Investigate the Assessment of Information Structures using Graphs

Yannick Allard
Michel Mayrand
OODA Technologies Inc.

Prepared By:
OODA Technologies Inc.
4891 Grosvenor
Montreal, QC H3W2M2

Contract Project Manager: Yannick Allard, 514-476-4773
PWGSC Contract Number: W7707-145677
CSA: Anthony Isenor, Defence Scientist, 902-426-3100 x106

Contract Report
DRDC-RDDC-2015-C162
March 2015

Investigate the Assessment of Information Structures using Graphs

Yannick Allard
Michel Mayrand

Prepared By: OODA Technologies Inc.
4891 Grosvenor
Montréal (Qc), H3W 2M2
514.476.4773

Prepared For: Defence Research & Development Canada, Atlantic Research Centre
9 Grove Street, PO Box 1012
Dartmouth, NS
B2Y 3Z7
902-426-3100

Scientific Authority: Anthony Isenor
Contract Number: W7707-145677
Call Up Number: 9; 4501234676
Project: 01da, Maritime Information Warfare
Report Delivery Date: March 31, 2015

This page is intentionally left blank.

Executive Summary

The rapid increase of information and communication technologies has made accessible large amounts of information stored in different application-specific databases. The key commonalities underlying database applications are that they use structured representations. In maritime domain awareness, several schemas and ontologies have been proposed. When database schemas for the same domain are developed by independent parties, they will almost always be quite different from each other. As an example, some are used within a more specific domain, such as track representation in a Command and Control System (CCS), while others, such as National Information Exchange Model - Maritime (NIEM-M) Information Exchange Package Documentation (IEPD), aim to be applicable in a wide range of maritime applications where the exchange of information is necessary.

As such, there is a need for schema matching, to identify similarities and differences, and schema integration, which is merging a set of given schemas into a single global schema. There are many known techniques for schema matching and integration amongst which we apply graph matching. This document investigates the use of graph theory for the comparison of schemas from the Maritime Domain Awareness (MDA) community.

The multiple schemas available from different communities of interest and for different underlying requirements enabled us to study the effect of multiple aspects of schemas within the particular task of schema matching and similarity measurements.

Multiple matching strategies were automatically run and evaluated using measures of performance typically used within the available literature. The comparisons that were performed highlighted many difficulties for automatic schema matching and similarity measurements.

This page is intentionally left blank.

Contents

List of Figures

This page is intentionally left blank.

List of Tables

This page is intentionally left blank.

C2	Command & Control
CCS	Command and Control System
EIEM	Enterprise Information Exchange Model
EPC	Electronic Port Clearance
IEPD	Information Exchange Package Documentation
MDA	Maritime Domain Awareness
MTB	MUSIC Test Bed
MUSIC	Multi-Sensor Integration within a Common Operating Environment
NIEM	National Information Exchange Model
NIEM-M	National Information Exchange Model - Maritime
NPR	National Position Repository
SQL	Structured Query Language
tf-idf	term frequency-inverse document frequency
TDP	Technology Demonstration Program
XML	Extensible Markup Language
XSD	XML Schema Definition

This page is intentionally left blank.

Part 1

Introduction

The rapid increase of information and communication technologies has made accessible large amounts of information stored in different application-specific databases and web sites [1]. The key commonalities underlying database applications that require semantic integration are that they use structured representations. Information structures typically represent how we store or exchange data and information. Many different kinds of structures can be considered as data/conceptual models [2] :

- description logic terminologies;
- XML-schemas;
- relational database schemas;
- catalogs and directories;
- entity-relationship models;
- UML diagram, and more.

In maritime domain awareness, several schemas and ontologies have been proposed (NIEM, MUSIC, NPR, Maritime Anomaly Detection, Electronic Port Clearance, etc). When database schemas for the same domain are developed by independent parties, they will almost always be quite different from each other [3]. From this and from the absence of a common supervising entity, the problem of semantic heterogeneities, the ambiguous interpretation of concepts which describes the meaning of data in heterogeneous data sources, is becoming more and more severe [1]. Semantic heterogeneity appears whenever there is more than one way to structure a body of data [4]. As such, there is a need for schema matching, to identify similarities and differences, and schema integration, which is merging a set of given schemas into a single global schema. Also, schema matching plays the central role in solving the problem of semantic heterogeneities.

There are many known techniques for schema matching and integration [5]. One of them is graph matching. In concept, the schema is comprised of objects (*i.e.*, node) and the relationships (*i.e.*,

edges) and can therefore be transformed into graphs. At that point, the analysis techniques that have been developed for graphs can be applied to schemas.

This document investigates the use of graph theory for the comparison of schemas from the Maritime Domain Awareness (MDA) community and is organized as follows :

- Section 2 presents an overview of applications domain where there is a comparison made between two or more graphs with a special emphasis on the schema and ontology matching problem.
- Section 3 presents a brief review of existing graph/schema comparison/matching software.
- Section 4 presents four schemas, from the topic area of maritime domain awareness and being used to store or exchange maritime data, that will be examined in more detailed for schema/ontology matching with graph theory.
- Section 5 presents the detailed results obtained using graph theory over the selected schemas.
- Section 6 presents the lessons learned during the realization of this call-up and the general conclusion of this document.

Part 2

Review of Graph Comparison Methods and Applications

This section presents an overview of the graph matching and analysis problem and is organized as follows:

- Section 2.1, presents the basics of graph matching.
- Section 2.2 gives an overview of the application domains of graph matching algorithms and technology.
- Section 2.3 discusses the problem of schema integration in more details.

2.1 Graph matching

When graphs are used for the representation of structured objects, the problem of measuring object similarity turns into the problem of computing the similarity of graphs, which is also known as graph matching [6].

Figure 2.1 displays a generic classification of all graph matching types.

The subgraph matching problem is similar to the graph matching one but occurs when you have a set of graphs and you are trying to extract a subset of nodes that are highly connected [7]. This problem comes up in several applications such as gene networks, social networks, and designing molecular structures. Approximated subgraph matching, where the connectivity within each subset of nodes is not exactly consistent between graphs might be more interesting for the schema analysis in MDA.

The following section provides an overview of graph similarity and subgraph matching measures and algorithms.

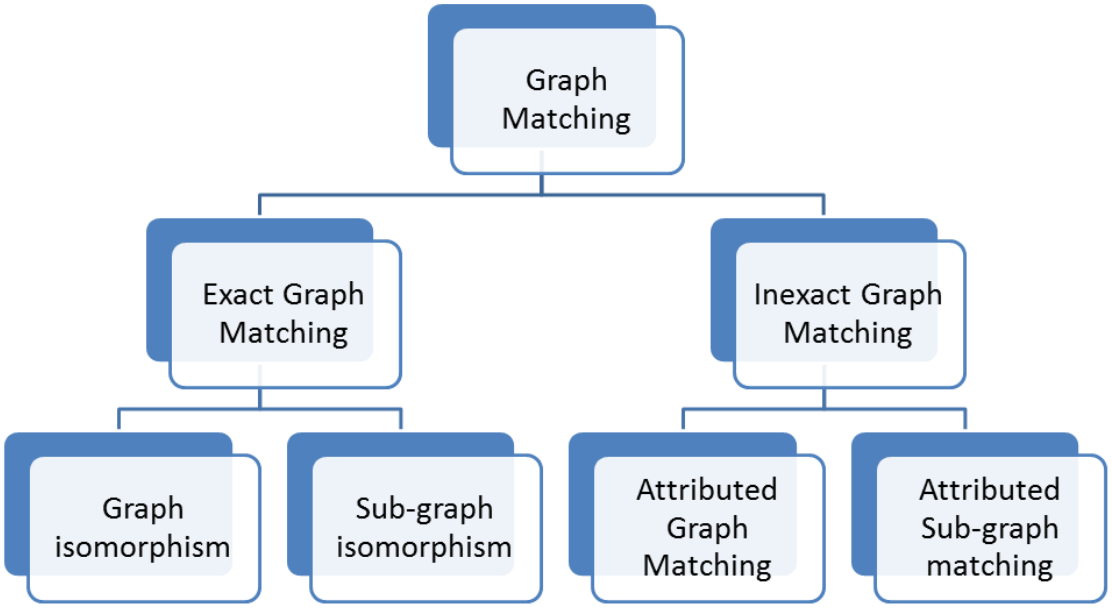

Figure 2.1: Classification of all the graph matching types

2.1.1 Graph similarity

Algorithms for graph comparison can be partitioned in three main categories [7]:

1. Edit distance / graph isomorphism;
2. Feature extraction;
3. Iterative methods.

Graph isomorphism is a bijective mapping from the nodes of G to the nodes of G' that preserves all labels and the structure of the edges [6]. It is a useful concept to find out if two objects are the same, up to invariance properties inherent to the underlying graph representation. Similarly, subgraph isomorphism can be used to find out if one object is part of another object, or if one object is present in a group of objects. Maximum common subgraph can be used to measure the similarity of objects even if there exists no graph or subgraph isomorphism between the corresponding graphs. Clearly, the larger the maximum common subgraph of two graphs is, the greater is their similarity [6].

A graph edit operation is either a deletion, insertion, or substitution (*i.e.* label change). Edit operations can be applied to nodes as well as to edges. The edit distance of two graphs, G and G', is defined as the shortest sequence of edit operations that transform G into G'. Obviously, the shorter this sequence is the more similar are the two graphs. Thus edit distance is suitable to measure the similarity of graphs [6].

Feature extraction matching is part of the attributed graph matching. Similar graphs probably share certain properties, such as degree distribution, diameter, eigenvalues. After extracting these features, a similarity measure is applied in order to assess the similarity between the graphs. These methods are powerful and scale well, as they map the graphs to several statistics that are much smaller in size than the graphs [7].

The iterative methods algorithms start with the assumption that two nodes are similar if their neighborhoods are also similar. In each iteration, the nodes exchange similarity scores and this process ends when convergence is achieved [7]. An example of such an approach, in the domain of schema matching, can be found in [8].

Several different measures of similarity between graphs are tested in [7].

2.2 Graph matching applications

Graph similarity has numerous applications in diverse fields (such as social networks, image processing, biological networks, chemical compounds, and computer vision), and therefore there have been suggested many algorithms and similarity measures [7]. This section presents an overview of the application domains that use graph matching to solve particular problems.

2.2.1 Schema matching

Schema matching is the task of identifying semantic correspondences between elements of two schemas [1]. A schema is a formal structure that represents an engineered artifact, such as a SQL schema, XML schema, entity-relationship diagram, ontology description, interface definition, or form definition [5].

Schema matching takes as input two schemas/ontologies, each consisting of a set of discrete entities (*e.g.*, tables, XML elements, classes, properties, rules, predicates), and determines as output the relationships (*e.g.*, equivalence, subsumption) holding between these entities [2].

The application domains of schema matching are numerous. For example ([2], [9]):

1. Schema integration: Since the schemas are independently developed in different real-world contexts, they often have different structure and terminology no matter if they are from different domains or even if they are from the same real world domain. Therefore, integration is required to translate the data from the original schemas into the integrated representation [9].

2. e-Business: for message translation as a message translator is required to convert messages between the different formats in order to exchange messages among buyers and sellers, and for catalog integration as catalogs are tree/graph-like structures, namely concept hierarchies with attributes. The catalog matching problem occurs when a single seller or a company wants to publish their products in the e-marketplace [9].
3. Data Warehouse: A data warehouse is a repository of integrated data sources, available for queries and analysis. Data are extracted from heterogeneous sources. Data from the source format has to be transformed into the warehouse format during the extraction [9].
4. P2P databases: Because of the high flexibility and dynamics of the P2P networks, a run time schema matching operation is applied in order to establish information exchange between peers. All the peers in the P2P networks are autonomous, and they might use different schemas. Therefore, a matching operation is required to identify and characterize the relationships between their schemas [9].
5. Agent Communication: By definition, software agents are autonomous pieces of software that are programmed to achieve a goal without supervision. For more complex goals, a single agent is often not sufficient and require the help of other agents, which requires the ability to communicate with them. As a consequence, when two autonomous and independently designed agents meet, they have the possibility of exchanging messages, but little chance to understand each other if they do not share the same content language and ontology. Thus, it is necessary to provide the possibility for these agents to match their ontologies in order to either translate their messages or integrate bridge axioms in their own models.
6. Web Services Integration: Web services are processes that expose their interface to the web so that users can invoke them. Some of those web services can be automated to interact with each other such as semantic web services which provide a richer and more precise way to describe the services through the use of knowledge representation languages and ontologies. Again, web services are mostly independently designed and the output ontology used for the first web service may not exactly fit the ontology used for the input of the second one. Henceforth, both for finding the adequate service and for interfacing services it will be necessary to establish the correspondences between the terms of the descriptions. This can be provided through matching the corresponding ontologies.
7. Life sciences applications and the semantic web: alignment of ontologies for gene or anatomical structures, alignment of patient records [5].

For most of the applications, schema matching is just one step in a multi-step process. That multi-step process involves other operators that manipulate schemas and mappings, such as schema merging and mapping composition.

Schema and ontology matching are treated with more depth in section 2.3

2.2.2 Music similarity

Graph similarity was also used in [10] to measure similarity in music. This aspect of graph matching is generally used for the automatic generation of playlists.

2.2.3 Pattern recognition / computer vision

Computer science has long used graph theory to represent network and data structures, but one of the most interesting applications of graph theory are pattern recognition and computer vision. Given two images, image processing software can often identify interesting features, and then describe the relationship among these features, for instance, in terms of distance and angle [11].

A number of applications rely heavily on graph representations and comparison, including optical character recognition and biometric identification. Many of these problems involve trying to find a subject graph within a database of graphs, the same task that arises with chemical compounds in databases [11].

2.2.4 Social Networking

Graph pattern matching is fundamental to social network analysis. Traditional techniques are subgraph isomorphism [12]. Social networks analysis introduce new challenges to graph pattern matching, from its definition to processing methods. For instance, social graphs are typically large. Facebook has more than 500 million users (nodes) with 65 billion links (edges) [12]. Also, social networks are frequently updated, which require recomputation quite often. As such, isomorphic methods cannot be applied as they are too expensive in computation time and memory.

Graph (or network) similarity has many uses in social networks analysis (marketing, intelligence, collaborative filtering, and more).

2.2.5 Biological networks

A biological network is any network that applies to biological systems. Biological networks provide a mathematical analysis of connections found in ecological, evolutionary, and physiological studies, such as neural networks [13].

