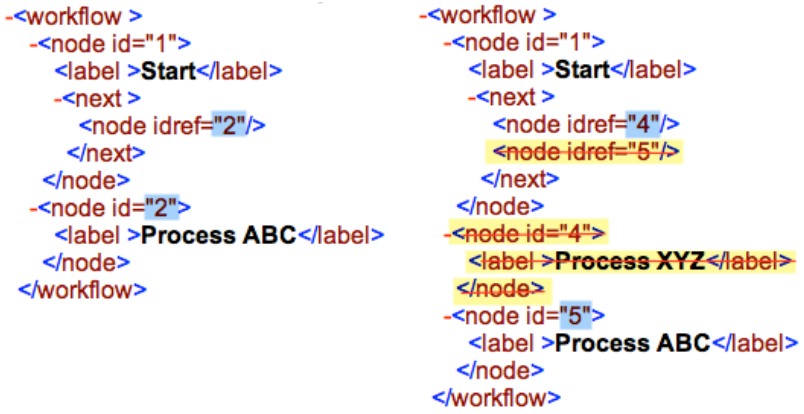
**Figure 3-2: Two XML sample documents showing the difference between non-normalized and reference-based normalized structures**

During a differencing process, the referenced structure should be considered as a part of the referring structure in the same place as the hyperlink.

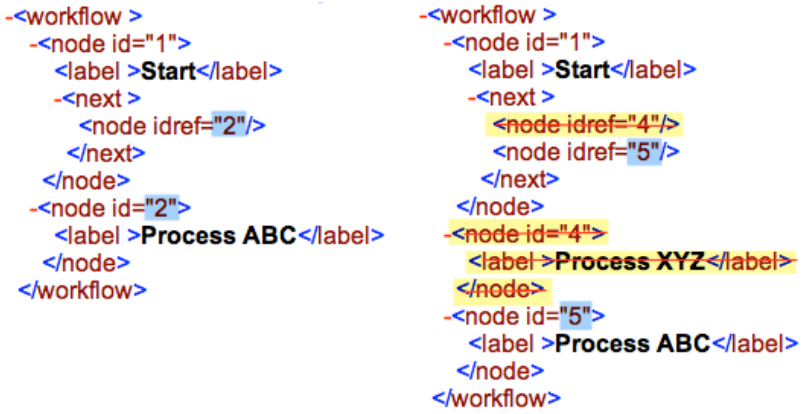
Similarly, when matching a document of the first type against a document of the second type, hyperlinks should logically be replaced by the referenced structures. In other words, element attributes constituting ID and IDREF should be handled differently than as just attributes. For instance, Figure 3-3 shows a simple example illustrating how the reference structure influences the validity of the produced results. This example describes two workflows: the first one has a start node that leads to one other node labeled "Process ABC". In the second workflow, the start node forks into two nodes "Process XYZ" and "Process ABC". The challenge here is that the nodes in the two workflows have different IDs. Figure 3-3 shows it is not difficult to detect that the node labeled "Process XYZ" was newly inserted, and that the id of the node labeled "Process ABC" has been changed from "2" to "5". However, the tricky part is in the reference inside the start node. The differencing tool should choose how to map `<node idref = "2"/>` either to `<node idref = "4"/>` (as in Figure 3-3 (b)) or to `<node idref = "5"/>` (as in Figure 3-3 (c)). Unless the attribute IDRef is treated specially, the solution will not be justifiable. A possible way to wisely resolve this situation is to follow the reference on both sides and compare the referenced nodes; comparing node 2 against node 4, and comparing node 2 against node 5 and choose the one that is closer. If this is applied properly, the correct solution should look like Figure 3-3 (c) where the solution shown in Figure 3-3 (b) should be perceived as incorrect for two reasons: (1) there is no reasonable justification why `<node idref = "2"/>` is mapped to `<node idref = "4"/>`, and (2) the reported results contradict with the rest of the results where it refers to a node "4" that is not mapped to the other side. While the solution based on reference model in Figure 3-3 (c) makes sense as it matches references to nodes that are the closest to each other.




**(a) Two sample workflows**




**(b) An example of undesired differencing results**

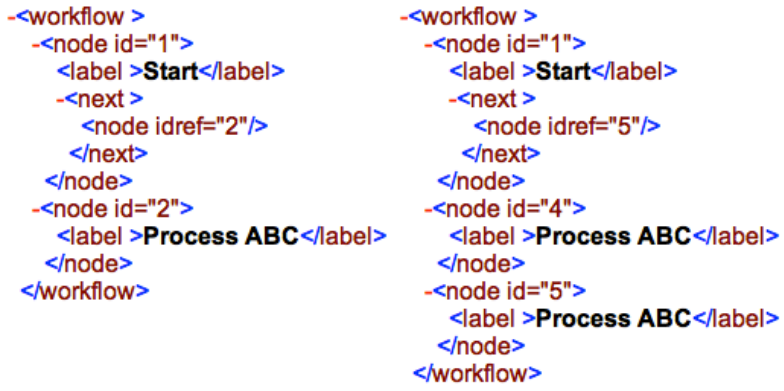



**(c) An example of desired differencing results**

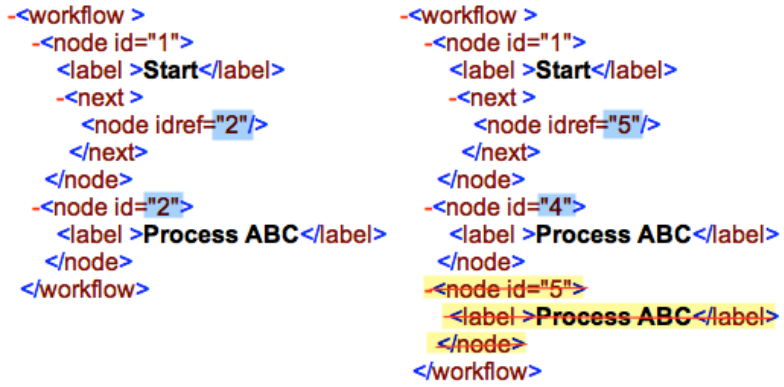
**Figure 3-3: An XML differencing example illustrating the importance of reference model**

**(9) Consider XML usage-context structure in the differencing process**

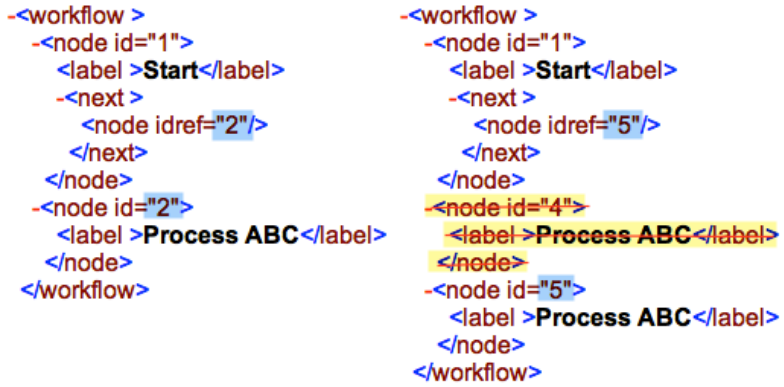
The XML reference model implies a two-way relationship: the previous example discussed one of them, namely outgoing references from an element of interest. The other direction, the so-called *usage-context*, is the referenced-by relationship, namely incoming references to the element of interest. Figure 3-4 illustrates a very simple example of the usage-context where there are two elements on the second tree that can be matched to an element in the first tree. The question will be which one to map it to. Such confusion would be resolved by all known methods through randomly picking any of the two choices. However, a smart differencing tool should resolve this confusion by picking the choice that maps elements used in similar contexts. A sound result should consider the fact that on the first document, node #2 is referenced from the start node, which implies that the correspondent in the second document should have a similar usage-context. In other words, the usage-context of a node should have an important role in the identification of that node. In that sense, the solution in Figure 3-4 (c) should be the right one as it maps nodes with the same usage-context. To the best of our knowledge, the known XML differencing tools neither consider the reference model nor the usage-context during the differencing process. This aspect is an important element of the applicability of the XML model for generic methods. Finally, it is evident that usage-context should be used in conjunction with a reference-aware approach to produce the best results.




**(a) Two sample workflows**




**(b) An example of undesired results**




**(c) An example of desired results**

**Figure 3-4: An XML differencing example illustrating the role of usage-context similarity in resolving matching ambiguities**

## 3.2 The Original Zhang-Shasha Algorithm

This section explains the original Zhang-Shasha algorithm [166] that is the base of the current implementation of VTracker. Given two ordered labeled trees, and a cost function, the Zhang-Shasha algorithm calculates the optimal edit distance to transform the first tree into the second tree.

Before explaining the algorithm, descriptions of some essential definitions and notations are offered. Let $T$ be a *rooted* tree, then:

- **Ordered tree**: a tree $T$ is called an ordered tree if a left-to-right order among siblings in $T$ is given.
- **Node index:** nodes are numbered in a post-order manner where children are visited from left-to-right before their parents. In other words, the index of the root node should be the same as the size of the tree that is denoted as $|T|$. Hence, $T[x_1..x_2]$ refers to the set of nodes with indexes between $x_1$ and $x_2$ inclusive. The left most leaf child of a node $x_i$ can be obtained by $lm(x_i)$. Hence, the sub-tree rooted by node $x_i$ can be represented as $T[lm(x_i)..x_i]$ that is short-handed as $T[x_i]$, and the whole tree can similarly be represented as $T[|T|]$.
- **Node label:** the label of a node $x_i$ is denoted by $l(x_i)$.
- **Labeled tree:** a rooted tree $T$ is called a *labeled tree* if each node $v$ is assigned a symbol from an alphabet $\Sigma$.
- **Edit operations:** an edit operation $s_i$ is represented as $(x_i , y_i)$ where $x_i$ is either a node in $T_1$ where $1< x_i < |T_1|$, or is $\lambda$ in case of no correspondence in $T_1$, and similarly $y_i$ is either a node in $T_2$ where $1< y_i < |T_2|$, or is $\lambda$ in case of no correspondence in $T_2$. Hence, edit operations can be formally described as follows:
    - Change operation: denoted as $(x_i , y_i)$ where $l_1(x_i)$ the label of node $x_i$ is mapped to $l_2(y_i)$. If $l_1(x_i) = l_2(y_i)$, it is pronounced as a *match* rather than as a *change* operation.

- Deletion Operation: denoted as ($x_i$ , $\lambda$) and means that node $x_i$ with label $l_1(x_i)$ in $T_1$ has no correspondence in $T_2$.
- Insertion Operation: denoted as ($\lambda$, $y_i$) and means that node $y_i$ with label $l_2(y_i)$ in $T_2$ has no correspondence in $T_1$.

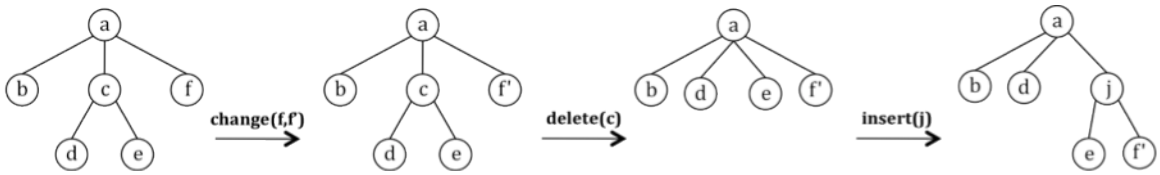



**Figure 3-5: A sample tree-edit script**

Figure 3-5 illustrates the tree-edit operations: (1) operation *change* label of node *f* to be *f'*, (2) operation *delete* node *c* where its children {*d*, *e*} became children of its parent, i.e. node *a*, and (3) operation *insert* a new node *j* to become an intermediate parent of some of node a children. It is very important to mention that an insertion operation is just the inverse of a deletion operation. The same operation when applied to the first tree, it is called a deletion but when applied to the second tree, it is called an insertion. Additionally, in a *change* operation, if the labels are the same, then it is not called a *change* but rather *match* operation.

- **Edit script**: an *edit script* is represented as $S_i = s_{i1}, \ldots, s_{ik}$ where $S_i$ is the $i^{th}$ edit script that is composed of a sequence of $k$ edit operations, and that is capable of transforming $T_1$ into $T_2$. An edit operation $s_{ij}$ denotes the $j^{th}$ edit operation of the $i^{th}$ edit script, and is represented as either a matching operation ($x_{ij}$,$y_{ij}$) such that $x_{ij}$ and $y_{ij}$ are nodes in $T_1$ into $T_2$ respectively, a delete operation ($x_{ij}$, $\lambda$), or an insert operation ($\lambda$, $y_i$) that satisfy the following conditions such as $(x_{ij_1}, y_{ij_1})$ and $(x_{ij_2}, y_{ij_2})$ are in $S_i$:
  - $x_{ij_1} = x_{ij_2} \Leftrightarrow y_{ij_1} = y_{ij_2}$ (one-to-one condition; no merge or split allowed).
  - $x_{ij_1}$ is an ancestor of $x_{ij_2}$ $\Leftrightarrow$ $y_{ij_1}$ is an ancestor of $y_{ij_2}$ (structure preserving condition).
  - $x_{ij_1}$ is to the left of $x_{ij_2}$ $\Leftrightarrow$ $y_{ij_1}$ is to the left of $y_{ij_2}$ (order preserving condition).

- **Tree-edit distance:** assume that given a cost function γ defined on each edit operation $s_i$, and is denoted as $\gamma(s_i)$. Then the cost of an edit script $S$ is calculated as $\gamma(S_i) = \sum_{j=1}^{k} \gamma(s_{ij})$, the sum of costs of operations in $S_i = s_{i1},\ldots,s_{ik}$.

- **An optimal *edit script*** between $T_1$ and $T_2$ is an edit script between $T_1$ and $T_2$ of the minimum cost, and is defined as:

  $$\delta(T_1,T_2) = \min\{\gamma(S_i) | 1 \le i \le n\}$$

  where $n$ is number of edit scripts that can transform $T_1$ into $T_2$. Hence, the tree-edit distance problem is to compute the cheapest edit distance and the corresponding edit script.

The Zhang-Shasha algorithm is based on a dynamic-programming approach that splits a tree-edit distance problem to a set of recursive sub-problems explained in Code 3-1. To accomplish that the algorithm divides a tree into a set of relevant sub-trees that are identified by a set of key roots. Key roots are defined as the set of nodes that includes the root of the tree in addition to all nodes that have at least one left sibling. The key-root set of each tree is then sorted according to the index of the key-root node. Hence, for all combinations of key sub-trees, the algorithm calculates the tree-edit distance starting from smaller sub-trees to bigger ones. The calculations of bigger sub-trees leverage results of smaller ones.

**Lemma 3-1: Tree-Edit Distance**

$$\begin{aligned} tdist(x,y) &= tdist(T_1[lm_1(x)...x], T_2[lm_2(y)..y]) \\ &= \min\begin{cases} fdist(T_1[lm_1(x)...x-1], T_2[lm_2(y)..y-1]) + \gamma(x,y) \\ fdist(T_1[lm_1(x)...x-1], T_2[lm_2(y)..y]) + \gamma(x,\lambda) \\ fdist(T_1[lm_1(x)...x], T_2[lm_2(y)..y-1]) + \gamma(\lambda,y) \end{cases} \end{aligned}$$

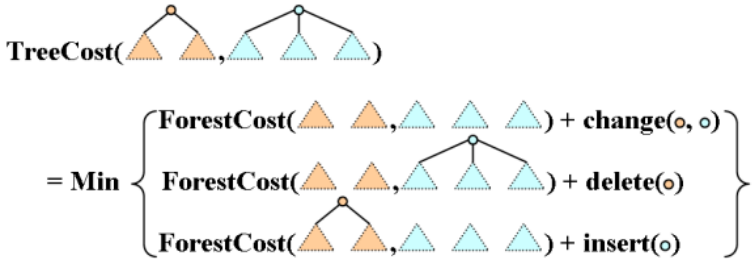
where the distance between two forests is defined as:

**Lemma 3-2: Forest-Edit Distance**

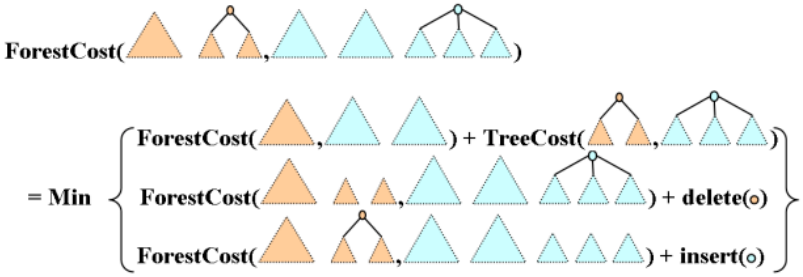
$$fdist(T_1[x_1..x_2], T_2[y_1..y_2])$$
$$= \min \begin{cases} fdist(T_1[x_1..lm_1(x_2)-1], T_2[y_1..lm_2(y_2)-1]) + tdist(x_2, y_2) \\ fdist(T_1[x_1..x_2-1], T_2[y_1..y_2]) + \gamma(x_2, \lambda) \\ fdist(T_1[x_1..x_2], T_2[y_1..y_2-1]) + \gamma(\lambda, y) \end{cases}$$

As illustrated by Figure 3-6, during each tree-edit distance calculation between $T_1[x]$ and $T_2[y]$, the Zhang-Shasha algorithm chooses the minimum cost option of the three following aspects:

- The cost of mapping node x to node y plus the cost of matching the remaining forests to each other.
- The cost of deleting node x plus the cost of matching remaining forest of first tree against the entire second tree.
- The cost of inserting node y plus the cost of matching entire first tree against remaining forest of the second tree.




**(a) Visualization of Tree-Edit Distance**




**(b) Visualization of Forest-Edit Distance**

**Figure 3-6: Visualization of Zhang-Shasha algorithm [46]**

```
DECLARE matrix tdist with size [|T1|+1] * [|T2|+1]
DECLARE matrix fdist with size [|T1|+1] * [|T2|+1]
FUNCTION treeDistance (x , y)
START
 lmx = lm1 (x) // left most node of x
 lmy = lm2(y) // left most node of y

 bound1 = x - lmx + 2 //size of sub-tree x + 1
 bound2 = y - lmy + 2 //size of sub-tree y + 1

 fdist[0][0] = 0

 // set the first column
 FOR i = 1 TO bound1 - 1
  fdist[i][0] = fdist[i-1][0] + cost(k,-1)

 // set the first row
 FOR j = 1 TO bound2 - 1
  fdist[0][j] = fdist[0][j-1] + cost(-1,l)

 k = lmx
 l = lmy
 FOR i = 1 TO bound1 - 1
  FOR j = 1 TO bound2 - 1
   IF lm1(k) = lmx and lm2(l) = lmy
   THEN // tree edit distance
    fdist[i][j] = min(fdist[i-1][j] + cost(k,-1),
              fdist[i][j-1] + cost(-1,l),
              fdist[i-1][j-1] + cost(k,l))
    tdist[k][l] = fdist[i][j]
   ELSE // forest edit distance
    m = lm1(k) - lmx
    n = lm2(y) - lmy
    fdist[j][j] = min( fdist[i-1][j] + cost(k,-1),
               fdist[i][j-1] + cost(-1,l),
               fdist[m][n] + tdist(k,l))
 RETURN tdist[x][y]
END
```

**Code 3-1: A pseudo code of Zhang-Shasha tree-edit distance algorithm**

### 3.3 The VTracker Approach

VTracker is a tree-edit distance algorithm that extends the Secondary and Primary RNA Comparison (SPRC) [103] algorithm, which was developed in the context of the author's work in RNA secondary structure alignment. Both SPRC and VTracker are based on the Zhang-Shasha tree-edit distance algorithm [166] which calculates the minimum edit distance between two trees given a cost function for different edit operations (e.g. change, deletion, and insertion). According to the

exact analysis of the algorithm performed by Dulucq and Tichit [46], Zhang-Shasha's algorithm is of complexity $|T_1|^{3/2}|T_2|^{3/2}$. Both VTracker and SPRC extend the Zhang-Shasha algorithm in two ways. First, they use an affine–cost policy that is that the cost of each operation may be adjusted based on the context in which it is applied. Second, in a post-processing step, they apply a simplicity-based filter to discard the more unlikely solutions from the solution set produced by the tree-alignment phase. But, unlike the Zhang-Shasha algorithm and SPRC, VTracker is both reference-aware and context-aware based on back cross-references between nodes of the compared trees. As shown in Figure 3-7, VTracker, given two XML documents and a cost model, produces the cheapest edit script that will transform the first document into the second one in conjunction with the edit script associated with the reported distance. This section presents the details of VTracker, and shows how VTracker meets all the requirements of generic XML differencing.

