Examining Standardised XML Validation Methods against an Industry Vocabulary:

Improvement of XLIFF 2 Roundtrip Workflows Through Effective Schema Design

A Thesis Submitted for the Degree of Doctor of Philosophy
By

Soroush Saadatfar

Department of Computer Science and Information Systems
University of Limerick

Supervisors: Dr. David Filip and Dr. Giuseppe Torre

Submitted to the University of Limerick, May 2017

Abstract

The concept of *automation* has dramatically changed after the machine-to-machine communication was established in mid 20th century and Information Systems were enabled to form networks and exchange data. Like any form of communication, *protocols* (standards) are an essential part of such networks and *compatibility* with the defined protocol is the most fundamental requirement for facilitating an effective exchange environment.

With the exponential advancement of processing technologies over the past years, contemporary information systems are capable of targeting complicated tasks within an automated workflow. The components of such workflow are in constant interaction and, therefore, reliable exchange standards for effective storage of the structured data play a vital role in enabling *interoperability*. However, as standards improve to cover the emerging requirements and to provide solutions for new demands, *conforming* to standards with complex data models is a non-trivial subject. To overcome conformance issues, which blocks *semantic interoperability*, the literature suggests defining the standards using machine-readable schema.

This research, conducted in line with design and development methodology, represents our work on delivering the machine-readable specification of OASIS XLIFF 2.1, a business-driven and XML-based standard. This standard is the most widely adopted exchange format in the localisation world and provides a platform for storage of translatable content and the associated metadata. The result of this work, an advanced validation platform based on standardised and declarative methods, is included in the Specification of the standard as *normative*. This work improves the state-of-the-art of the standard by providing machine-readable and platform independent artefacts that can be used by all users to guarantee an interoperable environment for XLIFF-based workflows.

This dissertation contributes to knowledge by providing a detailed study of XML validation methods, discovering gaps in the related literature and also by examining schema languages, especially Schematron, against business-driven rules. Finally, this work attempts to highlight the expressivity and capability of standardised XML schema languages, which lack fundamental research, and we hope that the results of this work would promote their wide adoption by providing useful guidelines. The validation platform for XLIFF 2.1 represents a unique work, by scale and official recognition, which is based on standardised and declarative methods.

Declaration

I hereby declare that this thesis is my own work and that it has not been submitted previously to this for the award of any other degree or academic award at any other institute.

Signature: [illegible]

Date: 28 May 2017

Acknowledgements

First and foremost I would like to express my sincere gratitude to my supervisor, Dr. David Filip, for his priceless help and support throughout my PhD program. His precious advices and guidelines, based on a unique vision on the world, alongside with his patience gradually led me towards independence in research. It was a great honour for me to carry out this research under his supervision.

I also would like to thank Dr. Giuseppe Torre, my second supervisor, for all his contributions of time, effort and suggestions to my work. Even though he joined my supervision team at the final stage, it is hard to imagine this dissertation brought to an end without his help.

The main technologies that this research is built on, i.e. Schematron and NVDL, are rather newcomers to the XML world and lack systematic research or comprehensive references in the literature. Many thanks to Professor Felix Sasaki and Mr. Yves Savourel for their feedback and suggestions during my work.

A big thank you to the ADAPT Centre (former CNGL) for the provided funding and facilities, especially to Professor David Lewis and Ms. Hilary Mc-

Donald who made my one-year secondment in Trinity College Dublin possible.

Last but not least, I would like to thank my parents and my little sister for the incredible support and the warm encouragement during the course of my PhD. I would never be able to express their fundamental contribution to this work by words, for which I am very thankful.

This PhD research was supported by the Science Foundation Ireland (Grant 12/CE/I2267) as part of the ADAPT Centre (www.adaptcentre.ie) at the University of Limerick, Limerick, Ireland.

Dedication

Even though this piece of research spans only a short period of my life, the scope of occurred events and changes is by no means proportional to the embraced time frame and here I would like to highlight the most significant and the happiest one of all; the birth of my daughter and an adorable angel, Sarina, who brought a new purpose to my life and granted me the motivation to walk this path through when I was stuck halfway. I am sincerely grateful to her and would like to dedicate this thesis to Sarina.

Contents

List of Tables

List of Figures

List of Abbreviations

API	Application Programming Interface
CAT	Computer-assisted Translation
CREAM	Conflict Resolution Environment for Autonomous Mediation
DOM	Document Object Model
DSDL	Document Schema Definition Languages
DSL	Domain-specific Languages
DTD	Document Type Definition
EIF	European Interoperability Framework
GALA	Globalization and Localization Association
HTML	HyperText Markup Language
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
ITS	Internationalization Tag Set
LISA	Localization Industry Standards Association
LSP	Localisation Service Provider
MT	Machine Translation

OASIS	Organization for the Advancement of Structured Information Standards
OSI	Open System Interconnection
RDF	Resource Description Framework
Relax NG	REgular LAnguage for XML Next Generation
RESTful	Representational state transfer
RNG	Relax NG
SaaS	Software as a Service
SAX	Simple API for XML
SGML	Standard Generalized Markup Language
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SRX	Segmentation Rules eXchange
TBX	TermBase Exchange
TC	Technical Committee
TM	Translation Memory
TMX	Translate Memory eXchange
W3C	World Wide Web Consortium
XLIFF	XML Localisation Interchange File Format
XML	eXtensible Markup Language

XML-DEV	XML Developers community
XSD	W3C XML Schema
XSLT	Extensible Stylesheet Language Transformations

Chapter 1

Introduction

1.1 Research Introduction

In this chapter we will provide a brief and general introduction into our work and its main underlying domains: localisation, interoperability and XML. We will further define the research questions and goals of our work as well as discussing the methodology used to address our research questions. This chapter also provides a a review of our research contribution and, finally, the outline of the thesis and its organisation is presented and discussed.

We will first define the fundamental concepts of our work, starting with localisation. This term originates from the word “locale”- a small area. “Localisation” has been associated with the language industry beginning from the early 1990s. The process of localisation has significantly evolved ever since and there has been a number of definitions proposed for it. According to the Localisation Industry Standards Association (LISA),

> Localisation involves taking a product and make it linguistically and culturally appropriate to the target locale (country/region and language), where it will be used and sold. (Cited in Esselink 2000)

A more precise definition was suggested later by Schaler (2009), adding more details he describes localisation as:

> linguistic and cultural adaptation of digital content to the requirements and locale of a foreign market, and the provision of services and technologies for the management of multilingualism across the digital global information flow.

However, all the definitions consider “adaptation” or “passive reproduction” the core and main purpose of localisation (Pym 2004). The fast-growing volume of digital content to be localised (e.g. news, software, user-generated content) makes the management and processing of the localizable data one of the key challenges for the localisation industry in the World Wide Web era, where the distributed data is easily accessible across the globe. In order to address increasing demands and deliver localised content within acceptable range of time, cost and quality the localisation domain has remarkably expanded over the past years; a localisation process spans enormous number of activities and technologies, created or adopted to increase level of automation and, thus, enhance productivity. Furthermore, localisation has affected and influenced other related fields, especially software development and content generation which had to be fundamentally re-designed to integrate localisation features. Most of the software is now being developed in accordance with principles of *internationalisation*, a new concept introduced to facilitate localisation support in the digital content. An *internationalised* software, for instance, is required to store the source code and localizable data in different places, as well as providing support for known natural language character sets (Esselink 2003).

A state-of-the-art localisation process can be generalised as presented in Fig. 1.1 where the roundtrip of localizable content is divided into 4 major steps. The starting point is the extraction of content from original format (e.g. HTML) and restoring it in the desirable container, i.e. a localisation-specific format. Such specific format generally provides advanced localisation capacities that are not represented/supported in original formats. The next set of tasks after extraction could be grouped in *enrichment* category. At this stage a variety of metadata (e.g. context information, annotation for non-translatable fractions and terms etc.) is associated with the payload to provide instructions for the following step of the workflow-translation. Notably, comprehensive, highly informative and rich metadata, produced at the enrichment level, can significantly contribute to the automation of the most expensive part of the entire project, i.e. translation, by facilitating efficient usage of Machine Translation (MT). Effective metadata, on the other hand, can ease creation and manipulation of Translation Memories (TM) and increase their reusability. After the translation stage is finalized, based on one or both of human translation and MT, the last step is merging the localised content back into the original format. It is essential for extractors and mergers to share knowledge of the original format for accurate and seamless creation of the final (localized) document. However, it is not necessary for other entities involved in the workflow to be aware of internal processing information of the preceding/following nodes.

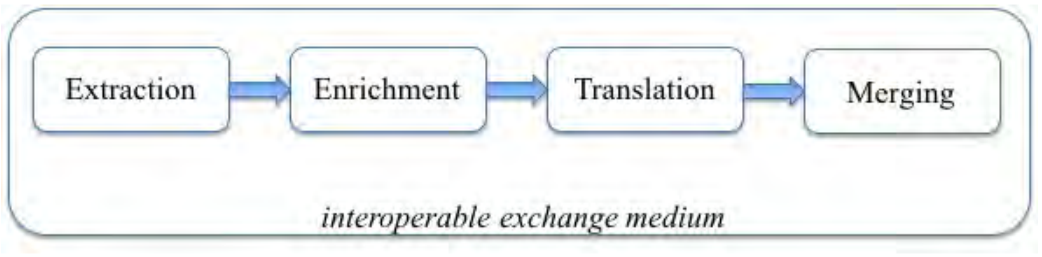


Figure 1.1: Processes of a contemporary localisation project

In order to enable a seamless exchange environment for the illustrated workflow in Fig. 1.1, a standardisation ecosystem started to evolve in the localisation world. Moreover, hiring standards would enable automation of the workflow and increase the efficiency of localisation to meet the current requirements of volume and speed. Exponential growth of localizable content, on the other hand, alongside with newly emerging tasks in each step of the process led to development of a number of solutions and formats. These formats have been adopted by different localisation entities and vary from common and general standards to small and specific technologies, developed by Localisation Service Providers (LSPs). Such diversity makes interoperability in a localisation ecosystem even more important; heterogeneity and inconsistency can cause tremendous damage as a localisation project might involve thousands of files and tens of activities and tasks. Standards were given special focus in building an *interoperable* environment for localisation components to integrate. Although most of early designed standards were initiated as proprietary, over time open standards seemed to gain popularity as they enable costumers and vendors to use the same format and therefore reduce the costs; according to Esselink (2003), adoption of open standards is the new tendency in the localisation. While this chapter represents a generic introduction, appropriate standards and formats of each localisation phase is provided in detail in chapter 2 (see p. 27).

The majority of localisation-related open standards, including the target of this research- XML Localisation Interchange File Format (XLIFF[1]), are defined using XML as the modelling environment. But before looking into this markup language, we will review the concept of interoperability to better understand the nature of needs for standards and standard platforms.

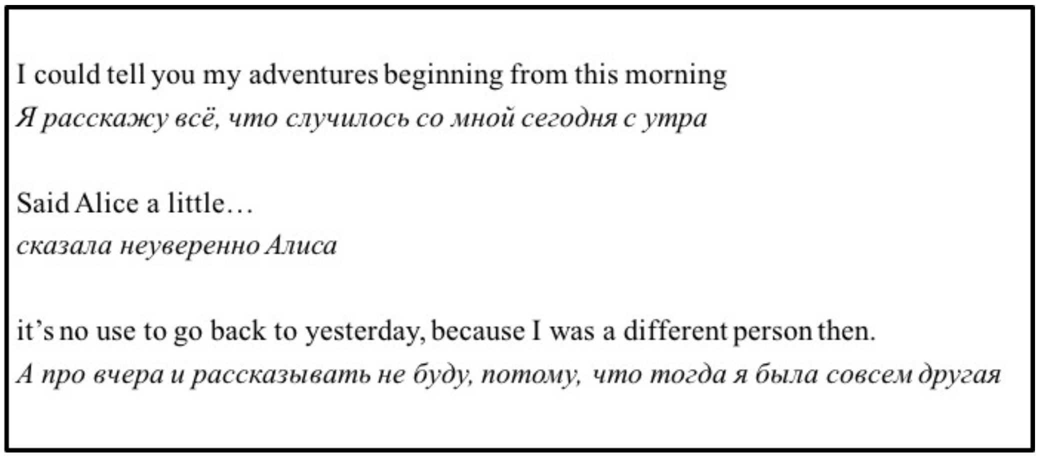
[1]Pronounced as 'exlif'

The emergence of collaborative systems in software engineering improved the overall efficiency and quality; Genesereth and Ketchpel (1994) report that *interoperation* enabled programs to solve those problems, which could not be addressed by an isolated software. The concept of *Agent-based software engineering* further was developed to facilitate interoperation of *agents*, as components of a larger system (Genesereth and Ketchpel 1994). The new scheme, nevertheless, created new challenges for interacting systems in terms of communication, known as interoperability issues (Neiva et al 2016). Although there has been a number of attempts to define and generalise interoperability issues, but the initial cause of such issues is the relation among incompatible systems (Naudet et al 2010). Interoperability has been also viewed as enabling co-existence and connection of heterogeneous software (Almeida et al 2010). According to the Institute of Electrical and Electronics Engineers (IEEE) (1991), interoperability "is the ability of two or more systems or components to exchange information and to use the information that has been exchanged". This being an early attempt, for Asuncion and van Sinderen (2011) interoperability is enabling effective usage of the supported services for interacting systems. Among all definitions, the one suggested by Lewis et al (2008) suits the context of this research the best:

The ability of a collection of communicating entities to (a) share specified information and (b) operate on that information according to an agreed operational semantics.

Interoperable tools must understand and use the exchanged data in an unambiguous way. This would allow various entities to integrate into a wrokflow for effective collaboration. Agreed structure and meaning, on the other hand, usually are achieved by defining standards for specific needs of a workflow. Standards are considered as one of the major solutions in enabling interoperability. In fact, if the involved entities conform to absolutely all the regulations and notations of the selected standard, they would not face interoperability issues due to homogeneity of the data-exchange environment.

The scope of this research is limited to interoperability in localisation workflows, where the XLIFF standard is being used as a common exchange format. XLIFF enjoys a great success and wide adoption in the industry mainly because of its two fundamental characteristics: it is a bitext format and it provides a comprehensive and expressive data model which is capable of adequately addressing needs of the entire localisation workflow presented in Fig. 1.1. Since the XLIFF data model and the functionalities it aims to deliver is a technical subject and covered in full in chapter 3 (see p. 77), in this introductory section we will focus on the XLIFF foundational attribute- bilinguality. XLIFF allows the content to be stored in *a* "source" language and *a* "target" language. This feature is reflection of a business requirement which is driven from classic translations, where the original content and its corresponding translation were visually aligned by sentences to ease readability, mapping and quality assessment. An example of classic bitext is illustrated in Fig. 1.2 where English sentences are aligned with Russian translation in a document.



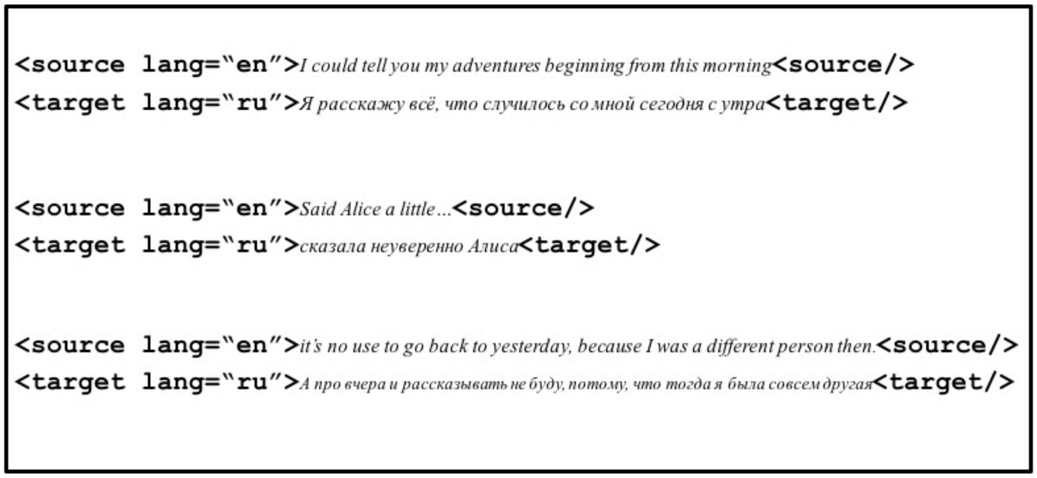
I could tell you my adventures beginning from this morning
Я расскажу всё, что случилось со мной сегодня с утра

Said Alice a little…
сказала неуверенно Алиса

it's no use to go back to yesterday, because I was a different person then.
А про вчера и рассказывать не буду, потому, что тогда я была совсем другая

Figure 1.2: A classic bitext artefact

Filip and O'Conchuir (2011) argue that such deposition is the most authentic representation of a linguistic (translator) mind and, in fact, the only way for proper quality assurance. The authors conclude that bitext is the essential core of localisation. Fig. 1.3 demonstrates how bitext content can be marked up and associated with appropriate metadata for effective manipulation. The artefact presented in Fig. 1.3 is a simplified example of XLIFF core logic and the full annotation with the same content is provided in Listing 3.1 (see p. 85) and its aspects are described and explored throughout chapter 3.



```
<source lang="en">I could tell you my adventures beginning from this morning<source/>
<target lang="ru">Я расскажу всё, что случилось со мной сегодня с утра<target/>


<source lang="en">Said Alice a little...<source/>
<target lang="ru">сказала неуверенно Алиса<target/>


<source lang="en">it's no use to go back to yesterday, because I was a different person then.<source/>
<target lang="ru">А про вчера и рассказывать не буду, потому, что тогда я была совсем другая<target/>
```

Figure 1.3: Effective bitext storage

The aim of our work is to deliver interoperability into XLIFF-based workflows and facilitate a seamless exchange of the localizable data and, therefore, appreciably improve the efficiency of the workflows by providing effective metadata for all of the nodes to secure correct usage of the exchange format at all levels.

We will now return to XML to provide introduction and discuss solutions for enabling interoperability in XML-based processes.

The history of markup languages goes as back as 1986, when the concept of Standard Generalized Markup Language (SGML) (ISO 1986) was first introduced. It was intended to address the need for a standardised approach to store the associated metadata in text files (Fawcett et al 2012). While SGML was rather received as a metalanguage (Maler and El Andaloussi 1995), it had a key role in creation of other markup languages such as Hyper Text Markup Language (HTML) and XML. The latter was released in late 1990s and unlike its ancestor, SGML, introduced a reduced level of flexibility. However, restrictions are minimalistic and the usage is very simple as XML is built only by two blocks: *elements* and *attributes* which shape *XML trees*. Nonetheless, XML can be hired not only for carrying data and metadata, but also to represent structured information through nested elements and attributes. This format has been designed to be easily read and interpreted by both machines and humans. XML soon and ubiquitously replaced traditional lists of name/value pairs and gained massive support to such an extend that today almost all of non-trivial computer applications implement XML at some level (Fawcett et al 2012). On the other hand, significant increase of the file size, because of the XML's verbose approach to store metadata, is considered as the biggest disadvantage of the markup language.

XML is generally used to define formats for effective storage of data and metadata in order to increase accessibility and reusability of the digital content. These advantages beside the XML support for internationalisation features made this markup language a suitable solution for developing localisation-related standards. Various standards have been released for major components of a localisation process (e.g. translation memory). However, in 2002, an OASIS standard, XLIFF, was introduced to facilitate seamless exchange of the localizable data between all bodies, regardless of the performed task. To achieve this goal, every application, intending to exchange information in XLIFF, has to conform to all of the specified regulations by the standard. In other words, the tool must be interoperable with the format.

Interoperability, from the XML perspective, is the proper adoption of a standard. This means that XML files must be *valid* and must not violate the defined constraints of the hired standard. Validation, a vital XML process, is carried out in order to check the structure and values of XML elements and attributes. Although a large number of technologies have been developed for this task, the scope of this work is limited to declarative and standardised approaches for validation of XML. Such methods, also known as *schema languages*, can be used to produce machine-readable, transparent, implementation- and platform-independent validation artefacts to *express* an XML format.

In order to justify the selected approach, we need to look into the foundation of XML schema languages. Van Deursen et al (2004) state that programming languages can generally be categorised as *generic* and *domain-specific* languages (DSL). Since the DSL are meant to solve specialised problems in a certain area, they generally offer a better solution for the domain of their focus. The main benefits of DSL are described as enhancement of productivity, reliability, maintainability and portability, whereas the cost of education, design and implementation is considered as the key disadvantages (Van Deursen et al 2004). DSL are usually small and declarative (Van Deursen et al 2004), which unlike *imperative* programming languages allows programmers to focus on the logic of the program and not the sequence of its execution (Coenen 1999). Another important characteristic of a DSL is its *expressive power*. This term is also referred to as "expressiveness" and "expressivity" in the literature and defines "expressibility or non-expressibility of programming constructs relative to a language" (Felleisen 1991).

Our research concentrates on delivering interoperability into localisation workflows, which are based upon a popular XML vocabulary- XLIFF 2 (Comerford et al; Eds.; 2014). In this work, we study and analyse the industry-driven structure and data model of XLIFF 2 from the XML prospective and aim to examine the expressive power of existing Document Schema Definition Languages (DSDL), developed for description of XML vocabularies, against the XLIFF 2 data model. Hiring such standardised methods for producing machine-readable artefacts could significantly improve interoperability across the tools. An optimal set of these artefacts, created for XLIFF 2, would be able to secure an interoperable environment for data exchange. XLIFF-based tools in the localisation workflow will only produce "clean output", which is valid and does not violate the standard's constraints or misuse the specified functionalities.

1.2 Aims and Objective

The primary question our research intends to address could be formulated as the following:

Are standardised declarative methods expressive enough to define XLIFF 2.1, a business-driven XML vocabulary?

Investigating this question includes answering to a number of sub-questions which we will explore and target throughout this thesis, but these sub-questions can be generalised as:

What is the optimal set of standardised XML validation methods to express the XLIFF 2 specification thoroughly?

The main objective of this research is to examine expressiveness capabilities of available standardised validation methods for XML against the demands of the modern and complex XML-based data models. The other objective of our work is to enhance the semantic interoperability in localisation workflows by delivering optimised set of standardised metadata for the most common exchange format in the area. The specified research question would help us to study targeted methods and identify their capabilities, limits and strength. On the other hand, our sub-question aims to allocate the specified constraint types to the best match out of the chosen schema languages. The main focus of the research sub-question is on optimisation and effective design.

The scope of this research is limited to the standardised description of XLIFF 2 data model and design and development of appropriate artefacts. This spans such areas like XML-based standards, XML validation techniques and semantic interoperability.

We consider research areas such as other XML standards, imperative validation approaches and other levels of interoperability as non-relevant to the defined research question and, therefore, out of scope of this work.