Biological systems are complex networks with many different kinds of interacting units that span several scales, from the population level to the molecular scale. In many ways, the forefront of biology is no longer the collection of experimental observations, but the intelligent analysis of existing data [11].

Different kinds of graphs constructed from biological data require different types of analyses. One example of biological data with a graph structure is a mapping of metabolic networks into enzyme graphs: biological enzymes serve as nodes in the graph, with an edge existing from enzyme e_1 to enzyme e_2 if e_1 catalyzes a reaction whose product is the substrate for e_2 [11].

In bioinformatics, matching has been used for network analysis of molecular interactions. In this domain, data instances represent metabolic networks of chemical compounds, or molecular assembly maps. Matching of molecular networks and biochemical pathways helps to predict metabolism of an organism given its genome sequence [8]. In [14], weighted graphs comparison is used for brain connectivity analysis.

For an example of global alignment of protein-protein interaction networks, see [15].

2.2.6 Chemistry

The application of graph theory to problems in chemistry has a long history. Chemical compounds have a natural graph structure, with nodes representing different types of atoms and edges representing inter-atomic bonds. When experimental chemists develop a new compound, their first step is to compare its molecular structure to a database of compounds and look for those which are structurally identical, or have structurally identical sub-components. In graph theory, this is referred to as the maximum common subgraph problem. There is also similarity measure for chemical structures that uses structural subgraph matching to identify biochemically meaningful features in a database of metabolic compounds [11].

2.3 Schema/Ontology matching using graph theory

Schema matching is considered an important step in integrating data from multiple sources [3]. Most of the work on matching, as presented in section 5.3, has been carried out among database schemas in the world of information integration, XML-schemas and catalogs on the web, and ontologies in knowledge representation. However, there is a slight difference between schema and ontology mapping. On the one side, schema matching is usually performed with the help of techniques trying to guess the meaning encoded in the schemas. On the other side, ontology matching systems try to exploit knowledge explicitly encoded in the ontologies [2].

Depending on matching purposes and application domains, different types of results can be obtained, *e.g.* a unified schema, a unified representation or a similarity value [9].

Figure 2.2 represents the general architecture of schema matching [16].

Schema matching is a challenging task for two main reasons [3]:

1. First, the same real world entity could have different representations and hence the semantic relationships between the schema elements of the independently developed data sources are unknown.
2. Second, attribute names and data values of different data sources may not have lexical similarities or may not be described using same the description language.

These are generally called semantic heterogeneity and it appears whenever there is more than one way to structure a body of data [4].

Differing structures are a by-product of human nature as people think differently from one another even when faced with the same modeling goal [4]. Other difficulties for schema matching and similarity assessment are posed by the high expressive power and versatility of modern schema languages, in particular user-defined types and classes, component reuse capabilities, and support for distributed schemas and namespaces [17].

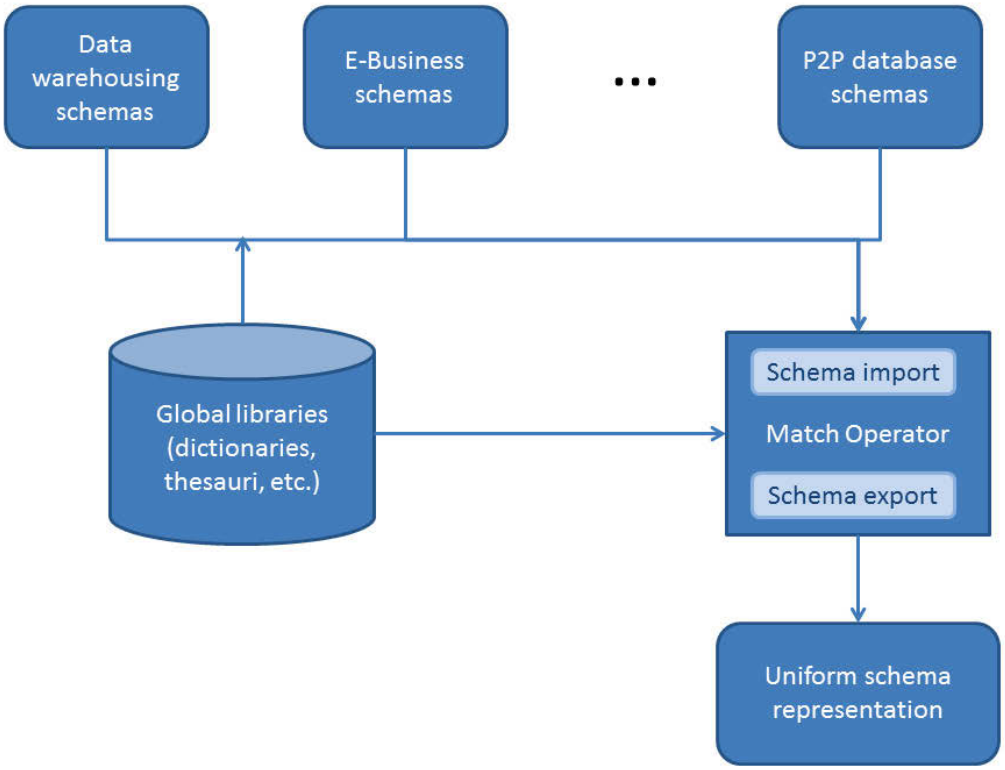

Figure 2.2: Generic architecture for schema matching

The problem of reconciling schema heterogeneity has been a subject of research for decades, but solutions are few. The fundamental reason that makes semantic heterogeneity so hard is that the data sets were developed independently, and therefore varying structures were used to represent the same or overlapping concepts [4].

Heterogeneity occurs not only in the schema, but also in the actual data values themselves. For example, there may be multiple ways of referring to the same ship. Hence, even though you are told that a particular ship in a database maps to ShipName, that may not be enough to resolve multiple references to this single ship [4].

To help solve the problem of schema heterogeneity, building a corpus of schema is suggested in [4], to help the software learn about:

1. Domain concepts and their representational variations: identify the main concepts in the domain and discover variations on how these concepts are represented.
2. Relationships between concepts: Given a set of concepts, we can discover relationships between them, and the ways in which these relationships are manifested in the representation. They can also be used to build a system that provides advice in designing new schemas.
3. Domain constraints: Find integrity constraints on the domain and its representations.

The first step in most schema matching approaches is to transform the schemas to be matched to a common model in order to apply the matching algorithm. Most of these approaches choose a graph data structure as the internal representation (ex: COMA, [18], [8], [16], [19]) [1]. This gives the choice of representing every schema feature as either a node or an edge of the graph, or to encode it using a node's or an edge's property [19]. Graph structure also helps reduce complexity of the problem [20]

There is a kind of a consensus, when using graph as the common model, to transform schema into directed and labeled graphs, which are sometimes synonymously referred to as (attributed) relational graphs or relational structures. Such a graph consists of a finite number of nodes, or vertices and a finite number of directed edges [6]. Labeled directed graphs with property sets are used in [19] while a rooted labeled graph is used in [1]. A rooted labeled graph is a directed graph such that nodes and edges are associated with labels, and in which one node is labeled in a special way to distinguish it from the graph's other nodes [1].

Schema matching using graphs can be defined as schema-based matching at the structural level following the classification of schema matching techniques found in [16] and [2].

Knowledge extraction for schema matching is done by exploiting two entities, (i) data and (ii) schema, *i.e.* the structure describing the data.

Schema based approaches only consider schema information, *e.g.* schema structure, attributes of schema elements such as data types, relationship types (partonomy, taxonomy) and constraints. A matcher first finds multiple match candidates, and then calculates the candidates' similarity represented by a normalized numeric value in the real interval [0,1]. Finally, the matcher finds the best match candidate by comparing the numeric values if they are available [9].

Structure-level matching refers to the combinations of elements in the structure. There are two kinds of structure-level matching, fully match and partial match. Fully match is the ideal case, *i.e.* all the components of the structures in the two input schemas map to each other. Partial match is also needed in some circumstances, *e.g.* matching sub-schemas from different domains [9].

Schema-matching techniques are based on the use of data interpretation and structural similarity. A schema matching problem can be reduced to a traditional graph matching problem by capturing hidden dependencies between attributes and structuring them labeled as a graph that takes into account the dependency relations among the attributes [21].

2.4 Summary

We have seen that matching schemas can be applied to a vast variety of domains and can be most valuable to resolve problems related to semantics between applications. Schema/ontology matching share a lot with graph matching and is the first but crucial step of a series of operations related to solve the above problems but remain a very difficult task because it remains a by-product of human thinking and can be expressed in so many different ways.

Part 3

Review of Schema Matching Software

In this section, we present a brief review of schema matching software followed by the selection of a tool to be used in this call-up and we describe the rationale behind this selection. This section presents an overview of the graph matching and analysis software and is organized as follows:

- Section 3.1, presents the categories of graph matching software.
- Section 3.2 gives a brief description of the available software.
- Section 3.3 discusses the software that was selected to be used in this investigation.

3.1 Matching Software categories

There are different kinds of schema matching software, for example, some are more specialized, other more generic. We describe here the main categories of tools and which ones are best suited for our task.

3.1.1 Structure vs instance

We distinguish between two categories of schema matching tools: schema-based (data structure) and instance-based (data values). For schema-based approaches, schema-level information is considered such as metadata, element names, data types, and structural properties/models whereas in instance-based approaches data and data content are considered. Generally, schema-based tools are the most common and the instance level matcher are typically used with schema-based matcher to boost the performance. There is a third category of matcher: hybrid matchers which combine several matching approaches. Most of these techniques employ additional information from dictionaries, thesauri, and user-provided match or mismatch information. Hence such techniques help determine match candidates based on multiple criteria or information sources. In this call-up, only maritime schemas (without data) were analyzed. Consequently, the tools that focus mostly on an instance-based approach were not selected.

3.1.2 Single vs multiple matcher algorithms

Each matching tool can use an individual matcher or a combination of them. The individual matching tool uses a single algorithm to perform the matching process. For combinational matchers, two types of matching can be done: (1) hybrid matchers take into account multiple criteria to perform the matching task, and (2) composite matchers run separate match algorithms on two schemas and combine the result. For this call-up, we thought that a multiple matcher approach could be useful in finding the most optimal ones (run in a batch mode). A single matcher approach could have given poor result if the tool were not optimal with the schemas to analyze.

3.1.3 Generic vs specialized

Some schema matching software are designed to address specific matching problems: protein/DNA match, social network analysis, etc. The design of specialized tools usually include support for typical graph topologies (and excluding others) as well as a list of adapted and optimized matching algorithm (and excluding others). The more generic matching tools will address a larger group of graph topologies and usually will contain a larger set of matching algorithms. In both cases, the ultimate performance is directly correlated with the capacity of the user to select the proper schemas to compare with the most optimal tool. Another aspect of generic matching tool is their capability to parse different file formats (XSD/XML, Relational/SQL, OWL/RDF). As we are at the beginning of investigating maritime schemas, it was more judicious to use a generic tool as we did not know yet the common characteristics of the schemas to compare. Also, the schemas to be analyzed had more than one format pointing to a generic tool as well.

3.1.4 Open-source vs commercial

Many of the schema matching software started as open-source code in university research groups in the early 2000's at the same as the field become popular. Some of these tools evolved to become commercial products. Other matching tools stayed within the university research group and grows from the contribution of the open-source community. The commercial schema matching software are usually permanently integrated in larger data analysis tool suite (ex: IBM and Microsoft) which make them quite expensive. Beside price and maintenance, there are not that many differences in quality and performance between commercial and open-source tools. In fact, most of the tools used and mentioned in papers refers to open-source ones. Also, open-source code allows the possibility of customization and creation of additional features. For all these reasons, an open-source tool would be more suitable for this call-up.

3.2 Software Description

We briefly describe some of the software package available. Table 3.1 at the end of this section includes web links and license type for all the tools described below.

3.2.1 Cupid/Biztalk

Cupid is a hybrid matcher that integrates element-based (linguistic matching with auxiliary information), structure-based, and reuse-based matching techniques. It is intended to be generic across data models and has been applied to XML and relational examples. Especially, CUPID also considers integrity constraints such as key and referential dependencies. It is intended to be generic across data models and has been applied to XML and relational examples. Cupid source code is not available anymore as it was acquired by Microsoft and integrated into the commercial application BizTalk Mapper. A single standard license costs around $2500US. (more than $10K for enterprise edition). Its user interface supports visualizing large schemas and complex matches. The Biztalk server is also incorporated in Microsoft Visual Studio and .NET framework.

3.2.2 Similarity Flooding/Rondo

Similarity Flooding is a Java library for schema matching which is integrated in the Rondo matching tool. It has been used with Relational, RDF and XML schemas. These schemas are initially converted into labeled graphs and SF approach uses fix-point computation to determine correspondences of 1:1 local and m:n global cardinality between corresponding nodes of the graphs. The algorithm has been implemented as a hybrid matcher, in combination with a name matcher based on string comparisons. First, the prototype does an initial element-level name mapping, and then feeds these mappings to the structural SF matcher.

3.2.3 OntoBuilder

Ontobuilder is a matching tool for web forms. It supports a GUI to crawl web forms, matching crawled forms, and generate an integrated form. OntoBuilder uses linguistic matching in combination with auxiliary information and some sequencing algorithms. Especially, it can also be used for matching RDF-based ontologies.

3.2.4 COMA/COMA++

COMA schema matching system is developed as a platform to combine multiple matchers in a flexible way and provides a large spectrum of individual matchers, in particular a novel approach aiming at reusing results from previous match operations, and several mechanisms to combine the results of matcher executions. COMA is a framework to comprehensively evaluate the effectiveness of different matchers and their combinations for real-world schemas. COMA is domain agnostic. COMA can parse different file formats: XSD/XML, SQL and OWL/RDF files. It supports a number of other features like merging, saving and aggregating match results of two schemas.

3.2.5 Clio/IBM InfoSphere Data Architect

Clio was developed at IBM and is the predecessor of the matching component in IBM Infosphere. Besides providing a comprehensive GUI and a matching engine, it has a schema management system that transforms and manages XML and SQL schemas. Clio uses a hybrid matching approach, which combines linguistic matching and learning algorithms. IBM released Infosphere in 2009 ($6,270 for enterprise edition). It has a mapping editor that supports linguistic matching and different types of databases like Oracle, DB2, Sybase, Microsoft SQL Server, MySQL.