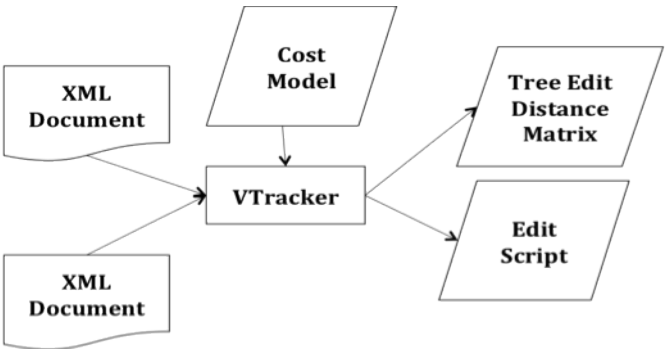



**Figure 3-7: VTracker's framework processing model**

### *3.3.1 XML Documents as Ordered Labeled Trees*

In VTracker, an XML tree is composed of a set of nodes, where each node is either a text node or an element node. A text node only has a value while an element node has a name, attributes, and/or children nodes. Each node has one parent. An attribute has a name and a value. A value is a literal value, an identifier, or a reference to an identifier. The reference model inside an XML document is either imposed by the underlying XML DTD or XML Schema, or

just logically embedded in the application. To be more specific, in DTD, an identifier attribute is declared as a type ID and a reference attribute is declared as an IDREF. Although, the XML referencing model is a critical player in the XML business, to the best of our knowledge, none of XML differencing approaches pays it the appropriate attention. One other thing to mention is that only text and element nodes are considered in VTracker since all other types of nodes, such as processing instructions and comments, do not add any value to the semantics of the document. Similarly, VTracker ignores empty text nodes and text nodes consisting of only white spaces.

### *3.3.2 The VTracker Cost Model*

The main contribution of VTracker is its innovative cost model. The cost model is the module responsible for assessing the cost of various edit-operations such as deleting a node, inserting a node, or changing a node label. The next few sub-sections discuss VTracker's context-oriented cost model such as change edit cost, deletion (or insertion) edit costs, and the relative weight between the change and deletion (or insertion) edit costs.

#### (1) Context-oriented Change Edit Cost

Given two tree nodes, a simple change edit cost assessment would follow a binary function that yields one of two values: zero in the case of perfect match, a constant value otherwise. However, in practice, two nodes that are not exactly the same may also not be entirely different. In VTracker, a matching cost is not a binary function but is an analog function where a matching cost value may range from zero, in the case of a perfect match, to a maximum constant, to indicate an impossible match. A simple implementation of such an analog cost function would measure the string distance between the two node names, their attributes, etc. However, some nodes that do not have similar names may have similar semantics, and vice versa, some nodes that may have literally similar names may have very distinct meanings. Therefore, in order to produce accurate solutions that are intuitive to domain experts, VTracker needs to be equipped with a domain-

specific cost function that correctly captures the understanding of subject-matter experts as to what constitutes similarity and difference among elements in the given domain. But, lacking such knowledge, a standard cost function can always be used as a default, which may, however, sometimes yield less accurate and non-intuitive results.

To address the challenge of coming up with a "good" domain-specific cost function, VTracker has an innovative method for synthesizing a cost function from the domain's XML schema by relying on the assumption that the XML schema captures in its syntax a substantial part of the domain's semantics. Essentially, VTracker assumes that the designers of the domain schema use their understanding of the domain semantics to identify the basic domain elements and to organize related elements into complex ones.

Once VTracker has been used first to develop a domain-specific cost function, it can be used to compare XML documents that are instances of the schema based on which the cost function has been developed. Figure 3-8 illustrates the bootstrapping process that should happen once, and for good, for each new domain. Given the domain's XSD along with the default cost model, VTracker is used to compare the schema elements against each other while trying to measure similarities, i.e. edit distance between them as if it is a regular XML document. VTracker then produces a distance matrix between defined elements. The distance matrix is the core of the cost model as it specifies the possibility that two elements are replaceable.

Table 3-1 depicts a sample of the cost model that was synthesized based on OWL/RDF XSDs. This sample shows all labels with distance more than 0%, and less than 8%. Each row shows the distance between two node labels followed by a percentage where 0.0% means a perfect match, and 100% means an impossible match. This distance is also interpreted as a similarity measure between nodes of the two given nodes. For instance, two nodes with a 15% distance would be more acceptable as a replacement of each other than those with

a 90% distance. As shown in this table, VTracker managed to uncover the semantics of the domain that are implicitly embedded in the underlying XSD, and was able to find only relevant matches. Then, the produced cost function is used to compare instances of this given XSD.


**Figure 3-8: VTracker domain bootstrapping process**

**Table 3-1: Sample of OWL/RDF synthesized cost model**

| | | |
|---|---|---|
| cardinality | maxCardinality | 2.78% |
| cardinality | minCardinality | 2.78% |
| subject | object | 3.70% |
| cardinality | qualifiedCardinality | 3.70% |
| cardinality | maxQualifiedCardinality | 4.63% |
| cardinality | minQualifiedCardinality | 4.63% |
| backwardCompatibleWith | incompatibleWith | 5.56% |
| maxCardinality | minCardinality | 5.56% |
| maxQualifiedCardinality | minQualifiedCardinality | 5.56% |
| maxQualifiedCardinality | qualifiedCardinality | 5.56% |
| minQualifiedCardinality | qualifiedCardinality | 5.56% |
| allValuesFrom | someValuesFrom | 6.48% |
| annotatedProperty | annotatedSource | 6.48% |
| annotatedProperty | annotatedTarget | 6.48% |
| annotatedSource | annotatedTarget | 6.48% |
| maxCardinality | maxQualifiedCardinality | 6.48% |
| maxCardinality | minQualifiedCardinality | 6.48% |
| maxCardinality | qualifiedCardinality | 6.48% |
| minCardinality | maxQualifiedCardinality | 6.48% |
| minCardinality | minQualifiedCardinality | 6.48% |
| minCardinality | qualifiedCardinality | 6.48% |
| sourceIndividual | targetIndividual | 6.48% |
| AsymmetricProperty | SymmetricProperty | 6.67% |
| IrreflexiveProperty | ReflexiveProperty | 6.67% |
| intersectionOf | unionOf | 7.41% |
| oneOf | unionOf | 7.41% |
| unionOf | oneOf | 7.41% |
| ReflexiveProperty | TransitiveProperty | 7.78% |

It is very important to mention here that the quality of the cost-model synthesizer largely depends on the richness and restriction of the given XSD. The richer and more restrictive the XSD the better quality of the cost-model achieved. If the given XSD does not capture the majority of the domain semantics, then the synthesizer will produce non-sense. It is essential to remember that it is always possible to manually configure the domain cost model, or even to fix the synthesized one. It is also important to mention that the bootstrapping process can help in building a cost function to translate between two different schemas. In this case, VTracker has to be provided by the two XSDs.

**(2) Context-oriented Deletion/Insertion Edit Cost**

A simple cost function assigns a uniform cost value to all deletion and insertion operations regardless of the context where the operation is applied. Thus, the cost of a node insertion/deletion is always the same, irrespective of whether or not that node's children are also to be deleted (or inserted). However, a parent node becomes less important if all its children are deleted. In order to produce more intuitive tree-edit sequences, VTracker uses an affine-cost policy.

The idea of affine-cost function was borrowed from the affine-gap cost function introduced in Bio-informatics sequence edit-distance problems [55]. Intuitively, the idea is that a single long insertion should be cheaper than several short ones of the same total lengths. For example, Figure 3-9 shows two possibilities of matching two strings “AUGCCUAGCCG” and “AUCG”. The first possibility has more gap fragments than the second. According to the affine-gap policy, the hypothesis is that “it is always cheaper by dozen.”, and that deletions and insertions tends to happen at contingent elements rather than dispersed ones.

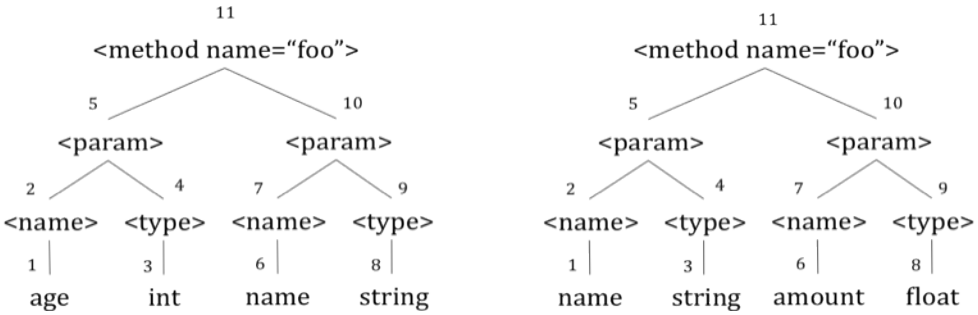
A U G C C U A G C C G
A - - - - U - - C - G
A U - - - - - - - C G

**Figure 3-9: A sample string-edit distance with affine-gap policy where dashes represent insertions and deletions**

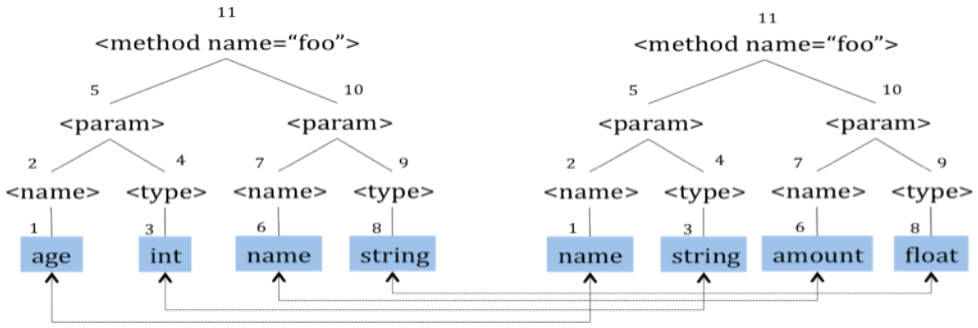
In VTracker, a node's deletion (or insertion) cost is context sensitive if all of a node's children are also candidates for deletion, this node is more likely to be deleted as well, and then the deletion cost of that node should be less than the regular deletion cost. The same is true for the insertion cost. To reflect this heuristic, the cost of the deletion or insertion of such a node is discounted by a certain percentage. Figure 3-10 illustrates the importance of an affine-cost function. First, assume a standard cost function where the cost of a deletion or an insertion is 3 while the cost of change is 6. Now, Let us consider the two trees of Figure 3-10 (a). According to the cost function, the cost of the differencing shown in Figure 3-10 (b) is 24 (four change operations) while the cost of the differencing of Figure 3-10 (c) is 30 (five deletion operations + five insertion operations). Therefore, according to this cost function, solution Figure 3-10 (b) is the optimal since it is cheaper. With a closer look at why Figure 3-10 (c) is so expensive, structure nodes like <param>, <name> and <type> are found to be more costly to delete, as they are so numerous. However, such structure nodes have no value if their contents are to be deleted. And, here comes the advantage of affine-cost function that discounts the edits to such structure-preserving nodes in case all their children are to be deleted. For example, according to a 66.6% discount policy, deleting or inserting any of the structure nodes will cost one unit instead of three each. In other words, applying affine-cost policy on the Figure 3-10(c), the cost will be 18 (two regular deletions of three units each + three discounted deletions of one unit each + two regular insertions of three units each + three discounted insertions of one unit each), which promotes the second solution to be the optimal one.

Then, the question is how to decide if a node is eligible for an affine discount. In other words, while calculating the edit cost between two nodes $x$ and $y$, the algorithm has to determine whether the children of $x$, $y$, or both are to be deleted. As shown in Code 3-2, for the cost function to decide whether this node is eligible for an affine policy discount, it has to leverage the distance calculations of this node's sub-forest. It checks if the cost of deleting the forest *fdist[0..x-*

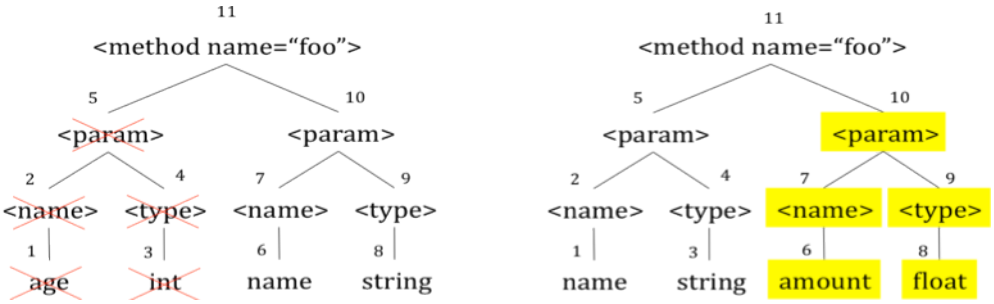
*1,0..y-1]* equals the summation of *fdist[0..lm$_1$(x)-1, 0..y-1]* plus the deletion cost of *T$_1$[lm$_1$(x)-1..x-1]*. A cell is eligible for a deletion affine discount, if and only if, either the cell is in the first column since the first column always means a full deletion, or the accumulated cost recorded with this node's children equals the cost of deleting the same children. The eligibility of the insertion affine discount is similarly calculated.




**(a) Two trees**




**(b) Differencing results of standard cost function**




**(c) Differencing results of affine-cost function**

**Figure 3-10: An example to illustrate the importance of affine-cost function**

```
FUNCTION IsDeleteAffineEligible (x, y)
START
 IF y = 0
 THEN // the whole tree is to be deleted
  RETURN true
 ELSE
  // Cost of matching the remaining forests to each other
  CostRemaingForest = fdist [lm1 (x)-1][y]

  // Cost of matching sub-forest is the actual cost minus
  // Cost of matching the remaining forests to each of the
  CostSubForest = fdist [x-1][y] – CostRemaingForest


   // Cost of deleting everything minus
   // Cost of matching the remaining forests to each other
   CostDelSubForest = fdist [x-1][0] – fdist [lm1 (x)-1][0]

  IF costSubForest = costDelSubForest
   RETURN true
  ELSE
 RETURN false
END
```

**Code 3-2: A pseudo code to check the eligibility of certain node for a deletion affine discount**

**(3) Relative Weight between Deletion and Change Edit Costs**

The previous two sections discuss the importance of context-sensitive cost functions on the dimensions of change, and deletion/insertion edit operations. The next question is what is the proper relative weight between these three types of operations? In practice, the cost value itself is not that important. Of greater importance is the relative cost between the different operations. In actuality, the cost of a deletion operation should always equal the cost of an insertion operation. Then the question becomes what is the relation between the cost value of deletion/insertion and the cost value of change. Many related works use a uniform cost model where deletion, insertion, and change operations have the same unit cost. However, this model gives the change operation more privilege over deletion and insertion. For example, if two nodes are totally different, but because of the uniform cost model matching them to each other will cost one unit while the cost of deleting the first node plus inserting the second node will cost two units, then

the match option will always be favored over deletion and insertion operations since it costs less. Therefore, in VTracker's cost model, the cost of change should be at least equal to the sum of the deletion and insertion costs; which gives a fair chance between all the three operations. In that way, if two nodes are:

- Perfect match, then their matching cost will be zero.
- Partially similar, then their matching cost will be prorated to the maximum matching cost, which should be less than the cost of deleting the first node plus the cost of inserting the second node.
- Entirely different, then the cost of matching them will equal the cost of deleting the first node plus the cost of inserting the second node, which gives both choices a fair chance to be favored by further calculations at subsequent nodes.

**(4) Basic Cost Functions**

VTracker uses a set of cost functions to measure the basic distance between different elements. One of the most common cost functions is the Levenshtein string-edit distance, and is used (a) to measure the distance between couples of string tokens, which is always normalized to the size of the two tokens; and (b) to measure the distance between two sets of tokens. In this case, Levenshtein's string-edit distance is used at two levels: once on the character level inside each token, and once more on the token-level for each set. Also Levenshtein's string-edit distance is used to measure the distance between attribute names, and between attribute values.

### *3.3.3 Considering Outgoing References*

The more fundamental advantage of VTracker over other differencing methods is the integration of the XML referencing structure into the XML containment structure, which enables VTracker to compare more complex structures (i.e. trees with back references) than others (that only compare proper trees). The approach presented in this thesis considers only references to nodes within the same document; references to external elements are currently ignored. However, a

workaround would include all external documents along with the main document into one tree structure.

A typical interpretation of such references is that the referenced element structure is meant to be entirely copied under the reference location; but, to avoid duplications, and inconsistencies, elements are reused through a reference to a common definition. SPRC, the precursor of VTracker, handled such referencing cases by just copying the content of the common element specification to every reference occurrence. This approach led to really large tree structures, especially in cases with many such cross-references. In addition to increasing the size of the tree and consequently increasing the time necessary for the computation, such "duplication" of elements to all their reference locations decouples them from each other and allows them to be treated as independent entities with just an "accidental" similarity in their internal structure and naming, which fundamentally misrepresents the intent of the schema designer.

The question then becomes how the cost function should be adjusted in order to compute the differences of two nodes in terms of the similarities and differences of the elements they contain and refer to. The answer to this question is straightforward: a referenced structure should be considered as an extension to the containment structure. As explained in Lemma 3-3, in order to assess the matching edit cost between two nodes $x$ and $y$, the following cases have to be considered:

- Neither node has a hyperlink attribute: a regular matching cost assessment is applied either through a domain-specific cost function or by applying a string-edit distance between the element names, attributes, and values.
- One node has a hyperlink attribute: a tree-edit distance measure is calculated between the referenced structure on the hyperlink side against the entire sub-tree on the other side.
- Both nodes have hyperlink attributes: a tree-edit distance measure is calculated between both referenced structures.

**Lemma 3-3: Reference-aware cost function**

In order to consider the reference-structure as a supplemental part of the tree-edit distance calculation, the cost function γ is modified to be:

$$\gamma'(x,y,p) = \begin{cases} \dfrac{tdist(x',y,p)}{tdist(x',\lambda)+\gamma(\lambda,y)}\gamma_{\max} & x \to x' \\ \dfrac{tdist(x,y',p)}{\gamma(x,\lambda)+tdist(\lambda,y')}\gamma_{\max} & y \to y' \\ \dfrac{tdist(x',y',p)}{tdist(x',\lambda)+tdist(\lambda,y')}\gamma_{\max} & x \to x' \& y \to y' \\ \gamma(x,y) & otherwise \end{cases}$$

Modifying a cost function to be reference-aware is conceptually a simple task. However, there are a few issues that have to be considered during the implementation. The following paragraphs discuss three challenges to be considered during the implementation of a reference-aware cost function.

**(1) Normalized values**

The expected output of the cost-assessment function is a value between zero and the maximum matching cost. However, following a hyperlink and involving a reference structure in the calculation may yield a distance value that is relative to the size of the referenced structures. Therefore, a normalized step is required to make sure that the reported matching distance is within range. As shown in Lemma 3-3, this can be accomplished by dividing the calculated tree-edit distance of the referenced structures by the cost of deleting them, which will yield a value less than, or equal to, 1.0. Finally, this value is multiplied by the maximum matching cost so that it is leveled with the normal matching cost.

**(2) Infinite Loops**

A challenge that arises when XML elements hold references to other elements is to prevent the algorithm from falling into an infinite loop, as it follows these references. Hence, the cost function should be equipped with a simple stack trace *p* that maintains the recursion path of the current calculations. Accordingly, the

cost function $\gamma'(x, y, p)$ accepts a recursion stack parameter $p$ to reassure not visiting the same state twice.