1.3 Methodology

In this section we will describe the research methodology hired to achieve the objectives of our work. However, the specific methodologies for each chapter will be presented in the corresponding chapter. Our methodology, for this thesis, would be of the kind, which is proposed by Johnson et al (2007) as mixed methods. Such methodology suggests usage of different elements of qualitative and quantitative approaches “for the broad purposes of breadth and depth of understanding and corroboration”. We will discuss the methodology strategies hired for different stages of this research within the following chapters, but in this section we will briefly review the research methods used to address the defined research questions. To address the primary question of our work, which requires examining a quality (expressivity) of a set of XML schema languages, we employed qualitative research methods. Such methodology was originally developed in social science to describe social phenomena using qualitative data such as documents, interviews etc. (Myers 1997). In computer science, Kaplan and Maxwell (1994) argue, qualitative methods are used in “evaluation studies, including evaluations of computer systems and information technology”. Among many available variations of qualitative methods, “Case Study Methodology” is claimed to be the most common for information systems (Orlikowski & Baroudi, 1991; Alavi and Carlson, 1992) which has been defined as “an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident” (Yin 1994). For the main question of our research we use a type of case study methodology, named “Positivist Research” which is “premised on a priori fixed relationships within the phenomena which are studied with structured instrumentation” (Orlikowski & Baroudi 1991). Positivist studies primarily aim to test hypothesis and theories and therefore was used to evaluate our target technologies- XML validation languages. The sub-questions of our work is investigated mainly using two research methodologies: *conceptual* and *design and development* research.

Based on abstract notions, conceptual research aims to “develop new concepts or reinterpret existing ones” (Kothari 2004). Design and development research, on the other hand, "results in production of some form of artefact" (Ellis and Levy 2010). Ellis and Levy (ibid) describe the methodology as:

> Such research begins with the initial conceptualisation of a problem and culminates in evaluation of the impact of one or more artefacts on ameliorating that problem. Design and development research focuses on building that bridging artefact that can serve to strengthen the interaction in the conceptualisation and evaluation cycle.

This research methodology must be distinguished from product development as the most important component of the latter is commercial success, whereas the objective of research has been defined as "addressing an acknowledged problem, building upon existing literature, and making an original contribution to the body of knowledge" (Ellis and Levy 2008).

We use other methodologies within this work for addressing secondary research questions, especially a sub-area of empirical methodology, named "Controlled Experiment" by Easterbook et al (2008), was hired as part of our methodology:

> A controlled experiment is an investigation of a testable hypothesis where one or more *independent variables* are manipulated to measure

their effect on one or more *dependent variables*. Controlled experiments allow us to determine in precise terms how the variables are related and, specifically, whether a cause-effect relationship exists between them. Each combination of values of the independent variables is a *treatment*

1.4 Research Contribution

The main and the most important contribution of our research is its output production: the effectively designed validation platform for OASIS XLIFF 2.x family. Within this thesis we discussed the increasing and extensive role of the family standard, its widespread adoption due to the comprehensive functionality it offers (or promises) and the growing interest in its usage as the exchange format in the localisation world. However the result of this research targets XLIFF 2.1, but the developed infrastructure allows easy adaptation for the further releases of the standard. Our proposed artefacts were approved by the XLIFF Technical Committee and will be officially included in the next version of the standard, 2.1, as the recommended validation solution. XLIFF 2.1 is currently in the final stages of release and our contribution to its development as a co-editor is in progress.

The remarkable advantages of our platform, standard-centric and transparency, would allow any XLIFF 2-based localisation workflow to integrate it as the validation component which can be used by all XLIFF 2 agents. This research contributes to elimination of the interoperability issues through advanced validation techniques and improved test-suite files. While the first will not allow invalid instances into the exchange environment, the latter provides a detailed guideline for proper usage of the standard. Our detailed study and review of XLIFF 2 revealed the limitations and issues of the current sample files and appropriate modifications were proposed and applied. We consider these modifications a valuable contribution of our research which inline with the validation platform represent the major achievement of this thesis and significantly improve the state-of-the-art of XLIFF.

On the other hand, we investigated an industry-driven data model from the academic point of view to systematically identify the existing (interoperability) issues and examine the proposed scientific solutions (standardisation) to address the problem. This aspect of our research which includes comprehensive use of standardised XML validation languages within a modular structure represents the originality and novelty of this work.

Finally, this research helped us to discover and highlight the gaps which exist in the related literature, mainly in the intersection of interoperability and XML areas. Our literature review revealed that the available research and resources around XML are behind the current demands of the modern data models. In the light of XML becoming an extreme popular exchange format, new research must be carried out to study the tendency of industry models, provide scientific and up-to-date classifications and explore the solutions for interoperability issues.

1.5 Thesis Outline

The rest of this dissertation is organised as follows:

- *Chapter 2*- provides a detailed literature review of the related areas of this work. This chapter starts with the discussion on localisation and the common standards in localisation. The chapter then provides a review on interoperability and XML, the other key focuses of this work. Our literature review aims to investigate and define the interoperability issues of XML-based workflows, or XLIFF in localisation in particular.

- *Chapter 3*- presents our work on detailed study, collection, analyses and classification of XLIFF 2.1 integrity constraints. This Chapter starts with description of the XLIFF data model and its specific non-XML behaviour and requirements. The other important task of this chapter is to recommend potential candidate (XML schema language) for the collected XLIFF constraints.

- *Chapter 4*- contains development process of the validation artefacts for XLIFF. Based on the detailed findings of Chapter 3, the appropriate schema will be developed and further optimised to form an interoperable and effective platform for advanced validation.

- *Chapter 5*- describes how the developed artefacts were evaluated and tested in practice. This chapter also provides a discussion on limitations of available implementations for the selected schema languages and describes how the advanced validation service for XLIFF 2.1 will be implemented in different projects.

- *Chapter 6*- draws conclusions of this thesis by summarizing and reviewing our work. Lastly, it provides some suggestions and directions for the future work.

Attached to this dissertation is a CD containing the following items:

- *Validation Artefacts*- contains the developed schemas for XLIFF 2.1.
- *Test Suite*- stores the XLIFF 2.1 files which were developed and used for evaluation of the validation platform.
- *Thesis*- contains a PDF version of this dissertation.

Chapter 2

Literature Review

2.1 Introduction

The underlying areas of this research are localisation, interoperability and XML which represent the main subjects of our literature review. Notably, the latter, XML, is rather an implementation platform and therefore we will explore the markup language only in the context of localisation and interoperability.

This chapter will start by reviewing the localisation domain in the related literature. We will then look into the concept of interoperability and examine possible solution for overcoming interoperability issues caused in the localisation industry due to the lack of effective metadata. Finally, a detailed discussion of enabling interoperability in XML-based environments will be presented.

2.2 Localisation

The concept and definition of localisation was previously introduced in chapter 1 (see p. 2) and we described how broad this domain is. As the scope of our research, when it comes to localisation, is limited to standards and interoperability in the workflows, in this section we will focus on these aspects of localisation in the literature.

2.2.1 Localisation Workflow

Localisation is a complicated process compounded of several different tasks carried out in a workflow. The preferable definition of workflow in the context of localisation, chosen by Lenker et al (2010), is proposed by Hollingsworth (1995) as “computerised facilitation or automation of a business process”. Hollingsworth states that initialisation and execution of workflows are tasks for workflow management systems which are based on software applications. Lewis et al (2009a) describe chain of activities in a localisation workflow in detail, covering all steps of a typical localisation project from extraction and segmentation of translatable text to creation of the target document. Lenker et al (2010), however, briefly list core components of the workflow highlighting translation as the core activity followed by terminology management, editing and post-editing, file conversion and testing.

With the fast growing amount of content that producers will to localize (e.g. web pages, software), volume of translatable data and its heterogeneity are main characteristics of localisation wokflows. These attributes make automation the main issue of the industry that directly affects efficiency of the process (Lenker et al. 2010). Authors define linguistic resources and lack of well-trained staff as secondary concerns. They claim that a standard and state-of-the-art workflow does not exist and this forces entities to create their own solutions. Finally, they suggest that interoperability of the workflow components, lack of standards as well as standardised procedures are topics requiring more research.

Lewis et al (2009a), on the other hand, conclude that “opaqueness and coarse granularity of the work” is the main issue of existing localisation workflows. Communications of stakeholders, various languages and platforms hired for implementation of the workflows are also examined in the literature, but the aforementioned subjects are out of the scope of this research since our focus is the XLIFF-based workflows only.

2.2.2 Localisation Standards

Savourel (2007) states that open localisation standards will keep entities away from proprietary solutions they might create for specific needs. Standards also have been always considered in interoperability improvements of the localisation domain (Lommel 2006). When it comes to the platform of standards in localisation, XML has become a well-established solution in the industry; various XML-based standard vocabularies were developed and successfully adopted since the inception of XML. XML's internationalisation capabilities, like the support of Unicode, offers a foundation to satisfy market needs (Sasaki 2011). The markup language also provides the most flexible and powerful built-in features to store and manipulate data in different languages. We will now look into the most common XML-based open standards which are developed to target various tasks of localisation projects mentioned in the previous section.

Segmentation Rules eXchange

The process of breaking the input text down into translatable units is referred to as *segmentation*. This process is typically the first step of the workflow after extraction of localizable data. *Segments* usually, but not necessarily, are sentences (Lewis et al. 2009a) which should be determined through automated algorithms. For certain languages (e.g. Japanese) defining such algorithms is a non-trivial task, although text segmentation in general is not a straight-forward process as it might seem at first (Milkowski & Lipski 2011).

Segmentation Rules eXchange (SRX) (Pooley and Raya Eds.; 2008) is an XML vocabulary which introduces a mechanism for defining segmentation rules to be applied to languages. It is a vendor-neutral standard that enables interoperability among different Computer-Aided Translation (CAT) tools in the context of segmentation algorithms.

Translation Memory eXchange

Segmented documents in the next step of the workflow are checked against existing translation memories for matches and translation reuse. This practice significantly decreases costs of a localisation project as the result, and therefore translation memories are considered as efficient and fundamental tool in localisation (Du et al. 2010).

In 2004 an XML standard, Translation Memory eXchange (TMX) (Lisa 2004), was introduced to address the main problem of this area, which was defined by Lisa as loss of data during the translation memory exchange between tools and/or translation vendors. This open standard also positions itself as a vendor-neutral format and provides a markup mechanism for effective structuralisation of original text and its corresponding translation in one or more languages (i.e. TMX represents a multilingual standard).

Term Base eXchange

Quality and speed of translation can be remarkably improved when glossaries of the related context are provided. In fact, terminology management is an important activity in the workflow and often defined at the level of content producers (i.e. localisation client) (Lenker et al. 2010).

This XML-based standard, Term Base eXchange (TBX) (ISO 2008), is an open format which is used for representation and interchange of terminological data and is designed to be processable by both human and automated tools. Its modular structure enables support of various *datacategories* making the standard a concept-oriented framework for terminology markup. A default set of popular data-categories is predefined in order to maximize interoperability while exchanging term bases. Like the previous standard, TBX is also a multilingual format.

Internationalization Tag Set

Metadata has an important role in the entire process of localisation (Moorkens 2013). Effective and rich metadata associated with the original text will improve level of quality for each component of the workflow. Internationalization Tag Set (ITS) (Filip et al; Eds.; 2013) aims to provide a markup mechanism, within a set of XML elements and attributes, for enriching the localizable payload with various internationalisation and globalisation information. Metadata in ITS 2.0 is deployed in 19 *datacategories* which can be applied to certain parts of the document using XPath as the underlying regular expression language. ITS 2.0 data categories vary from basic information, like the “translate” notation which defines whether a part of text should be translated or not, to complex ones including the “Text Analysis” data category, meant for annotation of contextual disambiguation. This W3C recommendation has been widely adopted in the industry and also well-integrated in HTML 5 to “enhance the foundation to integrate automated processing of human language into core Web technologies”, as claimed within the ITS 2.0 specification.

XML Localisation Interchange File Format

The XML Localisation Interchange File Format was initiated as an OASIS standard in 2001 by various industry representatives to develop an open standard which can enable interoperability among different tools in the workflow. Its main purpose is to facilitate a lossless roundtrip of localizable data and provide infrastructure for all tasks of a localisation projects, especially stable ground for data exchange (Lewis et al 2009b). The latest version of XLIFF, 2.0, took a different approach than its previous major release, 1.2 (Savourel et al; Eds.; 2008), but kept the bilingual nature of the standard. It introduced a modular structure compounded of core elements and various optional modules for localisation tasks. Modules, which are meant to be gradually added for next releases of 2.x, significantly solve interoperability issues as users do not need to define custom extensions that cannot be regulated and controlled by the standard specification. The XLIFF Technical Committee carries out detailed research for industry needs to introduce appropriate modules for upcoming versions of XLIFF 2, although the core structure remains unchanged for purpose of backward compatibility among all further XLIFF 2 releases. A good example of such work is the ITS module for XLIFF 2.1 (Filip and Saadatfar 2016). As SRX and TMX has been left unsupported till their latest releases and the need for modified and improved versions of those standards exists, there are discussions to adopt them for the following versions of XLIFF 2 (FEISGILTT 2016). Hayes et al (2015) introduced a mapping mechanism from the TBX standard to glossary module of XLIFF 2.

To conclude, we would like to highlight the comprehensivity of XLIFF 2 family standard and its capability to not only handle major defined tasks of the localisation workflow (presented in chapter 1, see p. 5), but to implement functionality of other common standards. Torres del Rey and Morado-Vazquez (2015) claim that XLIFF, as an exchange format, is expressive enough to represent localizable content throughout a localisation process and provide an effective structure for all of the involved tools and parties. The aforementioned features of XLIFF 2 promoted this standard and enhancement of interoperability in XLIFF-based workflows as the main scope of this research.

2.3 Interoperability

In this section, we will review the literature to identify obstacles for interoperability and various methods of solving such issues in the context of XML-based localisation systems.

Interoperability, as a concept, has been defined in multiple ways. In Chapter 1 (see p. 2) we had a brief look at some of the most accepted ones. Van Lier (cited in Folmer and Verhoosel 2011) proposes what these definitions have in common is the necessity of making agreements on several levels, which makes the domain of interoperability broader, than a technical-only issue. There are a number of proposals in the literature for classification of layers of interoperability, varying from 3 main layers (Technical, Semantic, Organisational) (European Commission 2004) to as many as six layers as in the ISO Reference Model for Open System Interconnection (OSI). We will look into levels of interoperability in the following section.

Beside the definition and layers, study of the root causes of interoperability issues is an important aspect of this concept which can help in finding proper solutions. Naudet et al (2010) state that *incompatibility* is the main cause, arguing “a relation is an existence condition for interoperability problems”. This general concept has been often divided into two subclasses of *heterogeneity* and *misalignment* (Ruokolainen et al 2007, Naudet et al. 2010), however, in early research *resource heterogeneity* was considered as the root cause of interoperability issues (Naudet et al. 2006).

2.3.1 Layers of Interoperability

In a comparison study of common interoperability frameworks and model proposals, Kubicek and Cimander (2009) suggest a 4-layer model for interoperability as the generalised model:

1. **Technical-** The purpose of this layer is providing a secure medium for data transfer and interchange on a technical (hardware) level;

2. **Syntactic-** At this layer heterogeneous systems are enabled to process the received data through agreed exchange data structures (e.g. XML vocabularies);

3. **Semantic-** Pushing the syntactic interoperability further, interoperable systems of this layer can interpret the exchanged data and extract meaningful information from it;

4. **Organisational-** This layer enables automatic linkage of processes (workflows) among systems.

Asuncion and van Sinderen (2010), on the other hand, claim that effective collaboration cannot be achieved solely by agreements on syntax and semantics, but also on "how the data is used". They introduce an additional layer of *pragmatic interoperability*, defining it as "the compatibility between the intended versus the actual effect of message exchange". However, this proposed layer has been associated with the semantic layer later by Folmer and Verhoosel (2011).

Technical and syntactic layers of interoperability are generally achieved through protocols and standards (Kubicek et al. 2011) with XML-based standards being preferred for the syntactic interoperability (Folmer and Verhoosel 2011). Although the top two layers are prerequisites to the higher levels of interoperability, these layers are out of scope of this research and, therefore, we will focus on semantic and organisational layers.

Semantic Interoperability

Ruokolainen et al (2007) state “semantic interoperability asserts that actual content of collaboration is understood by the senders and receivers”. Sartipi and Yarmand (2008) describe this layer as “interoperability of terms and concepts between different information organisations. Overall, all proposed definitions (Heiler 1995; European Commissions 2004; Berges et al 2010; Asuncion and van Sinderen 2010; Kubicek et al. 2011) for this layer of interoperability seem to agree on “correct interpretation of exchanged data”, as Ruokolainen et al conclude. This means that at this level, also known as knowledge-level interoperability (Park & Ram 2004), the received data becomes *information* (Kubicek & Cimander 2009; Kubicek et al 2011).

Semantic issues of interoperability, according to Park and Ram (2004), occur due to “differences in implicit meanings, perspectives and assumptions” and classify the semantic conflicts into two subgroups of *data level* and *schema level*. The data-level issues are described as “differences in data domain” and they arise when the same data can be represented or interpreted in various ways; data, Park and Ram (2004) report, can be referred to both object’s values/properties as well as to the object itself at the entity level. Schema-level conflicts, on the other hand, are the result of inconsistencies in the linked metadata. Some examples of such problems are naming conflicts and entity-identifier conflicts.

While Sartipi and Yarmand (2008) claim that semantic interoperability issues can be solved by standardised ontology systems and services, designed for exchange of domain-specific applications and workflows, Ouskel and Sheth (1999) in a broader sense introduce four "enablers" of this layer:

1. **Ontologies and Terminology Transparency-** As ontologies are capable of precise representation of domain models, it is important to enable both user and provider to choose an ontology or create one for their specific needs;

2. **Context-** Providing context-sensitive information allows systems to *understand* the context of data. Representing context to systems can help to avoid information overloads;

3. **Rules of Interaction Mechanisms-** These rules are meant to specify a framework for interacting parties to agree on commitments, responsibilities and norms;

4. **Information Co-relations-** Representing semantically related information at a higher abstraction level, regardless of the media type used.

They suggest that solutions for overcoming semantic interoperability issues must contain three components: metadata, contexts and ontologies.

Park and Ram (2004) generalise attempts for achieving semantic interoperability in three main categories: 1) mapping-based ; 2) intermediary-based and 3) query-oriented approaches. They identify limitations of each method and propose a generic approach, Conflict Resolution Environment for Autonomous Mediation (CREAM), as a comprehensive solution. This framework is designed on the basis of "embracing the advantages" and "overcoming the limitations" of existing approaches for enabling the semantic interoperability at both data and schema levels.

For Kubicek et al (2011), semantic interoperability can be achieved if all users agree on a common exchange format for specific services. The key challenge in this domain, nevertheless, remains integration and re-usage of legacy systems (Sartipi and Yarmand 2008; Kubicek et al 2011; Naudet et al 2010; Park and Ram 2004). Finally, when it comes to XML-based workflows, beside a well-designed XML vocabulary as the exchange format, XML schemas and validation artefacts play an important role in delivering the semantic interoperability (Kubicek et al. 2011).

Organisational Interoperability

According to the first version of European Interoperability Framework (EIF), "organisational interoperability implies integrating business processes and related data exchange" from the practical perspective (European Commissions 2010). This layer, which is named *enterprise interoperability* as well in the literature, has the least progress in comparison with previous layers as it lacks clear agreement on the definition and related issues among the available frameworks. Moreover, attempts for classification of available solutions are not at a mature state (Kubicek and Cimender 2009). Esper et al (cited in Folmer and Verhoosel 2011) , on the other hand, claim that organisational interoperability can be achieved if enterprises "share the same goal and have compatible management strategies". European Commissions (2004) define requirements for organisational interoperability through services which are "customer-oriented, transparent, and follow the so-called one-stop shop approach". It was later proposed by Kubicek et al (2011) that only "technical standards for the linkage of business processes" must be considered as the next level of interoperability after technical, syntactic and semantics layers and it should be renamed as *Business Process Interoperability*. Setting the goal as transforming multiple sub-functions to one automated workflow, this layer must address only the technical side of this automated process. Authors suggest such layer of interoperability could be achieved by hiring same service structures and keeping interoperability enabled on the three lower levels. Using Service-Oriented Architecture (SOA) is highlighted as a potential solution, where inter-organisational process can be expressed by standardised notions.

2.3.2 Interoperability in Localisation

Localisation is considered as a decentralised industry, where the localizable data and process information are required to seamlessly travel among multiple components of the supply chain (Tingley 2015). The heterogeneous tool-chains of the localisation ecosystem must conform to high requirements of interoperability (Lewis et al 2015) with the lossless information exchange being a priority. Hiring open standards enables interoperability when it comes to data exchange. Open standards also contribute to the transparency of the processes and stabilisation of the market in long-term perspective (Almeida et al 2010; Anastasiou 2011). Defining requirements for localisation data exchange information is a non-trivial task due to the needs for both fixed and inflexible standards to guarantee a strong level of interoperability, as well as the need for customizability and extensibility of standards (Tingley 2015). As we pointed out earlier, XLIFF is a well-known open standard in the industry. Its earlier version, 1.2, in spite of ambitious intention of promoting a single interchange file format for the localisation tools, faced serious interoperability issues in the practice. Tingley (2015) reports that implementations of XLIFF 1.2 are often not interoperable because some of the features of the standard have become implementation-dependant. This could be caused by overlapping functionality of features, which were hired by different tools in various ways and, most often, partially in accordance with particular business cases (Filip and Wasala 2013). As Tingley reports, the most important cause of interoperability issues for 1.2 is lack of provided normative information with regards to correctness of a certain implementation which is left to tool vendors. To address the occurred issues of interoperability, the XLIFF Technical Committee introduced a new major release of the standard, XLIFF 2, with a modular structure to satisfy both rigid and flexible requirements of the industry. Based on proposal of Filip and Wasala (2013), a process and agent classification-based interoperability model is implemented in XLIFF 2.0. The main aim of such work is to ensure that different *agents* will use the same feature set for implementing a functionality and therefore enabling interoperability among the tools, even if the standard is not supported completely. However, the complex nature of XLIFF would still leave room for different interpretations and delivering semantic interoperability remains an open issue.

2.4 XML

Earlier in this chapter, the characteristics of XML, which have turned it to a well-established and widely adopted solution in the localisation world, were highlighted. We also looked into a number of common XML-based standards, including XLIFF which is the main target of our work. In this section we will first analyse the XML data model and its integrity constraints in the literature and then will review declarative technologies for implementing the defined constraints. The review will attempt to describe how to deliver semantic interoperability in the XML-based workflows and particularly the localisation workflow based on the XLIFF roundtrip.

2.4.1 XML Data Model

XML nowadays is considered as the standard format for data exchange in the information systems. Fan (2005) reports that XML is superseding relational model not only in the data exchange, but also is being increasingly used as "uniform model for data integration". Hu and Tao (2006) state that XML has become a common format because it is "slef-describing, human and machine readable, extensible, flexible and platform neutral", even though XML has a very simple, but at the same time universal and generalised nature as a data model: a) it is composed of only two objects: XML nodes and their values b) data model expressed through value and structure of nodes c) it defines a minimum set of rules in terms of syntax (Saadatfar and Filip 2016). However XML is being more often preferred among relational database technologies for describing structured data models, it was initially defined more as a textual language than data model (Goldman et al. 1999). This bias to textual language is because of obligatory implicit order of XML documents (ibid). The *Document Object Model (DOM)* (W3C 1998a) was therefore introduced as a standard interface which represents the XML data as a tree structure. While DOM creates object representation of the entire XML document in memory, another approach for retrieving and accessing XML data (*parsing*), known as Simple API for XML (SAX), provides *event-based* parsing mechanism, i.e. the document is being read and loaded only partially and an event is triggered when a structural block is encountered (e.g. start element tag). Although SAX is considered as a common parsing method nowadays for large XML files, it has never been officially standardised and it doesn't have a formal specification. It was developed by XML-DEV members and originally was a Java API only, but later implemented in other environments as well.