3.2.6 Auto Mapping Core

Auto Mapping Core (AMC) is a generic framework that supports a customizable matching process. The matching process can be defined, executed, debugged, and visualized with highly flexible components. Existing matching tools can be plugged into AMC for a uniform evaluation and reuse. Especially, it supports various aggregation operators (e.g. sum, max, min) to combine the matching results of individual matchers.

3.2.7 YAM/YAM2

Yet Another Matcher (YAM) is a generic prototype that supports both matching tasks and post-match effort. One of its key features is the ability to dynamically combine similarity measures according to machine learning techniques, with the benefit of automatically tuning the thresholds for each measure. Its extended version, YAM++ [NB12], also supports ontology matching using machine learning approach.

3.2.8 Altova Mapforce

This tool has been released from 2008, with the price of 799 Euro for the enterprise edition. It has a graphical schema mapping interface which supports XML, relational databases, flat files, EDI and Microsoft Excel. Compared to other tools, it supports more user interaction features such as matching filters, structural matching, functional matching (union, intersection, sum, etc. operators).

3.2.9 JitterBit

This tool has been released in 2009, with the price of $4000/month for enterprise edition. Beside user-friendly interfaces and wizard tools, Jitterbit supports not only XML but also Web services. JitterBit focuses on data integration in the context of point-to-point application integration, ETL and SOA. It consists of two main components, namely an Integration Environment and an Integration Server.

3.2.10 Harmony OpenII

Harmony is the matching tool inside the open-source OpenII framework. It provides a graphical user interface and supports many well-known matching techniques. Especially, it supports composite matching where different confidence values (proposed by individual matchers) are aggregated at schema-level. Harmony is known to be able to match large schemas having about a thousand attributes.

3.2.11 NetWeaver

NetWeaver is a commercial tool which is the result of the research collaboration between SAP and the University of Leipzig. Consequently, it is strongly based on the COMA++ and also Auto Mapping Core (AMC), a framework conceived for supporting the integration of multiple schema matching approaches. NetWeaver provides the Matching Process Designer, that not only reports candidate matches but also helps users to select the techniques to adopt for finding correct matches.

3.2.12 Summary

Table 3.1 presents a summary of the different available software for schema matching.

It is important to notice that the matching operation typically constitutes the first step towards schema/ontology integration, web services integration or meta data management.

3.3 Tool selection

COMA++ was selected for this call-up. The main advantages are:

1. Free Community version available with source code.
2. Extension/Enhancement of COMA, open multi-component architecture;
3. Comprehensive graphical user interface;
4. Generic data model to uniformly support schemas and ontologies written in different languages (W3C XSD, XDR, OWL, Relational);
5. Repository for uniform management of schemas, ontologies and match results;
6. High-level operators for manipulating match results, e.g. compose, merge, and compare mappings;
7. Flexible combination of matchers to construct new matchers and match strategies;
8. Optimized implementation of matchers for fast execution times, essential improvement over COMA;

Name	**License**	**Inputs**	**Created**	**Notes**
Cupid, Biztalk	Commercial (Microsoft)	XSD	2001	Instance-based. Name matching and past user matchings
Onto-Builder	CLI, GUI	CLI, GUI	2004	
Similarity Flood-ing/Rondo	Java, GUI	XML, SQL	2001-2003	
COMA, COMA++	GUI, Java API	XSD, OWL, Relational	2002-2012	
Clio, IBM InfoSphere Data Architect	Commercial (IBM)	XML, Relational	2009	Typically schema-based (no instances support). Name matching and external thesauri.
YAM	GUI, CLI	XSD, OWL	2009	
Harmony (OpenII)	GUI, Open-source	XSD, OWL, Relational	2010	Tokenization/Stem-ming
AMC	Commercial, GUI	XSD	2011	
Altova Mapforce	Commercial (Altova)	XML, DB/SQL, EDI, Excel, XBRL, SOAP/REST	2008	Based on Name Matching
JitterBit	Open-source and Commercial	XML, Relational, WSDL	2009	Based on Name Matching
NetWeaver Process Integration	Commercial (SAP)	XML, Relational, OWL	2010	

Table 3.1: Ontology/Schema matching tools

9. New approaches for ontology, reuse-, context-, and fragment-based matching;
10. Platform for comparative evaluation of matchers and match strategies.

Designed to be generic, COMA++ was perfect for our first investigation of maritime schemas. Written in Java, it was easy to modify the code and add features to suit the purpose of this call-up. Most prototypes combine multiple criteria and properties in a hybrid algorithm, making it difficult to extend and improve. COMA 3.0, is one of a few tools following the composite approach to combine independently executed matchers. Accordingly, the extensibility of the composite prototypes consists of adding new matchers or learners and new methods for combining their results. Only COMA 3.0 supports a flexible infrastructure for constructing new matchers and match strategies from existing ones.

On the COMA website, it mentions a commercial version of the tool that we tried to acquire but our email requests remain unanswered. We found out that some features that we were looking for in the commercial version were in fact commented out in the base code in the free version and were easily reactivated.

This page is intentionally left blank.

Part 4

Maritime Domain Awareness Schemas

This section presents an overview of the schemas that are used to test the selected software for schema similarity and matching purpose. It is organized as follows:

1. Section 4.1 presents an overview of the National Information Exchange Model (NIEM) schema. It is the more complete one and we will use IEPDs as references to which we will compare the other available schemas.
2. Section 4.2 presents an overview of the database schema used in the National Position Repository (NPR).
3. Section 4.3 presents the schema of the Electronic Port Clearance information exchange (ISO-28005).
4. Section 4.4 presents the track database schema as part of the Multi-Sensor Integration within a Common Operating Environment (MUSIC)-Technology Demonstration Program (TDP).
5. Section 4.5 presents the summary of the characteristics of these schemas.

4.1 National Information Exchange Model - Maritime

NIEM is an Extensible Markup Language (XML)-based information exchange framework for sharing data between partners with like missions or information needs.

NIEM exchange models are published as Information Exchange Package Documentation (IEPD). Each IEPD contains the artifacts required to define the XML model. The IEPD is registered in the NIEM IEPD clearinghouse and available to mission partners.

IEPDs are the cornerstone for maritime information exchanges. Each IEPD within NIEM-Maritime defines a particular XML message, which is the basic unit of shared information. As requirements emerge, IEPDs can be created and altered for specific community's needs.

The NIEM-M schema contains elements to effectively share and exchange information about the maritime domain awareness. The data elements of this schemas are used to create IEPD.

As an example, the Notice Of Arrival IEPD defines a Notice of Arrival message using a number of definitions, including the definition of Vessel, Person, Arrival, Departure and CDC Cargo from the Maritime Enterprise Information Exchange Model (EIEM), where each of them include a finer detail of information.

The data elements represented in this domain are those necessary to exchange complete information to give partners a complete common operating picture of maritime security. This domain is the result of extensive efforts to harmonize content from the Maritime Information Exchange Model 1.0 (MIEM) and bring it into conformance with the NIEM Naming and Design Rules.

Within this project, the NIEM-M schema is the largest. However, we will not use this schema directly but rather the IEPD as these are the elements that are used to share and transmit information. The NIEM is just a set of building blocks to be used according to one's own requirements.

This will also enable us to restrict the study of similarity to meaningful portions of this huge schema.

NIEM-M has five already defined IEPDs which are:

- Indicators and Notifications: Indicators are information used to inform or contribute to an analytical process. Notifications include warnings of a possible event and alerts about the execution of an event.
- Levels of Awareness: Defined and standardized levels characterizing how much is known about a vessel (and associated people, cargo, and infrastructure) at any time.
- Notice of Arrival: A 96-hour advance notice that all vessels inbound to US ports are required to submit, which lists vessel, crew, passenger, and cargo information.
- Position: A geospatial position, course, heading, speed, and status of a vessel at a given time. A series of position reports can be combined to produce track information.
- Vessel Information: Static vessel characteristics information, aka Vessel Tombstone Data.

Within this project, only the Position exchange and Notice of arrivals IEPD XML Schema Definition (XSD) schema will be used and compared with other available schema.

4.2 Naval Position Repository

The NPR aims at storing the contact reports of ships for either real time or deferred used. Ship identification is stored in one table while the complete set of contact reports is stored within another table.

The NPR schema is under an SQL format and consists of multiple tables where each of them describes a particular aspect of an Entity. In this project, similarity of this schema is compared to

the position IEPD and the MUSIC schema as the Electronic Port Clearance (EPC) schema is not sharing any application purpose with the NPR.

This schema is meant for use within a very small community. As such the names of different elements and vocabulary is less self-descriptive than for schemas aimed at effective information exchange.

4.3 Electronic Port Clearance - ISO 28005-2

The EPC schema is the specification of the messages to be exchanged for Electronic Port Clearance. It is composed of the following elements:

- Pre-arrival notification / general declaration;
- Cargo manifest notification;
- Ship's store notification;
- Crew / Passenger list;
- Dangerous goods manifest notification.

ISO 28005 contains the definition of core data elements for use in electronic port clearance (EPC) messages. It contains definitions of core data elements for electronic messaging between ships and shore in the areas of safety, security and marine operations. It does not define any structuring of messages or provide any guidance on what information is required for a particular purpose; it is rather a general data dictionary for safety, security or operation-related maritime information.

EPC does not normally include cargo clearance for import or export.

ISO 28005 will be linked to IMO's e-Navigation initiative through the International Hydrographic Office's S-100 specification.

Table 4.1: Description of ISO 28005-2 Data Elements

ISO 28005-2 Data element	**Description**
Agent	Contact information of ship's agent
CallPurpose	Purpose of call
CargoData	Cargo description list
CargoOverview	Brief description of cargo
Certificate (RegistryCertificate)	Certificate of registry

Table 4.1: Description of ISO 28005-2 Data Elements

ISO 28005-2 Data element	**Description**
Certificate	International ship security certificate
Company	Company name - IMO company ID. no.
ContactInfo	Company security officer information
CrewList	Crew list
CurrentShipSecurityLevel	Security level
DutiableCrewEffects	Crew effects list
EPCMessageHeader	Person / date / reporting system
ETA	Date and time of arrival
ETD	Date and time of departure
GrossTonnage	Gross tonnage
HasSecurityPlan	Approved security plan
InmarsatCallNumber	Inmarsat call number
LastPortOfCall	Last port of call
Location	Location where report is made
NameOfMaster	Name of master
NetTonnage	Net tonnage
NextPortOfCall	Next port of call
PassengerList	Passenger list
PeriodOfStay	Period of stay
PersonsOnboard	Number of crew and passengers

Table 4.1: Description of ISO 28005-2 Data Elements

ISO 28005-2 Data element	**Description**
PortCalls	Last 10 port calls
PortOfArrival	Port of arrival / position of ship in port
PortOfDeparture	Port of departure
RegistrationPort	Flag state
Remarks	Remarks
ShipID	Ship name / IMO number / call sign
ShipStatus	Course and speed / pilot on board
ShipStores	Ship stores list
ShipToShipActivityList	Most recent ship-to-ship activities
ShipContent	Ship type
VoyageDescription	Brief description of voyage/cargo
VoyageNumber	Voyage number
WasteDisposalRequirements	Waste and residue disposal requirements
WasteInformation	Waste and residue detailed information

4.4 MUSIC-TDP Track Database Schema

The database schema of the MUSIC-TDP is used to represent a track. It represents essentially a vessel's attributes and positional history. Its structure is closely related to a Java object with a naming convention to match the global system requirements it is used in.

The vocabulary used is typical of a military Command & Control (C2) environment by using the term Track instead Ship and/or Vessel, since it is not meant for MDA information only, as well as incorporating attributes specific to military used such as Force Code and Military Id. The content of the schema also reflects the design choices of the MUSIC Test Bed (MTB), which is assigning the track to a particular community.

Since this schema is solely designed to represent a vessel with its identification information and positional history, we will compare its similarity with the NPR and the IEPD-M Position IEPD schemas.

As this schema is not meant to be used outside a very tiny group of initiates, the name of the different information element is much less descriptive than in schemas required for effective information exchange between multiple organizations.

4.5 Summary

As shown, multiple schemas exist for the representation of MDA related information. Some are used within a more specific domain, such as track representation in a Command and Control System (CCS), while others, such as NIEM, aim to be applicable in a wide range of maritime applications where the exchange of information is necessary.

This difference in scope has a direct consequence on the naming convention and vocabulary and as such will have a direct effect on the quality of the schema similarity measures.

The multiple schemas available at this point, from different communities of interest and for different underlying requirements will enable us to study the effect of multiple aspects of schemas within the particular task of schema matching and similarity measurements. These findings are presented in the next section.

Part 5

Schema comparison using COMA 3.0

This section presents the steps required to compare schemas within COMA 3.0 and is organized as follow:

- Section 5.1, presents the installation procedures of COMA++ 3.0 along with the configuration of MySQL.
- Section 5.2, presents a brief introduction on the usage of COMA 3.0.
- Section 5.3 gives a detail description of the algorithms and measures of performance used along with the results obtained in matching the schema from the different source.
- Section 5.4 provides a method to add additional schemas in the comparison framework.
- Section 5.6 discusses the possibility of schema merging and source combination.
- Section 5.7 highlights the difficulties encountered and the lessons learned during the schema comparison.
- Section 5.8 serves as a summary for this section.

5.1 Software Installation

This section presents the details for the installation of the COMA 3.0 software. First, section 5.1.1 lists the steps to follow to correctly install and configure MySQL so that COMA 3.0 can use it as its underlying database. Then, section 5.1.2 presents the installation steps of COMA 3.0 for a Windows 7 OS. Note that COMA can be installed on Linux platforms as well.

5.1.1 MySQL Installation

COMA 3.0 relies on MySQL as its underlying database. As such, prior to the installation of COMA 3.0, one must install and configure MySQL. For Windows 7 systems, download the MySQL

installer[1] and follow the default instructions of the installation wizard[2].

Some configurations are necessary so that COMA 3.0 can create and modify entries in MySQL. Open a command line window and do the following:

```
mysql -u root -p
<enter the root database password>
CREATE USER 'comaUser';
SET PASSWORD FOR 'comaUser'@'localhost' = PASSWORD('comaPwd');
GRANT ALL PRIVILEGES ON *.* TO 'comaUser'@'localhost' WITH GRANT OPTION;
FLUSH PRIVILEGES;
```

where `comaUser` and `comaPwd` are the value of the respective property in the `coma.properties` file located in the `coma-gui` directory of COMA 3.0.