**(3) Performance**

One last thing to discuss here is the performance of reference-aware edit distance calculation. The performance of the original Zhang-Shasha algorithm largely relies on the order in which sub-trees are compared to each other. The algorithm has a very specific order by which it calculates sub-problems so that rework is avoided or at least minimized. The Zhang-Shasha algorithm uses the concept of key-trees, on top of the dynamic programming model, to decide the order in which sub-problems should be solved so that no recursion is required, in part, because recursive calculations dramatically affect the amount of memory space required to solve the problem. However, VTracker, in addition to the containment hierarchy, also follows the reference relations between elements which affect the actual dependencies between sub-trees and consequently impacts the "proper" order in which sub-tree mappings should be calculated. References can unexpectedly happen from any node to any other node which can dramatically change the order in which sub-trees are compared and which dynamically increases the degree of recursion required to solve the problem.

To mitigate this problem, VTracker sorts key sub-trees based on their references, following the following two sorting criteria.

a) ***Popularity of the node***: the number of inbound references. Sub-trees with more inbound references should be considered before others with a fewer number of inbound references. This criterion guarantees that high-demand nodes are always calculated before low-demand ones so that calculations of high-demand sub-trees are always ready first which in turn dramatically decreases the number of possible recursions
b) ***Pre-requisites of a node***: the number of outbound references. Nodes with many out-bound references (i.e. hyperlinks) are harder to calculate especially if their referenced sub-trees have not been calculated at that time. Therefore,

sub-trees with many pre-requisites should be delayed to the end so that most of their referenced sub-trees are calculated first.

### *3.3.4 Considering Usage-Context (Incoming References)*

In usage-context matching, VTracker considers not only the internal and referenced structure of an element but also the context in which this element is used, namely the elements from which this element is being referenced. As discussed earlier, usage-context distance is used to resolve confusions that may happen in the regular tree-edit distance calculation.

In a post-calculation process, usage-context distance measures are calculated and combined with standard tree-edit distance measures into a new context-aware tree-edit distance measure. For each two nodes $x$ and $y$, two context-usage sets are established from nodes that having references to node $x$ and node $y$, respectively. Then, the usage-context distance between the two sets is calculated as the Levenshtein distance [86] between elements of the sets, where the distance between any two elements is the tree-edit distance between them, and the total Levenshtein distance is then called the *usage-context distance* between $x$ and $y$. Finally, the context-aware tree-edit distance measure is the average between the usage-context distance and the tree-edit distance measure.

### *3.3.5 Selecting the Optimal Edit Script*

The original Zhang-Shasha algorithm describes only the process of calculating a tree-edit distance and does not describe the proper way of recovering the edit script associated with this distance. Under the conditions of a perfect cost function there should be only one optimal edit script that transforms one tree into the other. In practice, such a perfect cost function is unlikely (even impossible) to exist leading to the fact that a tree-edit distance may have multiple corresponding edit scripts all with the same cheapest total cost value. It is important to mention that these various edit scripts may be quite different and they may report very different solutions.

For example, Figure 3-11 (a) shows segments from two RNA secondary structures represented in two kinds of tree structures. This example is interesting because in both tree representations, there are three possible edit scripts, i.e. solutions all with the same cost shown in Figure 3-11 (b) - (d). Each of these edit scripts corresponds to a different sequence of evolutionary operations that may have led to the production of one tree rather than another. The question is then which one should be reported as the differencing result. VTracker uses an innovative set of simplicity heuristics, which is designed to discard the unlikely solutions from the possible set. During this phase, three different simplicity criteria are applied to decrease the set's cardinality by eliminating solutions that do not meet the criteria.

**(1) Path Minimality**

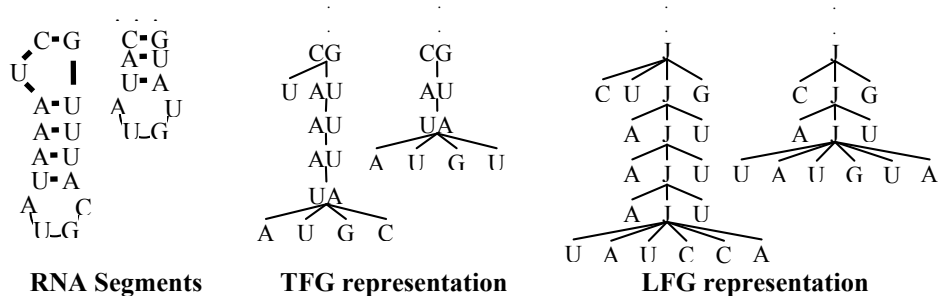
Intuitively, the first simplicity criterion eliminates "non minimal paths". When there is more than one different path with the same minimum cost, the one with the least number of deletion and/or insertion operations is preferable. This criterion aligns very well with the requirement of having a minimal delta, i.e., a minimal edit script. In the example of Figure 3-11, since all solutions have the same number of edit-operations, no solution is discarded in this phase of filtration.

**(2) Vertical Simplicity**

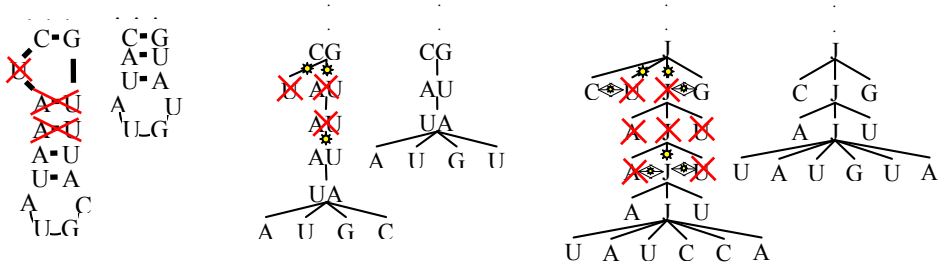
The second simplicity heuristic eliminates any edit sequences in which "non-contiguous similar edit operations" exist. Intuitively, this rule assumes that a contiguous sequence of edit operations of the same type essentially represents a single mutation or refactoring on a segment of neighboring nodes. Thus, when there are multiple different edit-operation scripts with the same minimum cost, and the same number of operations, the one with the least number of changes (refractions) of operational types along a tree branch is preferable.

This heuristic is implemented by counting the number of *vertical refraction points*. A vertical refraction point is defined as a node where the editing operation applied to its parent differs from the operation applied to this node. For example,

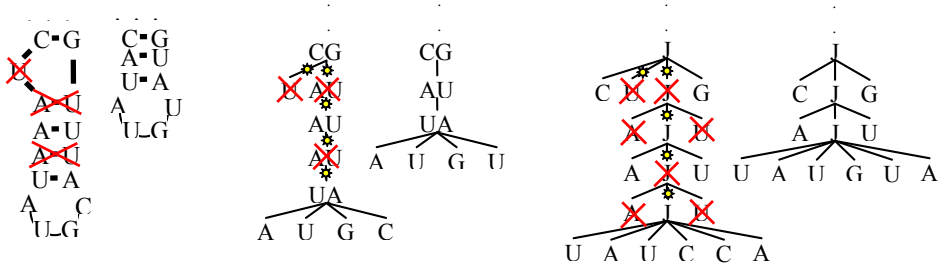
solution two has five vertical refraction points; contrast this with either solution one or solution three that each has three vertical refraction points only. Therefore, solutions one and three are simpler than solution two as they have fewer vertical refraction points; hence, solution two is discarded while solutions one and three pass this filtration step.



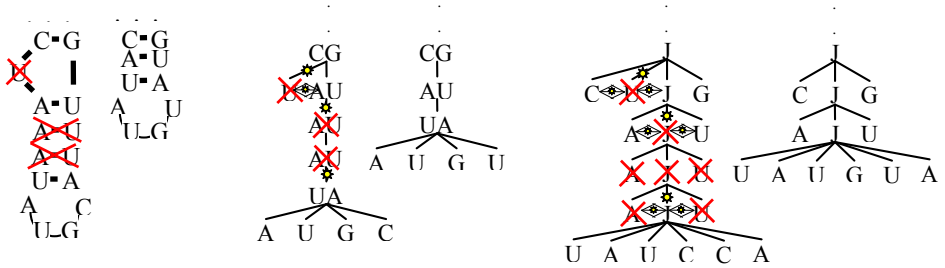

**(a) Two RNA segments**



**(b) Solution 1**



**(c) Solution 2**



**(d) Solution 3**

**Figure 3-11: An RNA comparison example showing the steps of the simplicity heuristic filtration process**

**(3) Horizontal Simplicity**

This filtering criterion is implemented by counting the number of *horizontal refraction points*. A horizontal refraction point is defined as a node where the operation applied to its sibling differs from the operation applied to this node. For example, in the case of the Tight Fine-Grained (TFG) tree representation solution one has no horizontal refraction points and solution three has one refraction points; in the case of the Loose Fine-Grained (LFG) tree representation solution one has four horizontal refraction points and solution three has six. Therefore, solution one is identified as the simplest edit script by having the most contiguous similar edit operations.

### *3.3.6 Domain–Aware Optimizations*

Unlike other contributions of VTracker, contributions presented in this section are specific to the Zhang-Shasha algorithm. Other contributions are generic and applicable to any other tree-edit distance approach.

Section 3.1 discusses one of the main inefficiency reasons of generic methods that are performing unnecessary steps in trying to match nodes that are not possible to map to each other. As mentioned before, the Zhang-Shasha algorithm is based on splitting each tree into a set of key sub-trees, and then to calculate the edit distance between all combinations of these sub-trees. Hence, VTracker, when provided with a domain-specific cost function that defines the similarity measure between various kinds of elements, optimizes the algorithm performance by deciding on the feasibility of a sub-tree-to-sub-tree correction process before carrying it out. In other words, it should not start matching two sub-trees if the roots of the two sub-trees are not of *replaceable* types. In this way, an unfeasible sub-tree will be skipped while focusing only on the feasible ones.

Formally speaking, the similarity measure $\rho$ is always true by default unless specified otherwise by the following formula:

$$\rho(x,y) = \begin{cases} false & \gamma(l_1(x), l_2(y)) > threshold \\ true & otherwise \end{cases}$$

where node-label distances are provided by the domain-specific cost function. In this way, the tree-edit distance between two sub-trees is skipped if the two root nodes are not replaceable which leads to an optimized version of the forest distance calculations Lemma 3-4.

**Lemma 3-4: Forest-edit distance with similarity measure**

$$
\begin{aligned}
&fdist(T_1[x_1 \ldots x_2], T_2[y_1 \ldots y_2]) \\
&= \min \begin{cases} \begin{cases} fdist(T_1[x_1 \ldots lm_1(x_2)-1], T_2[y_1 \ldots lm_2(y_2)-1]) + tdist(x_2, y_2) & \rho(x_2, y_2) \\ \infty & otherwise \end{cases} \\ fdist(T_1[x_1 \ldots x_2 - 1], T_2[y_1 \ldots y_2]) + \gamma(x_2, \lambda) \\ fdist(T_1[x_1 \ldots x_2], T_2[y_1 \ldots y_2 - 1]) + \gamma(\lambda, y_2) \end{cases}
\end{aligned}
$$

**Proof**

When nodes $x_2$ and $y_2$ are not replaceable, the first option in the *tdist* (Lemma 3-1) formula becomes very expensive, and will be discarded, which will leave two options only:

$$
\begin{aligned}
tdist(x_2, y_2) &= tdist(T_1[lm_1(x_2) \ldots x_2], T_2[lm_2(y_2) \ldots y_2]) \\
&= \min \begin{cases} fdist(T_1[lm_1(x_2) \ldots x_2 - 1], T_2[lm_2(y_2) \ldots y_2]) + \gamma(x_2, \lambda) \\ fdist(T_1[lm_1(x_2) \ldots x_2], T_2[lm_2(y_2) \ldots y_2 - 1]) + \gamma(\lambda, y_2) \end{cases}
\end{aligned}
$$

Substituting these two options with Lemma 3-2 results in the following forest distance formula representing the case when $x_2$ and $y_2$ which are not replaceable.

$$
\begin{aligned}
&fdist(T_1[x_1...x_2],T_2[y_1..y_2]) \\
&= \min\begin{cases}
fdist(T_1[x_1...lm_1(x_2)-1],T_2[y_1...lm_2(y_2)-1]) + fdist(T_1[lm_1(x_2)...x_2-1],T_2[lm_2(y_2)..y_2]) + \gamma(x_2,\lambda) \\
fdist(T_1[x_1...lm_1(x_2)-1],T_2[y_1...lm_2(y_2)-1]) + fdist(T_1[lm_1(x_2)...x_2],T_2[lm_2(y_2)..y_2-1]) + \gamma(\lambda,y_2) \\
fdist(T_1[x_1...x_2-1],T_2[y_1..y_2]) + \gamma(x_2,\lambda) \\
fdist(T_1[x_1...x_2],T_2[y_1..y_2-1]) + \gamma(\lambda,y_2)
\end{cases}
\end{aligned}
$$

Now, in order to prove Lemma 3-4, it is necessary to prove that the first two options in the above formula are not necessary since they are considered in the other two options. In other words, it is necessary to prove that

$$
\begin{aligned}
&fdist(T_1[x_1...lm_1(x_2)-1],T_2[y_1...lm_2(y_2)-1]) + fdist(T_1[lm_1(x_2)...x_2-1],T_2[lm_2(y_2)..y_2]) \geq \\
&\quad fdist(T_1[x_1...x_2-1],T_2[y_1..y_2])
\end{aligned}
\qquad ...1
$$

Finally, Lemma 3-4 will enhance the performance of a generic differencing method to skip unnecessary sub-tree matching. In this way, there is a decrease in the complexity of being $O(n^4)$ in worst case and $O(n^3)$ in average case, to be $O(n^2)$ in average case and reduce the possibility of the worst case even if its complexity remains the same.

### 3.4 VTracker as a generic XML differencing

This section discusses how VTracker meets the requirements of being a generic XML differencing approach.

- Not domain specific: by definition VTracker is designed to handle any kind of XML differencing problem. Yet, it is capable of becoming domain-aware, using a domain-specific cost function, and constructed in the bootstrapping process described in Section 3.3.2, in order to produce results that are sound and reasonable in terms of the domain knowledge and semantics.
- Meaningful minimal edit script: VTracker accomplishes this objective in many ways such as the affine-cost policy described in 3.3.2, the simplicity heuristics in 3.3.5.

- Hierarchal data structure: VTracker views an XML document as an ordered labeled tree since it is based on the Zhang-Shasha algorithm described in Section 3.2. Also, mapped elements should obey both the ancestor-child and siblings.
- Changes anywhere: VTracker does not favor certain kinds of changes over other kinds. VTracker is capable of detecting changes happening to internal structure nodes as efficiently as changes happening to leaf nodes. VTracker does not favor certain patterns of changes or edit operations.
- Object Identity: in VTracker, an element is identified by its name, attributes, value, and structure. VTracker also uses ancestor and siblings relationships to identify an element. Although VTracker does assume or require that given XML documents have some kind of atomic IDs, it utilizes key attributes if specified by the domain configuration. Moreover, it uses both the reference and usage-context structure to reinforce the identity of a certain element.
- No prior change tracking: by definition VTracker does not require edits to be done through a certain tool, utility, or IDE. It is also capable of comparing documents originated from different sources, or by different vendors.
- Efficiency: VTracker provides an optimization technique that is based on a domain-specific cost function; it focuses on comparing trees that are replaceable as described in Section 3.3.6.
- Reference structure: VTracker views the XML reference structure as a part of referring structures as described in Section 3.3.3.
- Usage-Context Structure: VTracker uses usage-context similarity as an extra measure to validate and reinforce the calculated tree-edit distance results as explained in Section 3.3.4.

# Chapter Four Applying VTracker to Specific Domains

Chapter Three explains the details of VTracker as a generic XML differencing method. Yet, VTracker is capable of being domain-aware through a domain-specific cost function. This chapter explains in detail how VTracker can be customized for a certain domain, and how its contributions such as affine-cost policy, reference-aware differencing, usage-context similarity assessment, simplicity heuristics for solution filtering, and synthesized cost function are applicable to each of these domains.

## 4.1 Applying VTracker to Ontology Matching

As previously explained, the ontology matching is the process of finding a semantic mapping between elements of two different ontologies. This thesis focuses on OWL/RDF as an example of ontology specification language. It was shown in Figure 2-3 how an OWL/RDF described in XML syntax can be represented as an ordered labeled tree.

### (1) Affine-Cost Policy

As discussed in the details of VTracker, affine-cost policy is important to prevent structural formality from having a negative influence on the quality of results. The objective of an affine-cost function is to assign a reduced cost when deleting or inserting internal nodes where all these children are deleted (or inserted) as well. The idea is based on the hypothesis that the purpose of an internal node is to group the structure of its content. Therefore, if its children are deleted, then this internal node loses its purpose and consequently needs to be deleted as well. The affine-cost function reduces its deletion cost to indicate to the algorithm the diminished importance of such a node.

Figure 4-1 illustrates the necessity of an affine-cost policy. This example matches two ontologies with two class definitions each, shown in Figure 4-1 (a) and (b). First in an OWL ontology definition, the number of structure nodes exceeds that of the text nodes, which means that structure nodes have the upper hand on the matching decision. However, in this example, structure nodes can

negatively influence such a decision. Let us compare the tree-edit distance of the two solutions of Figure 4-1 (c) and (d) when following a fixed deletion insertion cost versus an affine-cost function. The first solution is where a part class matches a collection class while the reference class is mapped to the part class, which is a rather counterintuitive solution. The second solution keeps the part class unchanged, deletes the reference class, and inserts the collection class.

The following calculations are based on the standard cost function where a deletion costs three units, an insertion costs three units, and a change costs six units. As shown in the table below, following a fixed costing policy, the number of internal nodes affects the total cost of the solution making it a very expensive choice. However, when following an affine-cost policy, the six internal nodes will receive a cost discount since their children are deleted as well. It should now be evident how an affine-cost policy would help to promote solutions that have a significant number of structure changes.

| **Solutions** | **Fixed cost policy** | **Affine cost policy** |
|---|---|---|
| **Solution 1:** | | 4.1.1.1.1 |
| 4 change Operations | 4 * 6 =24 | 4 * 6 =24 |
| 2 attribute changes | 2 * 2 = 4 | 2 * 2 = 4 |
| | Total = 28 units | Total = 28 units |
| **Solution 2:** | 4 * 3 = 12 | 4 * 3 = 12 |
| 4 leaf node deletions | 6 * 3 = 18 | 6 * 1.5 = 9 |
| 6 internal node deletions | Total = 30 units | Total = 21 units |

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE rdf:RDF >
-<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#" xmlns:rdf="&rdf;" xmlns:rdfs="&rdfs;">
  -<owl:Class rdf:ID="Part">
     <rdfs:label xml:lang="en">Part</rdfs:label>
     <rdfs:comment xml:lang="en">A part of something (either Book or Proceedings). </rdfs:comment>
   </owl:Class>
  -<owl:Class rdf:ID="Reference">
     <rdfs:label xml:lang="en">Reference</rdfs:label>
     <rdfs:comment xml:lang="en">Base class for all entries </rdfs:comment>
   </owl:Class>
 </rdf:RDF>
```

**(a) Ontology # 1**

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE rdf:RDF >
-<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#" xmlns:rdf="&rdf;" xmlns:rdfs="&rdfs;">
  -<owl:Class rdf:ID="Collection">
     <rdfs:label xml:lang="en">Collection</rdfs:label>
     <rdfs:comment xml:lang="en">A book that is collection of texts or articles.</rdfs:comment>
   </owl:Class>
  -<owl:Class rdf:ID="Part">
     <rdfs:label xml:lang="en">Part</rdfs:label>
     <rdfs:comment xml:lang="en">A part of something (either Book or Proceedings).</rdfs:comment>
   </owl:Class>
 </rdf:RDF>
```

**(b) Ontology # 2**

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE rdf:RDF >
-<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#" xmlns:rdf="&rdf;" xmlns:rdfs="&rdfs;">
  -<owl:Class rdf:ID = "Collection"  >
     <rdfs:label xml:lang="en">Collection</rdfs:label>
     <rdfs:comment xml:lang="en">A book that is collection of texts or articles.</rdfs:comment>
   </owl:Class>
  -<owl:Class rdf:ID = "Part"  >
     <rdfs:label xml:lang="en">Part</rdfs:label>
     <rdfs:comment xml:lang="en">A part of something (either Book or Proceedings).</rdfs:comment>
   </owl:Class>
 </rdf:RDF>
```

**(c) Differencing results without a fixed deletion/insertion cost**

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE rdf:RDF >
-<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#" xmlns:rdf="&rdf;" xmlns:rdfs="&rdfs;">
  -<owl:Class rdf:ID="Collection">
     <rdfs:label xml:lang="en">Collection</rdfs:label>
     <rdfs:comment xml:lang="en">A book that is collection of texts or articles.</rdfs:comment>
   </owl:Class>
  -<owl:Class rdf:ID="Part">
     <rdfs:label xml:lang="en">Part</rdfs:label>
     <rdfs:comment xml:lang="en">A part of something (either Book or Proceedings).</rdfs:comment>
   </owl:Class>
  -<owl:Class rdf:ID="Reference">
     <rdfs:label xml:lang="en">Reference</rdfs:label>
     <rdfs:comment xml:lang="en">Base class for all entries </rdfs:comment>
   </owl:Class>
 </rdf:RDF>
```

**(d) Differencing results following an affine-cost policy**

**Figure 4-1: An OWL/RDF matching example emphasizes the importance of an affine cost function**

### (2) Ontology Reference Structure

The example in Figure 4-2 shows two ontologies: the first ontology, shown in Figure 4-2 (a), defines two classes, a Resource and a Monograph where a Monograph is a sub class of a Resource; the second ontology, Figure 4-2 (b), defines four classes, Part, Reference, Chapter, and Book where Chapter and Book are sub classes of Part and Reference, respectively. Intuitively, the Resource and Reference classes are very similar in terms of their labels and comments, and should be matched to each other. Now, one of the two classes of the first ontology is successfully matched.