We will review the specifics of the XLIFF structure in the following sections, but its object model has been developed by Microsoft and it is expected to courage development of tools and increase of support for the XLIFF standard. Recently, a new OASIS Technical Committee, XLIFF Object Model and Other Serialisations (XLIFF OMOS[1]), was initiated to work on creation of a serialisation independent object model for the following releases of the standard.

2.4.2 XML Constraints

Any data model defines a set of constraints over its objects in order to semantically enrich the data. This concept comes from relational databases where different constraints have been well studied and classified. However, the research on definition, generalisation and normalisation of XML constraints is ongoing. The main challenge for this topic is caused by the hierarchical structure of XML data, which enables a new dimension of constraints, *relative*, on the top of *absolute* constraints for relational databases (Fan 2005). In other words, because XML can store both *relations* and *values* of objects, it is hard to agree on a common *normal*

[1]available: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xliff-omos [accessed 23 Jun 2016]

theory for XML, unlike relational databases. The suggested proposals, therefore, are considered more as notions for XML design enhancement rather than "normal XML". Libkin (2007) identifies three issues that an XML model with "good design" must overcome: a) storage redundancies b) lossless decompositions and c) update anomalies. Nevertheless, in this section we will review the studied types of XML constraints which are in the scope of our research, i.e. used for designing XLIFF 2.

Primary Keys and Foreign Keys

This type of constraints is defined as a mechanism, through which a tuple can be uniquely identified in a relation or referred to from another relation (Fan 2005). Buneman et al (2002) consider both types of keys as "fundamental to data models and conceptual deign". In the XML world, however, this type has been studied in two subcategories: absolute and relative, with absolute keys being a special case of relative ones. This classification is preferred in the XML-related research due to the hierarchical nature of XML. Although the theoretical work and proposed notions for keys in XML (Buneman et al 2002 ; Arenas et al 2002) seem sound, the common XML solution in practice, ID and IDREF attributes, are not expressive enough for capturing the suggested notions (Fan and Simeon 2000).

Fan (2005) claims that the ID/IDREF mechanism behaves more like a "pointer" than a key, because the concept is not well scoped and limits objects to only one identifier at a time.

Functional Dependencies

Vincent et al (2000) define functional dependencies as the "oldest and the most well-studied integrity constraint in relational databases". In fact, this category of constraints has a wide domain with the concept of "keys" extracted from it (Lee et al 2002). Although functional dependencies have been well clarified for relational data models, this definition is restricted to "flat and tabular relational data" and, therefore, cannot be directly extended to XML (ibid). Chen et al (2010) identify challenges of such extension as "structural differences" and reasonable definition of "value equality for complex [XML] elements". Nevertheless, there are a number of works dedicated to generalise and define a *normal form* for XML functional dependencies. According to Fan (2005), the proposals by Arenas and Libkin (2004) and Vincent et al (2000) are the most accepted ones in the literature; while the first treat XML like "tree tuple", the authors of the latter use path-based approach to define functional dependencies for XML. Investigating the implication and propagation issues of such constraints in XML is a core component of both proposals. However, while neither of them address *relative functional dependencies* (property of XML) and despite other works, Chen et al (2010) point out lack of consensus on a definition for this type of constraints in the XML world. However, a sub-area of functional dependencies, named *co-occurrence constraints*, has been defined for XML. This type, also known as co-constraints in the literature, represents a special case of functional dependencies which can be formulated as "if ... then ... " statements (Fawcett et al 2012). Co-constraints are usually used when value or presence of a node defines the content model of another node (van der Vlist 2003).

Progressive Constraints

A subarea of this category has been defined as studied as *incremental constraints* for XML (Abrao et al 2004; Benedikt et al 2002). This type of constraint is applied to an XML instance after some updates or modifications. All the research so far has been developed on the assumption that the constraints remain unchanged for pre- and post-modified files. As modern data models grow in complexity, the need for *dynamic constraints* has emerged. For instance, XLIFF specifies different stages for the life cycle of the file and each stage identifies a different set of constraints to be applied (named *Processing Requirements* in XLIFF 2). This type of constraints can be considered as the most recent and, therefore, not captured in the current literature.

2.4.3 Semantic Interoperability in XML Data

As we showed earlier in this chapter, hiring protocols and standards (e.g. an XML vocabulary) can facilitate syntactic interoperability in Information Systems. However, in order to ensure proper usage of data by users the data must be semantically interoperable as well. Fawcett et al (2012) describe interoperability as “one of the main driving forces behind XML” claiming that “the XML output by one application will most likely be used as the input to another”. Vergara-Niedermayr et al (2014) state that conforming to the predefined rules and constraints (i.e. semantic interoperability) enables syntactic interoperability between resources in an XML environment. They claim that semantically interoperable data has unambiguous structure and so can be manipulated by users correctly.

The process of examining XML documents against the published specification is called *validation* in the XML world. In other words, valid XML documents represent semantically interoperable data in information systems. In the following sections we will review existing approaches and technologies for XML validation.

2.4.4 XML Validation

It is crucial at this point to distinguish two concepts; well-formed and valid XML document. Investigating well-formedness in XML is to check whether the W3C's XML Recommendations have been followed during the creation of the document, which eventually determines if the output is XML or not (Fawcett et al 2012). An XML instance, on the other hand, is valid when it conforms to the constraints set by the data model it is representing. Such set of constraints is also referred to as *specification* of an XML vocabulary.

Neushul and Ceruti (2004) describe XML validation as a "binary decision-making mechanism" which specifies if an XML document conforms to a predetermined structure. The validation task can be performed in multiple ways, each following different logic and with various levels of efficiency. Generally, the structure of an XML vocabulary is described by some *schema language*, also known as *XML grammar*, which essentially provides a set of definitions and declarations for XML vocabularies. XML grammars aim to define the elements and attributes of the documents they describe, as well as element/attribute structural positions and the rules they adhere to in the documents (Tekli et al 2011). Nevertheless, there is a belief that XML schema languages are not "Turing complete" and, therefore, they cannot "verify every possible constraint" of an XML vocabulary (Harold 2004). Such common belief has lead users to hire another major category of XML validation approaches for verification of advanced and complex XML integrity constraints. This approach is based on direct use of programming languages (e.g. Java, Python) which would manipulate and check the XML document against the specified rules.

While schema languages produce transparent and declarative validation artefacts, which represent effective metadata required for enabling semantic interoperability, programmatic approach for XML validation is merely executable compiled code, highly dependant on platform and the developer party. Not only the latter is out of scope of this work, but avoiding such methods will significantly decrease interoperability issues in the light of transparent and sufficient schemas. Although we describe the entire XLIFF 2 specification in standardised and declarative languages and tend to disagree with the aforementioned common doubt about expressiveness of available schema languages, but proving this claim wrong from the point of view of Information Theory and Computability Theory is out of scope of our work and is rather subject for future research.

In the following section we will review the developed techniques for enhancement of declarative validation of XML and will have a detailed look into the schema languages we use within this research and alos explore their expressive power.

Finally, it is worth mentioning that XML validation and schema-based validation in particular are generally an expensive process, especially when an advanced validation, including datatypes and complex constraints, is performed (Maruyama 2002). Hines et al (2008) report that required memory and processing power beside the management of these resources remarkably increase costs of validation. Nevertheless, the benefits of enabling interoperability in large-scaled information systems justify the price, at which it can be achieved.

2.4.5 Standardised Technologies for XML Interoperability

In this section we will review the technologies developed to express and implement XML integrity constraints. Essentially, a schema language aims to encode the specification of the defined XML vocabulary into machine-readable scripts. Describing the structure of an XML is the first step and the core of this task which, in the context of standardised technologies, is expressed by *regular tree* *languages* (Murata et al 2005). Language members of this family perform on the basis of regular expressions by defining generic tree patterns for the target XML vocabulary. Expressivity power of this group, therefore, is limited to structural properties and datatypes (Tekli et al 2011; van der Vlist 2003; Murata et al 2005). Basic expressive power of these languages is driven by the fact, that they can define only limited relations of nodes, i.e. parent-child and sibling, in a tree structure (Dodds 2001). In the following we will introduce 3 languages of this type (DTD, XSD and RelaxNG) and will argue that despite significant differences they are built on the same principle and essentially describe structural properties of XML vocabularies.

Regular tree languages are not capable of describing constraints that affect particular areas of an XML tree and such constraints are target of a fundamentally different approach, known as *rule-based*. Such rules aim to define more advanced and complex constraints over some nodes of the XML vocabulary rather than the tree in general and, therefore, provide an advanced expressive power (Fawcett et al 2012). Since navigation through the XML tree and the ability to express complex nodes relations (further than parent-child and sibling) is the essential component of such approach, we will now introduce related technologies for XML query and then move on to schema languages in the following sections.

XML Query Languages

Access to specific subsets of structured data is a vital requirement in relational databases. Such access enables navigation through the document, selection and extraction of specified blocks of data. In XML, the blocks (i.e. XML nodes) can be reached from the root of the XML tree by a unique path, also named *node address* in the literature (Buneman et al 2000). Although a large number of languages have been developed for querying XML data, the scope of our work is limited to one query language- XML Path Language (XPath) (W3C 1999). The reason behind this selection is the XPath's usage and implementation in the schema languages hired by this research. *XPath expressions* are extremely common and the XPath syntax is widely adopted by various technologies around XML, including other query languages, like XQuery (W3C 2010), as the core of data query. XPath provides a powerful and flexible mechanism, following which one can access the desirable part of an XML document. It also includes a library of supported functions and operations, which can be hired to improve queries through enhanced *predicates*.

XPath has been evolving over time to meet the emerging demands of the XML world. Its latest release, at the moment of writing this thesis, XPath 3.0 (W3C 2014), is capable of selecting XML nodes based on advanced and very complex expressions. Fan and Simeon (2000), on the other hand, report that such complexity makes "reasoning about constraints defined with XPath highly intricate, if not impossible". Finally, Marx and de Rijke (2004), after analysing expressivity power of XPath from the Mathematical Logic point of view, have characterised the language as *not first-order complete*, i.e. not every query is definable by Core XPath (the navigational part) in first-order logic. Nevertheless, they point out that XPath is generally "a very well-designed" and "useful" language.

Later in this chapter we will describe how XPath has empowered different schema languages with increased expressivity through a highly flexible and effective query mechanism.

Document Type Definition

Initially, the structure of XML vocabularies was described in a machine-readable form using Document Type Definition (DTD) (W3C 1998b), which was adopted from the XML's predecessor- SGML. Although this method is a fundamental part of the markup language, the increasing requirements of modern data model designing soon revealed its limitations, most notably poor expressivity power beyond the structural validation and lack of support for XML namepaces which wourld mark DTDs as rather an old technology in the XML world (Fawcett et al 2012). The XLIFF 2 data model is far beyond the domain of DTDs, especially with the modular nature of the standard which combines multiple namespaces. This technology, therefore, has not been used in our work and is out of scope this research. In the following parts we will review the techniques employed during our research for design and development of validation artefacts for XLIFF 2.

W3C XML Schema

Limitations of DTDs were soon reached by growing number of data models shifting to XML with more advanced integrity constraints, especially in terms of handling keys (Buneman et al 2002; Fan 2005). To address the occurred issues and also to give XML a formal schema language (DTD was originally designed for SGML) W3C (2001) introduced its XML Schema as a recommendation (also referred to as XML Schema Definition, XSD), which aimed to "substantially re-construct and considerably extend the capabilities" of DTDs. Besides validation purposes, XSD may be used for documentation or processing automation as well (van der Vlist 2002). Fawcett et al (2012) consider the followings as major XSD benefits: its XML syntax, support of XML namespaces and enabling programming concepts, such as reusable content and inheritance.

XSD 1.0 is now the most common language for specifying an XML vocabulary, moreover, the concept of Data Type, introduced within the second part of XSD, provides a rich library of data types which is widely used and referenced by other schema languages. Users can apply different restrictions to values of elements or attributes.

In terms of expressing integrity constraints, XSD can only define absolute primary keys and foreign keys. This limitation is due to partial implementation of XPath, where regular expressions are significantly simplified and limited. With respect to functional dependencies, XSD is not capable of describing this type of XML integrity constraints. Although the later version, XSD 1.1 (W3C 2012), did make progress in addressing some of the emerged issues and adding new features, but the applied changes are rather minor as backward compatibility is maintained. Reasoning about its success, however might be too early.

Overall and in the light of XSD's limitations, Vergara-Niedermayr et al (2014) suggest that the schema language is not sufficient on its own to "enable semantically interoperable resources". A W3C XML Schema is provided for XLIFF 2 by the Technical Committee to define relations of the data model nodes. In the following section we will review the technologies we hired to enhance the state-of-the-art of XLIFF 2 validation.

The Document Schema Definition Languages Framework

In addition to formal W3C recommendations for XML specification, there has been a tremendous number of schema languages proposed and developed by different parties; according to Lee and Chu (2000), only as of June 2000 there were "about dozen of XML schema languages". Fawcett et al (2012) , however, report that "most experts agree that there is not a single technique that can cope with every validation rule" and "expect" that combined usage of schema languages will enable complete definition of XML specifications. In order to provide an interoperable platform for interconnection of different languages, the Document Schema Definition Languages (DSDL) (ISO 2001) project was initiated, as described in its specification, "to create a framework within which multiple validation tasks of different types can be applied to an XML document in order to achieve more complete validation results than just the application of a single technology". DSDL is a multipart standard which brings together a number of schema languages and also positions itself as an interoperable framework and extensible platform, flexible to accommodate new validation processes. Among its parts, the current DSDL contains 6 schema languages for various validation purposes. Notably, the 9th part introduces “Namespace and datatype declaration in Document Type Definitions (DTDs)”. For the purpose of our research we used three of DSDL languages, which we will explore in the following sections.

RelaxNG

REgular LAnguage for XML Next Generation (Relax NG) was first developed by an OASIS technical committee (2001), but later was standardised under the DSDL project as *Part 2: Regular-grammar-based validation — RELAX NG* (ISO 2003). Available in two syntaxes, XML and compact, Relax NG is a grammar-based schema languages (like XSD), which performs on the logic of matching documents with the predefined patterns. Unlike W3C XML Schemas, which are claimed to be “extremely complicated”, Relax NG provides a very easy and intuitive XML schema language (Fawcett et al 2012). This simplicity can also be noticed at its limited support of built-in datatypes (Datatypes from other vocabularies such as W3C XML schema can be integrated instead). Van der Vlist (2003) states that structural validation always was the main focus of the development team, which has made Relax NG an efficient schema language, targeting only one problem.

The schema language's popularity has been always growing since it was introduced and today many standards (e.g. ITS 2.0) choose to provide Relax NG as the only or part of official validation solutions. Finally, as mentioned earlier, in terms of expressivity, Relax NG is very limited and can target only some minor functional dependencies with no support for keys at all due to lack of navigational mechanism, yet a very powerful language for XML structure definition with a "solid theoretical basis" (Fawcett et al 2012), backed by the mathematical logic of regular expressions (van der Vlist 2003). Notably, the role of regular expressions is so fundamental that the term is referenced in the name of the schema language.

Schematron

Like the previous schema language, Schematron joined the DSDL framework after it was first created by Rick Jelliffe in 1999 (Lee and Chu 2000). Later in 2004, it was included in DSDL as *Part 3: Rule-based validation — Schematron* (ISO 2004). This schema language offers a different validation approach than other aforementioned methods which allows one to define *assertions* for XML sub trees. In the other words, the schema developer would define rules asserting what and where should be present or absent. Most importantly, Schematron integrates a full implementation of XPath as its navigational mechanism. Usage of XPath enables applying predicates in node selection and, therefore, more precise regular expressions can be expressed. This makes Schematron the most expressive and powerful schema language among all existing ones which can target all types of XML constraints, especially complicated functional dependencies. Such advanced expressiveness is mostly thanks to XPath *node axes* (Dodds 2001) which provide capability to describe complex relations of nodes. Unlike regular tree languages with limited access to parent, children and siblings of a node, XPath axes enable expression of the following relations and paths: child, descendant, parent, ancestor, following-sibling, preceding-sibling, following, preceding, attribute, namespace, self, descendant-or-self, ancestor-or-self. XPath functions, on the other hand, empowers Schematron to define additional relations, e.g. position of a node, or apply advanced constraints, e.g. text analyses.

Another advantage of Schematron is its customizable error messaging which allows creating user-friendly and enriched reports for *broken rules* (van der Vlist 2007). However, some of its useful features (e.g. providing a URI for error reporting), designed for advanced functionalities are defined as “optional” in the Schematron’s specification, which means that implementations are not required to support them.

Finally, neither of Relax NG nor Schematron aims to replace W3C Schema, but actually are meant to be used in conjunction with each other (Fawcett et al 2012). For instance, Schematron rules can be embedded in XSD for better performance. It is generally better to use each schema language for that task of validation, which they perform the best and they were initially developed for.

NVDL

The 4th Part of DSDL, Namespace-based Validation Dispatching Language (NVDL) (ISO 2006), presents a schema language which addresses a completely different and recently-emerged task of XML validation. With XML vocabularies tending to allow integration of other models within them (to enable extensibility), XML documents are often seen to be compound of multiple namespaces (e.g. XLIFF 2) and NVDL aims to facilitate validation of such hybrid XML instances. The other issue that this language targets is applying different schemas to the same document in the case, if the XML vocabulary is defined by various validation artefacts (e.g. XSD and Schematron). Overall, it provides a mapping and bridging mechanism among supported schema languages and, as a crucial part of the DSDL framework, creates an interoperable environment for advanced validation with combined schemas.

Chapter 3

Analysis of XLIFF 2 Data Model

3.1 Introduction

This chapter represents an extended study of XLIFF 2.1 data model, aims to directly address the research question of our work and tends to confirm the initial assumption about possibility of creating a machine-readable version of the XLIFF 2.1 specification based on declarative validation languages. The first and the most essential outcome of this detailed study is identification and extraction of the standard's integrity constraints for further analyses and research. Such collection of scattered XLIFF 2 constraints, being the first step of this work, would enable systematic review and scientific analyses of the data model objects and their interrelations. This study will target the constraints on the individual basis to accurately extract their components required for development, although the constraints had been initially sorted according to conventional XML classification of integrity constraints (see p. 42). On the other hand, documenting the XLIFF 2.1 rules altogether, where every constraint is represented by a detailed and well-studied profile of essential XML properties and analytic information, could be considered a useful and reliable reference for further development of the standard through scientific research.

We will start investigating the research question from exposition of the hired methodology for this part of our research. A short review of the XLIFF background and the evolution of its model will be provided, followed by a detailed discussion of its current (v. 2.1) structure, modules and functionality. We will analyse the standard and point out areas where the demands and common use-cases of the localisation world fall beyond the possibilities of XML, and XLIFF, being an XML vocabulary, had to create solutions to overcome the current limitations of XML. However, these solutions do not violate the XML rules, but rather propose legal and innovational techniques that highlight the feasibility of one of the main aspects of XML- extensibility. We will discuss the validation challenges of the XLIFF 2 "non-traditional" behaviour in Chapter 4 and within the current chapter we attempt to discover and state the synthesised problem for the rest of this research. After describing the standard's structure, we will have a detailed review of the integrity constraints defined over the XLIFF 2 data model. Notably, this study will help us to obtain a deep understanding of the standard and to discover those integrity constraints which are not spelled out in the specification because either they imply from the other rules or represent a common use case in the localisation world, but not recognised from the XML perspective. Finally, XLIFF 2.1 integrity constraints will be categorised into major types (in accordance with the classification we introduced in Chapter 2), where they will be analysed further, enriched with the necessary metadata for development and examined against the selected schema languages to identify potential candidates which are expressive for each constraint.

3.2 Methodology

For deep understanding of XLIFF 2.1 constraints we gathered all the normative parts of the standard through a meticulous review of its specification and then applied appropriate analyses to each and every integrity constraint. The study was designed in line with a sub-area of qualitative methodology, the *constant comparison method*. Seaman (1999) states that the main characteristic of this methodology is *coding*, which he describes as “attaching codes, or labels, to pieces of text which are relevant to a particular theme or idea that is of interest in the study”. Such “coding”, in the context of our research is implemented in form of investigating XLIFF integrity constraints, discovering and exploring their boundaries and connections between them and, finally, identifying the essential XML constituents of each constraints. The further steps of this method is described as follows:

Passages of text are grouped into patterns according to the codes and sub-codes they have been assigned. These groupings are examined for underlying themes and explanations of phenomena.

(Seaman 1999)

The notes and information produced by the analyses helped us to classify the integrity constraints into conventional XML integrity constraints. Categorised constraints further were reformulated in accordance with proposed notation by Saadatfar and Filip (2016) and broken down into predefined components. The study allowed us to form an adequate assessment of the complexity of relations and dependencies of XLIFF elements and attributes, on the basis of which various functionalities of the standard are implemented. This aspect of XML vocabularies, co-occurrence dependencies, was discussed in Chapter 2 (see p. 42) and we will review the XLIFF constraints of this type later in this chapter. Our "coding" of the XLIFF 2.1 specification, on the other hand, led us to shape requirements for the following steps of this work, in particular for the selection of schema languages. Analysing the results of this study, we identified main technical challenges, from the XML perspective, of XLIFF validation which provides vital information for the following parts of this research- effective design of an optimised validation platform.

The XLIFF standard contains a W3C XML Schema for structural validation purposes. This schema language is capable of describing the structure of XML vocabularies and representing the specified datatypes, but does not address other kinds of constraints (e.g. functional dependencies). However, it is the most common and reliable schema language in the XML world and it is intended for structural description of XML vocabularies, interrelations of nodes and definition of their datatypes.

For advanced validation of XLIFF 2.1, we selected the Document Schema Definition Languages (DSDL) framework, which provides a comprehensive set of schema languages for various validation tasks of XML vocabularies. This standardised initiative enables different schema languages to perform in an interoperable platform, which was discussed in detail in Chapter 2. In order to detect the necessary parts of DSDL for our work, we assigned constraints to appropriate DSDL block, conceptually matching the expressivity requirements of the constraints, indicated in the previous step, and expressivity power of the schema languages, as well as their main purposes of development (presented in Chapter 2). Following the conceptual methodology (see p. 16), we chose three parts of the DSDL framework:

1. **RelaxNG-** This schema language can describe the structure of XLIFF nodes, but also covers few constraints of other types. We chose RelaxNG for structural validation of XLIFF and a number of minor constraints (i.e. co-occurrence);

2. **Schematorn-** Rule-based approach of Schematron empowers this part of DSDL to a high level of expressivity of XML constraints. The underlying logic of Schematron makes it capable of, at least theoretically, expressing all types of XML integrity constraints, regardless of their complexity. Schematron was selected for our work as it enables us to target remaining constraints of the standard, which are not expressible in any other XML schema language;

3. **NVDL-** This part is designed for mapping purposes when target XML documents specify various XML namespaces (e.g. XLIFF 2 modules have their own namespaces) or when multiple validation schemas used (e.g. a combination of RelaxNG and Schematron artefacts). As both of mentioned functionalities are required for validation of XLIFF 2, NVDL was also picked to enable the cross-namespace validation and to apply appropriate schema to the corresponding namespace of an XLIFF 2 instance.

We identified other parts of DSDL as not relevant for our goals, however, the aforementioned selection, as our study revealed, represents the sufficient set of tools to satisfy the needs of our work. In the following section of this chapter we will provide a brief history of XLIFF and discover how it evolved to its latest version- 2.1.