Another prerequisite is to have JAVA (version 1.6 or higher) installed[3]. Make sure that the environment variable `JAVA_HOME` points to your Java directory and that the PATH includes java binaries directory.

5.1.2 COMA 3.0 Installation, Configuration and Start

Now create a new directory in your home directory (for example, COMA 3.0) or in another location if you have the right privileges. Unpack the zip file under this directory. Start COMA 3.0 with the batch file coma.bat under the unpacked directory `coma-gui`. This file sets a new `CLASSPATH` environment variable including all required jar files.

The `coma.bat` script sets the maximal amount of memory allocated to COMA 3.0 at 1GB. Consider increasing this value if you experience *OutOfMemoryException* (*e.g.*, when matching large schemas).

Alternatively, you can run the *ant* command (on Linux environment for example but also available for Windows 7[4]) over the provided `build.xml` file to recompile and run COMA 3.0. If you do so, it will recompile everything before running the software.

5.2 Software Description

When the application starts, the graphical interface is displayed like the one presented in Fig 5.1.

The tool is easy to use although a bit of practice is necessary to get familiar with the different features. There are two tabs on the left:

[1] `http://dev.mysql.com/downloads/installer/`
[2] `http://blog.mclaughlinsoftware.com/2013/04/25/mysql-5-6-install-steps/`
[3] `https://www.java.com/en/download/manual.jsp?locale=en`
[4] `http://ant.apache.org`

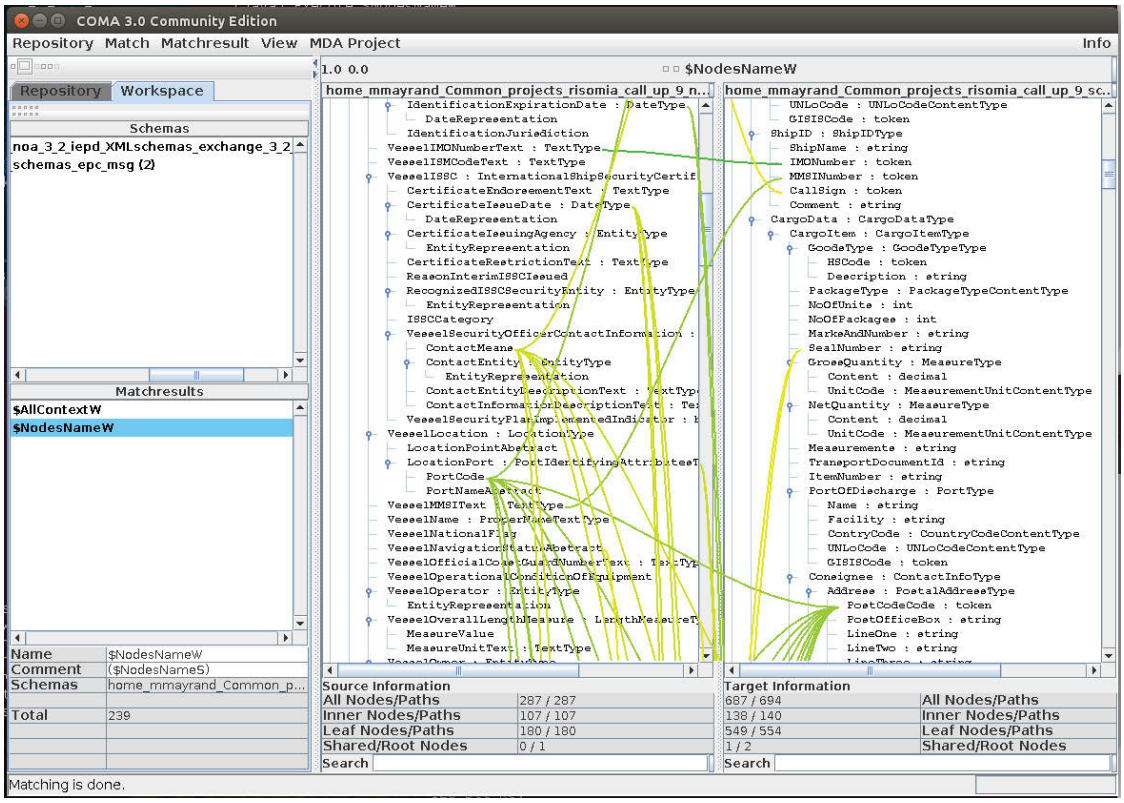

Figure 5.1: COMA graphical interface

1. the `Workspace` which is set in memory and will be erased at the end of the session.
2. the `Repository` which is localized in the MySQL database and will persist after the end of a session.

Schema files are loaded in the COMA database using the **Repository→Schema→Import File** menu. In order to load the schemas into the workspace, the user chooses and double clicks a schema listed in the Repository Tab. The first selection will be the source schema and the second selection will become the target schema. The **Repository** menu allows the deletion of any items in the database, either schemas or match results. If one double clicks a match result from the `Repository` tab, the associated schemas will be loaded in the workspace and the node relationships will be displayed as well (Fig 5.1. The menu **Repository→Instance** uses data instances (in opposition to schema) to enhance the matching results. Having only access to pure schemas, this feature was not used in our study.

The **Match→Execute Workflow** menu executes specific matching workflow algorithm. The result is displayed in the workspace, curving lines between nodes of the two schemas shows the relationship. By clicking on a starting node of a relationship, the value of the match is display. Using the third button of the mouse, the value can be manually modified or erased for a single relation or a multiple one. The matching algorithms use many parameters which can be modified with the menu **Match→Workflow Variables**.

The **MatchResult** menu provides the functionalities to move the matching results from the workspace to the database and vice-versa. Another set of command allows comparing and merging matching results.

The **View** menu displays different information and statistics related to the schemas (see Fig 5.2) and matching results (see Fig 5.3).

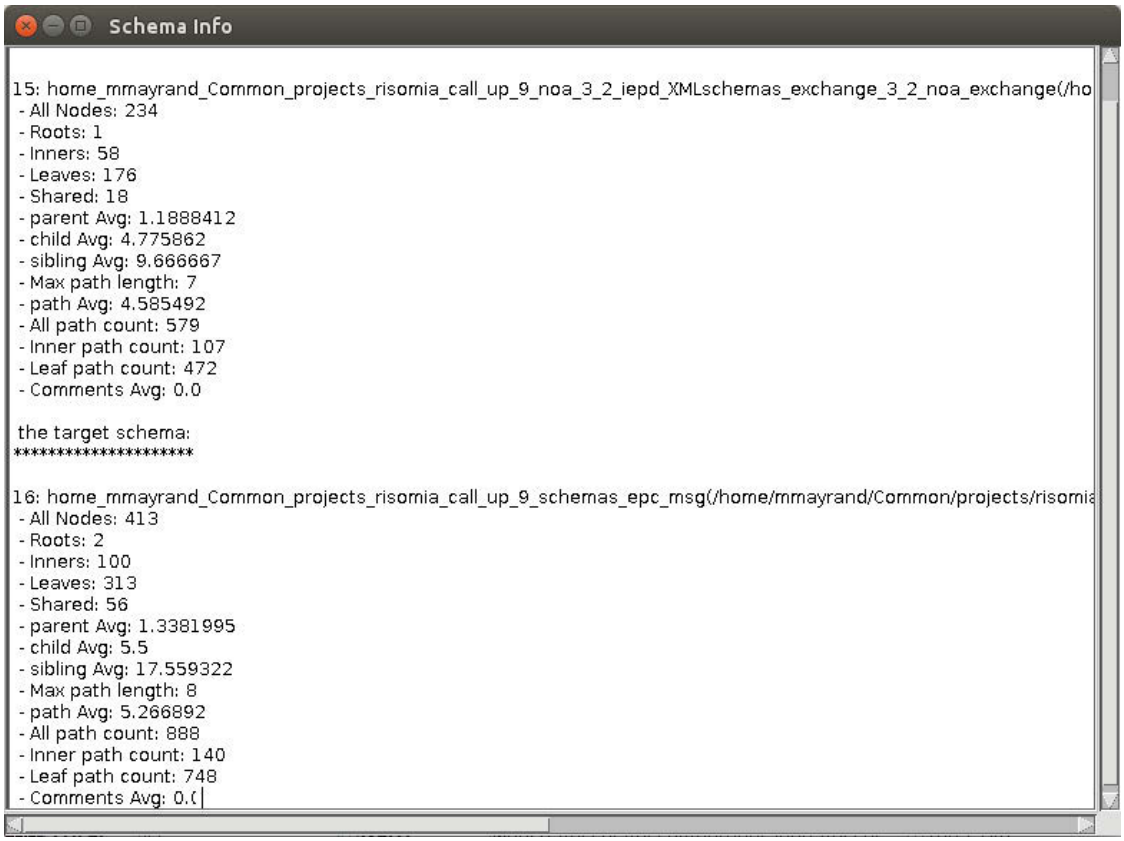

Figure 5.2: Information and statistics on the source and target schemas

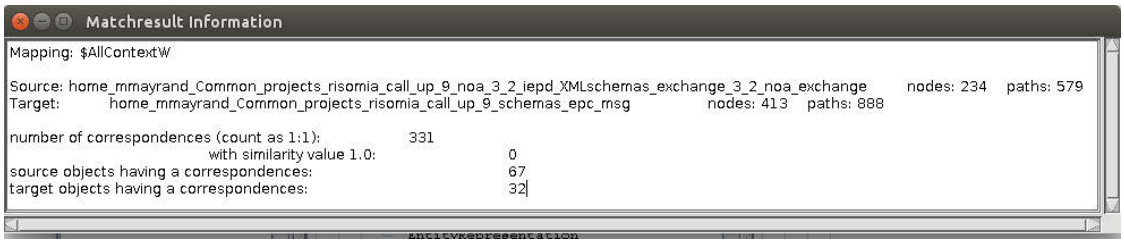

Figure 5.3: Information and statistics on a given matching result

The **MDA Project** menu provides functionalities added specifically for this project and is described in the following sections.

For more detailed descriptions of COMA, please see the following references: [22], [23], [17] and [24].

5.3 Schema Similarity and Matching

In order to assess the similarity of schema elements, the following information are usually used as input [23]:

1. Schema information: The input schemas already provide various kinds of information, such as element names, description, data types, schema structure and other relationships between elements, etc., which can be examined to characterize and compare the semantics of schema elements.
2. Instance data: In many applications, such as data integration and transformation, instance data is available for the schemas to be matched and can also be exploited to characterize the content and semantics of schema elements.
3. Auxiliary information: This category comprises all other kinds of information which can be exploited to detect similarity between schema elements. For example, we can look up semantic relationships like synonymy and hypernymy between element names in existing (general-purpose or domain-specific) dictionaries and thesauri.

In this study, only schemas and auxiliary information (synonyms and acronyms/abbreviations tables which were manually created) are available for matching the elements of the schemas when transformed into graphs. In COMA, the synonyms and abbreviations can be viewed in the **Repository→Auxiliary Info** menu. See for examples Fig 5.4 and Fig 5.5.

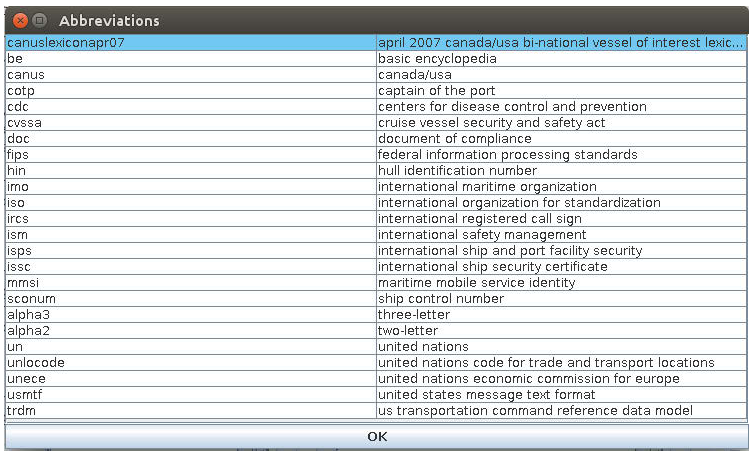

Figure 5.4: List of abbreviations/acronyms used by the matching algorithm.

Table 5.1 summarize the characteristics of the schemas, as graphs, used within this evaluation.

Figure 5.6 represents the general architecture of schema matching [23].

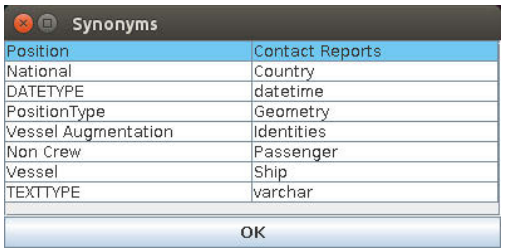

Figure 5.5: List of synonyms used by the matching algorithm

Name	Type	Nodes/Paths	Root/Inner/Leaf/Shared Nodes	Max / Average Path Length
NIEM-M NOA	XSD	287 / 287	1/ 107 / 180 /0	7 / 4.2
NIEM-M Position	XSD	100 / 100	1 / 33 / 67 / 0	7 / 4.6
EPC	XSD	687 / 694	2 / 138 / 549 / 1	8 / 5
NPR	SQL	143 / 143	19 / 19 / 124 / 0	2 / 1.9
MtbTrack (MUSIC)	XSD	33 / 33	2 / 3 / 30 / 0	3 / 2.6

Table 5.1: Graph characteristics of the schemas used within this evaluation

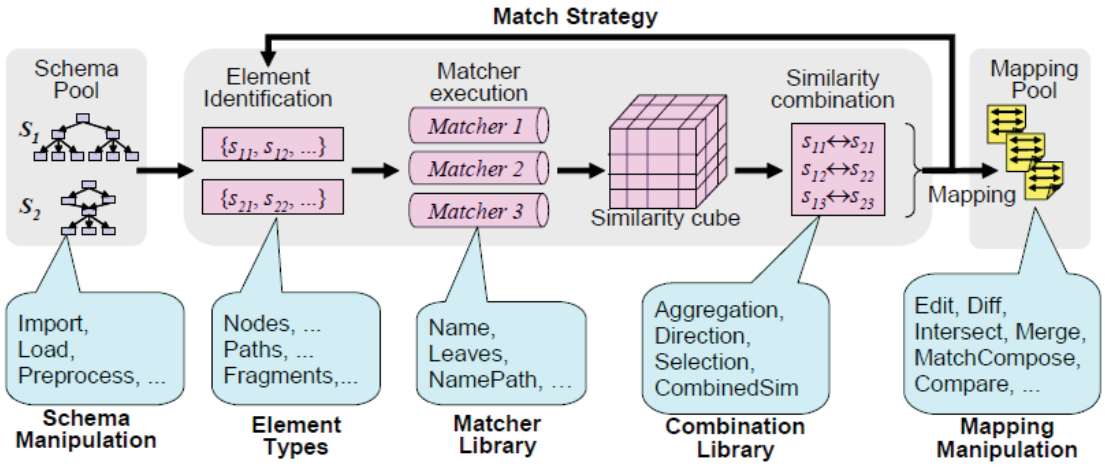

Figure 5.6: COMA schema matching process

The internal terminology for schema matching within COMA 3.0 is called a workflow. Workflows are composed of one or more match strategies. As shown on figure 5.6, a match strategy is composed of

- The determination of relevant graph elements for matching (graph's nodes, path or attributes

such as the names).