The next question is which class in the second ontology should be mapped to the Monograph class in the first ontology. Comparing the Monograph class definition against the remaining three classes Part, Chapter, and Book, Monograph has keywords that are similar to the three classes. Therefore, due to such confusion, a solution could randomly map Monograph to any of the three classes. Figure 4-2 (c) shows one of such random solution. However, since VTracker is reference-aware, it easily resolved this confusion based on the sub-class relation between the Monograph and Resource classes that are mapped to the Reference class. In another way, the Monograph should be mapped to a sub-class of the Reference class. As Figure 4-2 (d) shows, the perfect solution occurs where Monograph is matched to Book since both are sub-classes of matched classes.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE rdf:RDF >
-<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#" xmlns:rdf="&rdf;" xmlns:rdfs="&rdfs;">
   -<owl:Class rdf:ID="Resource">
      <rdfs:label xml:lang="en">Reference</rdfs:label>
      <rdfs:comment xml:lang="en">Base class for all entries </rdfs:comment>
    </owl:Class>
   -<owl:Class rdf:ID="Monograph">
      <rdfs:subClassOf rdf:resource="#Resource"/>
      <rdfs:label xml:lang="en">Monograph</rdfs:label>
      <rdfs:comment xml:lang="en">A book that is a single entity, as opposed to a
collection.</rdfs:comment>
    </owl:Class>
 </rdf:RDF>
```

**(a) Ontology # 1**

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE rdf:RDF >
-<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#" xmlns:rdf="&rdf;" xmlns:rdfs="&rdfs;">
   -<owl:Class rdf:ID="Part">
      <rdfs:label xml:lang="en">Part</rdfs:label>
      <rdfs:comment xml:lang="en">A part of something (either Book or Proceedings).
</rdfs:comment>
    </owl:Class>
   -<owl:Class rdf:ID="Reference">
      <rdfs:label xml:lang="en">Reference</rdfs:label>
      <rdfs:comment xml:lang="en">Base class for all entries </rdfs:comment>
    </owl:Class>
   -<owl:Class rdf:ID="Chapter">
      <rdfs:subClassOf rdf:resource="#Part"/>
      <rdfs:label xml:lang="en">BookPart</rdfs:label>
      <rdfs:comment xml:lang="en">A chapter (or section or whatever) of a book having its
own title.</rdfs:comment>
    </owl:Class>
   -<owl:Class rdf:ID="Book">
      <rdfs:subClassOf rdf:resource="#Reference"/>
      <rdfs:label xml:lang="en">Book</rdfs:label>
      <rdfs:comment xml:lang="en">A book that may be a monograph or a collection of
written texts</rdfs:comment>
    </owl:Class>
 </rdf:RDF>
```

**(b) Ontology # 2**

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE rdf:RDF >
-<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#" xmlns:rdf="&rdf;"
xmlns:rdfs="&rdfs;">
   -<owl:Class rdf:ID="Part">
      <rdfs:label xml:lang="en">Part</rdfs:label>
      <rdfs:comment xml:lang="en">A part of something (either Book or Proceedings).
</rdfs:comment>
    </owl:Class>
   -<owl:Class rdf:ID = "Reference"  >
      <rdfs:label xml:lang="en">Reference</rdfs:label>
      <rdfs:comment xml:lang="en">Base class for all entries </rdfs:comment>
    </owl:Class>
   -<owl:Class rdf:ID = "Chapter"  >
      <rdfs:subClassOf rdf:resource = "#Part"  />
      <rdfs:label xml:lang="en">BookPart</rdfs:label>
      <rdfs:comment xml:lang="en">A chapter (or section or whatever) of a book having its
own title.</rdfs:comment>
    </owl:Class>
   -<owl:Class rdf:ID="Book">
      <rdfs:subClassOf rdf:resource="#Reference"/>
      <rdfs:label xml:lang="en">Book</rdfs:label>
      <rdfs:comment xml:lang="en">A book that may be a monograph or a collection of
written texts</rdfs:comment>
    </owl:Class>
  </rdf:RDF>
```

**(c) Not reference-aware differencing**

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE rdf:RDF >
-<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#" xmlns:rdf="&rdf;" xmlns:rdfs="&rdfs;">
   -<owl:Class rdf:ID="Part">
      <rdfs:label xml:lang="en">Part</rdfs:label>
      <rdfs:comment xml:lang="en">A part of something (either Book or Proceedings).
</rdfs:comment>
    </owl:Class>
   -<owl:Class rdf:ID = "Reference"  >
      <rdfs:label xml:lang="en">Reference</rdfs:label>
      <rdfs:comment xml:lang="en">Base class for all entries </rdfs:comment>
    </owl:Class>
   -<owl:Class rdf:ID="Chapter">
      <rdfs:subClassOf rdf:resource="#Part"/>
      <rdfs:label xml:lang="en">BookPart</rdfs:label>
      <rdfs:comment xml:lang="en">A chapter (or section or whatever) of a book having its
own title.</rdfs:comment>
    </owl:Class>
   -<owl:Class rdf:ID = "Book"  >
      <rdfs:subClassOf rdf:resource = "#Reference"  />
      <rdfs:label xml:lang="en">Book</rdfs:label>
      <rdfs:comment xml:lang="en">A book that may be a monograph or a collection of written
texts</rdfs:comment>
    </owl:Class>
  </rdf:RDF>
```

**(d) VTracker Reference-aware differencing**

**Figure 4-2: An OWL/RDF matching emphasizes the importance of reference structure**

## 4.2 Implementation

This section explains the implementation outline of VTracker as a generic XML differencing approach. VTracker is implemented using Java 2 Standard Edition (J2SE), and therefore is it portable to different operating systems and platforms, and it is also capable of running as a standalone or a web application.

A typical deployment of VTracker is composed of a mandatory component VTrackerCore in addition to one or more domain-specific modules. VTrackerCore is main component of VTracker and it implements all the contributions presented in this thesis. The VTracker core is composed of the following components.

(1) **TreeEditingSuggestor:** that is given two XML documents and a cost function it produces a tree-edit distance matrix, and optionally an edit scripts associated with the calculated distances. This class is responsible for implementing the tree-edit distance algorithm. VTracker includes different implementations of the tree-edit distance algorithm such as the basic algorithm, the basic algorithm with affine-cost computation, and the algorithm that can be configured with domain-specific parameters for efficiency improvement.

(2) **CostAssessor:** the cost function is provided to TreeEditingSuggestor in the form of an instance that implements the abstract class CostAssessor. This CostAssessor is responsible for assessing the cost of deleting or inserting a certain node, in addition to deciding the cost of replacing one node with another. VTrackerCore provides two types of CostAssessors: XMLCostAssessor and RefXMLCostAssessor. Each domain then decides which one to use according to whether the domain may include references or not. In this way, VTracker distinguishes between the approach and the cost function. It is also important to mention that RefXMLCostAssessor is the component that is responsible for assessing the reference structure similarity.

Given two nodes *x* and *y*, in order to assess the similarity measure, RefXMLCostAssessor checks if:

- Neither node has a hyperlink attribute: a regular matching cost assessment is applied either through a domain-specific cost function or by applying a string-edit distance between the element names, attributes, and values.
- One node has a hyperlink attribute: a tree-edit distance measure is calculated between the referenced structure on the hyperlink side against the entire sub-tree on the other side.
- Both nodes have hyperlink attributes: a tree-edit distance measure is calculated between both referenced structures.

(3) **Edit Script backtracker**: is an optional module that runs when the tree-edit script is required. In application domains where the edit distance should be accompanied with an edit script, this module is responsible for building the edit script that is associated with the calculated tree-edit distance. Tree-edit traces map is recorded during the distance calculation process. These maps are matrixes where each cell records how the corresponding edit-distance was calculated; which one(s) of three edit choice led to that distance. In this way, a calculated edit distance can be tracked back to determine the sequence of edit operations involved in such a distance. In cases where multiple edit scripts are possible, this module employs the three-filtration steps of the simplicity heuristics.

(4) **Advanced Comparison**: this module is responsible for recognizing move operations as a combination of deletion from one place and insertion at another place. This module starts with calling the TreeEditingSuggestor for the two given XML documents in order to recognize deletions, insertions, and change operations. The advanced Comparison Module then strips out the two trees from all nodes except from sub-trees that are entirely deleted or inserted. Then, this module calls TreeEditingSuggestor on the stripped trees trying to

find if there are any possible matches, if yes, these are recognized as moves; otherwise they are reported as regular deletions or insertions.

### 4.3 The Configuration Process

This section discusses various configuration options of VTracker, and how they affect the end results. The following options have to be provided by the domain expert through a configuration file.

(1) Cost function: It can be manually composed, or automatically generated by VTracker in a bootstrapping step from the domain XSD. A cost model is the module responsible for assessing the cost of various edit-operations such as deleting a node, inserting a node, or changing a node label. A simple cost function would assign the same cost to all operations, with deletions and insertions having the same cost as changes. A better cost-function assigns costs based on the importance of the edit operating on the semantics of the document such that deleting important nodes should be more expensive than deleting optional or less important nodes. VTracker uses a context-oriented cost model where the cost of deleting or inserting a node is determined in the context of other edit operations happening around this node. Section 3.3.2 described VTracker's context-oriented cost model, and the relative weight between the change and deletion (or insertion) edit costs.

(2) Key elements (optional): is a list of schema element names that appear in an XML document. In some domains, a user is not interested in detailed edit operations that may happen to all types of elements. Instead, the domain expert is only interested in changes happening to some particular elements. This configuration option will not affect the calculation process but will be used in the solution report phase to filter out elements that are not key elements. For example, in OWL/RDF the objective is to find mapping between Class DatatypeProperty, and ObjectProperty but not Restriction nor subClassOf, etc. Hence, the elements Class DatatypeProperty, and owl:ObjectProperty should be considered key elements.

(3) Key attributes: this configuration option is used to give VTracker a hint about the relative importance of some attributes. In other words, attributes specified in this option are given more importance than other types of attributes. For example, since key attributes such as @id, @attribute, etc are relatively more important than values of other attributes, changing or deleting any of these attributes costs double the changing or deleting regular attributes. Similarly, a perfect match between two key attributes is rewarded as double as matching regular attributes.

(4) Meta Elements: a list of elements such as scripts and comments in HTML, XML instructions and comments, etc. that should not be considered during the differencing process. These elements will be suppressed during the differencing process.

(5) Meta attributes: is a list of attributes, similar to meta elements, such as identifiers used by IDEs, or those used for reverse engineering backward compatibility that are not to be considered during the differencing process.

(6) Reference Structure: the domain expert must decide whether the provided XML documents will include a reference structure, in which case the following two options are to be provided.

- ID attributes: a list of attribute names that are used as object IDs. In many cases, this list of attributes overlaps with the list provided as Key attributes.
- IDRef attributes: a list of attribute names that will reference, and have hyperlinks to, objects identified by ID attributes.

Table 4-1 shows the configuration necessary to customize VTracker for domains of interest. As shown in this table, the process of customizing VTracker to a certain domain is a simple process.

**Table 4-1: VTracker's system configurations for various domains**

| General | OWL | WSDL | BPEL | UML XMI | XHTML | RNA |
|---|---|---|---|---|---|---|
| Cost Function | Synthesized | Synthesized | Synthesized | Synthesized | Synthesized | |
| Affine-Cost Policy | Yes | Yes | Yes | Yes | Yes | |
| Key Element(s) | owl:Class<br>owl:DatatypeProperty<br>owl:ObjectProperty | operation<br>message<br>xsd:element | | packagedElement<br>ownedOperation | | |
| Key Attribute(s) | rdf:ID<br>rdf:about | @name | @name | @name | @id<br>@name<br>@href | |
| Meta Element(s) | | | bpws:import | eAnnotations | meta<br>script<br>link | |
| Meta Attribute(s) | | | | @xmi:id<br>@type<br>@general | | |
| **Reference Structure** | Yes | Yes | Yes | Yes | No | No |
| ID Attribute | rdf:ID<br>rdf:about | @name | @name | @xmi:id | | |
| IDRef Attribute(s) | rdf:resource<br>rdf:parseType<br>rdf:datatype<br>*element name** | @type<br>@element | @partnerLink<br>@linkName<br>@portType<br>@operation | @type<br>@general | | |

* In OWL/RDF class or property definition can be referenced by instantiating a new element of the class or property. In this case, the referencing happens through the instance element name.

# Chapter Five Evaluation

Chapter Three explains VTracker as a generic XML differencing method capable of becoming domain-aware. This chapter supports that statement with a set of empirical experiments that illustrate the contributions of VTracker over related work. The following set of experiments starts by evaluating basic features of VTracker, and moves gradually towards increasingly complex ones.

In the following experiments, the quality of results is measured in terms of Precision and Recall. Given a set of target mappings and a set of calculated ones, a precision is defined as the probability of a (randomly selected) calculated mapping to be in the target set. Similarly, a recall is defined as the probability of a (randomly selected) target mapping to be in the calculated set. In this way, a precision is calculated by dividing the number of correct calculated mappings by the size of the calculated set; while a recall is calculated by dividing the number of correct calculated mappings by the size of target set. Additionally, a precision and recall can be combined into a single measure called F-Measure that is calculated as twice the precision times recall divided by the summation of precision and recall.

## 5.1 General Quality Evaluation Experiment

The objective of this experiment is to evaluate the feasibility of the concept of domain-aware optimization explained in Section 3.3.6. The hypothesis behind this kind of optimization shows that the performance of the tree-edit distance algorithm can be improved by specifying a general similarity measure between different kinds of node labels. This similarity measure is a Boolean function that, given labels of two sub-tree roots, determines whether these labels can possibly be mapped to each other. Given such a measure, a tree-edit distance method can be smart enough to avoid comparing sub-trees that are impossible to match to each other.

The dataset used in this experiment was synthesized by XMark [124]. XMark is an XML benchmark framework that is capable of generating XML

documents of various sizes based on a size input parameter. The produced XML documents model an auction web site. This benchmark is originally intended for evaluating XML management approaches. In this experiment it is used as an unbiased source of random XML documents. In this experiment XMark was ran 20 times with different size parameters starting from 0.0001 all the way to 0.0028 that produced 28 XML documents of various sizes ranging from 29 KB to 217 KB. Then a random deformer was applied 40 times on each of these documents, which resulted in 40 different versions of each document. The job of a deformer is to randomly change node labels, delete existing nodes, or insert new nodes in a given XML document with total edit probabilities uniformly distributed between various kinds of edit operations. The 40 versions were deformed with various total probabilities ranging from "0.5" to "1.0." Each deformed version is then saved along with a record of edit operations that were randomly applied it.

The experiment is to compare each of the 28 XMark generated documents against each of its 40 deformed versions, and to measure the quality of the result and the time required to finish each comparison job. In this experiment, the task of VTracker is to compare each deformed document against the original version, and to produce the edit script that transforms the original document to the deformed one. The produced set of edit operations is then compared against the recorded ones, and measuring the precision and recall of each comparison; where a precision is ratio between the number of true positive edit operations divided by the number of produced edit operations. Similarly, recall is calculated as the ratio between the number of true positive edit operations divided by the number of recorded edit operations.

VTracker was requested to run in two different configuration setups: a default standard setup, and a domain-aware optimized setup. In the default standard setup, VTracker uses the standard tree-edit distance algorithm explained in Section 3.2, a default cost function, no domain-specific configurations, and is not reference-aware. In the domain-aware optimized setup, VTracker uses the domain-optimized tree-edit distance algorithm explained in Section 3.3.6, a

simple cost function that allows only for perfect match between node labels, no other domain-specific configurations, and is not reference-aware. In the latter setup, VTracker determines two nodes are similar enough to proceed with comparing their sub-trees if the two nodes have the same label; otherwise, it is impossible to map them to each other.

There are two important observations concerning the evaluation results of this experiment. First, both setups produced the exact tree-edit distances for all the 1120 test cases, which implies that this optimization technique did not affect the quality of the result. In other words, the optimization technique did not miss cheaper tree-edit scenarios. Figure 5-1 shows the run times required by both setups across different problem sizes; where a problem size is the multiplication of sizes of the two given trees. Figure 5-2 calculates the percentage of improvement in those various problems. In each of these cases, a percentage of improvement is calculated as the saved time, i.e. optimized runtime minus basic runtime, divided by the basic runtime. The second observation is that this optimization technique consistently improves the runtime performance by 25% in average. Combining this observation with the former one, it is concluded that domain-optimized tree-edit distance technique consistently improves the runtime performance without compromising the quality of calculated edit-distance that supports the hypothesis of Section 3.3.6.