3.3 Evolution of XLIFF Data Model

The concept and objectives of the XML Localisation Interchange File Format were introduced and discussed in Chapters 1-2 and in this section we will keep our focus on the technical and structural progress of the standard for better understanding of the foundations of its latest data model. Our review does not include the very first versions, which were not developed under OASIS, due to fundamental differences which make the pre-OASIS XLIFF irrelevant to this research.

Vesrion 1.2

XLIFF 1.2 was the first release to gain status of OASIS standard and, as its specification suggests, the 1.x family is “loosely based on the OpenTag version 1.2” and “borrows from the TMX 1.2”. While TMX was introduced and explained in Chapter 2, exploring OpenTag is out of scope of our research as it is not being maintained any more and is rather the retired ancestor of XLIFF. As the result of XLIFF 1.2 being successfully adopted by a large number of localisation entities, its limitations and issues were soon revealed and reported to the XLIFF TC. Savourel (2014) and Bly (2010) report that the need for a new major release, XLIFF 2, was driven by several factors including specification ambiguities, unorganised and unclear constraints and processing requirements (named “Processing Instructions” in XLIFF 1.x), lack of examples and limited features in comparison with the contemporary state-of-the-art of the localisation tools. The XLIFF Technical Committee applied comprehensive changes to the data model to satisfy the new requirements and fix the existing issues.

Version 2.0

Schnabel et al (2015) report that XLIFF 2 was eventually “redesigned from the ground up”, introducing a modular structure compounded of a compulsory Core and optional Modules. While the 2.0 Core follows the same logic of structural hierarchy (shown in Fig. 3.1) as 1.2, it has been significantly simplified to embrace only essential localisation tasks. XLIFF 2 modules, on the other hand, are supposed to replace some advanced (but not frequently used) features of XLIFF 1.2 and also limit extensibility of the XLIFF structure through support and regulation of common functionalities. Nevertheless, extensibility has been always a core feature of XLIFF to allow integration of user-defined structured data into the standard. Although such approach for data modelling offers wider flexibility to exchange additional information, it comes at the cost of reduced interoperability (Zhu and Wu 2011). Notably, the XLIFF 1.2 specification strongly recommends to avoid using extensions whenever possible. The objective of modular XLIFF 2 data model is to gradually add modules to the upcoming versions (i.e. 2.x) and provide support for newly emerging demands of the localisation industry under the defined namespaces of XLIFF 2. For instance, XLIFF 2.1 will introduce a new module for ITS 2.0, which will be discussed shortly. Overall, comparing legal "extension points", which was decreased from 7 to 3 for elements and from 21 to 7 for attributes (from XLIFF 1.2 to XLIFF 2.0 respectively) can highlight the progress of reduced extensibility in the XLIFF data model. Although, the Metadata module, one of the 8 XLIFF 2.0 modules, is intended to store custom data and generally is preferred over non-XLIFF structures.

Besides clarified constraints and large number of added examples, another significant improvement of XLIFF 2 specification is the strict definition of the standard conformance, lack of which in 1.2 led to interoperability issues in localisation workflows, including misinterpretations and compliance problems (Schanbel et al. 2015). Filip and Wasala (2013) point out “the lack of a formal Conformance Clause” in XLIFF 1.2 which led the XLIFF editors to solve this issue by explicitly defining the application conformance. The research around interoperability issues of XLIFF 1.2 revealed that the depth of the standard adoption, i.e. number of implemented features, varies in a wide range among the localisation tools which implement XLIFF. A detailed study of XLIFF 1.2 support was conducted by Filip and Wasala (2013), where the authors proposed to classify the users, named *XLIFF Agents*, in accordance with their functionality in the workflow and define separate conformance rules for each of them. The proposal, which is based on the generalised Business Process Model, was accepted and implemented in XLIFF 2.0 and as result the standard now divides the users into 5 major subgroups of Agents. The XLIFF 2.0 specification describes an Agent as “any application or tool that generates (creates), reads, edits, writes, processes, stores, renders or otherwise handles XLIFF Documents”. It has been reported that the agent-based approach of XLIFF 2 makes the exchange format suitable for the entire localisation workflow and “lowers the cost of standard-based interoperability” through reduced compliance requirements for agents (Filip and Wasala 2013). XLIFF 2 Agents are classified as Writers, Extractors, Enrichers, Modifiers and Mergers. While the first category, Writer Agents, defines a broader scope of tasks, the other types represent agents with more specific functionality (as their names suggest) in the workflow declared by the specification:

- **Extractor Agents-** encode localizable content from a native content or User Interface format as XLIFF payload;
- **Enricher Agents-** append the extracted XLIFF payload with metadata using the modules or extension points;
- **Modifier Agents-** change the content of an existing XLIFF file in one or another way;
- **Merger Agents-** perform the task of importing the localised XLIFF payload back to the original format.

Although, **Writer Agents** are described as "an agent that creates, generates, or otherwise writes an XLIFF Document for whatever purpose, including but not limited to Extractor, Modifier, and Enricher Agents". The need for this type of agents is driven by the fact, that XLIFF is an exchange format and applications usually "read" the XLIFF files into internal processing formats upon receiving it. Applications will then convert the processed content back into the exchange format for further roundtrip in the workflow and, therefore, representing a Writer Agent. With respect to application conformance of XLIFF 2.0, all Writer Agents are required to produce conformant XLIFF Documents, i.e. *well formed* and *valid*, in order to be an XLIFF 2 compliant Agent. With all XLIFF Agents, involved in an XLIFF-based workflow, throwing only conformant XLIFF documents to the roundtrip and complying to their specific Processing Requirements, a homogeneous exchange environment is guaranteed. Such workflow can be considered interoperable, where the applications can operate independently and are not required to have any knowledge of the other components of the workflow, whether or not they are directly exchanging data. In fact, the only XLIFF Agents required to share specific processing information are Extractors and Mergers, who are expected to have the same knowledge of the native format, from which the initial data was extracted into XLIFF (e.g. HTML in case of web pages). This is necessary to secure seamless merging after the localisation process is over, however, the knowledge is not related to the exchange standard. All the Agents are required to preserve XLIFF-related data, even if they cannot handle it and are advised to preserve the unfamiliar custom data where possible to protect the integrity of the XLIFF documents.

Finally and with regard to the standard backwards compatibility, XLIFF 2.0 has a different compliance declaration and, therefore, is not compatible with XLIFF 1.x. However, the XLIFF 2 family will keep the compatibility among its releases as the XLIFF 2 Core is to remain unchanged.

Version 2.1

At the time of writing this thesis XLIFF 2.1 draft has been approved by the Technical Committee and sent for its first public review, which is an OASIS requirement for Committee Specifications to allow all interested parties to provide comments or feedback to the TC. Except for some editorial changes in the specification text and from the XML point of view, the XLIFF 2.0 Core has not changed and is still defined in the same namespace as its predecessor;
`"urn:oasis:names:tc:xliff:document:2.0"`.
The major breakthrough of this release is considered the Advanced Validation Platform, the outcome of this work, as well as two new modules (Filip 2016)- a modified Change Tracking Module (based on the feedback of implementers and users) in a new namespace, and a brand new module for integration of ITS 2.0, which is defined under OASIS namespace. Notably, other 7 Modules of 2.0 have not been changed. As this work is an official and TC approved part of XLIFF 2.1, we will

keep our focus on this release of the exchange format and will discuss its technical features within this chapter. However, the study is applicable to the Core for the entire XLIFF 2 family and, except for the aforementioned Modules, to XLIFF 2.0 in particular.

3.4 XLIFF 2.1 Structure

In this section we will look into the specification of XLIFF 2.1 and explore its structural blocks, defined fields and nodes, the functionality they perform in the workflow, the roles they represent in the data model and their relation between each other. The study of XLIFF 2.1 hierarchy, for the Core and Modules, will help us to better understand the standard integrity constraints, identify the scope of affected nodes to which constraints apply through the structural relations and investigate the validation and technical aspects of the XLIFF data model. This step is in line with our methodology and will provide valuable "codings" on the XLIFF 2.1 structure, conforming to which is the first and fundamental element of the validation process.

The objective of XLIFF 2 family, as described in its specification, "is to store localizable data and carry it from one step of the localisation process to the other, while allowing interoperability between and among tools". The data model therefore defines a set of fields and properties (XML elements and attributes), within a hierarchical structure, to provide capacity to target various tasks of a localisation process. Earlier in this chapter we explained the multimodularity of the standard which is given by a Core and nine Modules, designed for storage of localizable data and metadata and for advanced functionality respectively. Although the term "multimodular structure", strictly speaking, is rather an abstract notion; from XML point of view the Core (defined in the ''`urn:oasis:names:tc:xliff: document:2.0`" namespace) allows XLIFF 2.1 Modules (defined in specific namespaces, see Table 3.1) to be integrated at certain points and as independent sub-trees, i.e. removing or adding Modules will not change the integrity of an XLIFF 2.1 document. The Modules' independence, however, is restricted to their structures and in terms of functionality Modules are peripheral in relation to the Core. All XLIFF 2.1 users, regardless of the agent type they represent, are required to implement the Core in order to guarantee interoperability while performing common localisation tasks, whereas Modules are left optional and might be hired for the specific use cases they aim to target. The rationale and advantages of this design approach was discussed earlier in this chapter. The XLIFF 2.1 data model, however, allows custom XML substructures to be embedded at few points of the data model. Such extensibility, even though restricted, enables users to expand functionality of the standard for specific needs. However, even the extension mechanism of XLIFF 2.1 specifies some constraints to clarify the application conformance. These rules require the extensions to define keys, e.g. by `xml:id`, within the targeted scope of the Core and confine the key datatype to `NMTOKEN`, the type of native XLIFF keys (see sec. 3.5.1). In the following sections we will explore the structural elements of XLIFF 2.1, but only those attributes which play crucial role in the functionality of each field or Module in general. Later in this chapter other important attributes, from validation perspective, will be introduced, although reviewing the entire package of attributes is out of scope this research.

XLIFF 2.1 Core

The major fields and main logical levels of the XLIFF data model is presented in Fig. 3.1. `<xliff>` is the root element which embraces the entire content of an XLIFF document and specifies such properties (attributes not shown in Fig. 3.1 for demonstration purposes) like the document version and languages. Unlike three lower main levels (file, group, unit), the XLIFF 2.1 root does not represent an extension point (specified by the `<other>` placeholder at legal points, where "*" is the "zero or more" type of occurrence).

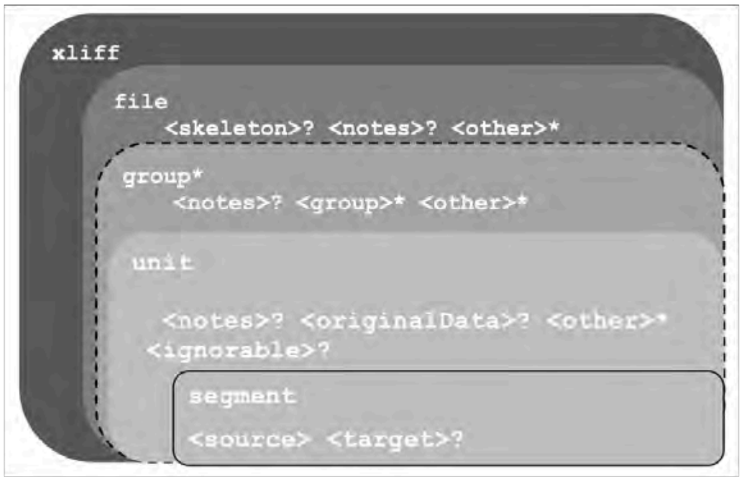

xliff
file
<skeleton>? <notes>? <other>*
group*
<notes>? <group>* <other>*
unit
<notes>? <originalData>? <other>*
<ignorable>?
segment
<source> <target>?

Figure 3.1: Main levels of an XLIFF 2.1 document

The lower level, `<file>`, represents an extracted document, divided into smaller portions of localizable material by `<unit>` and `<segment>`, where the first contains a linguistically logical block of the content (i.e. paragraph) and the latter stores the minimum fraction of the text to be translated. In Chapter 2 we reviewed this process in localisation and argued that the segmentation task is performed through breaking the text down to sentences (Somers 2003) and as result, the 'segment' element usually contains a sentence, represented in the source and target languages, within its two children elements; `<source>` and `<target>` respectively. We consider these four elements, 'xliff', 'file', 'unit' and 'segment' (the "umbrella" element for `<source>` and `<target>`) as the primary fields of XLIFF 2.1 which together represent the minimum valid structure. The optional 'group' element, being recursive, is designed to organize and "group" units with large volume or complicated structure. Its recursivity enables virtually any grouping pattern through nested elements and the only restriction is general XML well-formedness, i.e. parent elements must contain both opening and closing tags of their children elements.

The secondary XLIFF elements, marked by "?" in Fig. 3.1 (representing "zero or one" occurrence type), are listed in Table 3.1, where their content and purpose are explained.

XLIFF Element	Content/Sub-content	Purpose
`<skeleton>`	`<other>` and text	Storage of non-translatable data
`<notes>`	`<note>` and text	Collection of comments
`<originalData>`	`<data>`, `<cp>` and text	Storage of original code
`<ignorable>`	`<source>`/`<target>`	Storage of non-translatable segments

Table 3.1: XLIFF 2.1 Secondary Fields

Finally, the inner most block of XLIFF 2.1 structure is embedded within 'source'/ 'target' pairs, which have the same structure: text and a set of *inline elements* (placeholders), defined for enrichment of the localizable text with metadata, annotations, localisation information, processing instructions or other markup/-code pieces from the original format, which can operate in alignment with other XLIFF Core elements, e.g. `<originalData>` or `<ignorable>`. Although, to be precise, the 'target' field is optional as at least one type of XLIFF 2 Agents, Extractors, may not use this field ('source' is used for the extracted content), but in practice Extractors usually implement this field by duplicating the sibling 'source' content or as an empty field. The XLIFF inline content, illustrated in Table 3.2, represents the most important block of XLIFF fields from the validation perspective. These elements can be reviewed and classified by few factors, such as content (empty and recursive) or usage (standalone or in pair), but for the objectives of this research we divide them into three subgroups with unequal number of members. Elements of each subgroup share the same functionality in our classification. We distinguish the 'cp' element in a separate group as it has a unique role among the inline content. The next group of elements, 'ph', 'pc' and 'sc/ec' pair perform the same task, replacing original code (e.g. an HTML tag), but are designed for different use cases at three levels: standalone code, spanning code and not well-formed

XLIFF Element	Name	Content	Purpose
`<cp>`	Code Point	Empty	Storage of invalid XML characters
`<ph>`	Placeholder	Empty	Placeholder for standalone original code
`<pc>`	Pair Code	Text and inline content	Placeholder for spanning original code
`<sc>/<ec>`	Start/End Code	Empty	Placeholder for spanning original code which is not well-formed
`<mrk>`	Marker	Text and inline content	Spanning annotation marker
`<sm>/<em>`	Start/End Marker	Empty	Annotation marker for not well-formed spanning

Table 3.2: XLIFF 2.1 Inline Content

code. The latter provides a mechanism for conversion and storage of non-XML formats, yet another business requirement for XLIFF to cover all the common scenarios in localisation workflows as the Rich Text Format (RTF), which unlike XML allows tags to overlap, is a usual platform for storage of content to be localised. The last group, 'mrk' and 'sm'/'em' pair follow the same logic of 'pc' and 'sc/ec', but are in fact "markers" for metadata enrichment and various annotations. Except for 'cp', all other inline elements can refer to other XLIFF 2.1 nodes or modules in some form. During the current and following chapters we will present and explain the defined functional dependencies to regulate the usage of "overlapping" nodes in XLIFF 2.1, however the specification clearly sets the conformance such, that Agents must not use XLIFF pair codes for standalone codes and overlapping codes for standalone XLIFF fields.

An XLIFF 2.1 document is demonstrated in Listing 3.1, where the localizable data, with translation and original code is represented in the XLIFF format. We will explore the functionality and relations of XLIFF objects in the following sections, after a short review of XLIFF 2.1 Modules.

Listing 3.1: An XLIFF 2.1 Instance

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xliff xmlns="urn:oasis:names:tc:xliff:document:2.1"
    xmlns:mtc="urn:oasis:names:tc:xliff:matches:2.0"
    version="2.0" srcLang="en" trgLang="ru">
    <file id="f1">
        <notes>
            <note>Alice's Adventures in Wonderland</note>
        </notes>
        <unit id="u1">
            <mtc:matches>
                <mtc:match ref="mrk1">
                    <source>Alice</source>
                    <target>Алиса</target>
                </mtc:match>
            </mtc:matches>
            <notes>
                <note>Chapter 5</note>
            </notes>
            <originalData>
                <data id="d1">&lt;i></data>
                <data id="d2">&lt;/i></data>
            </originalData>
            <segment id="seg1">
                <source>"I could tell you my adventures—
                    beginning from this morning," said
                    <mrk id="mrk1" type="mtc:match">Alice</mrk>
                    a little ...</source>
                <target>Я расскажу все, что случилось со
                    мной сегодня с утра, — сказала неуверенно
                    <mrk id="mrk1" type="mtc:match">Алиса</mrk>.
                </target>
            </segment>
            <ignorable id="ig1">
                <source> </source>
            </ignorable>
            <segment id="seg2">
                <source>it's no use going back to
                    <pc id="pc1" dataRefStart="d1"
                        dataRefEnd="d2">yesterday</pc>, because
                  I was a different person then.</source>
                <target>— А про <pc id="pc1"
                    dataRefStart="d1" dataRefEnd="d2">
                    вчера</pc> и рассказывать не буду,
                    потому что тогда я была
                    совсем другая.</target>
            </segment>
        </unit>
    </file>
</xliff>
```

XLIFF 2.1 Modules

The process of localisation might be complex and sometimes requires additional functionalities. In the XLIFF 2.1 specification, an XLIFF Module is described as follows:

A module is an optional set of XML elements and attributes that stores information about a process applied to an XLIFF Document and the data incorporated into the document as result of that process.

Table 3.3 lists XLIFF 2.1 Modules, the namespace each is defined in and the corresponding XML prefix. We will now review the structure, basic functionality and interaction with the Core for each XLIFF 2.1 Module.

No	Name	Prefix	Namespace
1	Translation Candidates	`mtc:`	`*:matches:2.0` [1]
2	Glossary	`gls:`	`*:glossary:2.0`
3	Format Style	`fs:`	`*:fs:2.0`
4	Metadata	`mda:`	`*:metadata:2.0`
5	Resource Data	`res:`	`*:resourcedata:2.0`
6	Change Tracking	`ctr:`	`*:changetracking:2.1`
7	Size and Length Restriction	`slr:`	`*:sizerestriction:2.0`
8	Validation	`val:`	`*:validation:2.0`
9	ITS	`itsm:`	`*:itsm:2.1`

Table 3.3: XLIFF 2.1 Modules

[1] *: urn:oasis:names:tc:xliff

1. **Translation Candidates Module-** Arguably the most widely implemented module, Translation Candidates (with prefix `mtc:`), is used in Listing 3.1 to provide translation suggestions for the specified part of the text. This Module is rooted at `<mtc:matches>` and its scope is ‘unit’, i.e. it contains potential translations for portions of the ‘unit’, to which it is attached. For effective storage the content of the candidates is wrapped by the XLIFF 2.1 Core elements of ‘source’ and ‘target’ (and inline elements consequently), which can be enriched by Core ‘originalData’ (for inline mapping) and the Metadata Module (for additional metadata). This Module, in fact, is consist of 3 namespaces and its structural tree is shown in Listing 3.2. The “+” sign indicates that the element must appear at least once and “1” represents one and only one occurrence. Each ‘mtc:match’ element must specify a `ref` attribute which points to the span of text within the same ‘unit’, to which the candidate applies. For instance, in Listing 3.1 the content of `mrk` element, “Alice”, is the target.

Listing 3.2: Structure of Translation Candidates Module

```
<mtc:matches>

    <mtc:match>+

        <mda:metadata/>?

        <xlf:originalData/>?

        <xlf:source/>1

        <xlf:target/>1

    </mtc:match>

</mtc:matches>
```

2. **Glossary Module-** This Module provides a mechanism to create simple glossaries and store it within XLIFF 2.1 documents. A glossary, as its tree from Listing 3.3 is presented, can have multiple entries where each entry can contain translations and/or definitions, however, at least one of the 'translation' or 'definition' must be present for the Module to be valid. Like the previous module, the glossary entries could target a span of text in the Core through a `ref` attribute on 'gls:glossEntry' or 'gls:translation' elements, although they are not required to. In case of interaction with the XLIFF Core, the Module must appear at the 'unit' level and it represents an extension point inside the Module itself ('other' element in Listing 3.3).

Listing 3.3: Structure of Glossary Module

```
<gls:glossary>
    <gls:glossEntry>+
        <term/>1
        <translation/>*
        <definition/>?
        <other>*
     </gls:glossEntry>
</gls:glossary>
```

3. **Format Style Module-** With only two attributes of `fs` and `subFs`, Format Style is the lightest Module in XLIFF 2.1.The attributes can be added to the Core elements, especially inline codes, to provide HTML-like metadata for formatting information of portions of text to ease generation of web pages. While `fs` values are set to predefined HTML elements they replace, e.g. "head", "strong", `subFs` can be used to enable HTML attributes as well. Pairs of name/value for intended HTML attributes must be set according to the introduced pattern in the specification, where pairs are separated by backslashes (\) and values follow names after a comma (,). The usage of this module is shown in Listing 3.4 where the HTML 'img' tag, replaced

by ‘ph’ in XLIFF, defines two attributes of ‘src’ and ‘width’ with values of “image.jpg” and “68” respectively. The validation challenge of this module is driven by the fact, that it does not declare any elements and, therefore, will require additional steps for integration in the XLIFF 2.1 validation platform which will be discussed in Chapter 4.

Listing 3.4: Format Style Module

```
<ph id="ph1" fs:fs="img"
    fs:subFs="src,image.jpg\width,68"/>
```

4. **Metadata Module-** The Metadata Module represents a flexible structure for storage of custom metadata, for instance the original codes of native formats. As shown in Listing 3.5, the ‘meta’ element contains the untranslatable text and recursive ‘metaGroup’ can be used for nested and hierarchical structures. This module can appear at all of three extension points of the Core to provide the required metadata for each level.

Listing 3.5: Structure of Metadata Module

```
<mda:metadata>
    <mda:metaGroup>+
        <mda:metaGroup/>*
        <mda:meta/>+
    </mda:metadata>
</mda:metadata>
```

5. **Resource Data Module-** External resources can be referenced and stored using this module. The Module might appear in different levels and even those instances at the 'unit' level can reference higher sub-trees of the module using the `ref` attribute of 'resourceItemRef' element. The Module's other element, 'resourceItem', on the other hand, either stores the resource data, e.g. contextual information of the text, or holds a reference to it for 'source' and 'target' (defined in the Module namespace, not Core). The structure of Resource Data Module is presented in Listing 3.6 and we will discuss its functional dependencies in the following sections as the Module performs a complex functionality.

Listing 3.6: Structure of Resource Data Module