- The execution of one or several complex matchers.
- The combination of the complex matchers output if more than one matcher was used in the strategy.

As such, a strategy realizes the sequence matching using one or more complex matchers or strategies and the combination of their outputs. Three different types of workflow can be used to assess the similarity of the schema and ultimately match its different elements:

- Type 1: a single complex strategy;
- Type 2: several complex strategies with optional combination of matching results (intersection, difference, merge);
- Type 3: complex strategy followed by another one where the output of the first is the input of the second.

While comparing schemas, there is high probability that source element can have more than one matches in the target schema. One match has to be ranked the best manually or automatically, for the mapping purpose [20]. In COMA 3.0, the automatic selection is made by using a selection method such as maximum similarity or maximum N candidates.

However, it was noted in [25] that the aggregation function entails several major drawbacks: computing all match algorithms decreases performance time and it can negatively influence the matching quality.

5.3.1 Overview of the similarity measures

The similarity measures available in COMA 3.0 can be divided in two groups. The first one is for the matching of short strings while the second is meant to perform better over long strings. As such, and depending of the name convention of the schema one wants to compare, some will be more suitable than others. This section presents an overview of the similarity measures available.

For short string similarity measures, the following similarity measures are available:

1. Edit distance: edit distance is a way of quantifying how dissimilar two strings (*e.g.*, words) are to one another by counting the minimum number of operations required to transform one string into the other.
2. N-Gram (Tri-gram): N-grams are typically used in approximate string matching by sliding a window of length N over the characters of a string to create a number of N length grams for matching. A match is then rated as number of N-gram matches within the second string over possible N-grams. In the case of COMA 3.0, the length is three, as such we call it a tri-gram.

3. Jaro-Winkler distance: a type of string edit distance, and was developed in the area of record linkage. Jaro-Winkler distance uses a prefix scale p which gives more favourable ratings to strings that match from the beginning for a set prefix length. Details can be found here [26].
4. Levenshtein distance: The Levenshtein distance between two strings is defined as the minimum number of edits needed to transform one string into the other, with the allowable edit operations being insertion, deletion, or substitution of a single character.
5. Equality: which is the strict correspondence between two strings.

For long string similarity measures, the following similarity measures are available:

1. Jaccard index, also known as the Jaccard similarity coefficient is a statistic used for comparing the similarity and diversity of sample sets. It is defined as the size of the intersection of the two documents (or strings) divided by the size of the union of the two documents (or strings).
2. Cosine similarity: Cosine similarity is a measure of similarity between two vectors of an inner product space that measures the cosine of the angle between them. For text matching, the attribute vectors A and B are usually the term frequency vectors of the documents. The cosine similarity can be seen as a method of normalizing document length during comparison. This does not include weighting of the words by term frequency-inverse document frequency (tf-idf), but in order to use tf-idf, you need to have a reasonably large corpus from which to estimate tf-idf weights.
3. TF-IDF: short for term frequency-inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in information retrieval and text mining.

5.3.2 Measures of performance

In schema matching and similarity evaluation, the most challenging research domain is the match quality evaluation. Measures like precision and recall (described below) have been borrowed from information retrieval domain. These metrics have been customized to quantify the quality of schema matching but still require a lot of work [20].

In order to evaluate the performance of automatic match strategies, we will use the measures proposed in [23]. *Precision* and *Recall*, from the Information Retrieval field, can be computed using the False negatives (A, the relative complement of the suggested correspondences), the True positives (B, the intersection between the real and suggested correspondences) and the False positives(C, the relative complement of the real correspondences) (see figure 5.7).

Precision, which reflects the share of real correspondences among all found ones, is defined as:

$$Precision = \frac{|B|}{|B| + |C|} \tag{5.1}$$

where $|B|$ represent the cardinality of the set B. Precision value run from 0 to 1, the latest being the goal to achieve.

Recall, on the other hand, specifies the share of real correspondences that is found:

$$Recall = \frac{|B|}{|A| + |B|} \tag{5.2}$$

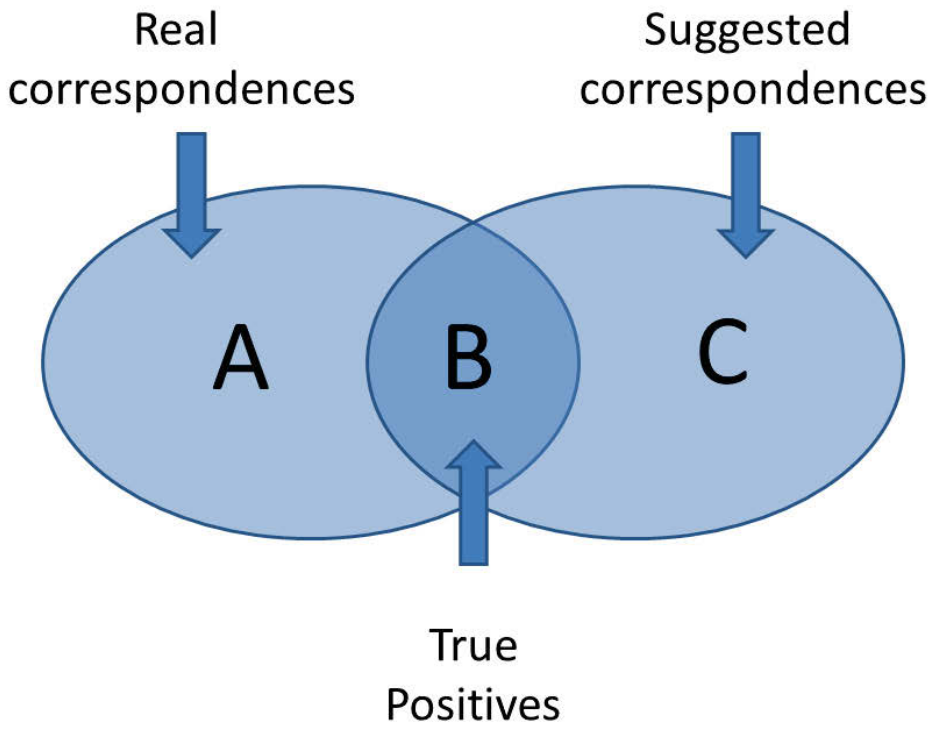

Figure 5.7: Real and suggested correspondences

Recall can easily be maximized at the expense of a poor *Precision* by returning as many correspondences as possible, for example, the cross product of two input schemas. On the other side, a high *Precision* can be achieved at the expense of a poor Recall by returning only few but correct correspondences. *Recall* value run from 0 to 1, the latest being the goal to achieve.

It is necessary to consider both measures or better yet, a combined measure of these two. *F-measure* represents the harmonic mean of *Precision* and *Recall* and is widely used in Information Retrieval.

$$F\text{-}measure = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{5.3}$$

F-measure value run from 0 to 1, the latest being the best value.

Accuracy (sometimes called *Overall*) was developed specifically in the schema matching context

and embodies the idea to quantify the post-match effort needed for adding false negatives and removing false positives.

$$Accuracy = Recall * \left(2 - \frac{1}{Precision}\right) \tag{5.4}$$

Its value run from $-\infty$ to 1, the latest being the best value.

Compactness can be measured as the number of correspondences [27] either as an absolute number or as the ratio of the expected number of matches and the measured number of matches:

$$Compactness = \frac{|A| + |B|}{|B| + |C|} \tag{5.5}$$

Compactness value can vary between 0 and something higher than 1 but the optimal value is 1.

These different measures will be used to quantify the performance of individual and combined matchers in order to compose a matching strategy with good performance for the maritime domain.

5.3.3 Algorithms performance in COMA++

Tables 5.2 and 5.3 present the available matchers within COMA 3.0.

Table 5.2: Hybrid Matchers in COMA 3.0

Technique	Matcher Name	Schema Info	Auxiliary Info	Default Matcher
String Matching	Levenstein	Elements Names	-	-
	NGram	Elements Names	-	-
	Jaro-Winkler	Elements Names	-	-
	EditDistance	Elements Names	-	-
Reuse-oriented	Synonym	Elements Names	External Dictionaries	-
	Taxonomy	Elements Names	Domain taxonomy	-
	Type	Data Types	Compatibility table	-
	Reuse	-	Previous Match Results	-
Structure	Statistics	Structural statistics	-	-

We have modified the software in order to evaluate these matchers in batch mode, apply all the possible combination of matcher and match selection, with vary threshold for matches acceptance with regards to the similarity measures.

As the semantic heterogeneity is present due to the different naming conventions retained by individual schema, a list of synonyms was made. COMA 3.0 was modified to handle these synonyms

Table 5.3: Combined Matchers in COMA 3.0

Technique	Matcher Name	Schema Info	Auxiliary Info	Default Matcher
Element-Level	Name	Elements Names	-	Synonym, Trigram
	Comment	Elements Comments	-	Synonym, Trigram
	NameType	Names + Data Types	-	Name, Type
Structure-Level	NameStat	Name + Statistics	-	Name,Statistics
	NamePath	Names + Path	-	Name
	Children	Child Elements	-	NameType
	Leaves	Leaf Elements	-	NameType
	Parents	Parent Elements	-	Leaves
	Siblings	Sibling Elements	-	Leaves

in a better way. Prior to the modification, one would have had to use the complete node names as synonym. As an example, one would have had to write in the synonyms file *VesselNameText* and *ShipName* instead of only writing *Vessel* and *Ship* to replace all the instance of *Ship* by *Vessel* within all the node names of the schemas to compare. This modification makes it easy to work with complex naming conventions.

Semantic heterogeneity is also present in the data types considered by different schemas. Some are using widely adopted types, such as *string*, while others more complex are using internally defined types, such as *TextType* in the case of NIEM (which ultimately is a string but the software do not go deeper than the node to evaluate the type). These decisions that were made at the schema design step have a large impact in the schema similarity evaluation. This can be partly resolved in COMA 3.0 by assigning a synonym to the data type but in some cases it is simply not possible.

The list of synonyms used for all the dual schema matching strategies are provided in Appendix B.

5.3.4 Difficulties of automatic schema matching

In order to understand the results obtained and highlight the difficulties of automatic schema matching and similarity assessment, consider the following three elements representing the same concept across different schemas, which is the name of the ship:

- *Notice.Vessel.VesselAugmentation.VesselNameText* from the Notice of Arrival Exchange IEPD

of NIEM:

Listing 5.1: VesselNameText Element from NIEM

```
<xsd:element name="Notice" type="noaex:NoticeType">
  <xsd:element name="Vessel" type="mda:VesselType">
    <xsd:element name="VesselAugmentation" type="m:VesselAugmentationType"
        substitutionGroup="s:Augmentation">
      <xsd:element name="VesselName" type="nc:ProperNameTextType" nillable="true
          ">
        <xsd:annotation>
          <xsd:documentation>The name of a vessel.</xsd:documentation>
        </xsd:annotation>
      </xsd:element>
    </xsd:element>
  </xsd:element>
</xsd:element>
```

- *EPCMesseage.EPCRequestBody.ShipID.ShipName* from the EPC schema:

Listing 5.2: Ship Name Element from EPC

```
<xs:element name="EPCMessage">
  <xs:element name="EPCRequestBody" type="epc:EPCRequestBodyType" minOccurs="0">
    <xs:element name="ShipID" type="epc:ShipIDType" minOccurs="0">
      <xs:element name="ShipName" type="epc:string" minOccurs="0" />
    </xs:element>
  </xs:element>
</xs:element>
```

- *MtbTrack.Attributes.Name* from the MUSIC-TDP schemas:

Listing 5.3: Track Name Element from MUSIC

```
<xs:element name="MtbTrack">
  <xs:element name="Attributes" type="AttributesType" minOccurs="0">
    <xs:element name="Name" type="xs:string" nillable="false" />
  </xs:element>
</xs:element>
```

Note that in the NPR schema this concept in defined as *dbo.Identities.IdentityName* with a *varchar* data type.

As mentioned earlier in section 5.3, the available information to achieve our goal are the schema information (element names, description, data types, structure (*i.e.* node-to-node path), etc.) and auxiliary information (synonyms and abbreviations in our case).

First, consider the element names (*VesselName*, *ShipName* and *Name*). Similarity measures that are applied in this case are based on String analysis. Without the support of auxiliary information

(*Ship* is as a synonym of *Vessel*), no algorithm would yield a satisfactory similarity measure. And even with the synonym, *Name* would not be matched to the other two unless we explicitly state in the synonym list that *Name* is a synonym of *VesselName* or if we lower the similarity threshold to accept the match which will lead to an increasing number of false matches.

In this first case, long to short string comparison is problematic. Complex naming conventions such as in the NIEM domain when compared with simpler models such as the one from MUSIC, will yield poor matching results and as such poor overall similarity for both schemas.

Second, let's add the parent nodes (*i.e.*, the graph structure above our leaf node) in our analysis. NIEM being a highly expressive schema and following the naming convention of the NIEM-Core is using the term *VesselAugmentation*. On the other hand, within the MUSIC-TDP schema, the language is closely related to the C2 community and *Name* is set as an *Attributes* of the vessel. Finally, the EPC puts it under *ShipID*. Here again, without the use of auxiliary information, no matches can be found. However, setting all these terms to be synonyms will yield a correct match result between the concepts of name.