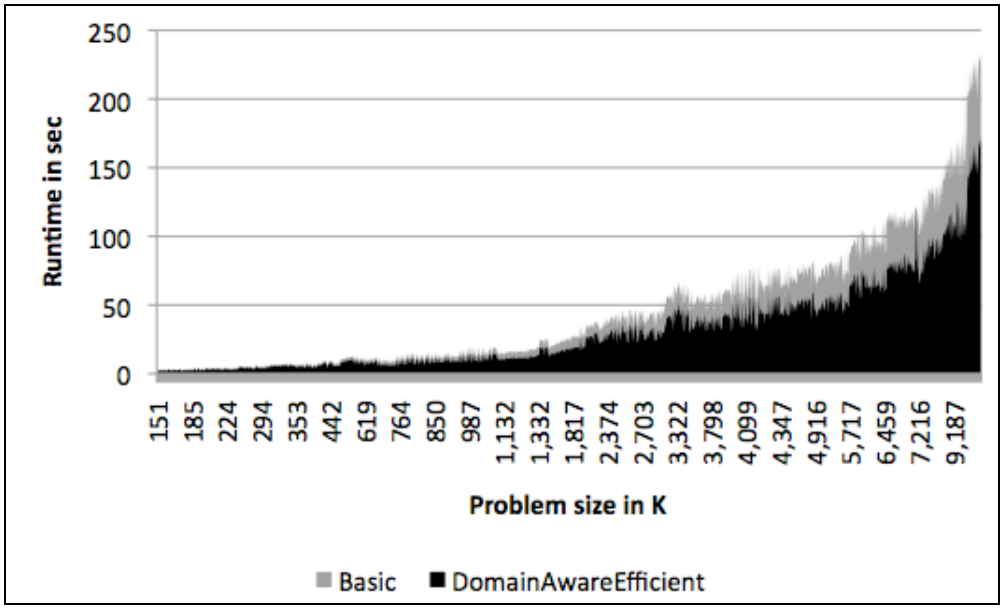



**Figure 5-1: Runtime of basic versus domain-aware optimized tree-edit distance algorithm**

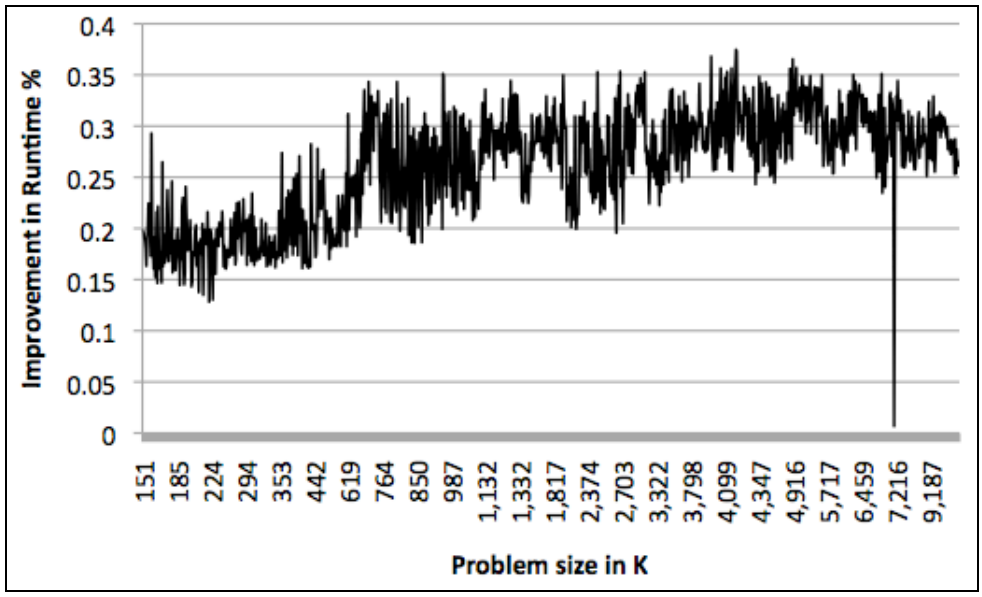



**Figure 5-2: Runtime performance improvement between basic and domain-aware optimized algorithms**

## 5.2 RNA Comparison Experiment

The objective of this experiment is to evaluate the importance of the simplicity heuristics. The set of simplicity heuristics is important to tree-edit distance problems where multiple optimal edit scripts have the same cheapest cost to transform one tree into another.

The dataset of this experiment is based on three "5S ribosomal" families (Szymanski et al.): Archeaa (91 structures), Eubacteria (756 structures), and Eukaryota (526 structures) [133]. "5S ribosomal" RNA is an integral component of the large subunit of all cytoplasmic and most organeller ribosomes. Its small size, and association with ribosomal and non-ribosomal proteins make it an ideal model RNA molecule for studies of RNA structure and RNA-protein interactions. Furthermore, multiple, biologically correct, sequence alignments of 5S ribosomal RNAs are known, where base pairs in phylogenetically conserved secondary structures are specified[5], thus providing target alignments against which computational alignments should be measured. In this way, each of these multiple alignments is decomposed into sets of pair-wise test cases (e.g. 4,186 pairs in Archeaa, 286,524 in Eubacteria, and 138,864 in Eukaryota).

In this experiment, RNA Secondary structures are represented as XML documents. Since there is not standard representation of an RNA tree structure, this experiment evaluated two different representation approaches: a Loose Fine-Grained (LFG) representation, and a modified version of the Tight Fine-Grained (TFG) representation by Mikhaiel and Stroulia [103]. As illustrated in Section 2.7.2, LFG is different from TFG tree structure in the way it represents stem loops. The former represent it as a joint node in addition to two nucleotide (base) nodes while the later represent the entire loop as one single node.

The used "5S ribosomal" dataset is provided in a tab-delimited multiple alignment formats. So, the first task of this experiment is for each structure to transform the tab delimited brackets, dashes, and symbols into an XML format that complies with the LFG and TFG RNA tree representations. Then, each pair of structures is fed to VTracker to compare them, produce the edit script, and to transform the edit script into a tab-delimited format again. Finally, the produced alignment is compared against the published one, and both precision and recall are measured where precision is the number of true positive aligned symbols divided

[5] http://www.man.poznan.pl/5SData/Alignments.html

by the number of produced symbol alignments. Similarly, recall is calculated as the number of true positive aligned elements divided by the number of published ones.

In this experiment, VTracker is configured to use the standard tree-edit distance algorithm, a manually developed domain-specific cost function, no other domain-specific configurations, and is not reference-aware. For each pair-wise matching problem in this experiment VTracker was requested to calculate the tree-edit distance A set of all optimal edit scripts associated with the calculated edit distance is built. Cardinality of the solution set is the number of edit-scripts in that set. The cardinality of each set is recoded before and after applying each of the three-filtration steps. Then, a filtration step is successful if it kept the target solution in the filtered solution set.

Figure 5-3 shows that *simplicity* heuristics are able to efficiently reduce the number of plausible minimum-cost alignments without excluding the biologically good ones for the Archeaa family. This graph shows the results of Archeaa family since it is more challenging than other families. It shows that the number of problems with high-cardinality solution sets was reduced, while the number of problems with low-cardinality solution sets was increased. The last category is especially interesting: for example, there were 1489 problems that produced a set of more than nine solutions – the corresponding number after the heuristics were applied was 242. Table 5-1 shows that, based on 429,574 test cases, the simplicity filtration process is capable of reducing (on average) the solution set size from 10.86 to 2.24 with 90.05% of keeping the best given solution in the filtered set.

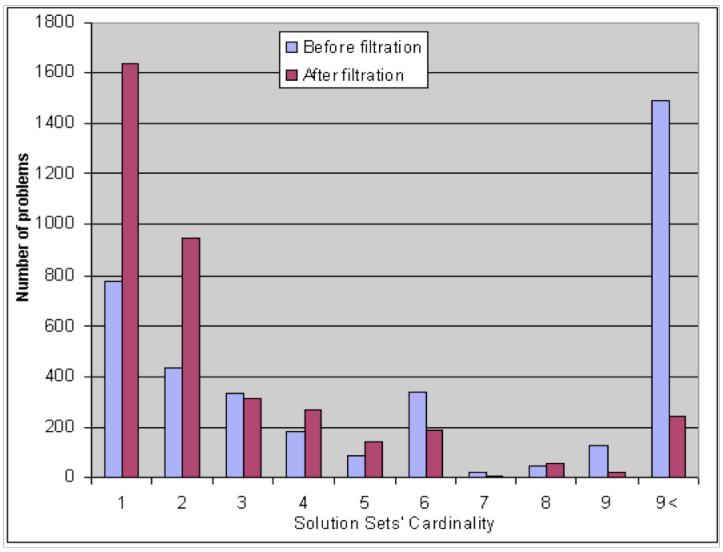



**Figure 5-3: Cardinality reduction for Archeaa family**

**Table 5-1: Evaluation results of simplicity heuristics in RNA Secondary Structure comparison measured by Harmonic Mean**

| Family | # of Problems | Avg Card. | Simplicity Heuristics | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | 1. Shortest Path | | 2. Vertical Simplicity | | 3. Horizontal Simplicity | |
| | | | Avg Card. | Quality | Avg Card. | Quality | Avg Card. | Quality |
| Archeaa | 4186 | 59.28 | 58.62 | 99.55 | 9.69 | 83.97 | 3.25 | 88.52 |
| Eubactria | 286524 | 14.50 | 14.34 | 99.9 | 5.10 | 95.8 | 2.58 | 92.57 |
| Eukaryota | 138864 | 1.88 | 1.87 | 100 | 1.81 | 99.87 | 1.50 | 96.04 |
| **H-mean** | | **10.86** | **10.74** | **99.93** | **4.08** | **97.00** | **2.24** | **93.65** |

To enable the comparison of the quality of the alignments produced by VTracker against these target alignments, the F-Measure that combines both the precision and recall of each comparison problem. The F-Measure of two alternative alignments of two RNA structures is calculated as twice the number of bases in the two compared structures that are edited similarly in each of the two alignments, divided by the sum of the two RNA structure lengths. Based on the above definition, F-Measure is a percentage, and when evaluating a computational alignment against a biologically plausible one, higher F-Measures are more desirable than smaller ones: when F-Measure = 1.00% the two alignments are identical (i.e. the calculated one is identical to the biologically published one).

The second part of this experiment measures the quality of the alignments produced by the two VTracker variants (i.e. VTracker applied to LFG and VTracker applied to TFG), and the two most well known RNA alignment tools at the time of the experiment (year 2007): RNA Align [33][6] and RNA Forester [59][60][61][7]. The RNA Forester tool is based on tree-edit distance approach and adopts an LFG representation – in fact, the team behind the tool is the first to propose this type of LFG representation. Therefore, RNAForester presents a good example of related work as it is uses a similar approach and was built specifically to answer this kind of particular RNA alignment questions. RNAForester is built on the tree alignment algorithm of ordered trees by Jiang et al. [68] and extended it to calculate local forest alignments, which is essential for finding local similar regions in RNA Secondary Structure. On the other hand, the RNA Align tool uses a sequence-based representation; which represents non-tree based approaches. RNA Align is based on an arc-based representation where joints between elements in secondary and tertiary structures are represented as arcs, and the objective is find the cheapest arc edit script that transforms one RNA structure into another. In this experiment, pairs of RNA structures were fed to the four tools to produce four corresponding alignments. Then, the four alignments were compared against the target solutions and their F-Measure metric is calculated. VTracker actually produces a set of possible alignments for each compared pair. For each family, Table 5-2 shows the percentage of cases where F-Measure is 1.00%, i.e., the calculated alignment and the biologically correct one are the same in addition to statistics of other cases, in which the computed alignment was not perfect Table 5-2 shows that both VTracker representations have an outstanding quality compared to those of RNA Forester and RNA Align. VTracker is capable in 26% of the cases of reporting the target solution 7% for RNA Forester and 11.5% for RNA Align.

[6] http://www.csd.uwo.ca/~kzhang/rna/rna_match.html

[7] http://bibiserv.techfak.uni-bielefeld.de/rnaforester/

**Table 5-2: Evaluation of VTracker against related work for RNA Secondary Structure Comparison**

| | Archeaa Family (4,186 test cases) | |
|---|---|---|
| | **Average F-Measure** | **Number of cases where F-Measure = 1.0** |
| **VTracker LFG** | 0.99 | 26.9% |
| **VTracker TFG** | 0.98 | 20.1% |
| **RNA Forester** | 0.97 | 7.0% |
| **RNA Align** | 0.97 | 11.5% |

## 5.3 Ontology Matching Experiment

The objective of this experiment is to evaluate the feasibility of considering the reference model as an essential part of the XML differencing problem. It also evaluates the feasibility of using a synthesized cost function versus using the default one. Finally, it compares the quality of VTracker's result against those state-of-the-art approaches.

VTracker was evaluated against results from the Ontology Alignment Evaluation Initiative's (OAEI-2010 Campaign). The Benchmark test library consists of 48 test cases over three sets. The simplest benchmark (1xx) contains three ontology instances, comparing the reference ontology with itself, with another irrelevant ontology or the same ontology in its restriction to OWL-Lite, The second benchmark (2xx) contains 43 instances obtained by discarding features (like name of entities, comments, specialization hierarchy, instances, properties) from the reference ontology. It aims at evaluating how an algorithm behaves when a particular type of information is lacking. Finally, the third benchmark contains four ontologies of bibliographic references (3xx) found on the web and left mostly untouched. It is important to state that VTracker had difficulties to process cases 206, 207, and 210 due to some XML encoding issues. Therefore, the following evaluation is based on results from the other 45 test cases.

Each of the dataset test cases is provided in the format of OWL/RDF ontology definitions and is described in XML syntax. Each of those ontologies

describes a set of classes, properties, relationships, and instances. The experiment objective is, given two such ontology definitions, to find which classes and properties in the first ontology can be matched to classes and properties of the second ontology, and with how much confidence. In this way, an edit script is required to determine which nodes are mapped to each other. This benchmark involves comparing 45 test cases. In each test case, the task is to compare the given ontology against the reference ontology, and to measure the quality of the produced results by the OAEI benchmark evaluation tool (EvalAlign[8]). EvalAlign then calculates the precision and recall given the reference ontology alignment and the produced one. For each test case, precision and recall were collected from evaluation results. Then the harmonic mean (H-mean) of precisions and recalls is calculated. In order to precisely characterize how useful each of VTracker's features is to its effectiveness we conducted a sequence of experiments, starting by applying the core VTracker algorithm and proceeding to incrementally enable each of the algorithm's features, and then repeating the same experiment. Table 5-3 shows the evaluation results for all the combinations.

Table 5-3 shows that pursuing an affine-cost model improved both precision and recall by between 1% and 4%. It also shows the affine-cost function makes more difference in the case of the default cost model than in the case of the domain-specific cost model. This may be interpreted that the domain-specific cost model inherently includes the semantics of affine-cost policy, and that internal structure nodes are not as important as other nodes. According to the experiment, enabling references improved both precision and recall by 2% to 6% i.e. H-Mean. Table 5-3 also shows that enabling context-awareness improved both precision and recall by 4% to 9% on average. An observation on these results is that context-aware differencing works better in conjunction with reference-aware models and especially with having affine-cost policy on the top. This experiment also evaluated the influence of having a domain-specific cost function,

[8] procalign.jar fr.inrialpes.exmo.align.util.EvalAlign -i fr.inrialpes.exmo.align.impl.eval.PRecEvaluator

synthesized by applying VTracker to the domain XML schema. Additionally, Table 5-3 shows that the best quality of result was produced with the following combination Reference-aware + context-aware + affine-cost policy + synthesized cost function. Finally, it is important to mention that, according to this experiment, none of the presented features negatively influenced the quality of results.

**Table 5-3: Evaluation of various VTracker Contributions**

| | | **Synthesized Cost Function** | | | | **Default Cost Function** | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | **Affine Policy** | | **Non-Affine Policy** | | **Affine Policy** | | **Non-Affine Policy** | |
| | | **Prec.** | **Recall** | **Prec.** | **Recall** | **Prec.** | **Recall** | **Prec.** | **Recall** |
| **Reference-aware** | **Context-aware** | 0.85 | 0.87 | 0.84 | 0.86 | 0.84 | 0.86 | 0.82 | 0.83 |
| | **Non-Context-aware** | 0.77 | 0.79 | 0.75 | 0.77 | 0.77 | 0.78 | 0.73 | 0.74 |
| **Non reference-aware** | **Context-aware** | 0.79 | 0.81 | 0.78 | 0.80 | 0.79 | 0.81 | 0.78 | 0.79 |
| | **Non-Context-aware** | 0.75 | 0.76 | 0.73 | 0.75 | 0.75 | 0.76 | 0.71 | 0.72 |

Table 5-4 shows the evaluation of VTracker against other systems from the 2010 Campaign of Ontology Alignment Evaluation Initiative. In this experiment VTracker was run with the following features enabled: (1) affine-cost policy option, (2) reference-aware model option, (3) context-aware option, and (4) domain-specific synthesized cost function. As shown in Table 5-4, in terms of recall, VTracker is the second top system with only 2% less than the top one (i.e. ASMOV) while in terms of precision, VTracker came in ninth place. The interpretation is that VTracker is the second top one in terms of finding relevant mappings, and the ninth in terms of finding only relevant mappings. The overall performance of VTracker is a harmonic mean of precision of 85% and recall of 87%. The combined F-Measure of precision and recall placed VTracker in the fourth place after ASMOV, RiMOM, and ArgMaker as shown in Figure 5-4. One

more observation about the quality of VTracker is that it is the best one in having balanced precision and recall which means that VTracker is very balanced between finding relevant mapping and discarding non-relevant ones. Finally, it is important to mention that having VTracker in the fourth position is such an achievement for two reasons: (1) the top three tools are especially built to answer this particular kind of ontology matching questions while VTracker is generic and not specially built for that purpose. Even though, VTracker did better than many other domain-specific tools, and (2) the top three tools are equipped with some kinds of lexical matching mechanism, which is not available in VTracker in its current version. We strongly believe that adding a lexical matching mechanism like WordNet to VTracker will improve the quality of produced results.

We have to mention that we did not participate in this benchmark contest. However, we used results of the contest published in [50]. For external validity, we used EvalAlign tool that measures the precision and recall for each of the ontology matching cases.

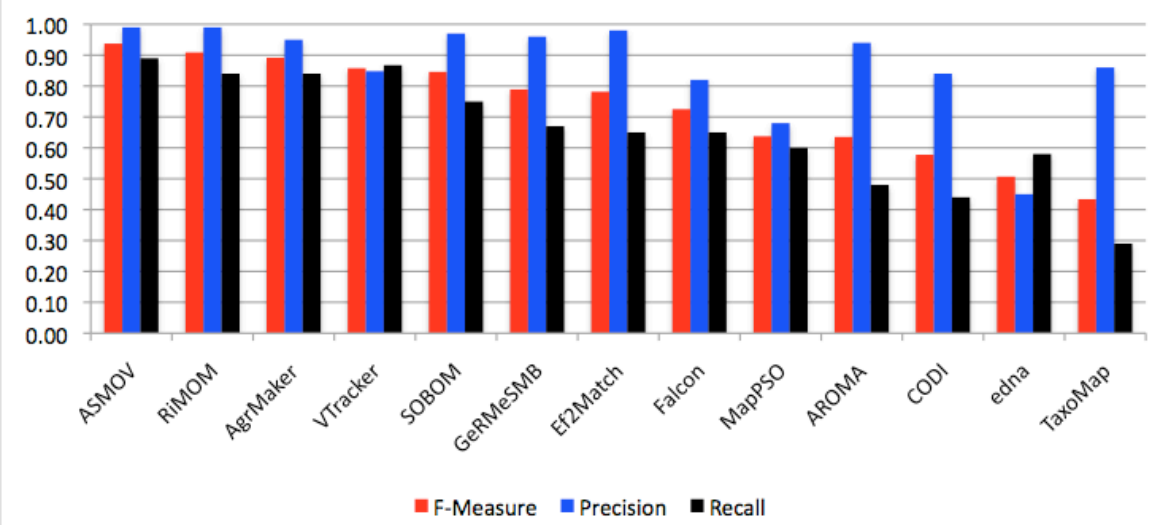



**Figure 5-4: Evaluation of VTracker's performance against benchmark results displaying H-Mean of precision, recall, and F-Measure sorted by F- Measure value**

**Table 5-4: Evaluation of VTracker against results from OAEI 2010**

| | | 1xx | 2xx | 3xx | H-mean |
|---|---|---|---|---|---|
| **refalign** | **Precision** | 1.00 | 1.00 | 1.00 | 1.00 |
| | **Recall** | 1.00 | 1.00 | 1.00 | 1.00 |
| **VTracker** | **Precision** | 1.00 | 0.87 | 0.48 | 0.85 |
| | **Recall** | 1.00 | 0.88 | 0.56 | 0.87 |
| **edna** | **Precision** | 1.00 | 0.43 | 0.51 | 0.45 |
| | **Recall** | 1.00 | 0.57 | 0.65 | 0.58 |
| **ArgMaker** | **Precision** | 0.98 | 0.95 | 0.88 | 0.95 |
| | **Recall** | 1.00 | 0.84 | 0.58 | 0.84 |
| **AROMA** | **Precision** | 1.00 | 0.94 | 0.83 | 0.94 |
| | **Recall** | 0.98 | 0.46 | 0.58 | 0.48 |
| **ASMOV** | **Precision** | 1.00 | 0.99 | 0.88 | 0.99 |
| | **Recall** | 1.00 | 0.89 | 0.84 | 0.89 |
| **CODI** | **Precision** | 1.00 | 0.83 | 0.95 | 0.84 |
| | **Recall** | 0.99 | 0.42 | 0.45 | 0.44 |
| **Ef2Match** | **Precision** | 1.00 | 0.98 | 0.92 | 0.98 |
| | **Recall** | 1.00 | 0.63 | 0.75 | 0.65 |
| **Falcon** | **Precision** | 1.00 | 0.81 | 0.89 | 0.82 |
| | **Recall** | 1.00 | 0.63 | 0.76 | 0.65 |
| **GeRMeSMB** | **Precision** | 1.00 | 0.96 | 0.9 | 0.96 |
| | **Recall** | 1.00 | 0.66 | 0.42 | 0.67 |
| **MapPSO** | **Precision** | 1.00 | 0.67 | 0.72 | 0.68 |
| | **Recall** | 1.00 | 0.59 | 0.39 | 0.6 |
| **RiMOM** | **Precision** | 1.00 | 0.99 | 0.94 | 0.99 |
| | **Recall** | 1.00 | 0.83 | 0.76 | 0.84 |
| **SOBOM** | **Precision** | 1.00 | 0.97 | 0.79 | 0.97 |
| | **Recall** | 1.00 | 0.74 | 0.75 | 0.75 |
| **TaxoMap** | **Precision** | 1.00 | 0.86 | 0.71 | 0.86 |
| | **Recall** | 0.34 | 0.29 | 0.32 | 0.29 |

## 5.4 UML Differencing Experiment

The objective of this experiment is to evaluate the performance of VTracker in the domain of Object-oriented model comparison. In this experiment an object-oriented model was divided into three kinds of design models: (1) a *containment* model that includes relationships between a class and its operations and attributes; (2) an *inheritance* model that includes relationships such as subclass and realization relationships; and (3) a *usage* model that includes dependency and association relationships such data types and calls of attributes and operations.