```
<res:resourceData>
    <res:resourceItemRef/>*
    <res:resourceItem>
        <res:source>?
           <other/>*
        </res:source>
        <res:target>?
            <other/>*
        </res:target>
        <res:reference/>*
    </res:resourceItem>
</res:resourceData>
```

6. **Change Tracking Module-** Storing revisions and tracking previous information of the files is especially useful in case of XLIFF, where the file will be processed and exchanged by multiple applications. This module is intended to provide such mechanism to track the changes made on the Core elements or attributes. As mentioned earlier, the Module has been redesigned for 2.1 after implementers reported limitations of the 2.0 version, which did not allow to track the inline content. The new structure, presented in Listing 3.7, enables this demand by providing 4 different options for the substructure of the 'ctr:item' and 'ctr:simpleItem' elements, suitable for those Core elements which are permitted to be tracked using the 2.1 version. Since it is not a standalone module, it interacts with the Core through the 'appliesTo' attribute of `<ctr:revisions>` which can have the following values, specifying the tracked Core element under this "revision": note, unit, segment,

ignorable, source or target. Notably, it is allowed at ‘group’ and ‘file’ levels to ensure that all the ‘notes’ can be tracked. Depending on the value of the ‘appliesTo’ attribute, the “item” children are restricted to either of 4 possible content, which are elements from the ‘xlf’ namespace or plain text. We will return to this module later in this chapter to introduce its integrity constraints. However, as the 2.1 Change Tracking is rather a complex data model itself, we will discuss its functionality and implementation of the constraints in Chapter 4 . Another aspect of the Module is its compliance requirements and backwards compatibility; Modifier Agents, which are the main users of this module, are required to store the revising data within the Module and are advised to treat the previous version of the module, 2.0, as an extension since the two versions are not compatible.

Listing 3.7: Structure of Change Tracking Module

```
<ctr:changeTrack>
    <ctr:revisions>+
        <ctr:revision>+
           <ctr:item/>+
           <ctr:simpleItem/>?
        </ctr:revision>
    </ctr:revisions>
</ctr:changeTrack/>
```

7. **Size and Length Restriction Module-** Defining restrictions on representation and storage size of content is common in localisation projects and this module intends to provide a capacity for such information to be stored within XLIFF documents. The restrictions can be profiled using the 'slr:profiles' field and the related data can be stored in the separate 'slr:data' element (demonstrated in Listing 3.8). The Module also specifies some attributes that can be added to Core elements for specific targets of defined restrictions. However, this module is considered a simple data model from the validation point of view and detailed review of its functionality is out of scope here, despite being an important part of the standard in general.

Listing 3.8: Structure of Size and Length Restriction Module

```
<slr:profiles>
    <slr:normalization/>?
    <slr:other/>*
</slr:profiles/>
<slr:data/>
```

8. **Validation Module-** The Validation Module enables application of linguistic rules to the textual content of ‘target’ elements. Rules can be applied to all the main levels of the Core depending on the extension point they appear in. The Module allows definition of validation rules through a number of attributes on the ‘val:rule’ element, as shown in Listing 3.9. For instance, to require all of the ‘target’ strings to start with a backslash (\) in a certain ‘unit’ one can add this rule at the unit level: `<val:rule startsWith="\"/>`. XML validation aspects of this module will be discussed in Chapter 4.

Listing 3.9: Structure of Validation Module

```
<val:validation>
    <val:rule/>+
</val:validation>
```

9. **ITS Module-** We introduced W3C ITS 2.0 in Chapter 2 and argued that it has been widely used as an extension for XLIFF 2.0 metadata enrichment. In order to enable an interoperable environment for the interaction of two technologies, XLIFF 2.1 provides a new module to host those functionalities of ITS 2.0 that cannot be performed by the existing XLIFF fields and properties. As the original vocabulary, ITS 2.0, essentially is a set of attributes, the ITS Module also is implemented by a large number (38) of attributes and 2 sub-tree structures, presented in Listing 3-10, for the ITS rules which are not expressible using attributes. The Module specifies a complex data model and interaction with the Core, as well as other XLIFF 2.1 Modules, which makes the validation task a challenging one. Integrity constraints of this module, defined to replicate original ITS functionalities, will be introduced later in this chapter.

Listing 3.10: Structure of ITS Module