As stated earlier, one can also rely on the data types of the nodes in order to increase the similarity values between the said nodes. However, if we have a look at the data types used by the different schemas for a simple element such as the names, we can see that, with the exception of the MUSIC-TDP schema which relies on the base type *xs:string*, the other two are using user-defined data types (*nc:ProperNameTextType* for NIEM and *epc:string* for EPC). Of course going a little deeper into the schemas, one can see that these are ultimately *xs:string*, but the graph-matching software will generally not find it. Here again, relying on auxiliary information to define synonyms between data types is necessary.

The description of the element is another option to increase quality of the match as the lexical description of a concept could be potentially closer than the language used to model it. However, it was observed that the description is often absent from the schema. And even if it is present, the vocabulary used is not the same and is highly dependent of the community of interest to which the schema is destined.

As demonstrated, unless the schemas to match are closely related to one another (evolution in time of a schema design for instance), the use of auxiliary information is mandatory to achieve good graph matching and similarity. However, the task of manually providing this information requires an in depth analysis of each schema one wants to match, which in the case of very different schemas is equivalent of performing manual investigation and matching of the different graph component.

To summarize, difficulties for schema matching and similarity assessment are posed by the high expressive power and versatility of modern schema languages, in particular user-defined types (*nc:TextType* in NIEM for instance) and classes, component reuse capabilities, and support for distributed schemas and namespaces.

A potential solution to these difficulties would be to use instance data about the same entity described under different schemas. This would be of great benefit in increasing match quality. Equality between instance data of different schema could be used to automatically derive the necessary auxiliary information. This would reduce the amount of manual analysis required. As noted in [20], if the schema information is very limited or not available, instance data can be

used to create a representation of the data and even if the schema is available, data instances can augment the schema matching by giving more insight about the schema element semantics.

5.4 Multiple schema comparison

This section describes the method for adding a new schema in the evaluation. Generally speaking, schema matching can be categorized into two types of problems depending upon the input [20]:

- two large size schemas (with up to thousands of nodes);
- a large set of schemas (with possibly hundreds of schemas and thousands of nodes).

Tools available today, including COMA 3.0, can be used for applications which require matching of two schemas. However as mentioned in [20], it was demonstrated in [17] that with some modification to the available tools/infrastructures, a solution to the matching of large set of schemas can be achieved.

In [23], it was noted that many schemas to be matched are often very similar to each other or to previously matched schemas. Likewise, in schema evolution, different versions of a schema may substantially overlap and only exhibit small changes. However, as shown in section 5.3.4, it was found that even when representing the same entities, the naming convention particular to each schema and the community who produced them or to which it is aimed, have a large impact on the similarity between two schemas.

Taking into account the difficulties related to schema matching identified earlier, it is nevertheless possible to add another schema in our analysis to transform it in a N-ary schema matching and similarity assessment. For instance, three schemas in this project are used to represent a ship along with its position information, namely:

- The NPR database structure;
- The *MtbTrack* schema from the MUSIC-TDP;
- The National Information Exchange Model - Maritime (NIEM-M) *Position Exchange* IEPD schema.

Consider the scenario in which one first compares the NIEM-M *Position Exchange* IEPD to the *MtbTrack* schema and then wishes to include the NPR Structured Query Language (SQL)-structure into the similarity analysis.

First, the results from the *Position Exchange* to *MtbTrack* are already available (*Mapping AB*). One could compare *Position Exchange* to NPR (*Mapping AC*) and then reuse the mapping AB to derive mapping BC automatically. Of course, these might be incomplete.

COMA 3.0 enables its users to perform a *reuse approach* to derive new mappings from existing ones. It evaluates a mapping path consisting of two or more mappings, successively sharing a common schema, to derive a new *Mapping BC*.

To summarize, one would have to perform the following tasks:

1. Load schemas in COMA 3.0.
2. Define synonyms between schemas A and B and schemas A and C.
3. Apply selected matching strategy to schemas A and B as well as schemas A and C.
4. Apply the available reuse strategy to find the correspondence between B and C.

In this case, the similarity value between the two is called transitive similarity. As noted by [23], a common approach to determine the transitive similarity, similarity between B and C in our example, is to multiply the individual similarity values. However, if two mappings have similarity of 0.7 and 0.8 respectively, this would lead to a transitive similarity of 0.56, which is not what one would intuitively expect. [23] therefore proposed to use operations such as *Average*, *Max* and *Min* in order to derive the transitive similarity values.

The steps proposed above might however lead to missing correspondence between B and C and as such perhaps a better solution would be to apply the matcher directly between B and C and merge the output with the output of the reuse strategy.

If more schemas and mappings are available, for example, as generated and collected over time, the potential for reuse increases with more possible mapping paths. However, they are also often longer involving multiple intermediary schemas, each of which in turn adds to the likelihood of false/missing correspondences and causes overhead for execution time. In order to avoid long mapping paths, [23] supports the utilization of a so-called pivot schema, *e.g.*, a standard schema or ontology in a domain, acting as the central schema, against which all new schemas are first matched. This approach is mainly motivated by the use of global schemas in data integration approaches, such as data warehousing and mediation, in which the schemas of new data sources to be integrated are uniformly matched against and merged to the global schema [23].

5.5 Maritime Schema comparison results

In this section, we present the comparison results between the different maritime schemas.

5.5.1 Creation of a reference matching set or Ground Truth

The first step is creating a reference matching set (Ground Truth) between two schemas. This is achieved by manually browsing each schema for matches that make sense. Then, for each match you need to create a triplet `source node, target node, similarity value`. The process can

be tidious but fortunately, COMA provides a way to manually create a set of triplets using its interface:

1. First, select the source schema by double clicking a selection in the Repository Tab and then, do the same for selecting the target schema. Remember that the order is important in COMA, matching is from the source to the target.

2. Create any MatchResult in the workspace by selecting one of the matching algorithm, for example `Match -> Execute Workflow -> $AllContextW` (do not save it to the repository, it will be done later). The produced MatchResult will serve as a template for creating the GroundTruth data.

3. When a MatchResult is created, you should see links between the two schemas (if not, you may have to scroll down to see them or maybe, no match were found).

4. The goal is to edit a MatchResult in order to create a Reference/GroundTruth/Expected MatchResult set. For creating a relation between the two schemas, select the related items in both schema by clicking once on each node, then using the third mouse button on one of the nodes, select `Create Correspondence`. This will create a link with a similarity measure of 1.0 (dark green line). If the link already exists but the value of the similarity is lower than 1.0, using the third button, select `Set Highest Similarity value`.

5. Repeat for all the links you want to create.

6. To remove links which do not belong in the Reference Match, select the node on the left where the links originate from and with the third button, select `Delete All Correspondences for this node`.

7. When completed, selecting the match result in the workspace, rename the match by clicking the Name input box, enter a detailed name, and save it by selecting the menu `MatchResult -> Save`.

8. You could also export your ground truth in a RDF file using the menu `Repository -> MatchResults -> Export RDF`. Copy/paste the content of the window into a file editor and save.

9. Repeat the previous steps for each pair of schema you want to compare.

Note that creating a ground truth or expected set introduce by definition a human subjectivity that can be disputed from another expert with a different view. The expected sets created for this study may be not perfect but they should be sufficient to achieve the objectives of this study. Another observation regarding the input data used for comparison is that the four maritime schemas have small common intersections between them. The common areas consist mostly of vessel identification information, cinematics and embarkation/debarkation crew/passenger information. Again, this increases the level of difficulty but should not prevent achieving our goals.

The RDF files for the ground truth for each pair are reproduced in the appendix A.

Nb of Nodes (Source)	Nb of Nodes (Target)	Ground Truth
100 (NIEM:Position)	33 (MtbTrack)	8
100 (NIEM:Position)	143 (NPR)	8
287 (NIEM:NOA)	687 (EPC)	21

Table 5.4: Proportion of ground truth match

5.5.2 Creation of synonym map

For each schema pair comparison, create a synonym file where there is one synonym pair per line, the synonyms within a pair are separated by a single comma. The synonyms can be a word or a group of words separated by spaces.

5.5.3 Batch running instructions

The set of instructions for executing a batch run are:

1. Edit the `coma-gui/coma.properties` file and select the name of the GroundTruth reference match in the database. In the same file, select the proper synonym and abbreviation files which must be located in the `coma-gui` directory;
2. Select the source schema by double clicking one of the schemas stored in the Repository tab. Remember that the source and target schemas of the ground truth reference match must coincide with the source and target schemas selected by the user.
3. Select the target schema by double clicking one of the schemas stored in the Repository tab;
4. In the menu `MDA Project`, select the command `Run Match Algorithms in batch`

When initiated, the GUI will not accept other commands until the completion of the batch command. The duration of a batch may depends on many factors: length of the schemas, configurable parameters, numbers of combination of matching algorithms, etc. For this study, a typical batch runs between a half-hour to four hours.

The output results of the batch mode are written in a CSV file named after the two schemas used for the analysis. The file is generated in the `coma-gui` directory. For speeding up the analysis, we created a Python script `analysis_csv.py` to extract from the large result file the most interesting results which are the lines with the similitude measure values closest to one. Since we have three different similitude measures, three subsets of results are extracted and written in a text file bearing the same file name as the input but with and additional `_analysis` extension.

```
python analysis_csv.py position-exchange_xsd-compare-npr3_tables_sql.csv
```

In the following subsections, we present the results extracted from the batch mode execution with the optimal (closest to 1) similitude measures (in bold font in the table). Each run was executed with an empty set of synonyms and a custom set of synonyms. In all cases, the same abbreviation file was used (Fig. 5.4)

5.5.4 Matching results between NOA with EPC

In this matching experiment, we compared in batch mode using multiple matching algorithm the Notice of Arrival from NIEM (source schema) with the Electronic Port Clearance (target schema). These two schemas are the two biggest maritime schemas compared in this study. The ground truth used for defining the `Expected` set is presented in section B.1.

Using synonyms	Expected (A+B)	Intersect (B)	Extracted (B+C)	F-Measure	Accuracy	Compactness	Best strategy/matchers
No	41	11	66	**0.206**	-1.073	0.621	NameCM-NameStatCM-SiblingsCM
No	41	4	6	0.170	**0.049**	6.833	NameCM-ChildrenCM-NameStatCM-SiblingsCM
No	41	8	41	0.195	-0.610	**1.000**	NameCM-NameTypeCM-SiblingsCM-ParentsCM
Yes	41	11	66	**0.206**	-1.073	0.621	NameCM-SiblingsCM-NameStatCMM
Yes	41	4	6	0.170	**0.049**	6.833	ChildrenCM-NameCM-SiblingsCM-NameStatCM
Yes	41	8	41	0.195	-0.610	**1.000**	NameCM-NameTypeCM-SiblingsCM-ParentsCM

Table 5.5: Similarity measures for NOA/EPC match

Table 5.5 shows that the number of perfect matches is low in comparison with the 41 matches expected. See Section 5.3.3 for a description of the last column (best strategy/matcher). Also, a lot of false positives is produced and the addition of synonyms did not improve the matching results.

5.5.5 Matching results between NIEM:Position and NPR

In this matching experiment, we compared in batch mode using multiple matching algorithm the Positional message from NIEM (source schema) with NPR (target schema). The ground truth used for defining the `Expected` set is presented in section B.2. Table 5.6 shows some good matching but

Using synonyms	Expected (A+B)	Intersect (B)	Extracted (B+C)	F-Measure	Accuracy	Compactness	Best strategy/matchers
No	8	5	35	**0.233**	-3.125	0.229	SiblingsCM-NameCM-ParentsCM
No	8	0	1	0.000	**-0.125**	8.000	NameTypeCM-NameStatCM-SiblingsCM
No	8	0	8	0.000	-1.000	**1.000**	NameTypeCM-NameStatCM-ChildrenCM
Yes	8	6	24	**0.375**	-1.500	0.333	SiblingsCM-ParentsCM-NameTypeCM
Yes	8	0	1	0.000	**-0.125**	8.000	ParentsCM-NameCM
Yes	8	1	9	0.118	-0.875	**0.889**	NameStatCM-ParentsCM

Table 5.6: Similarity measures for POS/NPR match

the price to pay is a certain amount of false positives. The introduction of synonyms in this case seems to lower the number of false positives and slightly increase the number of good matches.

5.5.6 Matching results between NIEM:Position and MtbTrack

In this matching experiment, we compared in batch mode using multiple matching algorithm the Positional message from NIEM (source schema) with the MTB track information (target schema). The ground truth used for defining the `Expected` set is presented in section B.3. Table B.3 shows

Using synonyms	Expected (A+B)	Intersect (B)	Extracted (B+C)	F-Measure	Accuracy	Compactness	Best strategy/matchers
No	8	5	5	0.769	0.625	1.600	SiblingsCM-NameStatCM-NameCM
No	8	5	8	0.625	0.250	1.000	SiblingsCM-ChildrenCM-NameCM
Yes	8	5	5	0.769	0.625	1.600	NameCM-NameStatCM-SiblingsCM
Yes	8	5	8	0.625	0.250	1.000	NameCM-ChildrenCM-SiblingsCM

Table 5.7: Similarity measures for POS/MtbTrack match

the best match results of this study. In this case, the number of false positives is quite low. The fact that two out of three similitude measures are optimized at the same time is a contributing feature.

5.6 Schema Combination and Grand Graph Construction

Combining multiple schemas into a unified view is called schema merging. The task becomes particularly challenging when the schemas are highly heterogeneous and autonomous. Classical data integration systems rely on a mediated schema created by human experts through an intensive design process [28].

Consider the scenario of merging multiple data sources to create a single unified query interface. Since the data sources are independently developed, their extensions, *i.e.*, explicitly stored data, usually do not conform to any inter-schema logical constraints [28].

The COMA 3.0 capabilities to perform sub-graph selection and merging are quite limited if nonexistent. It is however claimed by the creators that some of these functionalities are available in a commercial version of the software, for which no observation could be made. A deeper investigation into [23] revealed that a capability to automatically merge graphs was present. A match result of paths was transformed into one of nodes by discarding all ascendants in the paths and retaining only the nodes at the deepest level. The nodes of the target schema, which do not have a match candidate, are successively added to the source schema. It was a rather simplistic merge where within the source schema, structure relationships were added between them and the matching nodes of their parents, between them and the matching nodes of their children, and retain all relationships among them.

The source code to perform this merge is available partly in the actual software and within [23].

Few function definitions are missing but they could be easily reintroduced in the software.