The dataset of this experiment is 13 successive versions of JFreeChart[9] starting from version 1.0.0 to 1.0.13. The ground truth of model evolutions were independently developed by two the authors: Tsantalis, N., Negara [136] with help of Eclipse IDE. For the containment model, the ground-truth included edit operations to add, remove, or rename an operation, and to add, remove, or rename an attribute. The inheritance model included operations to add, remove, or rename a generalization, and to add or remove a realization. Finally, a usage model included operations to add, remove, or replace an operation call, and to add, remove, or replace an attribute access. In this experiment, the task of VTracker is, given two UML modules, to try find proper matching between elements of the two modules and report different kinds of edit operations explained above.

In this experiment, source-codes of Java classes were parsed and represented as XML documents. The XML representation used in this experiment is a simplified version of UML/XMI described in Section 2.5.2. The standard UML/XMI representation combines the three models into one coherent UML model. However, for the purpose of this experiment, the three models were represented separately using a simplified UML/XMI representation. One big difference is that the standard UML/XMI representation largely depends on XML reference-structure to share element definitions between various models while the simplified representation does not have such a reference-model as it is only based on XML containment structure. Code 5-1 shows an example of the simplified XML representation that describes the containment model of class "org.jfree.chart.block.BlockContainer". Accordingly, VTracker is configured to use the domain-optimized tree-edit distance algorithm, a standard cost function, some domain-specific configurations, and is not reference-aware. This experiment uses three kinds of domain-specific configurations. First, it specified attributes *className*, *operationName*, and *paramName* as *keyAttributes*. As explained in Section 4.3, key attributes give VTracker hint on the relative importance of

[9] http://sourceforge.net/projects/jfreechart/

different attributes. In this experiment, these attributes are relatively more important than other attributes as they logically identify a model element. Secondly, this experiment, configure attributes named *ID* as meta-attribute that should be ignored during the differencing process since it is just used to find correspondence between edit operations and original model elements. In the third, since this experiment is only interested in edit operations happening to parameters, and classes, the configurations of this experiment specified them as key elements. As explained in Section 4.3, configuring key elements do not influence the differencing process but it determines the desired outcome of the differencing process. Generally, VTracker reports the tree-edit distance and the edit script associated with this distance. Having this kind of configuration, VTracker is instructed to also report tree-edit distance matrix between all key elements. The produced matrix contains distances between all sub-trees rooted by key elements. Finally this experiments applies the stable marriage algorithm in order to find the optimal mapping between various elements. In this way, VTracker overcomes the limitation of ordered trees and allow mapping between nodes that are not in the same order. In other words, VTracker uses the tree-edit distance algorithm to measure the distance between various key sub-trees, and then it uses these distances to find the best mapping solution.

In this experiment VTracker was compared against UMLDiff [161]. UMLDiff is one of the state-of-the-art methods in comparing UML models. UMLDiff is based on purpose built heuristics, and matching techniques, to serve UML differencing in particular. Therefore, the experiment evaluates the performance of VTracker as generic method against a domain-specific method like UMLDiff. This evaluation of this experiment was independently performed by Tsantalis and Nigara. in [136]. Table 5-5, borrowed from [136], shows that VTracker has similar precision and recall to UMLDiff in matching elements of the containment model while VTracker does much better than UMLDiff when it comes to matching elements of the inheritance or the usage models. VTracker deals with all models on the same basis. It does not favor one over the other. That