```
<itsm:locQualityIssues>
    <itsm:locQualityIssue/>+
</itsm:locQualityIssues>
<itsm:provenanceRecords>
    <itsm:provenanceRecord/>+
</itsm:provenanceRecords>
```

To summarize the structural analyses of XLIFF 2.1 and highlight important validation points, we conclude that the standard's structure is driven by the organisation of current localisation workflows and common practices of the localisation industry in general. This industry bias, as result, has led XLIFF to distance from some of XML concepts and to define alternatives in the areas where "traditional XML" does not provide specific solutions. These "anomalies" are caused by three major factors:

- XLIFF is bilingual- XML elements can use the naive `xml:lang` attribute to specify the natural language (i.e. `xml:lang="en"` for English) of the content, but XLIFF needs two *properties* (attributes) to track both source and target languages. Consequently, and if the source and target languages have different directionalities (i.e. Right-to-Left and Left-to-Right), the `xml:dir` would not be able to address them both alone. However these XML attributes ('lang' and 'dir') enable internationalisation features for XML, they are not compatible with bitextual content. We will discuss the XLIFF properties (attributes) later in this chapter as they directly define functional dependencies of the standard and are less relevant to structural analyses;
- XLIFF's logical levels are rather independent- This directly affects the scope of XLIFF 2.1 primary keys and foreign keys which will be reviewed in the

following section. Besides, the supported recursivity of certain XLIFF fields complicates the defined scope by increasing its depth virtually to unlimited level;

- XLIFF provides support for non-XML structures- We already showed the basics of this mechanism and pointed out the needs behind such support. However, the functional dependencies on the “overlapping” codes will be presented in the following.

3.5 Integrity Constraints of XLIFF 2.1

After studying the XLIFF 2.1 “building blocks” and their functionalities, in this section we will investigate the integrity constraints of the standard. So far in this chapter we explained how XLIFF 2 represents the common hierarchy of localisation process by its structure (main levels of the data model) and highlighted the standard’s unique features from the XML perspective. This section will present how the features of the standard are defined and regulated through the specified integrity constraints of various types. We will discuss and analyse all of the defined constraints to facilitate the next step of our work with enough information for development of the validation platform (in agreement with the methodology of this chapter). This section will explore constraints by the classification, specified in Chapter 2, which consists of primary keys, foreign keys, business rules and progressive constraints.

3.5.1 Primary Keys and Foreign Keys

We reviewed two types of XML key constraints in Chapter2; absolute and relative, with absolute keys being a special case of relative keys, where the level of relativity is 0. In other words, the level of relativity defines the *scope* of the key, within which it is unique or uniquely refers to a key if it is a foreign key. The XLIFF 2.1 data model specifies keys with various and even combined levels of relativity based on the usage (29 keys and foreign keys defined for XLIFF 2.1 Core and Modules in total).

Before analysing the standard's keys, we will explore this category of XML Integrity Constraints in detail and discuss the existing notations and concepts. XML specifies a native attribute (in the XML namespace) for identification of elements inside a data set. The `xml:id` attribute, as a W3C Recommendation, aims to provide a unified and XML-specific identifier to conceptualize the mechanisms used in DTD and XML Schema by creating a supported by XML 1.0 and

XML 1.1 attribute type; ID (implemented in W3C XML Schema as `xsd:id`, a *simple datatype* that can be leveraged in RelaxNG as well). Annotating an XML vocabulary element with `xml:id`, on the other hand, creates certain limitations, mostly because it is set to be unique within the entire document. In other words, the native XML ID mechanism is designed only for absolute keys and if a data model requires relative keys, it must declare new attributes for this purpose and declare them as such. The other constraint of the concept is that an XML element can define only one 'id' attribute which causes limitations when relative keys are needed. Although XML does not define any particular attribute for foreign keys, but this datatype was initially introduced by DTD and later implemented in XML Schema (and consequently available in RelaxNG) as `xsd:IDREF`, which can be assigned to any attribute. Like keys, foreign keys of `xsd:IDREF` type are only applicable to absolute ones. The vast majority of keys and foreign keys of the XLIFF 2.1 data model are relative and therefore it defines its own attributes for keys and foreign keys and, as the XLIFF 2.1 specifies, "XLIFF Documents do not follow the usual behavior of XML documents when it comes to element identifiers". To allow a robust referencing and reliable navigation across different levels of the document, XLIFF provides a mechanism, *Fragment Identification* (FragID), as a unified format for referring to XLIFF defined elements with keys. This notion is also driven by another constraint of the ID datatype which states that each element can have up to one identifier and, in this way, navigational expressions can be created to point to specific elements, which can be useful for both internal (for foreign keys) and external access. The FragID expressions specify the path towards the targeted element (`note`, `data`, `segment`, `ignorable` and inline elements are defined as the lowest level) starting form the highest level of the data model, `file` element, and passing through `group` and/or `unit` elements down to the referenced node, whether it is in the Core or any of the Modules. The following notion, for instance, is the FragID expression which "points" to the first `<data>`, child of `originalData` element in Listing 3.1;

`#f=f1/u=u1/d1`.

We will use the FragID notions in the design of our error reports to provide useful, standardised and precise information on the caught errors which will ease the process of post-validation troubleshooting. The related discussion will be presented in Chapter 4.

Noteworthy, another major difference of XLIFF 2.1 keys and foreign keys with XML `ID` and `IDREF` concepts is the assigned type; while XML keys are strictly of *NCName* (Non-colonised name) type, XLIFF 2.1 keys have various types, but not NCName. In the following sections we will review XLIFF 2.1 keys and foreign keys in the increasing order of relativity level.

XLIFF 2.1 Primary Keys

Table 3.4 presents the collection of primary keys we extracted from the specification alongside with the analytic validation-related information our study revealed. For reference convenience throughout the rest of this work, we labelled the keys with identifiers. The next three columns contain components of each key followed the level of relativity each key defines, where “A” stands for “Absolute Key” and the number of “R”s, for “Relative Keys”, indicates the depth. Potential candidates which can express the key is represented in “Schema Language” column and, finally, its datatype is stored in the last column. While XML keys are usually identifiers, three of XLIFF 2.1 keys, K7, K12 and K14, unlike the rest, do not represent an identification tag for the host elements and are included in this category due to similarity to this type of integrity constraints. The role of ‘order’ attribute (K7) and the complete functional dependency over this property will be discussed in the following section, but with regard to the part of the constraint which is considered an XML key, its value must be unique among ‘target’s within the scope of the enclosing ‘unit’. However, strictly speaking, since the attribute has default value and ‘target’s do not specify any other id, ‘order’ can be considered the explicit identifier of a ‘target’. Two other keys are both in the Change Tracking Module and their values represent additional *non-replicable* information on type and version of their host elements, which do specify their own ids. These attributes must be unique in the scope of the Module.

Table 3.4: XLIFF 2.1 Primary Keys

ID	Node(s)	Attribute	Scope	Relativity	Schema Language	Datatype
K1	file	id	xliff	A	XSD SCH	NMTOKEN
K2	group	id	file	R	XSD SCH	NMTOKEN
K3	unit	id	file	R	XSD SCH	NMTOKEN
K4	note	id	file group unit	R/RR	SCH	NMTOKEN
K5	data	id	unit, mtc: matches	RR/ RRR	XSD SCH	NMTOKEN

ID	Node(s)	Attribute	Scope	Relativity	Schema Language	Datatype
K6	segment, ignorable, (mrk, sm, pc, sc, ec, ph) in source, (mrk, sm, pc, sc, ec, ph) in target without correspondent in the sibling source	id	unit	RR	SCH	NMTOKEN
K7	target	order	unit	RR	SCH	Positive Integer
K8	mtc:match	mtc:id	mtc: matches	RRR	SCH	NMTOKEN

ID	Node(s)	Attribute	Scope	Relativity	Schema Language	Datatype
K9	gls:glossEntry and gls:translation	gls:id	gls: glos-sary	RR/ RRR	SCH	NMTOKEN
K10	mda:metadata and mda:metaGroup	mda:id	mda: metadata	RR/ RRR	SCH	NMTOKEN
K11	res:resourceItem and res: resour-ceItemRef	res:id	res: resource-Data	RR/ RRR	SCH	NMTOKEN
K12	ctr:item and ctr:simpleItem	ctr:property	ctr: change-Track	RR/ RRR	SCH	String
K13	ctr:revisions, ctr:revision, ctr:item and ctr:simpleItem	ctr:id	ctr:revision	RR/ RRR	SCH	NMTOKEN

ID	Node(s)	Attribute	Scope	Relativity	Schema Language	Datatype
K14	ctr:revision	ctr:version	ctr: revisions	RR/ RRR	SCH	NMTOKEN
K15	itsm: locQuality-Issues and itsm: provenanceRecords	itsm:id	file group unit	RR/ RRR	SCH	NMTOKEN

As shown in the table, the `id` attribute, defined in the XLIFF 2.0 namespace, is the most frequently used key within the Core elements. Its type is of NMTOKEN (Name Token), which unlike NCName that limits the starting character to letter, underscore and colon, does not define any restrictions on the beginning character. The attribute has different levels of relativity based on the parent element it is used for. The other key for XLIFF Core is the `order` attribute, used in 'target' to specify the element's sequence within the `unit`, if the "targets" are disordered. This attribute has optional occurrence and is a positive integer. Six of XLIFF 2.1 Modules, on the other hand, specify their own keys which is also named `id`, but is defined in Translation Candidates, Glossary, Metadata, Resource Data, Change

Tracking and ITS module namespaces. Although, Change Tracking Module defines two additional to 'id' keys, one of which is of string datatype.

Beginning from the very top level (and the only XLIFF 2.1 absolute key), `id` is used in `file` elements to uniquely represent them within a document. Further down the "XLIFF tree", `<unit>` and `<group>` keys (id) are required to be unique within the enclosing 'file', or, in other words, the level of relativity is equal to 1, which is as many steps as required to reach these nodes from the root element. Although, strictly speaking, due to recursivity of `group` elements the number of steps technically might be more, but we define "steps" in accordance with the XLIFF specification, where the main levels of an XLIFF 2.1 document are "file" and "unit". The next group of keys, K4-K7 in Table 3.4, are required to have unique identifiers within the scope of their ancestor, the `unit` element and, therefore, are at the second level of relativity (i.e. relative to 'unit's, which are relative to 'file's). Even though 'data' and 'note' elements appear within a wrapper element, 'originalData' and 'notes' respectively, they are still and in fact relative to 'unit' as those wrappers can occur only once in each 'unit'. However, those 'note' elements which appear at the 'file' level, are only relative to this level and thus have the first order of relativity in this case. The special case of 'data' keys with 3rd order will be discussed shortly.

Although expanding the scope of identifiers uniqueness to the entire document (i.e. only absolute keys) would significantly simplify the data model and "scientifically" is the desirable approach, this constraint, limiting the scope to the 'unit' level, is purely an industry driven rule. The reason behind such requirement is that "Units are one of the basic objects in any localisation tool" (Savourel 2013), which was actively and in detail discussed by the XLIFF Technical Committee during the standard development. Applying this restricted scope to the XLIFF 2 data model leads to number of complex and entangled integrity constraints, including many elements to share keys and inline elements to duplicate identifiers in `<source>`/`<target>` pairs. We will discuss the duplication subject later as it is rather a business rule, but with regard to keys, we identify the following elements to share the same key (K6), `id` attribute, within the enclosing `unit` element: segment, ignorable, inline codes (mrk sm pc sc ec ph) in source, inline codes (mrk sm pc sc ec ph) in target without corresponding element in source. Notably, all of the specified nodes are accessible by XPath expressions, which will be discussed in the next chapter.

Finally, the rest of XLIFF 2.1 keys, K8-K15, are defined for Modules and level of their relativity depends on the extension point where the Module is inserted. With 'unit' and 'group' being the usual level for most of the Modules, these keys have 3rd level of relativity. Those 'xlf:data' elements which appear in the Translation Candidates Module are also at the same level. It is worth mentioning that like the depth of relativity with 'group' and 'unit', our interpretation of XLIFF 2.1 Modules' keys being of 3rd level is based on the logic of XLIFF data model, even though the actual level might vary depending on which of the XLIFF Core nodes a module occurs in. However, from the Module's namespace perspective all the Modules' keys are absolute, but from the XLIFF document point of view modules are embedded in the Core.

With respect to the implementation of XLIFF 2.1 keys, only K1-K3 and K5 are expressible in W3C XML Schema and others, because of optional id attribute, shared key or conditional keys can be described using Schematron rules only. However, none of the available keys for XSD is implemented and therefore is to be developed in Schematron. RelaxNG does not provide key definition mechanism, although the `xsd:ID` of XSD datatypes can be assigned to attributes.

XLIFF 2.1 Foreign Keys

The complete list of XLIFF 2.1 foreign keys is presented in Table 3.5. Each foreign key is given an identifier and then the constraint is represented by the name of the attribute, elements they can appear in, the referenced element and the scope of each one. The last column contains some notes for the development stage which can be helpful since all of the XLIFF 2.1 foreign keys will be expressed in Schematron.

Table 3.5: XLIFF 2.1 Foreign Keys

ID	Attribute	Used in	Refers to	Scope	Type	Schema	Notes
FK1	copyOf	ph, pc, sc, ec	ph, pc, sc, ec	unit	RR	SCH	This implies an FD that elements must be of the same type
FK2	dataRef	ph, sc, ec	data	unit	RR	SCH	
FK3	dataRefEnd	pc	data	unit	RR	SCH	
FK4	dataRefStart	pc	data	unit	RR	SCH	
FK5	ref	mrk, sm	note	unit	RR	SCH	This attribute can have other roles than foreign key
FK6	startRef	ec, em	sc, sm	file	R	SCH	
FK7	subFlows	ph,sc, ec	unit+	file	R	SCH	IDREFS
FK8	subFlowsEnd	pc	unit+	file	R	SCH	IDREFS
FK9	subFlowsStart	pc	unit+	file	R	SCH	IDREFS
FK10	mtc:ref	mtc:match	segment, mrk, pc	unit	RRR	SCH	

ID	Attribute	Used in	Refers to	Scope	Type	Schema	Notes
FK11	gls:ref	gls: glossEntry, gls: translation	segment, mrk, pc	unit	RRR	SCH	
FK12	res:ref	res: resourceItemRef	res: resourceItem	file	R	SCH	
FK13	itsm: locQualityIssuesRef	mrk, sm	itsm: locQualityIssues	unit	RR	SCH	
FK14	itsm: provenanceRecordsRef	file, goup, unit, mrk, sm, mtc:match and ctr:revision	itsm: provenanceRecords	file/ group/ unit	R/ RR	SCH	This attribute can have different levels of relativity

Table 3.5 illustrates absence of absolute foreign keys in the data model. Like primary keys, the XLIFF foreign keys do not follow the `IDREF` concept of XML;

the scope of referencing is ‘file’ or ‘unit’ and all are of NMTOKEN type, with the exception of `xlf:ref- anyURI` and `mtc:ref, gls:ref- IRI`, because they can perform various roles depending on the functionality they are engaged in (functional dependency). At the first level of relativity XLIFF has five properties, FK6-FK9, FK12 and FK15, which function in the scope of ‘file’. It was mentioned earlier in this chapter that XLIFF facilitates non-well formed codes by the ‘sc/ec’ and ‘sm/em’ pairs. However, to regulate behaviour of these elements and ensure the proper usage, the specification defines a wide spectrum of constraints, one of which is of foreign key type. Ending tags, i.e. ‘ec’ and ‘em’, declare their corresponding starting tags, i.e. ‘sc’ and ‘ec’, using the ’startRef’ attribute (FK6). The next attribute at the ‘file’ level, ‘subFlows’, represents an `IDREFS` as it refers to more than one element, in this case a list of ‘unit’ identifiers, indicating the order of sub-flow of inline codes. FK8 and FK9 perform the same feature in pair for ‘pc’ elements, which implies a business rule that they must be used together only. FK15 is the last key which can be defined at the ‘file’ level and links the host element to the element in ITS Module which contains information about the provenance record of the content. However, it is at the first level only when used in ‘file’. Finally, FK12 references a resource which is located at the ‘file’ level.

The rest of Core foreign keys, FK1-FK5, are defined at the 'unit' level. While 'dataRef' and the pair of 'dataRefStart'/'dataRefEnd' can be used by inline elements to reference the code they are replacing in XLIFF and which is stored in 'data' elements, 'copyOf' directly points to the inline code which is being replicated. These two foreign keys also imply a functional dependency each; 'dataRefStart' and 'dataRefEnd' must be used together and an inline element can be only copy of the same kind of inline code. FK5 defines a way for inline markers, 'mrk' and pair of 'sm'/'em', to refer to 'note' elements within the same 'unit', although this attribute can contain reference to external sources as well.

The remaining XLIFF foreign keys are defined in Modules' namespaces. FK10 and FK11 associate the corresponding module fields with a span of text inside the Core to which they apply (for translation candidate or term definition) and thus are defined in the scope of 'unit'. Other Module foreign keys are internal pointers and perform on the reversed logic; associating Core elements with Modules. The foreign key of the Size and Length Restriction Module, FK13, points to an element defined in a custom namespace and therefore it cannot be implemented or enforced by XLIFF standard, however this constraint must be met by Agents who implement this feature. Finally, FK14 allows annotation markers, ‘mrk’ and ‘sm’, to refer to an element in ITS Module for the report on localisation quality issue, a native ITS 2.0 datacategory covered by the module in XLIFF 2.1. With respect to development of XLIFF foreign keys, W3C XML Schema allows a restricted usage of XPath, where *XPath Predicates* are not implemented and therefore relative foreign keys are not expressible in this schema language. Instead, and to run the prerequisite constraints first, the XLIFF 2.1 foreign keys can only be validated by Schematron.

3.5.2 Business Rules

Unlike the previous category of Integrity Constraints, XML business rules have a very broad domain and rather vague definition and conceptualisation. As a result, analyses of this type of constraints is difficult in the lack of clearly defined components and agreed synthesis. However, we argued in Chapter 2 that many of such rules can be expressed by XML co-occurrence constraints to define various functionalities for nodes of an object model. The deep study of XLIFF 2.1 business rules helped us to identify co-constraints, hired by the standard to apply required business rules, and also to track a general pattern in those constraints. We gathered 37 rules from the Specification which can be considered as co-occurrence constraints. Table 3.6 demonstrates this collection for XLIFF 2.1, where each row conforms to the following pattern: “Variation of a Subject Node defines valid variations of the Object Node” and the table is organised according to this convention. The first column assigns an identifier to rules which start with “CC” (Co-occurrence Constraint) and does not follow any explicit order. We generalised the variations of subject nodes under 3 options: the node’s presence, absence or its specific value (within double quotation marks), where variations of objects are defined by the previous values and “Allowed” which means that the object is allowed to be present, but not required to. The business rules can be considered as logical conditional (if ... then ...) and biconditional (if and only if) statements, for instance, the first row of Table 3.6 could be read as: “if ‘target’ element is present, the ‘trgLang’ attribute of ‘xliff’ must be present” (attributes are preceded by “@” symbol in accordance with XPath expressions). This rule, CC1, represents a conditional statement, marked by (1) in the last column, which indicates that the rule does not apply in the reverse order as presence of ‘trgLang’ attribute does not make ‘target’ elements compulsory. CC3 represents an example of a biconditional statement: “if and only if the ‘isolated’ attribute is set to ‘yes’, an ‘id’ attribute must be present”, which is a part of the functional dependency over ‘sc’/‘ec’ usage.

Table 3.6: XLIFF 2.1 Co-occurrence Constraints

ID	Subject Node	Variation	Object Node	Variation
CC1	target	Present	xliff/@trgLang	Present (1)
CC2	skeleton children	Absent	skeleton/@href	Present
CC3	ec/@isolated	“yes”	self/@id	Present
CC4	ec/@isolated	“no”	self/@startRef	Present
CC5	ec/@isolated	“yes”	self/@dir	Allowed
CC6	@subState	Present	self/@state	Present (1)
CC7	@subType	Present	self/@type	Present (1)
CC8	@copyOf	Present	self/@dataRef or the pair of ‘dataRefStart’ and ‘dataRefEnd’	Absent
CC9	@canReorder	“no” or “firstNo”	self/@canCopy and self/@canDelete	“no”(1)
CC10	sm/@type, mrk/@type	“generic”	self/@translate	Present (1)

ID	Subject Node	Variation	Object Node	Variation
CC11	sm/@type, mrk/@type	“mtc: match”	self/@ref	Absent (1)
CC12	@dispStart	Present	self/@dispEnd	Present
CC13	@equivStart	Present	self/@equivEnd	Present
CC14	@subFlowsStart	Present	self/ @sub-FlowsEnd	Present
CC15	@dataRefStart	Present	@dataRefEnd	Present
CC16	@mtc:subType	Present	self/@mtc:type	Present (1)
CC17	ec/@isolated	“yes”	self/@fs:fs	Allowed
CC18	@fs:subFs	Present	self/@fs:fs	Present (1)
CC19	res:source and res:target chil-dren	Absent	parent:: res:resourceItem/ @res:mimeType	Present
CC20	res:source chil-dren	Absent	self/ @res:href	Present
CC21	res:resourceItem/ @context	“no”	child: res:source	Present (1)

ID	Subject Node	Variation	Object Node	Variation
CC22	res:target children	Absent	self/ @res:href	Present
CC23	@slr:sizeInfo	Present	self/ @slr:sizeInfoRef	Absent (1)
CC24	ec/@isolated	“yes”	self/ @slr:sizeInfoRef	Allowed
CC25	val:rule/ @existsInSource	“yes’	self/ @isPresent or @startsWith or @endsWith	Present (1)
CC26	mrk/@itsm: annotatorsRef, sm/@itsm: annotatorsRef	Present	self/@type	“itsm:generic” (1)
CC27	mrk/@itsm: domains, sm/@itsm: domains	Present	self/@type	“itsm:generic” (1)

ID	Subject Node	Variation	Object Node	Variation
CC28	mrk/@itsm: localeFilterList, sm/@itsm: localeFilterList	Present	self/@type	"itsm:generic" (1)
CC29	sm|mrk/ @itsm: locQuality Rat-ingScore	Present	self/@type	"itsm:generic" (1)
CC30	sm|mrk/ @itsm: locQuality Rat-ingVote	Present	self/@type	"itsm:generic" (1)
CC31	sm|mrk/ @itsm:lang	Present	self/@type	"itsm:generic" (1)
CC32	sm|mrk/ @itsm: mtConfidence	Present	self/@type	"itsm:generic" (1)
CC33	@itsm:locQuality RatingScore	Present	self/@itsm: loc-QualityRating ScoreThreshold	Allowed

ID	Subject Node	Variation	Object Node	Variation
CC34	@itsm:taIdent	Present	self/@itsm: taSource	Present
CC35	@itsm: locQuality RatingScore	Present	self/@itsm: locQuality Rat-ingVote	Absent
CC36	@subType	"xlf:b" "xlf:i" "xlf:u" "xlf:lb" "xlf:pb"	self/@type	"fmt" (1)
CC37	@subType	"xlf:var"	self/@type	"ui" (1)
CC38	@val:occure	Present	self/ @val:isPresent	Present(1)

In terms of expressibility in schema languages, the co-occurrence constraints are, at least theoretically, expressible in RelaxNG. Although, implementing those constraints where subject and object nodes are far from each other in the tree (e.g. 'xliff' and 'target' in CC1) would significantly increase the volume of the schema.

Schematron, on the other hand, offers a better capacity for creating validation artefacts based on the data of Table 3.6. Selection of schema language and implementing constraints are not the goal of this chapter and will be discussed in chapter 4.

Unlike primary keys, foreign keys and implied constraints, XLIFF 2.1 co-occurrence constraints are stated in detail within the specification and their functionality and reasons behind them are clearly described and, therefore, we consider their feature discussions out of scope of this work.

The next group of business rules is consist of those constraints, that are applied to values (content) of nodes and can be expressed and defined via regular expression patterns. This small group embraces the following attributes: ‘hex’ (used in ‘cp’), ‘itsm:annotatorsRef’ (allowed at multiple elements) and attributes of Validation Module which represents a complicated interrelation of the module’s properties.

The “Code Point” element, ‘cp’, is meant to serve as a placeholder for invalid XML characters, storing the illegal content within its ‘hex’ attribute. These characters are represented by hexadecimal range of [0000 10FFFF]. The next attribute of this group has a more complex constraint since its value contains structured information. This attribute is defined by the ITS Module to implement Tools Annotation datacategory of ITS 2.0, which facilitates storage of information about the tool or tools who produced the ITS metadata. Since multiple applications can be involved in creation of ITS metadata for different datacategories, this attribute is required to address them all by providing ordered and informative value. To meet the requirements, this attribute must contain space-separated list of ITS datacategories and an IRI (Internationalized Resource Identifier) of the hired tool, distinguished by a vertical line character (|). Additionally, datacategories must be ordered alphabetically in case, if the attribute contains information about more than one tool. For instance, the following represents a valid usage of the attribute: `itsm:annotatorsRef="provenance|IRI1 translate|IRI2"`. This constraint is directly driven by ITS 2.0 specification and the corresponding XLIFF-defined attribute is allowed to be added to the following elements: file, unit, group, mrk, sm, mtc:match, ctr:revisions and ctr:revision.

The last set of attributes, whose constraints can be expressed by regular expressions, are 'startsWith', 'endsWith', 'isPresent' and 'isNotPresent', all being used only in `val:rule` element. The first attribute was presented earlier in this chapter and the rest are meant to perform on the same logic; to specify a pattern for the translated text to end with or (not) to be present in the text. However, the module provides additional attributes for advanced functionality and rule definition and, therefore, the scope of the constraints over this module extends to a number of functional dependencies. The first member of the module's secondary fields is the 'occur' attribute which can appear only with 'isPresent' (CC38 in Table 3.6) to set the exact number of times the pattern can be met in the target text. The next attribute, 'caseSensetive' indicates whether or not the test pattern is case sensitive and can be applied to any of 4 major properties. Another attribute, 'existsInSource', is specified by the module to expand the test to the 'source' content as well and it must appear with only one of the first 3 patterns (CC25 in Table 3.6). The last property is the 'disabled' attribute which represents an important aspect of the Module- inheritance. If the module is included in XLIFF at the 'file' level, it means that the rule applies to the scope of the entire 'file' and all of 'unit' and 'group' descendants and if any of lower levels need to avoid the "global" rule, they "disable" it by specifying it at the required level.

While the first two of the described constraints can be expressed by regular expressions, the integrity constraints of the Validation Module would require taking into account other constraints that affect the test patterns. Overall, XSD, RelaxNG and Schematron all support regular expression patterns and selection of the optimal schema is subject of the following chapter.

The rest of XLIFF 2.1 business rules are presented in Table 3.7 and as they cannot be further generalised and analysed collectively, each rule must be targeted on the individual basis. The rules are extracted directly from the specification and as result the wording and order matches the document.

Table 3.7: XLIFF 2.1 Business Rules

ID	Rule
BR1	A ‘unit’ must contain at least one ‘segment’ element
BR2	When a ‘target’ element is a child of ‘segment’ or ‘ignorable’, the explicit or inherited value of the optional xml:lang must be equal to the value of the trgLang attribute of the enclosing ‘xliff’ element.
BR3	The attribute isolated must be set to yes if and only if the ‘ec’ element corresponding to this start marker [sc] is not in the same ‘unit’, and set to no otherwise
BR4	The values of the attributes canCopy, canDelete, canReorder and canOverlap must be the same as the values of the ones in the ‘ec’ element corresponding to this start code [sc]

ID	Rule
BR5	The attribute isolated must be set to yes if and only if the ‘sc’ element corresponding to this end code [ec] is not in the same ’unit’ and set to no otherwise
BR6	The value of the attribute canReorder must be no if the value of canReorder is firstNo in the ‘sc’ element corresponding to this end code [ec]
BR7	The values of the attributes canCopy, canDelete and canOverlap must be the same as the values the ones in the ‘sc’ element corresponding to this end code [ec]
BR8	Comment Annotation: The type attribute is required and set to comment, If the value attribute is present it contains the text of the comment, If and only if the value attribute is not present, the ref attribute must be present and contain the URI of a ‘note’ element within the same enclosing ‘unit’ element that holds the comment
BR9	The inline elements enclosed by a ‘target’ element must use the duplicate id values of their corresponding inline elements enclosed within the sibling ‘source’ element if and only if those corresponding elements exist

ID	Rule
BR10	To be able to map order differences, the ‘target’ element has an optional order attribute that indicates its position in the sequence of segments (and inter-segments). Its value is an integer from 1 to N, where N is the sum of the numbers of the ‘segment’ and ‘ignorable’ elements within the given enclosing ‘unit’ element. When Writers set explicit order on ‘target’ elements, they have to check for conflicts with implicit order, as ‘target’ elements without explicit order correspond to their sibling ‘source’ elements
BR11	Modifiers must not delete inline codes that have their attribute canDelete set to no
BR12	Modifiers must not replicate inline codes that have their attribute canCopy set to no
BR13	The start marker, sm, must appear before the end marker em
BR14	A glossEntry element must contain a translation or a definition element to be valid
BR15	ITS Text Analysis Annotation: At least one of the followings: itsm:taClassRef or one of (A pair of a itsm:taSource and itsm:taIdent both set or itsm:taIdentRef), the type attribute is optional and set to itsm:generic.
BR16	Exactly one of the following attributes must be set in any val:rule element: isPresent, isNotPresent, startsWith, endsWith or a custom rule defined by attributes from any namespace.

ID	Rule
BR17	When the optional xml:lang attribute [of res:source] is present, its value must be equal to the value of the srcLang attribute of the enclosing xliff element
BR18	When the optional xml:lang attribute [of res:target] is present, its value must be equal to the value of the trgLang attribute of the enclosing xliff element
BR19	In case the required ctr:appliesTo attribute value is set to source or target and there is more than one xlf:source or xlf:target element in scope, the value of the ctr:ref attribute must be equal to the NMTOKEN value of the xlf:id attribute of the parent xlf:segment or xlf:ignorable element of the referenced xlf:source or xlf:target element
BR20	In all other cases [note, segment, ignorable for ctr:appliesTo] where multiple elements of the same name are in scope, the ctr:ref attribute value must be equal to the value of the xlf:id attribute of the referenced element itself
BR21	For ctr:item and ctr:simpleItem elements with the attribute ctr:property other than content only Content Type D is allowed and no XLIFF Core Inline Elements must be used

ID	Rule
BR22	For all ctr:item and ctr:simpleItem elements with the attribute ctr:property set to content: 1) If and only if ctr:appliesTo equals unit, the grandchild must be a ctr:item element of the Content Type A; 2) If and only if ctr:appliesTo is set to segment or ignorable, the grandchild must be an ctr:item element of the Content Type B or C; 3) If and only if ctr:appliesTo is set to note, source, or target the grandchild may be a ctr:simpleItem element. In case the ctr:item element is used, Content Type D is required
BR23	In case the enclosing ctr:revisions grandparent has the attribute ctr:appliesTo set to note, the ctr:item must not contain any XLIFF Core Inline Elements
BR24	ctr:item must not contain any unhandled orphaned XLIFF Core Inline Elements
BR25	In case the enclosing ctr:revisions grandparent has the attribute ctr:appliesTo set to note, the ctr:simpleItem must not contain any XLIFF Core Inline Elements
BR26	ctr:simpleItem must not contain any unhandled orphaned XLIFF Core Inline Ele-ments

ID	Rule
BR27	ITS Localization Quality Issue Annotation: Exactly one of the fol-lowings: itsm:locQualityIssuesRef or (itsm:locQualityIssueType and/or itsm:locQualityIssueComment), the type attribute is optional and set to itsm:generic, and if itsm:locQualityIssuesRef is set, the followings must not be used: itsm:locQualityIssueSeverity, itsm:locQualityIssueProfileRef, and itsm:locQualityIssueEnabled
BR28	ITS Provenance Annotation: The following attributes must not be set if and only if itsm:provenanceRecordsRef is declared, otherwise at least one the following must be set: itsm:org, itsm:orgRef, itsm:person, itsm:personRef, itsm:revOrg, itsm:revOrgRef, itsm:revPerson, itsm:revPersonRef, itsm:revTool, itsm:revToolRef, itsm:tool, itsm:toolRef, the type attribute is optional and set to itsm:generic.
BR29	At least one of the following itsm attributes must be set for itsm:provenanceRecord element: itsm:org, itsm:orgRef, itsm:person, itsm:personRef, itsm:revOrg, itsm:revOrgRef, itsm:revPerson, itsm:revPersonRef, itsm:revTool, itsm:revToolRef, itsm:tool, itsm:toolRef.

ID	Rule
BR30	At least one of the attributes itsm:locQualityIssueType or itsm:locQualityIssueComment must be set for itsm:locQualityIssue element

Schematron is the only candidate among schema languages to be considered for expressing the aforementioned remaining constraints and our goal within this chapter is to discover whether or not these rules could be expressed by the schema language. As was discussed in Chapter 2 and reported by Saadatfar and Filip (2016), the essential parts of business rules from Schematron perspective are context and test paths, similar to subject/object convention used in co-occurrence constraints. In other words, to determine if a rule is expressible in Schemtaron, the first step of investigation must start finding whether or not the nodes are accessible through XPath navigational expressions, taking into account the relation of nodes. It means that if the path from subject node to the object is expressible in XPath, the rule may be expressible in Schematron. The other step would be applying the required check (e.g. presence, value check) by matching the the defined functional dependency with available XPath functions. Table 3.8 demonstrates the analytic data which is required for Schematron development purposes, providing information for context and test paths as well as additional notes which might be considered in the development stage (rules can be mapped to Table 3.7 by IDs for the text).

Table 3.8: Analyses of XLIFF 2.1 Business Rules

ID	Context Path	Test Path	Notes
BR1	unit	segment	
BR2	target elements with segment or ignorable parent and with xml:lang	xliff	CC1 is prerequisite for this FD; comparison conducted in accordance with BCP 47
BR3	sc	corresponding ‘ec’ in the same unit or file	This FD implies separate check for sc elements in source and target
BR4	sc	corresponding ‘ec’	BR3 is prerequisite for this FD; This FD implies 8 checks- each attribute in source and target separately; An implied case is when ec is not in the same unit

ID	Context Path	Test Path	Notes
BR5	ec	corresponding ‘sc’ in the same unit or file	This FD implies separate check for ec elements in source and target. It is also implied that the end code appears after the start code
BR6	sc with canReorder=“firstNo”	corresponding ‘ec’	BR5 is prerequisite for this FD; An implied case is when ec is not in the same unit; This FD overrides BR4 and CC9 for special case of canReorder
BR7	ec	corresponding ‘sc’	BR5 is prerequisite for this FD; This FD implies 8 checks- each attribute in source and target separately; An implied case is when ec is not in the same unit
BR8	annotation markers (sm/mrk) with type=‘comment’	self	This constraints contains a co-occurrence and a foreign key to be distinguished

ID	Context Path	Test Path	Notes
BR9	inline elements in target	inline element of the same kind in the sibling source	K6 is prerequisite for this FD; This FD also implies that if the corresponding code doesn’t exist, the id must be unique
BR10	‘targe’ with order attribute	‘targets’ within the same unit	This constraint implies a dependant value check. K7 is prerequisite
BR11	inline elements in ‘source’ with ‘canDelete’=’no’	corresponding inline element in the sibling ‘target’	This constraint is a PR, but applies to all agents
BR12	any element with copyOf	the corresponding inline element	This constraint implies that the scope of replication is ‘unit’
BR13	sm	corresponding ‘em’	This is an implied FD
BR14	gls:glossEntry	children	Implementing this constraint in RelaxNG is not reasonable due to volume

ID	Context Path	Test Path	Notes
BR15	annotation markers (xlf:sm/xlf:mrk) with itsm:taClassRef, itsm:taSource, itsm:taIdent or itsm:taIdentRef	self	It can partialy be grouped in SCH implementation with CC26-32; requires several checks for all restrictions
BR16	val:rule	self	
BR17	res:source with xml:lang	xliff	The comparison to be conducted in accordance with BCP 47
BR18	res:target with xml:lang	xliff	The comparison to be conducted in accordance with BCP 47
BR19	ctr:revisions with appliesTo="source" or "target"	parent xlf:unit	This constraints contains an FD and a foreign key to be distinguished
BR20	ctr:revisions with appliesTo= "segment|ignorable|note"	parent xlf:unit/ parent xlf:file/ parent xlf:group	Identifying the scope of the module is prerequisit to this FD

ID	Context Path	Test Path	Notes
BR21	ctr:item and ctr:simpleItem with ctr:property not “con-tent”	children	This means that the elements can contain text only
BR22	ctr:revisions	decendandts	This FD embraces 3 FDs
BR23	ctr:revisions with ctr:appliesTo=“note”	decendants	
BR24	ctr:item	xlf desendants	
BR25	ctr:revisions with ctr:appliesTo=”note”	decendants	
BR26	ctr:item	xlf desendants	

ID	Context Path	Test Path	Notes
BR27	annotation markers (xlf:sm/xlf:mrk) with itsm: locQualityIssuesRef, itsm: locQualityIssueType or itsm: locQualityIssueComment	self	It can partially be grouped in SCH implementation with CC26-32, BR15; requires several checks for all restrictions; implies a foreign key
BR28	annotation markers (xlf:sm/xlf:mrk) with the declared attributes	self	It can partially be grouped in SCH implementation with CC26-32, BR15, BR27; requires additional check on co-occurrence
BR29	itsm:provenanceRecord elements	self	Twisted co-occurrence constraint
BR30	itsm:locQualityIssue element	self	Twisted co-occurrence constraint

Describing functionality and features of XLIFF 2.1 business rules is out of scope of our work, although some of the most important and complex business rules will be explained at the development stage in the next chapter as successful and comprehensive implementation of them is not possible without knowledge of their purpose. The study within this chapter will be referred to again later in this thesis for optimal schema creation. However, we will look into the integrity constraints which apply to the XLIFF document shown in Listing 3.1. It represents a valid instance and, therefore, satisfies all the applicable functional dependencies: the document contains ‘target’ and the root element specifies ‘trgLang’ (CC1 in Table 3.6); the ‘type’ attribute of the ‘mrk’ element declares (by its value, ‘mtc:match’) that the element is being used as ‘Translation Candidate Annotation’ and must follow the appropriate pattern (e.g. not contain ‘ref’ attributes) (CC11); inline codes of ‘target’ are using the duplicate ids of their corresponding elements in the sibling ‘source’ (BR9) and the other attributes of the elements also match. Notably, the last constraint is an example of XLIFF implied constraints which are not explicitly set, but nevertheless exist and violating them must be reported as a validation error.

3.5.3 Progressive Constraints

The XLIFF 2.0 specification sets a number of rules, known as “Processing Requirements”, which apply to a certain group of XLIFF Agents and specify the permitted changes they can commit while processing XLIFF documents. Many of the Processing Requirement rules are rather instructions to properly design and develop XLIFF agents for conformance purposes, a few of these rules apply to all agents and, therefore, can be considered as integrity constraint (e.g. BR11-12 in Table 3.7), some of them emphasis certain constraints for specific agents and only few represent what we consider as "Progressive Constraints" and expressible by validation methods. For instance, the following describes how "Writer" agents must implement the 'cp' element (Code Point) which is offered by XLIFF to store invalid XML characters (Section 4.2.3.1, XLIFF 2.1 specification):

> Writers must encode all invalid XML characters of the content using 'cp';
>
> Writers must not encode valid XML characters of the content using 'cp'.

The following Processing Requirement from Section 4.2.2.3 of XILIFF 2.1 specification, on the other hand, implies a static constraint which in fact is a clarification to the 'skeleton' constraint (CC2 in Table 3.6):

> Extractors creating an XLIFF Document with a 'skeleton' element must leave the element empty if and only if they specify the attribute 'href'.

As the Extractor Agents represent the very start of the XLIFF roundtrip, this Processing Requirement actually defines the ‘skeleton’ valid behaviour throughout the workflow: “it must either have the ‘href’ attribute and be empty, or include content (e.g. text) without any attribute”. This implication is embedded in CC2 (Table 3.6) for further implementation.

The XLIFF 2.1 Progressive Constraints are gathered in Table 3.9, where the responsible Agents and the modification restrictions are identified.

Thanks to the `document()` function of XPath, which enables cross document checking, the XLIFF 2.1 Processing Requirements are expressible in Schematron with its full support of XPath functions. We will discuss their implementation in the following chapter, however we would highlight the most challenging aspect of this type of rules here, which is locating the correct node in both files. For implementing P2, P3 and P5 (Table 3.9) it is crucial to navigate through the exact element in the document by matching their identifiers and also identifiers of ancestors for relative primary keys. For instance, ‘type’ attribute (P3) can be hosted by the pc, ph, sc and ec elements and in order to reference them, it is important to track identifiers of enclosing ‘file’ and ‘unit’ levels as well as the elements’ id itself.

Table 3.9: XLIFF 2.1 Processing Requirements

ID	Rule	Agent	Context Path	Test Path
P1	Modifiers and Enrichers processing an XLIFF Document that contains a ‘skeleton’ element must not change that element, its attributes, or its content	M, En	skeleton	skeleton in the original file
P2	Writers updating the attribute state must also update or delete subState	W	xlf:segment [@state]	corresponding segment in the original file
P3	Modifiers updating the attribute type must also update or delete subType	M	xlf:*[@type]	corresponding element in the original file

ID	Rule	Agent	Context Path	Test Path
P4	When removing a given inline code, the user agents must remove its associated original data, except if the original data is shared with another inline code that remains in the unit	M, En	data	inline codes referencing the element
P5	Writers updating the attribute fs must also update or delete subFs	W	xlf:*[@fs:fs]	corresponding element in the original file

3.6 Conclusion

Within this chapter we presented our extended study of XLIFF 2.1 and also had a detailed review of its structure, data model and integrity constraints. The main objectives of this chapter can be summarised as targeting our research question and producing enough information and guidelines for the following part of our work, where we will be developing the identified and analysed integrity constraints. As of the research question, we showed in this chapter that the DSDL methods are expressive enough to define the XLIFF 2.1 specification within appropriate validation artefacts. The study and deep analyses we conducted in this chapter, on the other hand, helped us to identify the important and challenging parts and aspects of the XLIFF 2.1 validation, especially in terms of the fields' interrelations. The findings of this chapter also provide explanation, notion and description for those integrity constraints which are not well researched in the literature, i.e. progressive constraints, conditional keys and various functional dependencies, but are very likely to be adopted by modern industry data structures as the result of defining intricate business rules. Another important contribution of this chapter is the scientific and scholar assessment of XLIFF 2.1, which, as we showed throughout the chapter, is affected by the industry requirements and demands in its nature, starting from the structural model and ending with its functionality and integrity constraints. Such fundamental and systematic approach helped us to specify "unusual" aspects of XLIFF behaviour from the XML point of view and produce valuable knowledge for further improvement of the standard, as XLIFF has been always developed in line with contemporary and progressive research.

Chapter 4

Development and Optimisation of Validation Artefacts

4.1 Introduction

After analyses of XLIFF 2.1 specification which revealed that each constraints has at least one candidate within the selected schema languages and, consequently, addressing the research question of our work, this chapter is in accordance with the directions of our sub-questions; delivering the appropriate artefacts within an optimised validation platform for advanced validation of XLIFF 2.1 instances.

In this chapter we will review the process of implementing the "machine-readable specification" beginning with the discussion of the hired methodology. We will then explain our work with regard to each schema language which was selected during our research: RelaxNG, Schematron and NVDL. This chapter also contains a discussion on the process of optimisation of the developed artefacts, which contributes to the effective design of the finalised validation platform through identifying duplications and suggesting alternative approaches for those artefacts built on a weak logic. Notably, these improvements do not expand the list of the covered XLIFF constraints, but rather target efficiency by defining a strong and sound pattern to be followed for the best practice. In order to keep the chapter relevant to our objectives, we will represent artefacts which are already modified for each schema language and will leave the optimisation discussion and the review of the proposed changes in a separate section.

It was mentioned earlier that the standard already offers a set of W3C Schema files for the Core and Modules, although we saw that such schema is generally used for structural validation, which alongside with the document's well-formedness is prior to advanced validation. These prerequisite and vital tasks are performed by W3C XML Schema (Any XSD processor runs a well-formedness check before applying the schema). In order to keep backwards compatibility among all versions of XLIFF 2.x family, the existing schema cannot be modified and even though some of expressible constraints are absent in the official schema (e.g. K1 in Table 3.4), we have to address them through an alternative solution. Finally, we will investigate the optimal set and organisation of artefacts for XLIFF 2.1 validation.

4.2 Methodology

The research methodology we use for this part of research is of design and development kind, which was introduced in Chapter 1. Our objective within this chapter is development of effectively designed artefacts. Henver et al (2004) describe the task of design science research as creation and evaluation of "IT artefacts intended to solve identified organisational problems", which is lack of interoperability in localisation workflows in the context of this work. A broader definition has been suggested by Vaishnavi and Keuchler (2004):

Design science research involves the creation of new knowledge through design of novel or innovative artefacts (things or processes) and analysis of the use and/or performance of such artefacts along with reflection and abstraction to improve and understand the behaviour of aspects of Information Systems.

While we will discuss the evaluation of our platform in the next chapter, the following sections of the current are dedicated to the process of design and development for each schema language.

At this stage, we have already identified the validation tasks of XLIFF 2 which could not be addressed using the existing XSD schema: primary and foreign keys, functional dependencies and dynamic (progressive) constraints. We developed a RelaxNG schema for XLIFF 2.1 as the next step and discovered that the contribution of such artefact to the enhancement of the overall progress is not essential- most of implemented constraints overlap with ones in XML Schema and only some minor functional dependencies could be applied and expressing some of the constraints for large-scaled vocabularies such as XLIFF is not reasonable in terms of volume, although theoretically possible. Furthermore, we developed *Schematron Rules* that express all the remaining constraints after XSD and RelaxNG. After assessment and comparison of the artefacts it was decided to omit RelaxNG for performance purposes as the improvement, to the overall validation process, made by this schema was not significant. Missed constraints, as the result, were re-assigned to Schematron and developed. Having the entire XLIFF 2.1 specification expressed in a combination of XSD schema and Schematron Rules, an NVDL mapping file was developed to match different namespaces of the XLIFF document with the appropriate schema, as well as applying all the developed artefacts to the file. Finally, we applied some improvements to the developed Schematron Rules to avoid overlapping functionality among them, simplify the notations and make use of the Schematron enrichment features for providing additional information on the occurred errors. The latter will significantly ease the process of troubleshooting through an advanced error report which contains enough information for locating and fixing the invalid nodes. During the process of optimisation, our goal was to ensure that every validation task is assigned to the most suitable schema language match and it is implemented in the optimised way.

We will now present development of the artefacts in each schema language, which would shape the final validation platform for the official OASIS XLIFF 2.1 specification.

4.3 RelaxNG Schema

The first step of schema development in this work is implementation of a RelaxNG schema and targeting those integrity constraints which were identified as expressible in the schema language by the findings of chapter 3; structural validation, co-occurrence constraints and regular expression patterns. We chose the *named pattern* approach in development of the schema for XLIFF 2.1 Core , which unlike the other one, "Russian Doll" style, allows recursion, an important aspect of XLIFF 2.1 structure, e.g. `<group>`, and is significantly easier to maintain and read for large schema (van der Vlist 2003). Name patterns are used in a modular fashion, where the patterns can be reused and, therefore, would remarkably decrease the overall volume of the schema. For instance, attributes of the same type or elements with the same structure (e.g. inline elements usage in `<source>` and `<target>`) can refer to one pattern instead of re-declaring the content.

Since the developed RelaxNG schema is a large document, we will only include important fractions of it in this chapter, but the full schema is available online in the OASIS repository of the XLIFF TC[1] and is also presented in the attached CD to this thesis.

The schema, in terms of XLIFF 2.1 structure definition, duplicates the official XSD schema to a large degree, except for certain improvements on accurate definition of the extension points. For instance, the `ec` element (an XLIFF 2.1 inline code) is allowed to contain attributes from other (non-XLIFF) namespaces, however, only a limited group of attributes from XLIFF 2.1 Modules namespaces are permitted at this point. Nonetheless, the XSD misses this restriction by using the following annotation:
“`<xs:anyAttribute namespace="##other" processContents="lax"/>`”. The “`##other`” wildcard simply enables every attribute, even illegal, to appear in the element. The same approach is used in XSD for other XLIFF Core elements and this issue is well covered by RelaxNG, as illustrated in Listing 4.1, where at first all the modules are prohibited and then exceptions are specified. Notably, further validation for the permitted attributes

[1]http://tools.oasis-open.org/version-control/browse/wsvn/xliff/branches/Soroush/xliff_core_2.0.rng [accessed 20 Sep 2016]

must be carried out by the appropriate module schema and passing the validation only by this schema does not guarantee that the document is valid from the module's perspective (e.g. `fs:fs` attribute can be included only if `isolated` attribute of the element is set to 'yes', i.e. its pair start code, 'sc', appears in a different 'unit'). Listing 4.1 presents the corresponding fragment of the schema for allowed attributes of the 'ec' element.

Listing 4.1: Restriction on Extension Point at 'ec'