As for a selective combination to create a grand graph, these capabilities are not available in COMA 3.0. However, with the experience acquired by playing in the source code of the software, it appears that this functionality could be easily added to the current build as the interface already allow for node to node selection and core algorithms and functionalities to achieve schema merging (or combination) are already present.

Now, let's consider that a great graph has been constructed and that one would want to populate it with instance data. As such, the data from different sources are effectively mixed but retain the source from which the information was taken as this might have an importance, as an example, for further processing or evaluation of information quality.

Conceptually, graphs are composed of nodes and vertexes. A node, which would effectively contain the instance data, can be populated with any number and type of attributes. As such, an attribute that represents the source of information can be easily added to any graph node.

In reality, on the programmatic point of view, it is easy to add any attribute to a node of a graph. If one examines the COMA 3.0 Java source code, or any graph-based software, he/she would see that the graph structure employed to store an Object, which class is an Element (similar to an XML element). This has the advantage of being able to assign any attribute to it, and within COMA 3.0, the source identification is already a property which can be transferred when two or more graphs are merged.

Simple modifications to the libraries of COMA 3.0 would enable one either to use the existing attributes or define new ones to keep and retrieve the provenance of the data or any other pertaining information based on the requirements to fulfill. Also, most graph libraries, if not all, already enable the user to fill any nodes and vertex with any information he/she deems necessary.

Figure 5.8 represents the concept of multisource information as graph with the source ID as node's attribute.

As single piece of information is represented by a node within a graph, even simple information, such as the vessel's name, can be related to its source, even if it has been incorporated in a sub-graph representing the information structure of a different source as root, parents and leafs nodes share the exact same internal structure.

One could also envisage an information exchange scenario where the graph and its instance data needs to be exported as an XML file. This would require the creation of new XSD files, modifying the complex type definition of some element but would ultimately be based on the initial available schemas.

5.7 Problems encountered

Several problems were encountered while using COMA 3.0 to measure schema similarity. These are summarized below.

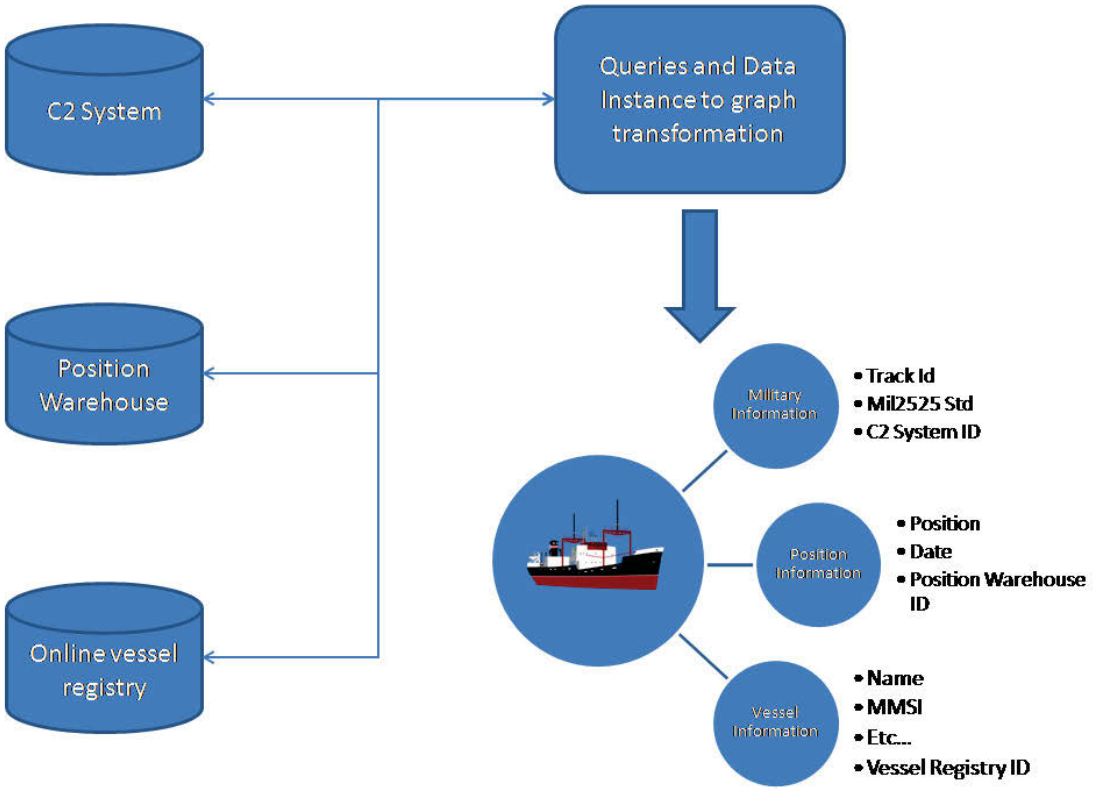

Figure 5.8: Conceptual representation of multisource information as graph with the source ID

- First, the parsing of the XSD files, in which the most complex schemas are stored, was deficient. As such, a new parser had to be implemented to take into account the import and include flags of the XSD files so that complex type definition could be resolved and imported within the internal graph structure of the software.
- The software, despite being quite flexible, offers no easy way to create and evaluate custom schema matching algorithm. As such, the availability of the source code made it possible to examine the processes and algorithm as well as the available functions and to design and implement additional functionalities to automatically create and evaluate schema matching strategy.
- Short names versus ontology-like schema matching yield poor performance unless extensive auxiliary information is provided. As demonstrated in section 5.3.4, the importance of using a common or close vocabulary for similarity measurement is of tremendous importance to achieve a high level of precision. The definition of an appropriate synonyms list demands a lot of manual effort and analysis of both schemas to be compared especially if the schemas are using a highly expressive language and were developed for a different target audience, even if they represent the same underlying concept.
- The synonyms that the user provided to the system via the synonym list were badly applied

to the node's name. Complete node names had to be provided. In order to solve this problem, modifications were made so that partial name or only part of a string would be substituted prior to similarity evaluation.

- The export feature of Microsoft SQL Server produced an SQL file containing square brackets and parenthesis. These characters were included when COMA was loading the SQL file. The special characters were removed manually in the database in other to prevent odd behaviour with COMA algorithms.
- There is limitations in schema matching. Take for example the term `Contact`. It means two different things completely depending of the context. For MUSIC, contact refered to a position container part of a track. In all the other schemas, contact refered to a person associated with a port, a ship or an organisation for communication purpose.When semantic is identical, it takes a very sophisticated algorithm for distinguishing between the two and declaring that it is not a match. This level of sophistication has not been achieved so far.

5.8 Summary

This section presented the comparison of MDA-related schemas with COMA 3.0. Some modifications were required to the COMA 3.0 XSD parser to correctly load XSD files with a lot of dependency but it the end, it was possible to load XSD and SQL schemas and compare them.

Multiple matching strategy were automatically run and evaluated using measures of performance typically used within the available literature. The comparisons that were performed highlighted many difficulties for an automatic schema matching and similarity measurements. Notably, the semantic heterogeneity is one of the greatest problem as different groups or communities have a different preferred language and hierarchical concept to express the same concept.

As it was mentioned by several authors, along with the semantic heterogeneity, the variety in data types, that can be user-defined adds an extra layer of complexity to the problem. These problems can be solved by the definition of a common language, a synonym list in COMA 3.0, but this requires a lot of manual effort and an in-depth analysis of the schemas to be merged. One should envisage this task in a long term perspective, we this is done once and the evolution in time of the multiple schemas can then be compared much more easily.

The step after comparing a schema is normally to take into account the strength and weaknesses of each schema and perhaps selectively take those parts and combine them into a grand graph. It was noted that while the capabilities for sub-graph selection are non-existent in COMA 3.0, it can be modified somewhat easily as the software already provide node selection capabilities through its interface and graph merging functionalities.

The resulting graph can then be populated with multisource information. The source of the information could easily be encapsulated within a node and retrieved at some point in further analysis of this information.

Part 6

Conclusion

As seen, several schemas have been proposed in the context of maritime domain awareness. As these database schemas for the same domain were developed by independent parties, they are quite different from each other. COMA 3.0 was used in order to assess the feasibility of automatic schemas matching, using graph theory, over very dissimilar schemas of multiple complexity.

From the four maritime schemas, three pairs were studied for matching comparison. In each case, the computation was done in batch mode in order to try different matcher combinations, trying to find the best one that optimizes one of the three similarity metrics computed. Each cases were executed with no synonym at first then with a custom set of synonyms. In spite of the dissimilarities, COMA was able to find matches that fit the ground truth (or expected) values define in Appendix A, some small and other more successful. The reasons for these differences were not investigated as it would have required more time interfering with the project schedule. Same thing on the impact of the usage of the synonyms, it some case synonyms improve the output results but not always. Again, further investigation could reveal the reasons behind these results. Probably, synonyms can contribute in some types of matcher but not all of them. Also, it is possible that some matcher can find the optimal matching results without the help of synonyms.

For this study, three similarity metrics were used for comparing matching results. Another complex metric exists ([27]) but was not implemented. When examining the best matcher results, some pattern can be found although the order in which they are executed does not seem to be that important. Although a single set of matchers was shown in the result tables, it is worth mentioning that for a constant set of similarity metrics, there were many possible sets of matchers which result to the same optimal metric values.

In other future investigation, it would be interesting to increase the granularity of the investigation. For example, to see if it is always the same set of nodes that are being matched and to understand why the other unmatched nodes resist the matching process.

Several problems were encountered while using COMA 3.0 to measure schema similarity. The selected software was not adequately dealing with today's complex schema definitions. Also, the wide variety of available schema made it difficult to measure similarity effectively between complex and simple naming convention. Fortunately, the open source nature of the software enables us to

overcome most of the problems encountered.

Despite being a difficult task, semi-automatic schema matching and merging using graph theory is possible. Multiple schema matching and the construction of a grand graph can also be realized. However, in order to do so with COMA 3.0, the latter would require some more modification. The experience acquired with the code base of COMA 3.0 would make those modifications available in a reasonable time frame.

From this investigation, the analysis demonstrate that maritime schemas differ very much from each other in term of structure, semantic and data type. For those reasons, the maritime domain cannot be compared with other schema matching field such as molecular analysis where structure and semantic are more universal and rigourous.

In the future, in order to avoid having to deal with many different schemas, it could be enforced, like in the United States with the NIEM first initiative, to use a pre-defined standard in all project requiring information storage and exchange. This would remove the necessity for schema matching on newly developed technologies and systems and ensure that efficient information exchange can be achieved. Adoption of a standard across government agencies can also lead to cost reduction as less time would be devoted to schema definition and information exchange implementation to match the different vocabularies used by different group of interest.

Bibliography

[1] Alsayed Algergawy, Eike Schallehn, and Gunter Saake. Understanding the schema matching problem. In *Proceedings of the 7th Conference on 7th WSEAS International Conference on Applied Computer Science*, volume 7, pages 59–68, 2007. URL http://www.wseas.us/e-library/conferences/2007venice/papers/570-446.pdf.

[2] Pavel Shvaiko and Jérôme Euzenat. A survey of schema-based matching approaches. In *Journal on Data Semantics IV*, pages 146–171. Springer, 2005. URL http://disi.unitn.it/~p2p/RelatedWork/Matching/JoDS-IV-2005_SurveyMatching-SE.pdf.

[3] K Amshakala and R Nedunchezhian. Schema Matching Using Directed Graph Matching. *WSEAS Transactions on Computers*, 12(9), 2013. URL http://www.wseas.org/multimedia/journals/computers/2013/a025705-243.pdf.

[4] Alon Halevy. Why your data won't mix. *ACM Queue*, 3(8):50–58, 2005. URL https://homes.cs.washington.edu/~alon/files/acmq.pdf.

[5] Philip A Bernstein, Jayant Madhavan, and Erhard Rahm. Generic schema matching, ten years later. *Proceedings of the VLDB Endowment*, 4(11):695–701, 2011. URL http://dbs.uni-leipzig.de/file/10yearBestPaper-BernsteinMadhavanRahm.pdf.

[6] Horst Bunke. Graph matching: Theoretical foundations, algorithms, and applications. In *Proc. Vision Interface*, volume 2000, pages 82–88, 2000. URL http://www.cipprs.org/papers/VI/VI2000/pp082-088-Bunke-2000.pdf.

[7] Danai Koutra, Ankur Parikh, Aaditya Ramdas, and Jing Xiang. Algorithms for graph similarity and subgraph matching, December 2011. URL https://www.cs.cmu.edu/~jingx/docs/DBreport.pdf. [Online, last accessed december 2014].

[8] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 117–128. IEEE, 2002. URL http://ilpubs.stanford.edu:8090/730/1/2002-1.pdf.

[9] Jing Jin. *Similarity of weighted directed acyclic graphs*. PhD thesis, University of New Brunswick, 2006. URL http://www.cs.unb.ca/agentmatcher/Jing%20system%20back%20up/wDAG/thesis%20and%20defense/0831JingThesisfinal%20without%20complexity.pdf.

[10] John C Platt. Fast embedding of sparse music similarity graphs. *Advances in neural information processing systems*, 16:571578, 2004. URL `http://research.microsoft.com/pubs/68916/fastsparsemusic.pdf`.

[11] Laura Zager. *Graph similarity and matching*. PhD thesis, Massachusetts Institute of Technology, 2005. URL `http://dspace.mit.edu/bitstream/handle/1721.1/34119/67618399.pdf?sequence=1`.

[12] Wenfei Fan. Graph pattern matching revised for social network analysis. In *Proceedings of the 15th International Conference on Database Theory*, pages 8–21. ACM, 2012. URL `http://homepages.inf.ed.ac.uk/wenfei/qsx/reading/icdt12.pdf`.

[13] Wikipedia. Biological network. `http://en.wikipedia.org/wiki/Biological_network`, 2014. URL `http://en.wikipedia.org/wiki/Biological_network`. [Online; accessed November-2014].

[14] Owen Macindoe and Whitman Richards. Graph comparison using fine structure analysis. In *Social Computing (SocialCom), 2010 IEEE Second International Conference on*, pages 193–200. IEEE, 2010. URL `http://people.csail.mit.edu/whit/macindoe_richards_soccom10.pdf`.

[15] Mikhail Zaslavskiy, Francis Bach, and Jean-Philippe Vert. Global alignment of protein–protein interaction networks by graph matching methods. *Bioinformatics*, 25(12): i259–1267, 2009. URL `http://bioinformatics.oxfordjournals.org/content/25/12/i259.full.pdf+html`.