is why VTracker did similarly in all models unlike UMLDiff that may be strong in matching some kinds of relationships but not the others. One more observation is that VTracker has consistent values of precision and recall, which implies that VTracker is very confident in the produced result.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<VirtualRoot name="VirtualRoot">
  <Class ID="63493" className="org.jfree.chart.block.BlockContainer"
    isAbstract="false" isInterface="false">
    <Operation ID="63512" operationName="add" param1="void"
      param2="org.jfree.chart.block.Block" param3="java.lang.Object"
      visibility="public">
      <Parameter paramKind="return" paramtType="void" />
      <Parameter paramKind="in" paramName="block"
        paramtType="org.jfree.chart.block.Block" />
      <Parameter paramKind="in" paramName="key" paramtType="java.lang.Object" />
      <MethodCall ID="10320" arg1="java.lang.Object"
        methodCallName="java.util.List.add" originClassName="java.util.List">
        <Argument argKind="in" argType="java.lang.Object" />
        <Argument argKind="return" argType="boolean" />
      </MethodCall>
      <MethodCall ID="10322" arg1="org.jfree.chart.block.Block"
        arg2="java.lang.Object"
        methodCallName="org.jfree.chart.block.Arrangement.add"
        originClassName="org.jfree.chart.block.Arrangement">
        <Argument argKind="in" argType="org.jfree.chart.block.Block" />
        <Argument argKind="in" argType="java.lang.Object" />
        <Argument argKind="return" argType="void" />
      </MethodCall>
      <FieldAccess ClassType="org.jfree.chart.block.Arrangement"
        ID="10321" OwnerClassName="org.jfree.chart.block.BlockContainer"
        fieldName="org.jfree.chart.block.BlockContainer.arrangement" />
      <FieldAccess ClassType="java.util.List" ID="10319"
        OwnerClassName="org.jfree.chart.block.BlockContainer"
        fieldName="org.jfree.chart.block.BlockContainer.blocks" />
    </Operation>
  </Class>
</VirtualRoot>
```

**Code 5-1: An example of a simplified XML representation of a containment model specification**

**Table 5-5: Evaluation of VTracker against UMLDiff**

| | VTracker | | UMLDiff | |
|---|---|---|---|---|
| | **Precision** | **Recall** | **Precision** | **Recall** |
| Containment | 0.99 | 0.98 | 0.97 | 0.97 |
| Inheritance | 1.00 | 1.00 | 0.88 | 0.88 |
| Usage | 0.95 | 0.94 | 0.91 | 0.84 |

## 5.5 Service Discovery Experiment

This experiment aims at evaluating VTracker in terms of two aspects. First, it evaluates the feasibility of considering the XML reference model in the domain of WSDL matching. Second, it generally evaluates the performance of VTracker in that domain.

The dataset of this experiment is based on SAWSDL-TC 3 that includes a set of 1080 WSDL services from seven different domains: education, medical care, food, travel, communication, economy, weaponry, geography, and simulation. The collection also includes a set of 42 WSDL queries from these domains. For each query-service combination the SAWSDL-TC 3 collection includes a matching grade to indicate the relevance of matching this service to that particular query. This grade is based on a 4-graded scale:

- Highly relevant: Any service that offers exactly what the user asked for (or even better).
- Relevant: Any service that might answer the request completely or partially does the requested job.
- Potentially relevant: any service that may be helpful
- Non-relevant: any service that is totally irrelevant to the query request.

For each query, VTracker is requested to calculate the edit-distance between the WSDL of the query and the 1080 WSDLs of the offered services. Then, calculated distances of each query are sorted in ascending order according to the calculated distance. Then, a matching relevance is determined by the order of the solution. Consider an example where SAWSDL-TC has a set of three highly relevant offers, then if any of the top three distances in the calculated list belong to this set, this distance is considered a true-positive result to that grade. This experiment evaluates results of 21 WSDL queries against the 1080 offered service WSDLs from SAWSDL-TC. For each query, Table 5-6 shows the four different grades along with the number of offered services in each grade. This table shows that in 90% of the queries, VTracker is able to recommend the highly

relevant offers at the top of the matching list. Secondly, VTracker is successful by 95% in ranking irrelevant offers at the bottom of the list. VTracker was also capable in 39% to rank relevant offers in proper position in the matching list. It is also capable in 37.5% to rank partially relevant offers in their proper positions. This table also shows that the using references improved the quality of ranking partially relevant offers from 34% to 37.5% of the cases. It is important to mention that VTracker did better in finding highly recommended and irrelevant offers than in finding relevant and partially relevant ones. In other words, VTracker was successful in finding perfect matches and absolutely different offers while it did not do that good when it comes to the gray area in between. This can be explained by the fact that VTracker does not include a lexical matching mechanism; it is only based on structural, content, context, and reference matching mechanisms. In the case of perfect match or an absolute different offers the VTracker mechanism are good enough to measure the similarity. However, when in comes to the gray area in the middle, a lexical matching is essential assess the relevancy between a request and an offer.

Figure 5-5 shows percentage of extra time required by the reference-aware algorithm compared to the time required by the basic one. This figure shows that the reference-aware tree-edit distance algorithm consistently requires more time than that is required by the basic algorithm. The question is "Can the required extra time be justified?". Indeed, yes it can. For example in the query named "governmentdegree_scholarship_service.wsdl", reference-aware requires almost double the time required by the basic approach while in that particular case, the quality of the partially-relevant grade was also doubled, i.e. improved from 55% to 100%. Similarly, in case of query named "getLocationOfCityState.wsdl", reference-aware approach required about 140% more than basic one while it improved the quality of the same grade from 23% to 31%. Therefore, we can conclude that while the reference-aware approach requires more time, it proportionally improves the quality of the results.

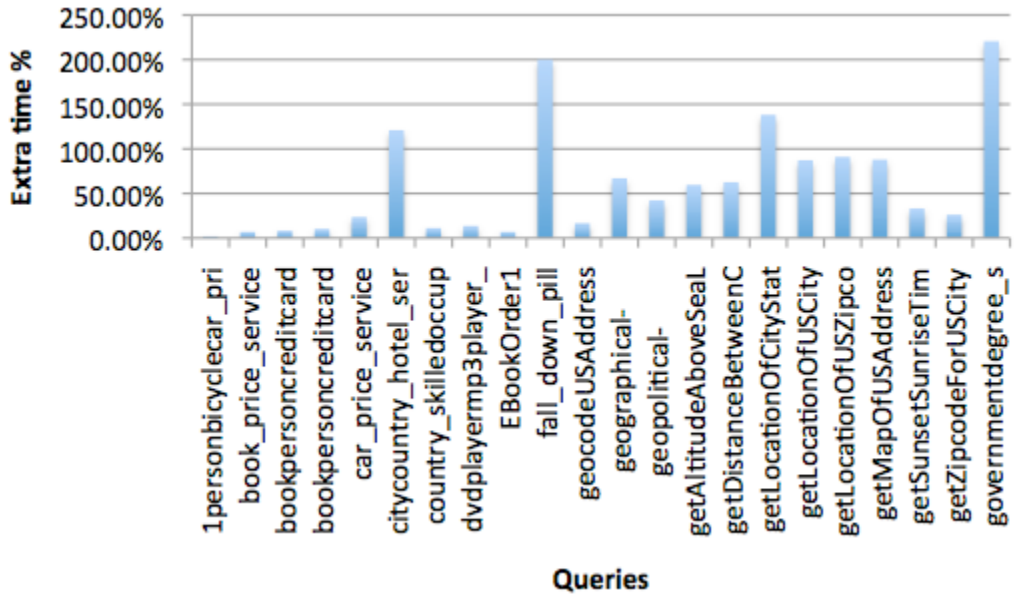



**Figure 5-5: Runtime of Basic versus Reference-aware algorithms in regards to SAWSDL-TC experiment**

**Table 5-6: Evaluation of VTracker in SAWSDL-TC Collection**

| | Highly Relevant | | | Relevant | | | Partially Relevant | | | Irrelevant | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | SAW SDL-TC | VTracker | | SAW SDL-TC | VTracker | | SAW SDL-TC | VTracker | | SAW SDL-TC | VTracker | |
| | | Ref | Basic | | Ref | Basic | | Ref | Basic | | Ref | Basic |
| 1personbicyclecar_price_service.wsdl | 11 | 45% | 55% | 20 | 50% | 45% | 61 | 77% | 75% | 988 | 89% | 89% |
| book_price_service.wsdl | 12 | 50% | 50% | 22 | 59% | 64% | 45 | 11% | 56% | 1001 | 88% | 90% |
| bookpersoncreditcardaccount__service.wsdl | 5 | 20% | 20% | 9 | 78% | 78% | 12 | 100% | 14% | 1054 | 95% | 92% |
| bookpersoncreditcardaccount_price_service.wsdl | 2 | 50% | 50% | 37 | 62% | 65% | 56 | 66% | 66% | 985 | 88% | 88% |
| car_price_service.wsdl | 14 | 29% | 29% | 35 | 54% | 54% | 44 | 9% | 84% | 987 | 85% | 88% |
| citycountry_hotel_service.wsdl | 8 | 25% | 25% | 8 | 25% | 25% | 23 | 17% | 17% | 1041 | 94% | 94% |
| country_skilledoccupation_service.wsdl | 21 | 24% | 24% | 46 | 46% | 46% | 22 | 23% | 18% | 991 | 86% | 86% |
| dvdplayermp3player_price_service.wsdl | 5 | 60% | 60% | 10 | 20% | 40% | 12 | 0% | 33% | 1053 | 95% | 96% |
| EBookOrder1.wsdl | 3 | 67% | 67% | 0 | 0% | 0% | 12 | 0% | 0% | 1065 | 97% | 97% |
| fall_down_pill.wsdl | 1 | 0% | 0% | 1 | 100% | 100% | 0 | 0% | 0% | 1078 | 100% | 100% |
| geocodeUSAddress.wsdl | 11 | 9% | 9% | 9 | 78% | 56% | 3 | 100% | 33% | 1057 | 96% | 96% |
| geographical-regiongeographical-region_map_service.wsdl | 4 | 50% | 50% | 2 | 100% | 100% | 12 | 33% | 33% | 1062 | 97% | 97% |
| geopolitical-entity_weatherprocess_service.wsdl | 3 | 67% | 67% | 30 | 67% | 63% | 4 | 0% | 100% | 1043 | 95% | 95% |
| getAltitudeAboveSeaLevelOfLocation.wsdl | 3 | 33% | 33% | 0 | 0% | 0% | 0 | 0% | 0% | 1077 | 100% | 100% |
| getDistanceBetweenCitiesWorldwide.wsdl | 1 | 0% | 0% | 2 | 50% | 50% | 17 | 35% | 6% | 1060 | 97% | 96% |
| getLocationOfCityState.wsdl | 1 | 0% | 0% | 4 | 25% | 25% | 13 | 31% | 23% | 1062 | 97% | 97% |
| getLocationOfUSCity.wsdl | 4 | 25% | 25% | 12 | 17% | 17% | 5 | 100% | 67% | 1059 | 97% | 97% |
| getLocationOfUSZipcode.wsdl | 7 | 14% | 14% | 4 | 0% | 0% | 9 | 22% | 56% | 1060 | 97% | 97% |
| getMapOfUSAddress.wsdl | 4 | 25% | 25% | 1 | 0% | 0% | 11 | 0% | 18% | 1064 | 97% | 97% |
| getSunsetSunriseTimeOfLocation.wsdl | 3 | 33% | 33% | 1 | 0% | 0% | 0 | 0% | 0% | 1076 | 99% | 99% |
| getZipcodeForUSCity.wsdl | 5 | 20% | 20% | 2 | 0% | 0% | 3 | 100% | 0% | 1070 | 99% | 98% |
| governmentdegree_scholarship_service.wsdl | 8 | 63% | 63% | 26 | 27% | 27% | 18 | 100% | 55% | 1028 | 93% | 92% |
| **Average** | | **32%** | **33%** | | **39%** | **39%** | | **38%** | **34%** | | **95%** | **95%** |

To conclude, this chapter explains the evaluation process of VTracker in different domains and in different setup and configurations. The first experiment, evaluates the general aspects of VTracker such as performance and quality of using the domain-optimized tree-edit distance algorithm that avoids matching infeasible sub-trees. It was shown that the optimized technique saves on average about 25% of the processing time while producing the same quality of results. The second experiment evaluates the matching quality of VTracker in the context of RNA Secondary Structure Comparison. It was shown that VTracker has an F-Measure value of 0.99, which exceeds the performance of the state-of-the-art methods at the experiment time, i.e. 2007. In this experiment VTracker using the LFG tree-representation performed better than RNA Forester that uses the same tree representation where VTracker was able to find the target solution in 26% of the cases while RNA Forester found the target solution in only 7% of the test cases. Additionally this experiment illustrated the importance of simplicity heuristics in finding the best optimal solution from within a set of optimal ones. Thirdly, the OAEI benchmark experiment evaluated the performance of VTracker in the context of Ontology Matching where VTracker comes in the forth-top place within systems those are especially built to serve this application domain. This experiment also illustrated the importance of domain-specific cost function and the XML reference-structure in improving the quality of the produced results. In the forth is the UML differencing experiment that was independently conducted by a third party to evaluate the performance of VTracker in the context of object-oriented model differencing against the-state-of-the-art in the domain which is UMLDiff. In this experiment VTracker superiorly competed with UMLDiff. Finally, VTracker is evaluated in the context of SAWSDL-TC bench mach of matching WASL queries against offered WSDL services where VTracker was able to find at least one of the best offers in 90% of the cases. This experiment also evaluates the influence of using the reference-aware algorithm in the quality of the produced results.

Finally, it is also worth to mention that VTracker has been applied to XHTML comparison through installing as a differencing component of Annoki, an open source of wiki.

# Chapter Six Discussion, Conclusion, and Future Work

This thesis, motivated by the importance of XML, a universal format for structured documents and data on the Web, focuses on the general problem of XML differencing. Instances of this general problem appear in various domains such as document management, service discovery and matching, system integration, semantic-web interoperability, and many other domains. In each of these domains special methods have been developed to solve the particular instance of the differencing problem for the domain in question. To mitigate the problem of effort duplication, this thesis presents VTracker, a generic differencing method that is capable of being domain-aware through a domain-specific cost function. VTracker views an XML document as an ordered labeled tree on which it can apply Zhang-Shasha's tree-edit distance algorithm.

This thesis makes two important contributions: first, an extension to of the original Zhang-Shasha algorithm with an XML reference structure (i.e. hyperlinks) on the top of the natural XML containment structure, and second, a domain-specific cost function that is capable of capturing domain knowledge and semantics. It has been illustrated by examples in Section 3.1 that the reference-structure plays a critical role in determining the semantics of a given XML document. In addition both the OAEI and SAWSDL-TC experiments shows how considering the reference-structure during the tree-edit distance calculation improves the quality of produced results. Similarly, the usage-context similarity measure is important to work in conjunction with the reference-aware algorithm to resolve matching ambiguities such as the example of Figure 3-4. VTracker also extends Zhang-Shasha with an affine-cost policy that prevents structural formality from having a negative influence on the quality of results, which was illustrated in the example of Figure 3-10 and another example in Section 4.1in the context of ontology matching. Additionally, VTracker is equipped with the mechanism, called simplicity heuristics, to handle situations where a calculated edit distance has multiple edit scripts that are all capable of transforming the first tree into the second tree. The set of simplicity heuristics is important in

applications that involve considerable amount of changes in the internal structure of the given trees. It has shown special significance in the context of RNA Secondary Structure Comparison. VTracker is also equipped with a domain-specific cost function mechanism where VTracker is used to match elements of the domain schema against each other calculating the edit distance between each two elements. The calculated distance matrix is then used as the domain-specific cost function when matching instances of this schema. This synthesizing mechanism assumes that the given XML schema is rich with domain semantics and knowledge. This assumption was proven true in cases like BPEL and OWL/RDF XML schemas. However, this assumption is not always true especially in cases where simple schema is provided such as WSDL and XMI.

Chapter Two explains how the reference structure is a critical component in the semantics of many applications of XML such as OWL/RDF, WSDL, BPEL, and UML/XMI. Both XHTML and RNA do not utilize this particular feature of XML since their semantics do not require this kind of association and dependency relationships. Chapter Three discusses the importance of another aspect of the XML reference structure and the concept of usage-context as a secondary measure of similarity between XML elements. This chapter also emphasizes the importance of using an affine-cost policy for giving a fair chance between deletions and changes. The objective of affine-cost policy is defined so as to promote edit scripts that group edit operations in neighbors. Similarly, the heuristic-based approach chooses the most optimal edit scripts in case of multiple ones. Chapter Four explores the applicability of VTracker in various problem domains. In Chapter Five, the heuristic-based approach was proven true in domains, such as RNA secondary structure, where one edit-distance may have multiple edit scripts. Also, VTracker proposes a method for bootstrapping the algorithm in a domain by automatically synthesizing a domain-aware cost function based on the underlying XML Schema Definition (XSD). The feasibility of the synthetic cost model was illustrated in Experiment Number Three where

employing the synthetic cost function improved the quality of the matching results.

Various aspects of VTracker were evaluated through a set of five experiments. The first experiment evaluated the general performance of VTracker along with the influence of the proposed optimization. Secondly, the feasibility of the simplicity heuristic set was illustrated in the context of RNA Secondary Structure Comparison. Thirdly, the importance of the reference model and the synthetic cost function were verified in the context of an OAEI Ontology Matching benchmark experiment. In this experiment, VTracker was also evaluated against state-of-the-art approaches in Ontology Matching domain, and VTracker competed successfully with the top tier systems. In the fourth, VTracker was evaluated against the state-of-the-art approach in object-oriented differencing, and VTracker showed an outstanding performance against UMLDiff. Finally, VTracker was evaluated in the domain of WSDL service matching where it also performed adequately.

## 6.1 Conclusion

The aim of this work is to demonstrate that differencing problems in various domains are similar in their essence, and can be solved through a generalized approach that takes into consideration a domain's specialties. The objective is to show that all domain specific differencing methods are simply trying to accomplish the same thing. It would be more beneficial for these domain-specific solutions to start from a generic method like VTracker, and focus more on the real problem of differencing semantics, which is captured through the cost function. VTracker was evaluated against state-of-the-art systems in each of those domain's differencing techniques. These evaluations should be considered positive if VTracker performs comparably to, or exceeds, methods especially built to serve those domains. As shown in the evaluation chapter VTracker competed very well with the more well known differencing methods in these domains.

Finally, it is important to mention that VTracker takes the differencing problem into a higher level of flexibility. A user can control the level of details on which VTracker works. For example in the case of UML model differencing, the experiment designer was allowed to decide which aspects of the model to compare: inheritance model, containment model, or usage model. The experiment designer had to provide an XML structure that captures only the desired aspects. The same flexibility applies in case of BPEL matching. If WSDL definitions are included in the BPEL specification XML document, then VTracker considers the BPEL workflow along with the underlying WSDL definitions to reach more accurate matching quality. In this way, VTracker allows a user to specify the desired level of details and aspects.

## 6.2 Future Work

There are many directions of future work in VTracker. Firstly, we plan to work towards improving the quality of the cost model in general and domain-specific cost functions in particular. On one hand, a big contribution would be to equip VTracker with a lexical matching mechanism such as WordNet that is capable of matching synonym terms and vocabularies. On the other hand, it is clear that XML Schema Definitions are undeniably rich with domain semantics and knowledge so another dimension of improving the cost model is to dig deeper into various domain XML schemas to uncover more implicit semantics, and to improve the current process of synthesizing domain-specific cost functions. A second future task is to automate the process of domain-specific configuration, using a domain schema to recognize ID/IDREF, key, and metadata attributes. Further experimentation is also necessary to validate the applicability of VTracker in other domains and to evaluate its effectiveness against benchmarks of these domains. Finally, we are interested in reusing and applying the innovations of VTracker in the context of other tree-edit distance differencing mechanisms.

# Bibliography


[1] Aalst, W.M.P. "The Application of Petri Nets to Workflow Management". The Journal of Circuits, Systems and Computers, Vol. 8, No. 1. (1998), pp. 21-66.

[2] Aalst, W.M.P., "Why workflow is NOT just a Pi-process". BPTrends, February, 2004.

[3] Agarwal, A., and Ankolekar, A., "Automatic Matchmaking of Web Services". In Proceedings of International Conference on Web Services, ICWS '06, pp. 45-54, 2006. .

[4] Allali, J., and Sagot, M., "A New Distance for High Level RNA Secondary Structure Comparison". IEEE/ACM Transactions. Comput. Biol. Bioinformatics , Vol. 2, No. 1 (2005), pp. 3-14.

[5] Allali, J., and Sagot, M., "Novel Tree Edit Operations for RNA Secondary Structure Comparison". In Proceedings of Proceedings of the 4th Workshop on Algorithms in BioInformatics, WABI '04, pp. 412-425, 2004.

[6] Altinel, M. and Franklin, M., "Efficient filtering of XML documents for selective dissemination of information". In Proceedings of the 26th International Conference on Very Large Data Bases (VLDB '00), pp. 53–64, 2000.

[7] Altschul, S., Gish, W., Miller, W., Myers, E., and Lipman, D., "Basic local alignment search tool". Journal of Molecular Biology, Vol. 215, No. 3 (1990), pp. 403-410.

[8] Altschul, S., Madden, T., Schaffer, A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D., "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs". Journal of Nucleic Acids Research, Vol. 25, No. 17 (1997), pp. 3389-3402.

[9] Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., and Weerawarana, S., "Business Process Execution Language for Web Services". Version 1.1, 2003, http://www-128.ibm.com/developerworks/library/specification/ws-bpel/

[10] Ankolekar, A. "DAML-S: Web Service Description for the Semantic Web". In Proceedings of International Semantic Web Conference, ISWC'02, pp. 348-363, 2002.

[11] Anthony, M. "A Study of Strategic Change, Process Alignment, and Notation: FNGC Tap Process." White paper from International Performance Group, Ltd. http://www.bptrends.com/publicationfiles/Anthony%20Study%20%2D%20FNGC%20Case%201%2DTEXT%20%2D%203%2D4%2D03%2Epdf (last accessed July 30, 2010).

[12] Antoniou, A., and Harmelen, F.V., "Web Ontology Language: OWL". Book titled: Handbook on Ontologies in Information Systems, publisher: Springer-Verlag, 2003.

[13] Apiwattanapong, T., Orso, A., and Harrold, M. J., "JDiff: A differencing technique and tool for object-oriented programs". Journal of Automated Software Engineering (ASE)., Vol. 14, No. 1(2007), pp. 3–36.

[14] Augsten, N., Barbosa, D., Bohlen, M., and Palpanas, T., "TASM: Top-k Approximate Subtree Matching", In Proceedings of 26th International Conference on Data Engineering (ICDE'10), pp. 353 – 364, 2010.

[15] Bae, J., Bae, H., Kang, S.H., and Kim, Y., "Automatic Control of Workflow Processes Using ECA Rules". IEEE transactions on knowledge and data engineering, Vol. 16, No. 8 (2004), pp.1010-1023.

[16] Bafna, V., Muthukrishnan, S., and Ravi, R., "Computing similarity between RNA strings". In Proceedings of Combinatorial Pattern Matching Conference '95,, pp. 1-16., 1995.

[17] Batra, S. and Bawa, S., "Web Service Categorization Using Normalized Similarity Score, " International Journal of Computer Theory and Engineering, Vol. 2, No. 1 (2010), pp. 139-142.

[18] Bernard, M., Boyer, L., Habrard, A., Sebban, M., "Learning probabilistic models of tree edit distance". Journal of Pattern Recognition, Vol. 41, No. 8 (2008), pp. 2611-2629.

[19] Berners-Lee, T. and Connolly, D. "Hypertext Markup Language - 2.0", RFC 1866. Proposed Standard, November 1995.

[20] Berners-Lee, T., Brickley, D., Miller, E., and Swick, R.R, "Frequently Asked Questions about RDF". W3C, http://www.w3.org/RDF/FAQ, 2004.

[21] Berners-Lee, T., Fielding, R., and Masinter, L., "Uniform Resource Identifier (URI): Generic Syntax". RFC 3986, January 2005, http://www.rfc-editor.org/rfc/rfc3986.txt

[22] Blanchet, W., Elio, R., Stroulia, E., "Conversation Errors in Web Service Coordination: Run-time Detection and Repair". In Proceedings of International Conference on Web Intelligence (WI'5), pp. 442 – 449, 2005.

[23] Blanchet, W., Elio, R., Stroulia, E. "Supporting Adaptive Web-Service Orchestration with an Agent Conversation Framework". In Proceedings of the third IEEE International Conference on Web Services (ICWS'5), pp.541-549, 2005.

[24] Brauer, M., Weir, R., and McRae, M., "OpenDocument. v1.1 specification", 2007, http://docs.oasis-open.org/office/v1.1/OS/OpenDocument-v1.1.pdf

[25] Breuel, T.M. "Information Extraction from HTML Documents by Structural Matching". In Proceeding of Second International Workshop on Web Document Analysis (WDA'03), ,pp.11-14, 2003.

[26] Brockmans, S., Ehrig, M., Koschmider, A., Oberweis, A., and Studer, R., "Semantic Alignment of Business Processes". In Proceedings of Eighth International Conference on Enterprise Information Systems, (ICEIS'6), pp.191-196, 2006.

[27] Chawathe, S., Rajaraman, A., Garcia-Molina, H. and Widom,J., "Change Detection in Hierarchically Structured Information". In Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'96), Vol. 25, No. 2(1996), pp. 493–504.

[28] Chawathe, S. and Garcia-Molina, H., "Meaningful Change Detection in Structured Data". In Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'97), Vol. 26, No. 2(1997), pp. 26–37.

[29] Chevalier, F., Auber, D., and Telea, A., "Structural analysis and visualization of c++ code evolution using syntax trees". In Proceedings of the ninth International Workshop on Principles of Software Evolution (IWPSE'7): in conjunction with the 6th ESEC/FSE joint meeting, pp. 90–97, 2007.

[30] Choi, N., Song, I.-Y., and Han, H., "A survey on ontology mapping". SIGMOD Record, Vol. 35, No. 3(2006), pp. 34–41.

[31] Cobéna, G., Abdessalem, T., Hinnach, Y., "A comparative study for XML change detection", Gemo Report number 221, 2002. http://www-rocqbis.inria.fr/verso/Gemo/PUBLI/all- byyear.php, ftp://ftp.inria.fr/INRIA/Projects/gemo/gemo/Gemo Report-221.pdf, accessed May 2005.