```
<zeroOrMore>
   <attribute>
       <anyName>
        <except>
            <nsName ns=""/>
            <nsName ns="urn:oasis:names:tc:xliff:fs:2.0"/>
            <nsName ns="urn:oasis:names:tc:xliff:matches:2.0"/>
            <nsName ns="urn:oasis:names:tc:xliff:glossary:2.0"/>
            <nsName ns="urn:oasis:names:tc:xliff:metadata:2.0"/>
            <nsName ns="urn:oasis:names:tc:xliff:resourcedata:2.0"/>
            <nsName ns="urn:oasis:names:tc:xliff:changetracking:2.1"/>
            <nsName ns="urn:oasis:names:tc:xliff:sizerestriction:2.0"/>
            <nsName ns="urn:oasis:names:tc:xliff:validation:2.0"/>
            <nsName ns="urn:oasis:names:tc:xliff:itsm:2.1"/>
        </except>
       </anyName>
   </attribute>
   <attribute ns="urn:oasis:names:tc:xliff:fs:2.0" name="fs"/>
   <attribute ns="urn:oasis:names:tc:xliff:fs:2.0" name="subFs"/>
   <attribute ns="urn:oasis:names:tc:xliff:sizerestriction:2.0"
   name="equivStorage"/>
   <attribute ns="urn:oasis:names:tc:xliff:sizerestriction:2.0"
   name="sizeInfo"/>
   <attribute ns="urn:oasis:names:tc:xliff:sizerestriction:2.0"
   name="sizeInfoRef"/>
</zeroOrMore>
```

With respect to the XLIFF 2.1 Core co-occurrence constraints (CC1-15 in Table 3.6), this category of constraints was partially expressed in RelaxNG with the exception of the followings: CC1 and CC9 . While the reason for exclusion of CC1 was already described in chapter 3, it was decided to leave CC9 ('canReorder' attribute) for Schematron as it is a part of a wide functional dependency, which will be discussed in the following section. Co-occurrence constraints can be addressed through a well-known and common pattern of RelaxNG. Listing 4.2 demonstrates how CC2 is implemented in RelaxNG. This constraint requires ‘skeleton’ element to store the non-translatable data either within its content or specify a URI of the content by an ‘href’ attribute.

Listing 4.2: Definition of skeleton Element in RelaxNG

```
  <define name="skeleton-content">
    <choice>
       <group>
           <attribute name="href">
               <ref name="URI-IRI"/>
           </attribute>
           <empty/>
       </group>
       <group>
           <interleave>
               <text/>
               <zeroOrMore>
                    <ref name="anyElement"/>
               </zeroOrMore>
           </interleave>
       </group>
    </choice>
</define>
```

Listing 4.3 represents the implementation of CC3-CC5 constraints, which were grouped for development since they affect one common element, ‘ec’, and the major property is ‘isolated’.

Listing 4.3: Co-occurrence Constraints of ‘ec’ attributes

```
<choice>
      <group>
             <attribute name="isolated">
                 <value>yes</value>
             </attribute>
             <attribute name="id">
                 <ref name="attid"/>
             </attribute>
             <optional>
                 <attribute name="dir">
                     <ref name="attdir"/>
                 </attribute>
             </optional>
             <optional>
                 <attribute ns="urn:oasis:names:tc:xliff:fs:2.0"
                                                   name="fs"/>
             </optional>
         </group>
```

```
                <group>
                    <optional>
                        <attribute name="isolated">
                            <value>no</value>
                        </attribute>
                    </optional>
                     <attribute name="startRef">
                        <ref name="attstartRef"/>
                    </attribute>
             </group>
      </choice>
```

The rest of co-occurrence constraints are developed on the basis of the same logic and, therefore, will not be presented here. With regard to the regular expressions, on the other hand, it was decided to assign this type of constraints to Schematron to implement the entire set of patterns within one schema. Later in this chapter we will discuss why RNG schema for XLIFF 2.1 Modules were not developed and the general role of this schema language in preparing the advanced validation technique for XLIFF 2.1. The following section will present the role of Schematron in our platform and discuss the development of appropriate rules for validation of XLIFF 2.1 integrity constraints.

4.4 Schematron Rules

The largest portion of the standard's constraints and complex rules is expressed in Schematron, including all of dynamic constraints, XLIFF 2.1 primary and foreign keys. In this section we will present how the rules were elaborated and developed based on the XLIFF 2.1 integrity constraints, which were collected and analysed within chapter 3. However, for effective design of the Schematron schema we first reviewed the business rules again and studied the provided notes and analyses to discover the interrelation of the rules and to ensure that the schema completely covers the scope of dependencies. This means that the Schematron rules would not necessarily match each identified rule, but rather provide an optimised and well-organised schema for validation of XLIFF 2.1 integrity constraints. The developed schema is referenced in the specification and is also available online[2]. A copy of the Schematron file is also attached and can be found in the provided CD. While the process of the schema development will be reviewed in the following, this chapter aims to represent the general process and, in terms of details, will only focus on the most important parts of the schema which are con-

[2]http://tools.oasis-open.org/version-control/browse/wsvn/xliff/trunk/xliff-21/schemas/#_trunk_xliff-21_schemas [accessed 22 Nov 2016]

sidered the most challenging or represent a common model for a set of integrity constraints.

We will review implementation of the integrity constraints starting with keys, as it is the only group, members of which are independent and do not cause any drawbacks on other constraints, but in fact are prerequisite for many of other rules and it is reasonable, regarding effective design, to validate these constraints at the first step.

4.4.1 XLIFF 2.1 Keys

Expressing key constraints in Schematron is a common, straightforward and rather a simple task. The standard approach is based on selecting the target node and then counting nodes which use the same key within the specified scope. The final stage would be declaring the "assertions" in accordance with the key definition. Besides all of the XLIFF 2.1 keys, which were discovered and presented in Table 3.4, we included BR9 (Table 3.8) in this group as well since the business rule represents a conditional key and those conditions can be applied by XPath predicates. All of the keys then were implemented in accordance with the standard approach and the optimal order, starting from higher levels and taking into account the prerequisites reported by the study in chapter 3. The Schematron rule for uniqueness of inline elements within ‘unit’ (K6 in Table 3.4) is presented in Listing 4.4 as an example. Although this rule is consist of multiple conditions and relations, but it has been simplified after clear indication of subject and object nodes. We first set the “context” to inline elements within ‘source’ and then “assume” that only 1 element with the same id is present in the same ‘unit’, which is the context node itself, and all other cases would be reported as error.

Listing 4.4: Schematron Rule for id duplication within ‘unit’

```
<sch:pattern>
    <sch:rule context="xlf:source[ancestor::xlf:segment|
              ancestor::xlf:ignorable]//xlf:*[@id]|
              xlf:segment[@id]| xlf:ignorable[@id]">
        <sch:let name="id" value="current()/@id"/>
        <sch:report test="count(ancestor::xlf:unit//xlf:*[@id=$id]
              [ancestor-or-self::xlf:segment| ancestor-or-self::xlf:ignorable]
              [not(ancestor::xlf:target)])!=1">
            id duplication found.
        </sch:report>
    </sch:rule>
</sch:pattern>
```

With respect to the XLIFF 2.1 foreign keys, it was decided to to implement them in group with other applicable constraints to the node, because they generally are involved in other and broader functional dependencies and it is more effective to target the entire FD at once, since the selected context in Schematron rule would be the same. However, these constraints are specified following the same logic as for keys, i.e. counting the referred nodes within the scope, but the assertion would be presence of *at least one* match, instead of only one match. The reason for different assertion in case of foreign keys is to avoid duplicating constraint checks and, therefore, delivering the effective schema where the matter of uniqueness is already targeted within the rules for keys at the previous step.

The next group of integrity constraints to be reviewed and implemented contains those constraints that should be addressed by regular expressions and were previously skipped by RelaxNG.

4.4.2 Regular Expression

The constraints of this type were introduced and described in chapter 3 and except for one of them, which will be discussed shortly, the others do not require further investigation, but only developing the appropriate regular expressions. Listing 4.5 illustrates implementation of the value range constraint for ‘hex’ attribute which is applied by the appropriate regular expression.

Listing 4.5: Schematron Rule for ‘hex’ attribute

```
 <sch:pattern>
    <sch:rule context="xlf:cp">
        <sch:assert test="matches(@hex,'^([0-9a-fA-F][0-9a-fA-F]
                                         [0-9a-fA-F][0-9a-fA-F] |
                                         [0-9a-fA-F][0-9a-fA-F][0-9a-fA-F]
                                         [0-9a-fA-F][0-9a-fA-F] |
                                         [01][0-9a-fA-F][0-9a-fA-F]
                                         [0-9a-fA-F][0-9a-fA-F][0-9a-fA-F])\$')">
            The value of 'hex' attribute must be in  the hexadecimal
            inclusive range of 0000 to 10FFFF.
        </sch:assert>
    </sch:rule>
</sch:pattern>
```

For validation of the next constraint of this type, ‘itsm:annotatorsRef’ attribute, the rule would be slightly more complex. We distinguished 2 general cases for this attribute; when its value contains a single ITS datacategory and, the other case, where multiple values are stored within the attribute. Such distinguishment is necessary to ensure that all possible errors are caught by the rule and the schema is deigned effectively. While for the first case it is only required to validate the name of the datacategory, multiple values must be validated against the alphabetic order and replication on top of that. This integrity constraint is implemented within two Schematron patterns to express the rule in full for both cases and also to provide advanced and informative error reporting. It is represented in Listing 4.6.

Listing 4.6: Implementation of ‘itsm:annotatorsRef’ constraint in Schematron

```
<sch:pattern>
    <sch:rule context="xlf:*[@itsm:annotatorsRef]
             [not(contains(@itsm:annotatorsRef, ' '))]">
        <sch:let name="ref" value="@itsm:annotatorsRef"/>
        <sch:let name="its-dc-id" value="substring-before(\$ref,'|')"/>
        <sch:report test="\$its-dc-id!='allowed-characters'
             and \$its-dc-id!='directionality' and \$its-dc-id!='domain'
             and \$its-dc-id!='elements-within-text'
             and \$its-dc-id!='external-resource' and \$its-dc-id!='id-value'
             and \$its-dc-id!='language-information'
             and \$its-dc-id!='locale-filter' and \$its-dc-id!='localization-note'
             and \$its-dc-id!='localization-quality-issue'
             and \$its-dc-id!='localization-quality-rating'
             and \$its-dc-id!='mt-confidence'
```

```
              and \$its-dc-id!='preserve-space'
              and \$its-dc-id!='provenance' and \$its-dc-id!='storage-size'
              and \$its-dc-id!='target-pointer'
              and \$its-dc-id!='terminology' and \$its-dc-id!='text-analysis'
              and \$its-dc-id!='translate'">
             Invalid id used for the ITS datacategory
             <sch:value-of select="\$its-dc-id"/>.</sch:report>
     </sch:rule>
 </sch:pattern>
 <sch:pattern>
     <sch:rule context="xlf:*[@itsm:annotatorsRef]
               [(contains(@itsm:annotatorsRef, ' '))]">
         <sch:let name="ids-string" value=
         "replace(@itsm:annotatorsRef, '\|\w+','')"/>
         <sch:let name="ids-tokens" value="tokenize(\$ids-string, ' ')"/>
         <sch:report test="count(\$ids-tokens)!=
                                 count(distinct-values(\$ids-tokens))">
             Each ITS data category identifier must not be used more than once.
         </sch:report>
         <sch:assert test="matches(\$ids-string, '^(allowed-characters)?\s*
                 (directionality)?\s*(domain)?\s*(elements-within-text)?\s*
                 (external-resource)?\s*(id-value)?\s*(language-information)?\s*
                 (locale-filter)?\s*(localization-note)?\s*
```

```
                (localization-quality-issue)?\s*(localization-quality-rating)?\s*
                (mt-confidence)?\s*(preserve-space)?\s*(provenance)?\s*
                (storage-size)?\s*(target-pointer)?\s*(terminology)?\s*
                (text-analysis)?\s*(translate)?\$')">
            The space separated triples are not ordered alphabetically
            as per the ITS Data category identifier or contain illegal value.
        </sch:assert>
    </sch:rule>
</sch:pattern>
```

The last business rule of this class is defined over the Validation Module, functionality of which was discussed in chapter 3 and requires a detailed review of the entire functional dependency since it targets the Module constraints in full. We recall that the `val:rule` element defines only attributes, 4 of which are meant for the description of the actual content validation function (isPresent, isNotPresent, startsWith and endsWith) and the others can be added to control these rules in one or the other way. Although a custom attribute can be defined to target functions beyond the provided ones, but, overall, each ‘rule’ element can set only one of the aforementioned major properties and since we cannot regulate other namespaces, we will only validate the 4 native functions. We will start validating this nested functional dependency with a check on proper usage of the main attributes. This means that each ‘va:rule’ element must specify only one of the attributes and if non is present, then there must be a foreign attribute for custom functionality (first pattern in Listing 4.7). After ensuring that the element is not violating the fundamental requirement of the module, we can move on to CC25 and CC38 (Table 3.6), where the first states that if the ‘existsInSource’ is specified, exactly one of the followings must be set as well: ‘isPresent’, ‘startsWith’ or ‘endsWith’ (first pattern in Listing 4.7), and the latter requires ‘occurs’ attribute to be accompanied by ‘isPresent’, as it is the only rule that ‘occurs’ can affect (second pattern in Listing 4.7). Notably, value of this attribute must be a positive integer which is covered by XSD. We will then “pick” those `val:rule` elements, which declare one of the 4 major properties and apply appropriate value and other checks in accordance with the defined functionality. The reason for considering this strategy the optimal is that the number of steps are minimised through maximum generalisation, in form of selecting nodes based on the main property and also to avoid interfering with alien namespaces. Before proceeding to the further process of the Module’s validation, it is necessary to highlight a very important aspect of this module which was mentioned in chapter 3 as inheritance. We remember that the Module is allowed to appear at any of the extension points (‘file’, ‘group’ or ‘unit’) and can be “switched off” (overridden) at lower levels through the ‘disabled’ property. The behaviour and scope of the Module for each level is defined by the specification as follows:

- When the `<validation>` element occurs at the ‘file’ level, rules must be applied to all `<target>` elements within the scope of that `<file>` element, except where overrides are specified at the ‘group’ or ‘unit’ level;
- When `<validation>` occurs at the ‘group’ level, rules must be applied to all `<target>` elements within the scope of that `<group>`, except where overrides are specified in a nested `<group>` element, or at the ‘unit’ level;
- When `<validation>` occurs at the ‘unit’ level, rules must be applied to all `<target>` elements within the scope of that `<unit>`.

This statement of use immediately implies that ‘disabled’ cannot appear for the ‘file’ level (third pattern in Listing 4.7). These constraints are illustrated in Listing 4.7 which represent the first block of this functional dependency implementation.

Listing 4.7: Four Schematron Patterns for Validation Module Functional Dependency

```
<sch:pattern>
    <sch:rule context="val:rule">
        <sch:report test="@isPresent and (@isNotPresent or @startsWith
        or @ endsWith)">
            Only one of isPresent, isNotPresent, startsWith or endsWith
            is allowed.
        </sch:report>
        <sch:report test="@isNotPresent and (@isPresent or @startsWith
         or @ endsWith)">
            Only one of isPresent, isNotPresent, startsWith or endsWith
            is allowed.
        </sch:report>
        <sch:report test="@startsWith and (@isPresent or @isNotPresent
        or @ endsWith)">
            Only one of isPresent, isNotPresent, startsWith or endsWith
            is allowed.
        </sch:report>
        <sch:report test="@endsWith and (@isPresent or @isNotPresent
        or @ startsWith)">
            Only one of isPresent, isNotPresent, startsWith or endsWith
```