[16] Erhard Rahm and Philip A Bernstein. A survey of approaches to automatic schema matching. *the VLDB Journal*, 10(4):334–350, 2001. URL `http://dbs.uni-leipzig.de/file/VLDBJ-Dec2001.pdf`.

[17] Hong-Hai Do and Erhard Rahm. Matching large schemas: Approaches and evaluation. *Information Systems*, 32(6):857–885, 2007. URL `http://disi.unitn.it/~p2p/RelatedWork/Matching/LargeSchemas-COMA.pdf`.

[18] AnHai Doan, Jayant Madhavan, Pedro Domingos, and Alon Halevy. Learning to map between ontologies on the semantic web. In *Proceedings of the 11th international conference on World Wide Web*, pages 662–673. ACM, 2002. URL `http://homes.cs.washington.edu/~pedrod/papers/www02.pdf`.

[19] Marko Smiljanić, Maurice Van Keulen, and Willem Jonker. Formalizing the XML schema matching problem as a constraint optimization problem. In *Database and Expert Systems Applications*, pages 333–342. Springer, 2005. URL `http://www.ub.utwente.nl/webdocs/ctit/1/000000f6.pdf`.

[20] Khalid Saleem, Zohra Bellahsene, et al. New Challenges in Data Integration: Large Scale Automatic Schema Matching. 2007. URL `https://hal.archives-ouvertes.fr/file/index/docid/273699/filename/SchemaMatchingSurvey.pdf`.

[21] S Ezhilin Freeda and TC Ezhil Selvan. Schema Matching with Inter-Attribute Dependencies Using VF2 Approach. *International Journal of Emerging Engineering Research and Technology*, 2:14–20, 2014. URL `http://www.ijeert.org/pdf/v2-i3/3.pdf`.

[22] Institute of Computer Science. *COMA 3.0 CE: Program Description*. University of Leipzig, August 2012. URL `http://www.mirrorservice.org/sites/downloads.sourceforge.net/c/project/co/coma-ce/Program%20Description.pdf`.

[23] Hong Hai Do. *Schema matching and mapping-based data integration*. PhD thesis, Universität Leipzig, 2006. URL `http://lips.informatik.uni-leipzig.de/files/2006-4.pdf`.

[24] Hong-Hai Do and Erhard Rahm. COMA: a system for flexible combination of schema matching approaches. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 610–621. VLDB Endowment, 2002. URL `http://dbs.uni-leipzig.de/file/COMA.pdf`.

[25] Fabien Duchateau, Zohra Bellahsene, and Remi Coletta. A flexible approach for planning schema matching algorithms. In *On the Move to Meaningful Internet Systems: OTM 2008*, pages 249–264. Springer, 2008. URL `http://disi.unitn.it/~p2p/RelatedWork/Matching/Duchateau_coopis08.pdf`.

[26] Wikipedia. Jaro-winkler distance. `http://en.wikipedia.org/wiki/Jaro%E2%80%93Winkler_distance`, 2015. URL `http://en.wikipedia.org/wiki/Jaro%E2%80%93Winkler_distance`. [Online; accessed January-2015].

[27] Jérôme Euzenat. Semantic Precision and Recall for Ontology Alignment Evaluation. In *IJCAI*, pages 348–353, 2007. URL `http://ijcai.org/papers07/Papers/IJCAI07-054.pdf`.

[28] Xiang Li, Christoph Quix, David Kensche, and Sandra Geisler. Automatic schema merging using mapping constraints among incomplete sources. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 299–308. ACM, 2010. URL `http://disi.unitn.it/~p2p/RelatedWork/Matching/merge-cikm10.pdf`.

This page is intentionally left blank.

Appendix A

Ground Truth RDF files

A.1 Ground Truth for matching NOA with EPC

```
<?xml version="1.0" encoding="iso-8859-1" ?>

<rdf:RDF xmlns="http://knowledgeweb.semanticweb.org/heterogeneity/alignment"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:xsd="http://www.w3.org/2001/XMLSchema#">

<Alignment>

<!--MatchResult of simMatrix [41,41] + Name: GroundTruth + Info:
($ComaOptS) + Source: null + Target: null + Matcher: null + Config: null-->

<xml>yes</xml>
<level>0</level>
<type>11</type>
<onto1>/home/mmayrand/Common/projects/risomia/call-up_9/noa-3.2.iepd/XMLschemas/exchange/3.2/noa-exchange.xsd</onto1>
<onto2>/home/mmayrand/Common/projects/risomia/call-up_9/schemas/epc-msg.xsd</onto2>
<uri1>null</uri1>
<uri2>null</uri2>
<map>
<Cell>
<entity1 rdf:resource="Notice.Location"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.Location"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.LastPortOfCall"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.LastPortOfCall"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.CrewList"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.CrewList"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.NonCrewList"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.PassengerList"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
```

```
</Cell>
<Cell>
<entity1 rdf:resource="Notice.Vessel.VesselAugmentation.VesselCallSignText"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.ShipID.CallSign"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.Vessel.VesselAugmentation.VesselClass"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.ShipClass"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.Vessel.VesselAugmentation.VesselClassificationSocietyName"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.ShipClass.SocietyName"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.Vessel.VesselAugmentation.VesselGrossTonnage"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.GrossTonnage"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.Vessel.VesselAugmentation.VesselIMONumberText"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.ShipID.IMONumber"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.Vessel.VesselAugmentation.VesselMMSIText"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.ShipID.MMSINumber"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.Vessel.VesselAugmentation.VesselName"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.ShipID.ShipName"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.Vessel.VesselAugmentation.VesselNationalFlag"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.ShipClass.Country"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.Vessel.VesselAugmentation.VesselNavigationStatusAbstract"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.ShipStatus.NavigationalStatus"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.Vessel.VesselAugmentation.VesselOverallLengthMeasure"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.LengthOverall"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.Vessel.VesselAugmentation.VesselBeamMeasure.MeasureValue"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.Beam"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
```

```
</Cell>
<Cell>
<entity1 rdf:resource="Notice.Location.LocationPort.PortNameAbstract"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.Location.Port.Name"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.Arrival.VisitReceivingFacilityName"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.PortOfArrival.Facility"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.Arrival.Port.PortNameAbstract"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.PortOfArrival.Name"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.Departure.VisitReceivingFacilityName"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.PortOfDeparture.Facility"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.Departure.Port.PortNameAbstract"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.PortOfDeparture.Name"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.LastPortOfCall.VisitReceivingFacilityName"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.LastPortOfCall.Facility"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.LastPortOfCall.Port.PortNameAbstract"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.LastPortOfCall.Name"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.NextPortOfCallList.NextPortOfCall"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.NextPortOfCall"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.NextPortOfCallList.NextPortOfCall.VisitReceivingFacilityName"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.NextPortOfCall.Facility"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.NextPortOfCallList.NextPortOfCall.Port.PortNameAbstract"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.NextPortOfCall.Name"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.CrewList.Crew"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.CrewList.CrewMemberData"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
```

```
</Cell>
<Cell>
<entity1 rdf:resource="Notice.CrewList.Crew.PersonCountryOfResidence"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.CrewList.CrewMemberData.CountryOfResidence"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.CrewList.Crew.PersonDebarkationDate"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.CrewList.CrewMemberData.DebarkationDate"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.CrewList.Crew.PersonEmbarkationDate"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.CrewList.CrewMemberData.EmbarkationDate"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.CrewList.Crew.PersonDebarkationLocation.LocationPort"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.CrewList.CrewMemberData.DebarkationPort"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.CrewList.Crew.PersonDebarkationLocation.LocationPort.PortNameAbstract"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.CrewList.CrewMemberData.DebarkationPort.Name"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.CrewList.Crew.PersonEmbarkationLocation.LocationPort"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.CrewList.CrewMemberData.EmbarkationPort"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.CrewList.Crew.PersonEmbarkationLocation.LocationPort.PortNameAbstract"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.CrewList.CrewMemberData.EmbarkationPort.Name"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.NonCrewList.NonCrew"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.PassengerList.PassengerData"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.NonCrewList.NonCrew.PersonCountryOfResidence"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.PassengerList.PassengerData.CountryOfResidence"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.NonCrewList.NonCrew.PersonDebarkationDate"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.PassengerList.PassengerData.DebarkationDate"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.NonCrewList.NonCrew.PersonEmbarkationDate"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.PassengerList.PassengerData.EmbarkationDate"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
```

```
</Cell>
<Cell>
<entity1 rdf:resource="Notice.NonCrewList.NonCrew.PersonDebarkationLocation.LocationPort"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.PassengerList.PassengerData.DebarkationPort"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.NonCrewList.NonCrew.PersonDebarkationLocation.LocationPort.PortNameAbstract"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.PassengerList.PassengerData.DebarkationPort.Name"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.NonCrewList.NonCrew.PersonEmbarkationLocation.LocationPort"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.PassengerList.PassengerData.EmbarkationPort"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Notice.NonCrewList.NonCrew.PersonEmbarkationLocation.LocationPort.PortNameAbstract"/>
<entity2 rdf:resource="EPCMessage.EPCRequestBody.PassengerList.PassengerData.EmbarkationPort.Name"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
</map>
</Alignment>
</rdf:RDF>
```

A.2 Ground Truth for matching NIEM:Position with NPR

```
<?xml version="1.0" encoding="iso-8859-1" ?>

<rdf:RDF xmlns="http://knowledgeweb.semanticweb.org/heterogeneity/alignment"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:xsd="http://www.w3.org/2001/XMLSchema#">

<Alignment>


<!--MatchResult of simMatrix [8,8] + Name: GroundTruthPosNpr + Info:
($FragSelectionS;$DownPathSelectionS) + Source: null + Target: null + Matcher: null + Config: null-->


<xml>yes</xml>
<level>0</level>
<type>11</type>
<onto1>/home/mmayrand/position-3.2.iepd/XMLschemas/exchange/3.2/position-exchange.xsd</onto1>
<onto2>/home/mmayrand/risomia/coma/coma-project/schema/npr3_tables.sql</onto2>
<uri1>null</uri1>
<uri2>null</uri2>
<map>
<Cell>
<entity1 rdf:resource="Message.Vessel.VesselAugmentation.VesselNavigationStatusAbstract"/>
<entity2 rdf:resource="[dbo].[ContactReports](.[NavStatusId]"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Message.Vessel.VesselAugmentation.VesselNationalFlag"/>
<entity2 rdf:resource="[dbo].[ContactReports](.[CountryFlag]"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
```

```
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Message.Position.PositionSpeedMeasure"/>
<entity2 rdf:resource="[dbo].[ContactReports](.[Speed]"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Message.Position.PositionCourseMeasure"/>
<entity2 rdf:resource="[dbo].[ContactReports](.[Course]"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Message.Position.PositionHeadingMeasure"/>
<entity2 rdf:resource="[dbo].[ContactReports](.[Heading]"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Message.Vessel.VesselAugmentation.VesselIMONumberText"/>
<entity2 rdf:resource="[dbo].[Identities](.[IMO]"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Message.Vessel.VesselAugmentation.VesselMMSIText"/>
<entity2 rdf:resource="[dbo].[Identities](.[MMSI]"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Message.Vessel.VesselAugmentation.VesselCallSignText"/>
<entity2 rdf:resource="[dbo].[Identities](.[CallSign]"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
</map>
</Alignment>
</rdf:RDF>
```

A.3 Ground Truth for matching NIEM:Position with MtbTrack

```
<?xml version="1.0" encoding="iso-8859-1" ?>

<rdf:RDF xmlns="http://knowledgeweb.semanticweb.org/heterogeneity/alignment"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:xsd="http://www.w3.org/2001/XMLSchema#">

<Alignment>


<!--MatchResult of simMatrix [8,8] + Name: GroundTruthPosMusic + Info:
($ComaOptS) + Source: null + Target: null + Matcher: null + Config: null-->


<xml>yes</xml>
<level>0</level>
<type>11</type>
<onto1>/home/mmayrand/position-3.2.iepd/XMLschemas/exchange/3.2/position-exchange.xsd</onto1>
<onto2>/home/mmayrand/MUSIC Schemas/MtbTrack.xsd</onto2>
```

```
<uri1>null</uri1>
<uri2>null</uri2>
<map>
<Cell>
<entity1 rdf:resource="Message.Position"/>
<entity2 rdf:resource="MtbTrack.Report.Position"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Message.Vessel.VesselAugmentation.VesselCallSignText"/>
<entity2 rdf:resource="MtbTrack.Attributes.Callsign"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Message.Vessel.VesselAugmentation.VesselCategory"/>
<entity2 rdf:resource="MtbTrack.Attributes.Category"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Message.Vessel.VesselAugmentation.VesselClass"/>
<entity2 rdf:resource="MtbTrack.Attributes.ShipClass"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Message.Vessel.VesselAugmentation.VesselHullNumberText"/>
<entity2 rdf:resource="MtbTrack.Attributes.Hull"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Message.Vessel.VesselAugmentation.VesselName"/>
<entity2 rdf:resource="MtbTrack.Attributes.Name"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Message.Vessel.VesselAugmentation.VesselNationalFlag"/>
<entity2 rdf:resource="MtbTrack.Attributes.Flag"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
<Cell>
<entity1 rdf:resource="Message.Vessel.VesselAugmentation.VesselSCONUMText"/>
<entity2 rdf:resource="MtbTrack.Attributes.SCONUM"/>
<measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
<relation>=</relation>
</Cell>
</map>
</Alignment>
</rdf:RDF>
```

This page is intentionally left blank.

Appendix B

Synonyms

B.1 Synonyms between matching NOA and EPC

To target	**From source**
Ship	Vessel
Passenger, Non Crew Attributes	Vessel Augmentation
Callsign	Call Sign
Port Call	Port Of Call
Contry Code	Country
Country	National Flag

Table B.1: Synonyms defined between the NIEM Notice of Arrival schema and the Electronic Port Clerance schema.

B.2 Synonyms between NIEM:Position and NPR

To target	**From source**
Ship	Vessel
Contact Reports	Position
Country	National
Identities	Vessel Augmentation
Nav	Navigation

Table B.2: Synonyms defined between the NIEM Position schema and the NPR schema.

B.3 Synonyms between NIEM:Position and MtbTrack

To target	**From source**
Passenger	Non Crew
Ship	Vessel
Attributes	Vessel Augmentation
Mtb Track	Vessel

Table B.3: Synonyms defined between the NIEM Position schema and the MTB schema.