[32] Cobena, G., Abiteboul, S., Marian, A., "Detecting Changes in XML Documents". In Proceedings of the 18th International Conference on Data Engineering (ICDE'02), pp. 41-52, 2002.

[33] Collins, G., Le, S., and Zhang, K., "A new algorithm for computing similarity between RNA structures". Journal Information Sciences: an International Journal, Vol. 139. No. 1-2,(2001), pp. 59-77.

[34] Contributors: IBM, BEA Systems, Microsoft, SAP AG Siebel System, "Business Process Execution Language for Web Services version 1.1, " May, 2003.http://www.ibm.com/developerworks/library/specification/ws-bpel/ (last accessed July 30, 2010).

[35] Contributers: IBM, Systinet, UnitSpace, Microsoft, LMI, SAP AG, Computer Associates, SeeBeyond Technology, and Oracle, "UDDI Version 3.0.2 - UDDI Spec Technical Committee Draft, Dated 20041019" OASIS Standard, http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm (last accessed July 10, 2011).

[36] Contributers: Adobe Systems, BEA, Deloitte, IBM, Individual, Intalio, Jboss Inc.. Microsoft, Oracle, SAP, Sterling Commerce, TIBCO Software, webMethods, "Web Services Business Process Execution Language Version 2.0, " OASIS Standard, 11 April 2007, http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html (last accessed July 30, 2010).

[37] Contributers: Adobe Systems, BEA, Deloitte, IBM, Individual, Intalio, Jboss Inc.. Microsoft, Oracle, SAP, Sterling Commerce, TIBCO Software, webMethods, "Schema for OASIS Business Process Execution Language (WS-BPEL) 2.0 - Abstract BPEL Common Base" OASIS Standard, http://docs.oasis-open.org/wsbpel/2.0/OS/process/abstract/ws-bpel_abstract_common_base.xsd (last accessed July 30, 2010).

[38] OASIS ebXML RegRep Version 4.0 Part 0: Overview Document. 12 May 2011. OASIS Committee Specification Draft 02 / Public Review Draft 01. http://docs.oasis-open.org/regrep/regrep-core/v4.0/csprd01/regrep-core-overview-v4.0-csprd01.odt.

[39] Corpet, F., and Michot, B., "RNAlign program: alignment of RNA sequences using both primary and secondary structures". Journal Comput. Appl. Biosci., Vol. 10, No. 4(1994): pp. 389-399.

[40] Corrales, J.C., Grigori, D., and Bouzeghoub, M., "BPEL processes matchmaking for service discovery". In Proceedings of OTM Conferences: Cooperative Information Systems (CoopIS'6), pp. 237 – 254., 2006.

[41] Cruz, I. and Sunna, W., "Structural Alignment Methods with Applications to Geospatial Ontologies". Transactions in GIS, special issue on Semantic Similarity Measurement and Geospatial Applications, Vol. 12, No. 6(2008), pp. 683–711.

[42] Cruz, I., Antonelli, F. P., and Stroe, C., "Efficient Selection of Mappings and Automatic Quality-driven Combination of Matching Methods". In Proceedings of international workshop on Ontology Matching (OM'09), Vol. 551 of CEUR Workshop Proceedings, CEUR-WS.org, 2009, 2009.

[43] Davenport, T.H. & Short, J.E., "The New Industrial Engineering: Information Technology and Business Process Redesign". Sloan Management Review, pp. 11-27, (1990 Summer).

[44] Davydov, E., and Batzoglu, S., "A computational model for RNA multiple structural alignment". Journal Theoretical Computer Science, Vol. 368, No. 3 (2006), pp.205-216.

[45] Doshi, P., Goodwin, R., Akkiraju, R., and Verma, K., "Dynamic workflow composition using markov decision processes". International Journal of Web Services Research, Vol. 2, No. 1(2005), pp. 1-17.

[46] Dulucq, S., and Tichit, L., "RNA Secondary structure comparison: exact analysis of the Zhang–Shasha tree edit algorithm". Journal Theoretical Computer Science, Vol. 306, No. 13(2003), pp. 471 – 484.

[47] El-Mabrouk, N. and Raffinot, M..,"Approximate matching of secondary structures". In Proceedings of the Sixth Annual international Conference on Computational Biology (RECOMB '02), 156-164, 2002.

[48] Eshuis, R. and Grefen, P., "Structural matching of BPEL processes". In Proceedings of the Fifth European Conference on Web Services (ECOWS '07), pp. 171–180, 2007.

[49] Euzenat, J. and Valtchev, P., "Similarity-based ontology alignment in OWL-lit.". In Proceedings of the 15th European Conference on Artificial Intelligence (ECAI'04), pp. 333-337, 2004.

[50] Euzenat, J., Ferrara, A., Meilicke, C., Pane, J., Scharffe, F., Shvaiko, P., Stuckenschmidt, H., Šváb-Zamazal, O., Svátek, V. and dos Santos, C.T., "First results of the Ontology Alignment Evaluation Initiative 2010". In Proceedings of the Fifth International Workshop on Ontology Matching (OM'10), the 9th International Semantic Web Conference ISWC, 2010.

[51] Ferrara, E. and Baumgartner, R., "Automatic Wrapper Adaptation By Tree Edit Distance Matching". In Proceedings of the 2nd International Workshop of In Combinations of Intelligent Methods and Applications (CIMA'10), pp 41-54, 2010.

[52] Fluri, B., Wuersch, M., PInzger, M., and Gall,H., "Change Distilling: Tree Differencing for Fine-Grained Source Code Change Extraction". IEEE Transactions on Software Engineering, Vol. 33, No. 11, pp. 725–743, 2007.

[53] Fokaefs, M., Mikhaiel, R., Tsantalis, N., Stroulia, E., "An Empirical Study on Web Service Evolution". In Proceedings of the 9th International Conference on Web Services (ICWS'11), pp. 49-56, 2011.

[54] Giunchiglia, F. , Yatskevich, M., Mcneill, F., "Structure preserving semantic matching". In Proceedings of the ISWC+ASWC International workshop on Ontology Matching (OM'07), pages 13–24, 2007.

[55] Gotoh, O. , "An Improved Algorithm for Matching Biological Sequences". Journal of Molecular Biology, Vol. 162, No. 3 (1981), pp. 705-708.

[56] Havey, M. "Essential Business Process Modeling". Book published by O'Reilly & Associates, ISBN 0-596-00843-0, 2005.

[57] Herrbach, C., Denise, A., Dulucq, S., Touzet, H., "Alignment of RNA secondary structures using a full set of operations", Rapport de Recherche LRI n° 1451, 2006

[58] Hinchcliffe, D., "Situational Software Platforms Begin to Emerge". ZDNet, http://blogs.zdnet.com/Hinchcliffe/?p=69, October 16th, 2006, last accessed on Jun 16th, 2008.

[59] Hochsmann, M., Toeller, T., Giegerich, R., and Kurtz, S., "Local Similarity of RNA Secondary Structures." In Proceedings of Computational Systems Bioinformatics (CSB'03): pp. 159-168, 2003.

[60] Höchsmann, M., Voss, B., and Giegerich, R., "Pure Multiple RNA Secondary Structure Alignments: A Progressive Profile Approach". IEEE/ACM Trans. Comput. Biol. Bioinformatics Vol. 1, No. 1 (2004), pp. 53-62.

[61] Höchsmann, M., "The tree alignment model : algorithms, implementations and applications for the analysis of RNA secondary structures". Dissertation, Universität Bielefeld, Technische Fakultät, 2005.

[62] Hofacker, I., Fontana, W., Stadler, P., Bonhoeffer, L., Tacker, M., and Schuster, P., "Fast Folding and Comparison of RNA Secondary Structures." Chemical Monthly, Vol. 125, No. 2 (1994): pp. 167-188.

[63] Hoffmann, C. M. and O'Donnell, M. J., "Pattern matching in trees". Journal of the Association for Computing Machinery (JACM), Vol. 29, No. 1(1982), pp. 68–95.

[64] Hollingsworth, D., "The workflow reference model". Workflow Management Coalition Specification TC00-1003, Workflow Management Coalition, Winchester Hampshire, UK, January 1995.

[65] Horrocks, I., "A Denotational Semantics for Standard OIL and Instance OIL". http://www.ontoknowledge.org/oil/downl/semantics.pdf, 2000.

[66] Isert, C., "The Editing Distance Between Trees." , 1999, http://citeseer.ist.psu.edu/isert99editing.html (accessed Aug. 2006)

[67] Jaeger, M.C., Rojec-Goldmann, G., Liebetruth, C., Mühl, G., Geihs, K., "Ranked Matching for Service Descriptions Using OWL-S". In Proceedings of Kummunikation in Verteilten Systemen (KiVS'5), pp. 91-102, 2005.

[68] Jiang, T., Wang, L., and Zhang, K., "Alignment for trees -an alternative to tree edit". In Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching, LNCS, 807: pp. 75-86, 1994.

[69] Jiang, T., Wang, L. and Zhang, K., "Alignment of trees – an alternative to tree edit". Theoretical Computer Science, Vol. 143, No. 1(1995), pp. 137-148.

[70] Jiang, Y., Stroulia, E., "Towards Reengineering Web Sites to Webservices Providers", In Proceedings of Eighth Euromicro Working Conference on Software Maintenance and Reengineering (CSMR'4), pp. 296-305, 2004.

[71] Jin, J., Sarker, B. K., Bhavsar, V. C., Yang, L., and Boley, H., "Towards a Weighted-Tree Similarity Algorithm for RNA Secondary Structure Comparison". In Proceedings of the Eighth international Conference on High-Performance Computing in Asia-Pacific Region (HPCASIA'5). pp. 639 - 644 , 2005.

[72] Kensche, D., Quix, C., Chatti, M. A., and Jarke, M., "GeRoMe: A Generic Role Based Metamodel for Model Management", Journal on Data Semantics, VIII, pp. 82–117, 2007.

[73] Kilpel ̈ainen, P. and Mannila, H., "Ordered and unordered tree inclusion". SIAM Journal of Computing, Vol. 24, No. 2, pp. 340–356, 1995.

[74] Kim, Y., Park, J., Kim, T., and Choi, J., "Web Information Extraction by HTML Tree Edit Distance Matching". In Proceedings of International Conference on Convergence Information Technology (ICCIT'07), pp.2455-2460, 2007

[75] Klein, P, Tirthapura, S., Sharvit, D. and Kimia, B., "A tree- edit-distance algorithm for comparing simple, closed shapes". In Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'00), pp. 696–704, 2000.

[76] Kruchten, P., "The 4+1 view model of architecture". IEEE Software, Vol. 12, No. 6, pp. 42–50, 1995.

[77] Küster, J. M., Gerth, C., Förster, A., and Engels, G., "Detecting and Resolving Process Model Differences in the Absence of a Change Log". In Proceedings of the 6th International Conference on Business Process Management (BPM'08). LNCS, Vol. 5240. pp. 244-260, 2008.

[78] Lah, T., "The Services Alignment Risk Factor: The Real Challenge for HP". Article is provided courtesy of Prentice Hall, 2001, http://www.phptr.com/articles/article.asp?p=23768&seqNum=5&rl=1

[79] Lah, T. E., "The Services Alignment Risk Factor: The Real Challenge for HP". www.informit.com, Oct 26, 2001, Article is provided courtesy of Prentice Hall, http://www.informit.com/articles/article.aspx?p=23768 (last accessed July 30, 2010).

[80] Laski, J. and Szermer, W., "Identification of program modifications and its applications in software maintenance". In Proceedings of Conference on Software Maintenance, pp. 282 – 290, 1992.

[81] Lassila, O., and Swick, R.R. "Resource Description Framework (RDF) Model and Syntax Specification". W3C Recommendation 22 February 1999, http://www.w3.org/TR/1999/REC-rdf-syntax-19990222.

[82] Le, D.-N., Nguyen, V.-Q., and Goh, A., "Matching WSDL and OWL-S Web Services", In Proceeding of the 2009 IEEE International Conference on Semantic Computing (ICSC'09), pp. 197-202, 2009.

[83] Le, S., Nussinov, R., and Mazel, J., "Tree graphs of RNA secondary structures and their comparison". Computational Biomedical Research, Vol. 22, No. 5 (1989), pp. 461-473.

[84] Le, S., Owens, J., Nussinov, R., Chen, J., Shapiro, B., and Maizel, J., "RNA secondary structures: comparison and determination of frequently recurring substructures by consensus". Comput. Appl. Biosci., Vol. 5, No. 3 (1989), pp. 205 - 210.

[85] Lee, M. L., Yang, L. H., Hsu, W. and Yang, X., "XClust: Clustering XML Schemas for Effective Integration". In Proceedings of the eleventh international conference on Information and knowledge management (CIKM'02), pp. 292-299, 2002.

[86] Levenshtein,V., "Binary codes capable of correcting deletions, insertions and reversals". Soviet Physics Doklady, Vol. 10, No. 8. (1966), pp. 707-710.

[87] Li,J., Tang,J., Li,Y., and LuoQ., "RiMOM: A dynamic multi-strategy ontology alignment framework". IEEE Transaction on Knowledge and Data Engineering, Vol. 21, No. 8(2009), pp. 1218–1232.

[88] Lin, G., Ma, B., and Zhang, K., "Edit distance between two RNA structures." In Proceedings of the Fifth Annual international Conference on Computational Biology. (RECOMB '01), pp. 211-220, 2001.

[89] Lindholm, T., Kangasharju, J., Tarkoma, S., "Fast and simple XML tree differencing by sequence alignment". In Proceedings of the 2006 ACM symposium on Document engineering (DocEng'06), pp. 75-84, 2006.

[90] Liu, B., Han, H, Noro, T., and Tokuda, T., "Personal News RSS Feeds Generation Using Existing News Feeds". In Proceedings of International Conference on Web Engineering (ICWE'09) , pp. 419-433, 2009.

[91] Liu, F., Shi, Y., Yu, J., Wang, T., and Wu, J., "Measuring Similarity of Web Services Based on WSDL". In Proceedings of International Conference on Web Services (ICWS'10), pp.155-162, 2010.

[92] Liu, J., Wang, J., Hu, J., and Tian, B., "A method for aligning RNA secondary structures and its application to RNA motif detection". BMC Bioinformatics. 6: 89, 2005.

[93] Liu, Y., Wang, J., Zhu, J., Liang, H., Tian, Z., and Sun, W., "Business Process Modeling in Abstract Logic Tree". IBM Research Report, IBM Research Division, China Research Laboratory, RC23444 (C0411-006), November, 2004.

[94] Ma, B., Wang, L., and Zhang, K., "Computing similarity between RNA structures". Theoretical Computer Science, Vol. 276, No. 1-2(2002), pp. 111-132.

[95] Mahleko, A., Wombacher, A., and Fankhauser, P., "A Grammar-Based Index for Matching Business Processes". In Proceedings of International Conference on Web Service (ICWS'5), pp. 21-30, 2005.

[96] Manola, F., and Miller, E., "RDF Primer". W3C Recommendation 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-primer-20040210/.

[97] Marian, A., Abiteboul, S., Cobena, G., and Mignet, L., "Change-centric management of versions in an XML warehouse". In Proceedings of the International Conference on Very Large Data Bases (VLDB'01), pp. 581-590, 2001.

[98] Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., Mcllraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne,T., Sirin, E., Srinivasan, N., and Sycara, K. "Owl-s: Semantic markup for web services". http: //www.daml.org/services/owl-s/1.1/overview/, 2004.

[99] McGuinness, D.L., Fikes, R., Stein, L.A., and Hendler, J.A, "DAML-ONT: An Ontology Language for the Semantic Web". Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential, The MIT Press, Cambridge, Massachusetts, ISBN 0262062321, pp. 65-94, 2003.

[100] McGuinness, D.L., and Harmelen, F.V., "OWL Web Ontology Language: Overview". W3C Recommendation, 10th of February 2004, http://www.w3.org/TR/owl-features/

[101] McIlraith, S., and Mandell, D., "Comparison of DAML-S and BPEL4WS (initial draft)". Knowledge Systems Lab, Stanford University, September 5, 2002.

[102] Mikhaiel, R., Stroulia, E., "Examining Usage Protocols for Service Discovery". In Proceedings of the 4th International Conference on Service Oriented Computing (ICSOC'06), LNCS 4294, pp. 496-- 502, 2006.

[103] Mikhaiel,R., Lin, G., Stroulia,E., "Simplicity in RNA Secondary Structure Alignment: Towards biologically plausible alignments". In Proceedings of the IEEE 6th Symposium on Bioinformatics & Bioengineering (BIBE '06), pp. 149 - 158 , 2006.

[104] Miller, E., "An Introduction to the Resource Description Framework". D-Lib Magazine, ISSN 1082-9873, May 1998.

[105] Milner, R., Parrow, J., and Walker, D., "A Calculus of Mobile Processes, Part I+II". Journal of Information and Computation, Vol. 100, No. 1 (1992), pp. 1-87.

[106] Mlýnková, I., "Similarity of XML schema definitions". In Proceeding of the eighth ACM symposium on Document Engineering (DocEng'08), 2008.

[107] Munzner, T., Guimbretiere, F., Tasiran, S., Zhang, L., and Zhou, Y., "TreeJuxtaposer: Scalable Tree Comparison using Focus+Context with Guaranteed Visibility". SIGGRAPH, ACM Transactions on Graphics, Vol. 22, No. 3 (2003), pp. 453-462.

[108] Musser, J., "Salesforce.com Launches ApexConnect".blog.ProgrammableWeb.comNovember, 27th, 2006, http://blog.programmableweb.com/2006/11/27/salesforcecom-launches-apexconnect/, last accessed on Jun 16th, 2008.

[109] Nguyen, B., Abiteboul, S., Cobena, G., and Preda, M., "Monitoring XML data on the web". In Proceedings of the 2001 International Conference on Management of Data (SIGMOD'01), pp. 437-448 ,2001.

[110] Nierman, A. and Jagadish, H. V., "Evaluating Structural Similarity in XML Documents". In Proceedings of the Fifth International Workshop on the Web and Databases (WebDB'02), pp. 61-66,2002.

[111] O'Reilly, T., "What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software".web article on O'Reilly net, Sep 30, 2005 (http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html access on Jan 3rd, 2009).

[112] Object Modeling Group (OMG), "XML Metadata Interchange (XMI)", OMG Formally Released Versions Of XMI, version 2.1.1, December 2007. http://www.omg.org/spec/XMI/2.1.1/

[113] Paoli, J., Valet-Harper, I., Farquhar, A., and Sebestyen, I., "ECMA-376 Office Open XML File Formats". 2006.

[114] Paolucci, M., Kawamura, T., Payne, T. R., and Sycara, K., "Semantic Matching of Web Services Capabilities". In Proceedings of the 1st International Semantic Web Conference (ISWC'03), pp. 333-347, 2002.

[115] Papavassiliou, V., Flouris, G., Fundulaki, I., Kotzinos, D., Christophides, V., "On Detecting High-Level Changes in RDF/S KBs". In Proceedings of the International Semantic Web Conference (ISWC'09), pp. 473–488, 2009.

[116] Payne, T.R., Paolucci, M., and Sycara, K., "Advertising and Matching DAML-S Service Descriptions". In Proceedings of Semantic Web Working Symposium (SWWS'01), 2001.

[117] Peters, L., "Change Detection in XML Trees: a Survey". 3rd Twente Student Conference on IT, Enschede June, 2005.

[118] Qeli,E., Gllavata, J., Freisleben, B., "Customizable detection of changes for XML documents using XPath expressions". In Proceedings of ACM Symposium on Document Engineering (DocEng'06), pp. 88-90, 2006.

[119] Ramesh, R. and Ramakrishnan, I.V., "Nonlinear pattern matching in trees". Journal of the Association for Computing Machinery (JACM), Vol.39, No. 2(1992), pp. 295– 316.

[120] Reis, D., Golgher, P., Silva, A. and Laender, A., "Automatic web news extraction using tree edit distance". In Proceedings of the 13th International Conference on the World Wide Web (WWW'04), pp. 502–511, 2004.

[121] RNAForester Online: http://bibiserv.techfak.uni-bielefeld.de/rnaforester/ (accessed Aug. September 2011).

[122] Salimifard, K., and Wright, M. "Petri-Net based Modeling of Workflow Systems: An Overview". In European Journal of Operational Research, Vol. 134, No. 3(2001), pp. 218–230.

[123] Sankoff, D., and Cedergren, R., "Simultaneous comparison of tree or more sequences related by a tree". The Theory and Practice of Sequence Comparison, Vol. 28(1983), pp. 253-263.

[124] Schmidt, A., Waas, F., Kersten, M.L., Carey, M.J., Manolescu, I., Busse, R., "XMark: A Benchmark for XML Data Management". In Proceedings of the International Conference on Very Large Data Bases (VLDB'02), pp 974-985, 2002.

[125] Selkow, S., "The tree-to-tree editing problem", Information Processing Letters, Vol. 6, No. 6 (1977) ,pp. 184-86.

[126] Shapiro, B., "An algorithm for comparing multiple RNA secondary structures." Comp. Appl. Biosci, Vol. 4, No. 3(1988), pp. 387-393.

[127] Sivaraman, E., and Kamath, M., "On the use of petri nets for business process modeling". http://citeseer.ist.psu.edu/535337.html

[128] Shapiro, B., and Zhang, K., "Comparing multiple RNA secondary structures using tree comparisons." Comput. Appl. Biosci., Vol. 6. No. 4 (1990): pp. 309-318.

[129] Stroulia, E., and Wang, Y., "Structural and Semantic Matching for Assessing Web-Service Similarity". International Journal of Cooperative Information Systems, Vol. 14, No. 4(2005), pp. 407-437.

[130] Sullivan, L., "eBay Developers Create Huge Software-As-A-Service Community". TechWeb Network (techweb.com), March 03, 2006, http://www.techweb.com/wire/ebiz/181500918, last accessed on Jun 16, 2008.

[131] Sun Microsystems. OpenOffice.org XML File Format. 1.0, Dec. 2002

[132] Syeda-Mahmood, T. F., Shah, G., Akkiraju, R., Ivan, A. A., and Goodwin, R., "Searching Service Repositories by Combining Semantic and Ontological Matching". In Proceedings of International Conference on Web Service (ICWS'05),pp. 13-20, 2005.

[133] Szymanski, M., Barciszewska, M., Erdmann, V., and Barciszewski, J. "5S ribosomal RNA database". Journal Nucleic Acids Research, Vol. 30, No. 1 (2002), pp. 176-178.

[134] Tai, K.C., "The tree-to-tree correction problem". Journal of the ACM, Vol. 26, No. 3(1979), pp. 422–433.

[135] The RNA match package: http://www.csd.uwo.ca/~kzhang/rna/rna_match.html (accessed Aug. 2006)

[136] Tsantalis, N., Negara, N., Fokaefs, M., Mikhaiel, R., and Stroulia, E., "A Domain-Agnostic Technique for Differencing Object-Oriented Models", to appear.

[137] Vagena, Z., Moro, M., and Tsotras, V., "Twig query processing over graph-structured XML data", In Proceedings of the 7th International Workshop on the Web and Databases: colocated with ACM SIGMOD/PODS, pp. 43-48, 2004,

[138] Virdell, M., "Business processes and workflow in the Web services world, article". IBM developerWorks, SOA Web Services, Jan, 2003, http://www-128.ibm.com/developerworks/webservices/library/ws-work.html#author.

[139] Wang, L., and Gusfield, D., "Improved approximation algorithms for tree alignment". In Proceedings of the 7th Combinatorial Pattern Matching conference, (CPM'96) pp. 220-233, 1996.

[140] Wang, Y., DeWitt, D. J. and Cai, J.-Y., "X-Diff: an effective change detection algorithm for XML documents". In Proceedings of the19th International Conference on Data Engineering, pp. 519–530, 2003.

[141] Wang, Y., Stroulia, E. "Flexible Interface Matching for Web-Service Discovery". In Proceedings of the 4th International Conference on Web Information Systems Engineering, pp. 147-156, 2003.

[142] Wang, Z., and Zhang, K., "Alignment between Two RNA Structures". In Proceedings of the 26th international Symposium on Mathematical Foundations of Computer Science, pp. 690-702, 2001.

[143] Wang, Z. and Zhang, K., "Multiple RNA Structure Alignment". In Proceedings Computational Systems Bioinformatics Conference (CSB 04): pp. 246-254, 2004.

[144] Webber,N., O'Connell,C., Hunt,B., Levine,R., Popkin,L., and Larose, G., "The Information and Content Exchange (ICE) Protocol". http://www.w3.org/TR/NOTE-ice.

[145] Wombacher, A., Fankhauser, P., and Neuhold, E. "Transforming BPEL into Annotated Deterministic Finite State Automata for Service Discovery." In Proceedings of International Conference on Web Services (ICWS'04), pp. 316-323, 2004

[146] World Wide Web Consortium (W3C), Cambridge, Massachusetts, USA. Document Object Model (DOM) Level 3 Core Specification, Apr. 2004. W3C Recommendation.

[147] World Wide Web Consortium (W3C), Cambridge, Massachusetts, USA. Extensible Markup Language (XML) 1.0 (Fifth Edition), W3C Recommendation 26 November 2008. http://www.w3.org/TR/2008/REC-xml-20081126/

[148] World Wide Web Consortium (W3C), Cambridge, Massachusetts, USA. OWL 2 Web Ontology Language, W3C Recommendation 27 October 2009.

[149] World Wide Web Consortium (W3C), Cambridge, Massachusetts, USA. OWL-S: Semantic Markup for Web Services, W3C Recommendation 22 November2004.

[150] World Wide Web Consortium (W3C), Cambridge, Massachusetts, USA. Remote Events for XML (REX) 1.0, Feb. 2006. W3C Working Draft.

[151] World Wide Web Consortium (W3C), Cambridge, Massachusetts, USA. Resource Description Framework (RDF), Feb. 2004. W3C Recommendation.

[152] World Wide Web Consortium (W3C), Cambridge, Massachusetts, USA. Scalable Vector Graphics (SVG). 1.1 Specification, Jan. 2003. W3C Recommendation.

[153] World Wide Web Consortium (W3C), Cambridge, Massachusetts, USA. Service Modeling Language, Version 1.1, May. 2009. W3C Recommendation.

[154] World Wide Web Consortium (W3C), Cambridge, Massachusetts, USA. SOAP Version 1.2 Part 0: Primer (Second Edition), Apr. 2007. W3C Recommendation.

[155] World Wide Web Consortium (W3C), Cambridge, Massachusetts, USA. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, Apr. 2007. W3C Recommendation.

[156] World Wide Web Consortium (W3C), Cambridge, Massachusetts, USA. XHTML 1.0 The Extensible HyperText Markup Language (Second Edition), Aug.2002. W3C Recommendation.

[157] World Wide Web Consortium (W3C), Cambridge, Massachusetts, USA. XML Information Set, 2nd edition, Feb. 2004. W3C Recommendation.

[158] World Wide Web Consortium (W3C), Cambridge, Massachusetts, USA. XML Path Language (XPath) 2.0, 2nd edition, Dec. 2010. W3C Recommendation.

[159] World Wide Web Consortium (W3C), Cambridge, Massachusetts, USA. XQuery 1.0 and XPath 2.0 Data Model (XDM), Nov. 2005. W3C Candidate Recommendation.

[160] World Wide Web Consortium, (W3C) Cambridge, Massachusetts, USA. Web Ontology Language: Overview (OWL), Feb. 2004. W3C Recommendation.

[161] Xing, Z. and Stroulia, E., "UMLDiff: an algorithm for object oriented design differencing, " in Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering, pp. 54–65, 2005.

[162] Xu, P., Wang, Y., Cheng, L., and Zang, T., "Alignment results of SOBOM for OAEI 2010", In Proceedings of Ontology Matching Workshop colocated with ISWC 2010.

[163] Xu, Y., Wang, L., and Deng, X., "Exact pattern matching for RNA secondary structures". In Proceedings of the Second Conference on Asia-Pacific Bioinformatics, ACM International Conference Proceeding Series, Vol. 55, pp. 257-263, 2004.

[164] Zhai, Y., Liu, B., "Web data extraction based on partial tree alignment", In Proceedings of the 14th international conference on World Wide Web (WWW'04), pp. 76-85, 2005.

[165] Zhang, K., "Computing Similarity Between RNA Secondary Structures". In Proceedings of the IEEE international Joint Symposia on intelligence and Systems, pp.126 – 132, 1998.

[166] Zhang, K., Stgatman, R., and Shasha, D., "Simple fast algorithm for the editing distance between trees and related problems". SIAM Journal on Computing, Vol. 18, No. 6, (1989), pp. 1245–1262.

[167] Zhang, K., Shasha, D. and Wang, J. T. L., "Approximate tree matching in the presence of variable length don't cares". Journal of Algorithms, Vol. 16, No. 1(1994), pp. 33–66.

[168] Zhang, K., Wang, J. T. L., and Shasha, D., "On the editing distance between undirected acyclic graphs and related problems". In Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching (CPM '95), pp. 395–407, 1995.

[169] Zhang, K., Wang, L., and Ma, B., "Computing similarity between RNA structures". In Proceedings of 10th Annual Symposium on Combinatorial Pattern Matching (CPM '99), pp. 281-293, 1999.