```
                is allowed.
            </sch:report>
            <sch:report test="not(@isPresent) and not(@isNotPresent)
             and not(@startsWith) and not(@ endsWith) and
             not(@*[namespace-uri()!='urn:oasis:names:tc:xliff:validation:2.0'])">
                When native module attributes isPresent, isNotPresent, startsWith
                 or endsWith are not used, a custom attribute must be set for
                 'val:rule' element.
            </sch:report>
        </sch:rule>
    </sch:pattern>
   <sch:pattern>
        <sch:rule context="val:rule[@existsInSource]">
            <sch:assert test="(@isPeresent and not(@startsWith) and not(@endsWith))
                      or (@startsWith and not(@isPeresent) and not(@endsWith)) or
                      (@endsWith and not(@isPeresent) and not(@startsWith))">
                The existsInSource attribute must be used with exactly one of
                isPresent, startsWith or endsWith.
            </sch:assert>
        </sch:rule>
    </sch:pattern>
    <sch:pattern>
        <sch:rule context="val:rule[@occurs]">
            <sch:report test="not(@isPresent)">
                The occurs attribute can only be used in pair with isPresent
            </sch:report>
        </sch:rule>
    </sch:pattern>
    <sch:pattern>
        <sch:rule context="val:rule[@disabled='yes']">
            <sch:report test="local-name(../..)='file'">
                The disabled attribute cannot be set to 'yes' when the
                Validation Module is at the 'file' level.
            </sch:report>
        </sch:rule>
    </sch:pattern>
```

Finally, we will proceed to the pattern checking while taking into account all the properties. However, it should be mentioned that these rules affect only the textual content of XLIFF and are not defined as normative constraints in the specification and, therefore, we would not include them in the final platform. It is up to CAT tools to implement and run the appropriate validation for the textual content. Although, since the functionalities of the Module's attributes represents a challenging task for Schematron, we will attempt to examine the expressivity of the schema language against these rules. XPath defines a number of functions to process strings and some of these functions, `starts-with`, `ends-with` and `contains`, could be used instead of regular expressions for startsWith, endsWith and isPresent attributes respectively (isNotPresent can be addressed by reversing the result of the `contains` function). The next step would be considering and applying all of the defined *parameters* for each major attribute. For instance, when a 'startsWith' attribute is set, it can be regulated through 'existsInSource' and 'caseSensitive' attributes. The first can be used to extend the pattern to the 'source' text (with default value of "no"), while the second attribute will determine how the pattern must be treated in terms of the character case (with default value of "yes"). The final check must be dedicated to whether or not the rule has been "disabled" or is still active at the point. In order to include all of the "switches" for the 'startsWith' attribute, we will distinguish the context node by the "caseSensitive" field. This will allow us to significantly minimize the volume and avoid extremely large blocks of predicates. It would be also useful to immediately exclude those "rule" elements with 'disabled' set to 'yes' already. With the context node chosen, we can proceed to the pattern check and apply the defined validation rule. Further we count the "eligible-targets", which are expressed by the following XPath predicated: child of 'segment' (to exclude Modules), no 'group' or 'unit' ancestors which have overridden the rule (the validation rule is still active and not “disabled”). The next step is to count those ‘targets’ which follow the rule, i.e. all the previous predicates and checking the start pattern. Finally and within the “test” expression we define the “assertion” of valid scenarios, taking into account the “existsInSource”: if the latter is set, the check will be expanded to ‘source’ elements as well. The corresponding Schematron pattern for ‘startsWith’ attribute is presented in Listing 4.8.

Lising 4.8: Schematron Rule for ‘startsWith’ constraint

```
<sch:pattern>
        <sch:rule context="val:rule[@startsWith][not(@disabled) or
               @disabled='no'][@caseSensitive='yes' or not(@caseSensitive)]">
            <sch:let name="start-pattern" value="@startsWith"/>
            <sch:let name="eligible-targets" value="count(../..//xlf:target
            [parent::xlf:segment][not(ancestor::xlf:group/val:validation/
            val:rule[@startsWith=\$start-pattern][@disabled='yes'])]
            [not(ancestor::xlf:unit/val:validation/val:rule
            [@startsWith=\$start-pattern][@disabled='yes'])])"/>
            <sch:let name="valid-targets" value="count(../..//xlf:target
            [parent::xlf:segment][not(ancestor::xlf:group/val:validation
            /val:rule[@startsWith=\$start-pattern][@disabled='yes'])]
            [not(ancestor::xlf:unit/val:validation/val:rule
            [@startsWith=\$start-pattern][@disabled='yes'])]
            [starts-with(.,\$start-pattern)])"/>
            <sch:let name="eligible-sources" value="count(../..//xlf:source
            [parent::xlf:segment][not(ancestor::xlf:group/val:validation/
            val:rule[@startsWith=\$start-pattern][@disabled='yes'])]
            [not(ancestor::xlf:unit/val:validation/val:rule
            [@startsWith=\$start-pattern][@disabled='yes'])])"/>
            <sch:let name="valid-sources" value="count(../..//xlf:source
            [parent::xlf:segment][not(ancestor::xlf:group/val:validation/
            val:rule[@startsWith=\$start-pattern][@disabled='yes'])]
            [not(ancestor::xlf:unit/val:validation/val:rule
            [@startsWith=\$start-pattern][@disabled='yes'])]
            [starts-with(.,\$start-pattern)])"/>
            <sch:assert test="((not(@existsInSource) or @existsInSource='no')) or
                (@existsInSource='yes' and \$eligible-sources=\$valid-sources))
                and \$valid-targets=\$eligible-targets">
                The startsWith pattern is violated.
            </sch:assert>
        </sch:rule>
    </sch:pattern>
```

It can be seen that the Schematron rule in Listing 4.8 addresses only the case-sensitive patterns and the other case can be implemented in a separate Schematron rule with modified context node selection and usage of `lower-case` (or `upper-case`) function of XPath before conducting the comparison. With regard to the other 3 fields, we consider all as expressible in Schematron following the same logic, however for the special case, when the 'occurs' attribute is set, a regular expression pattern must be used since XPath does not provide any function for counting the occurrence of a string pattern.

Overall, schema development for the Validation Module, where a pattern checking led to a large number of validation checks, is a good representative of complexity of relations and constraints which are set as the result of business rules and industry requirements. It also illustrates how it is important to to investigate the scope of a functional dependency when designing appropriate Schematron rules for expressing the constraint. Finally, the aforementioned artefacts examine limitations of the schema language and while processing functions, in this case for strings, might not provide enough instruments, but when it comes to "picking" specific nodes, even very complex conditions can be applied. Although the two mechanisms of "advanced navigation" to the desirable node and processing functions of its content are responsible for the expressivity power of Schematron, they are both driven by XPath. The schema language, on the other hand, provides the mechanism for support and effective usage of XPath through variables and the ‘test’ attribute to define XML integrity constraints. It was shown that how extremely complex relations can be expressed by combination of XPath predicates and while a relation of nodes might be defined starting from more than one place in the “tree”, the best way is to first identify the subject node directly (‘context’ attribute in Schematron) and then navigate to the object node from it by specifying the conditions. Such approach allows complete coverage of the constraint scope and ensures the correct selection of the nodes and since the relation is described directly, it represents the shortest way. For this particular constraint, ‘val:rule’ elements are the subject and in order to reach affected nodes (objects), we must first navigate to its grandparent, the first ‘xlf’ ancestor which can be any of file, group or unit, and then select all of the grandparents ‘target’ children. This strategy, that is followed throughout the Schematron rules development and is based on detailed research of the constraints, justifies the optimal design of the artefacts.

We will continue the Schematron discussion by reviewing the implementation of the rest of XLIFF 2.1 integrity constraints.

4.4.3 Business Rules

This section presents the mapping from the remaining XLIFF 2.1 functional dependencies to appropriate Schematron artefacts. We will follow the same strategy for this part by identifying the underlying functional dependency of the constraints and grouping them for development. Notably, we will only describe in detail those constraints that deserve individual attention from the XML validation point of view and the rules that can be targeted as standalone and by standard development patterns (e.g. foreign keys, co-occurrence) will be skipped.

The first important XLIFF constraint we would like to discuss in this chapter is related with one of XLIFF-specific behaviours as an XML vocabulary- bitext. While this constraint was presented (BR2 in Table 3.7) and explained in chapter 3, we will focus on the development of Schematron rule here. The constraint requires the optional 'xml:lang' property of the 'target' field to be equal to the 'trgLang' of the root element. We argued that literal comparison of these properties will cause valid scenarios, such as `trgLang="en", xml:lang="en-ie"` to be also reported as error. Fortunately, XPath has a function, `lang()`, which compares language codes in accordance with the BCP 47 regulations and returns 'true' when the language property of the context node ('target' element) is either the same language code of the target content (‘trgLang’ of ‘xliff’) or a variant of the language, but not vice versa. As this function performance exactly matches our needs, it was used to express this business rule in Schematron as shown in Listing 4.9.

Listing 4.9: Schematron Rule for ‘xml:lang’ in XLIFF 2.1

```
<sch:rule context="xlf:target[@xml:lang]
    [parent::xlf:segment | parent::xlf:ignorable]">
    <sch:let name="trgLang" value="/xlf:xliff/@trgLang"/>
    <sch:report test="not(lang($trgLang))">
        'xml:lang' attribute of the 'target' element and 'trgLang'
        attribute of the 'xliff' are not matching.
    </sch:report>
</sch:rule>
```

The same approach was used for implementing the similar constraints, BR17-18 in Table 3.7, defined for the Resource Data Module.

The next constraint that will be included in this section is over the ‘order’ attribute that is allowed at ‘target’ elements. This attribute was introduced in chapter 3, where we discussed the first part of this constraint, primary key, and in this section we will focus on the remaining part of the constraint. However, it is necessary to understand the rationale behind this business rule (BR10 in Table 3.7) in order to adequately design the corresponding validation artefact.

After a text has been segmented, the translated sentences might need to appear in a different order to shape the same paragraph of the source language. This means that the sequence of 'segments' (i.e. smallest blocks of the content) does not necessarily remain the same after translation, or, in other words, the order of "target segments" might be distinct from the one of 'sources'. The pattern of this common phenomena in the localisation industry is illustrated in Listing 4.10, where the translated content in French has a mixed order, B-C-A, in comparison with the order of the original (source) text in English, A-B-C (Example from XLIFF 2.1 specification). This means that the translated text must be inserted in a different order to shape the translated document. This process, known as "merging" in the localisation world, is performed by XLIFF 2 Merger Agents. The "order" attribute provides the solution for such use case by explicitly defining a different position for the `<target>` in the sequence of 'segments' and 'ignorables' within the scope of a 'unit'. The attribute is optional and if not explicitly set, its default value, i.e. the order of the sibling 'source', is implied.

Listing 4.10: Content With Shifted Segment Order After Translation

```
<p lang='en'>Sentence A. Sentence B. Sentence C.</p>

<p lang='fr'>Phrase B. Phrase C. Phrase A.</p>
```

For this mechanism to be hired properly and to perform the intended practical application in the localisation tools, a number of constraints are defined in 2 different sections of the specification and we differentiate the integrity constraints as:

1. The value of optional ‘order’ attribute must not be greater than the number of ‘segment’ and ‘ignorable’ elements combined within the enclosing ‘unit’ (Functional Dependency);
2. The value of optional ‘order’ attribute must be unique in the scope of ‘unit’ (Key type);
3. For every `<target>` with explicit ‘order’ attribute, the ‘target’ element whose sequence position is claimed must also explicitly set the ‘order’ attribute (Functional Dependency).

For instance, the XLIFF document in Listing 4.11 stores the content within 5 segment/ignorable blocks, with ‘ids’ A to E, and while the original text is organised by the sequence of A-B-C-D-E, the translation must be extracted in a different order: C-A-B-D-E. The 4th and 5th `target` elements are not required to set 'order' as they are in the same position with their sibling `<source>`. The XLIFF 2.1 instance of Listing 4.11 satisfies all of the defined rules and, therefore, is a valid document from the point of view of this integrity constraint.

Listing 4.11: An XLIFF 2.1 Document With Explicit “order” Attribute

```
<xliff xmlns="urn:oasis:names:tc:xliff:document:2.0"
    version="2.1" srcLang="en" trgLang="ru">
    <file id="f1">
        <unit id="1">
            <segment id="A">
                <source>“Tut, tut, child!”</source>
                <target order="2">"Малышке"</target>
            </segment>
            <ignorable id="B">
                <source> </source>
                <target order="3">:</target>
            </ignorable>
            <segment id="C">
                <source>said the Duchess</source>
                <target order="1">Герцогиня сказала</target>
            </segment>
            <ignorable id="D">
                <source> </source>
                <target> </target>
            </ignorable>
            <segment id="E">
                <source>“Everything’s got a moral, if only you
can find it.”</source>
                <target>"Во всем есть своя мораль, нужно только
уметь её найти!"</target>
            </segment>
        </unit>
    </file>
</xliff>
```

Listing 4.12 demonstrates how the 1st and 3rd parts of the aforementioned rules are developed in Schematron, where the XPath expressions are developed on the basis of the simplified version of the constraint (the additional predicate, `[ancestor::xlf:segment | ancestor::xlf:ignorable]`, is required to exclude the "XLIFF targets" which appear in other namespaces, e.g. Translation Candidates Module).

Listing 4.12: Schematron Rules for 'order' attribute

```
<sch:rule context="xlf:unit">
    <sch:let name="maxOrder" value="count(xlf:segment|xlf:ignorable)"/>
    <sch:let name="unit-id" value="@id"/>
    <sch:report test=".//xlf:target[@order>$maxOrder]
        [ancestor::xlf:segment|ancestor::xlf:ignorable]">
        Invalid value used for order attribute of 'target' element(s).
        It must be an integer from 1 to <sch:value-of select="$maxOrder"/>.
    </sch:report>
</sch:rule>
<sch:rule context="xlf:target[@order]">
    <sch:let name="order" value="@order"/>
    <sch:let name="actual-pos" value="count
        (../preceding-sibling::xlf:segment|
        ../preceding-sibling::xlf:ignorable)+1"/>
    <sch:let name="unit-id" value="ancestor::xlf:unit/@id"/>
    <sch:assert test="ancestor::xlf:unit//xlf:target[@order=$actual-pos]
        [ancestor::xlf:segment | ancestor::xlf:ignorable]">
        The corresponding 'target' element, 'order' attribute of which
        must be '<sch:value-of select="$actual-pos"/>', is missing in the
        enclosing 'unit'.
    </sch:assert>
</sch:rule>
```

Finally, with regard to the XLIFF 2.1 Processing Requirements, we produced one Schematron file for each group of XLIFF 2.1 Agents. In the previous chapter we explained how they are expressible in Schematron using the `document()` function of XPath, which enables the schema to provide a cross document validation with the schema to be applied to the modified file while the original document is referenced inside the Schematron rule. However, one of the rules, P4 (see Table 3.9), does not require access to the original document. Instead, for implementing this constraints, it is sufficient to look for those ‘data’ elements in the modified file which are not referenced at all, i.e. there is no inline code with a respective foreign key. The most important task of implementing the remaining constraints is to locate the exact corresponding elements in both original and modified files. Since an XLIFF 2 document may contain only one ‘skeleton’ (target of P1), choosing correct elements for P1 is a straightforward task. For the rest of the constraints, P2, P3 and P5, on the other hand, we must extract the precise “address” of the node. In order to locate the corresponding ‘segment’ in the original file (P2) we will use the identifiers of its ‘file’ and ‘unit’ ancestors as well as its position in the ‘unit’, since ‘id’ is optional for this element. This information will help us to uniquely identify and locate the needed ‘segment’. The context of P3, ‘type’ attribute, can appear at a number of inline elements, which all must specify an ‘id’ and, therefore, must be located in the same ‘file’, in the same ‘unit’ and have the same ‘id’ attribute. Listing 4.13 illustrates how this progressive constraint is implemented in Schematron with the original XLIFF document (original.xlf) being imported. Finally, and for P5, we must distinguish the host elements of ‘fs:fs’ by their corresponding scope. We will target ‘file’, ‘unit’, inline codes and ‘note’ elements separately to be able to track the corresponding nodes in the original file. However, it is inevitable to divide ‘note’ elements as well, since they can appear at three levels (file, group or unit) and also to use their position as ‘id’ is an optional field for these elements. All of the developed artefacts for progressive validation can be found on the attached to this thesis CD.

Listing 4.13: An XLIFF Processing Requirement Described in Schematron

```
<sch:pattern id="P3">
        <sch:rule context="xlf:*[@type]">
            <sch:let name="original" value="document('original.xlf')"/>
            <sch:let name="id" value="current()/@id"/>
            <sch:let name="type" value="current()/@type"/>
            <sch:let name="subtype" value="current()/@subType"/>
            <sch:let name="unit-id" value="ancestor::xlf:unit/@id"/>
            <sch:let name="file-id" value="ancestor::xlf:file/@id"/>
            <sch:let name="original-type" value="$original//
            xlf:*[@id=$id][ancestor::xlf:unit/@id=$unit-id]
            [ancestor::xlf:file/@id=$file-id]/@type"/>
            <sch:let name="original-subtype" value="$original//
            xlf:*[@id=$id][ancestor::xlf:unit/@id=$unit-id]
            [ancestor::xlf:file/@id=$file-id]/@subType"/>
            <sch:assert test="$type=$original-type or
            not($subtype=$original-subtype)">
                When 'type' attribute is modified, the 'subType'
                attribute must be updated or removed.
            </sch:assert>
        </sch:rule>
    </sch:pattern>
```

After targeting all of the XLIFF 2.1 integrity constraints, we will now provide a discussion on the optimisation of the artefacts to explain how the validation platform was finalised and implemented.

4.5 Optimisation

The topic of optimisation was mentioned at the beginning of this chapter and also was referred to during the Schematron artefacts presentation. We argued in the previous section that the chosen approaches in formation of blocks of Schematron rules and patterns are considered the optimal strategy of constraint expression in the schema language. This section will expand this discussion to explore the process of modifications which were applied to the Schematron rules to ensure the effective design and organisation, as well as usage of the advanced features of the schema language for informative error reporting. The other aspect of this topic, which this section aims to target, is to identify the optimal combination of the developed schemas to be used in the final validation platform.

To address the most important optimisation task, the selection of schemas, we divided the validation process of XLIFF 2.1 into two major parts: structural and integrity constraints. As we showed earlier, the RelaxNG schema showed a large overlap, in terms of functionality within the platform, with the XSD, which was already developed and is being widely used by XLIFF Agents. Based on this, our choice was to exclude RelaxNG from the final set of schemas. As the result, RNG schema was not further developed for XLIFF 2.1 Modules and we had to re-assign the RelaxNG-specific constraint (co-occurrences) to Schematron. So, our validation platform is based on an NVDL mapping file which handles multiple namespaces and modularity of XLIFF 2.1 instances, a set of W3C XML Schema file for structural and datatype validation of each namespace, and, finally, a number of Schematron files for checking the integrity constraints. It was decided to provide separate Schematron files for the Core and Modules to conform to the independence of the namespaces which is highlighted in the specification. The NVDL script will be described in the following section. This platform is represented in an interoperable environment, where the interaction of the hired technologies are standardised and, therefore, the performance is guaranteed to produce the same result, regardless of the implementation platform. The functionality of our validation solution, on the other hand, is non-redundant which highlights its effective design and development through detailed study of the constraints interrelations and scopes.

The main contribution to the aforementioned qualities of our platform is made by the optimised design of the developed Schematron rules. This schema language, unlike RelaxNG and W3C Schema, is not a well-researched technology. This lack of research might be the result of the fact, that XML functional dependencies and particularly business rules, which represent the main target of Schematron, is also not a well-studied subject yet. However, in this sections we will present the important findings of our research on development and design of Schematron rules for expression of XLIFF 2.1 integrity constraints which, as we discussed in the previous and current chapters, represent business rules of different complexity. Our hope is that this pilot study would provide valuable information for the further research.

Rule-based approach for XML validation is the most fundamental characteristic of Schematron. This means that each "scenario" that the integrity constraints defines, whether it is valid or invalid, must be considered individually in the design of appropriate Schematron rules. For comparison, RelaxNG and XSD define general patterns or templates that an XML instance must match in order to be valid. On the other hand, Schematron rules are applied through two key mechanisms; assert and report. The first approach will raise an error if the defined assertion is failed, whilst ‘report’ specifies illegal content. It has been claimed that usage of these methods is rather a matter of style and it is up to the developer to choose whether to “make assertions” or “report errors”. Although, technically speaking and as the available literature suggests, these fields perform on the same logic with ‘report’ being a ‘reversed assertion’ and, therefore, they can be used interchangeably. In practice, however, and for some cases this presumption is not true. It is important to distinguish that ‘assert’ specifies valid scenarios, when ‘report’ defines invalid scenarios and as we argued earlier about the necessity of considering all scenarios, it is reasonable and purposeful to choose the ‘assert’ approach when there are more invalid scenarios implying from the integrity constraint. The essential part of Schematron rules development is deep understanding of the constraint and its propagation domain to effectively define the Schematron properties (e.g. ‘test’ attribute) and use a proper set of ‘assert’ and/or ‘report’. For instance, the functional dependencies over the XLIFF 2 ‘sc’/‘‘ec’ fields represent a large number rules, BR3-BR7 in Table 3.7 and CC3-CC5 in Table 3.6. To express this constraint we hired the following strategy using 6 Schematron rules:

1. Select ‘sc’ elements in ‘source’ with ‘isolated’ attribute set to ‘yes’ and check a) if there is any ‘ec’ within the same ‘unit’, in a ‘source’ that refers to this ‘sc’ element by the ‘startRef’ attribute (invalid scenario- report) b)

if there is one, and only one ‘ec’ in a different ‘unit’, appears after the ‘sc’ element, is in ‘source’ and its ‘id’ equals to the ‘id’ of the context ‘sc’ (valid scenario- assert) c) the ‘canCopy’, ‘canDelete’, ‘canOverlap’ and ‘canReorder’ attributes of the ‘sc’ have the same values as the corresponding (described in the previous statement) ‘ec’ (assert);

2. Select ‘sc’ elements in ‘target’ with ‘isolated’ attribute set to ‘yes’ and check the same as above with the ‘source’ being changed to ‘target’;

3. Select ‘sc’ elements in ‘source’ with ‘isolated’ attribute set to ‘no’ or missing and check a) if there is one, and only one ‘ec’ element in the same ‘unit’, in a ‘source’ and refers to the ‘sc’ by its ‘startRef’ attribute (assert) b) the ‘canCopy’, ‘canDelete’, ‘canOverlap’ and ‘canReorder’ attributes of the ‘sc’ have the same values as the corresponding (described in the previous statement) ‘ec’ (assert)

4. Select ‘sc’ elements in ‘target’ with ‘isolated’ attribute set to ‘no’ or missing and check the same as above with the ’source’ being changed to ‘target’;

5. Select ‘ec’ elements in ‘source’ and check a) proper usage of ‘id’/’startRef’ attributes based on the value of ‘isolated’ field (assert) b) if the ‘isolated’ is yes, there must be one, and only one ’sc’ element in a different ‘unit’, in a

‘source’, has the ‘isolated’ attribute set to ‘yes’ and which appears before the ‘ec’ and refers to it by the ‘id’ attribute (assert) c)if ‘isolates’ is set to ‘no’ or missing, there must be one, and only one ‘sc’ in the same ‘unit’, in a ‘source’, which appears before the end code, has the same value for the ‘isolated’ field and refers to this ‘ec’ by the ’id’ attribute;

6. Select ‘ec’ elements in ‘target’ with ‘isolated’ attribute set to ‘no’ or missing and check the same as above with the ’source’ being changed to ‘target’.

It is clear that the ‘canCopy’, ‘canDelete’, ‘canOverlap’ and ‘canReorder’ attributes of ‘ec’ do not need to be checked against their corresponding attributes as this check is already performed for ‘sc’ and we ensured that every ‘ec’ has one, and only one corresponding ‘sc’, so defining the same check for ‘ec’ will be redundant. Our Schematron rules were developed on the basis of detailed analyses of the target constraints and their implications, sound and effective deployment of the constraint in terms of logical parts of the regular expressions and using the optimal set and order of Schematron features to conceptually describe the constraint, which means to ensure that all the possible valid and invalid scenarios of the constraint are defined and covered in the appropriate Schematron rule. The ‘sc’/‘ec’ example is a good representative of our design strategy in development of the validation platform for XLIFF 2. While the aforementioned rules are too large in volume to be included in this chapter, Listing 4.14 shows a fraction of Rule 2, which checks if the ‘canReorder’ attribute of ‘sc’ elements of ‘target’ and with ‘isolated=yes’ matches the value of the corresponding ‘ec’ element.

Listing 4.14: Schematron Rule for validation of the functional dependency over ‘canReorder’ property

```
<sch:rule context="xlf:sc[ancestor::xlf:target][@isolated='yes']
    [ancestor::xlf:segment | ancestor::xlf:ignorable]">
    <sch:report test="@canReorder='no' and
        not(following::xlf:ec[@id=$id][ancestor::xlf:target]
        [ancestor::xlf:file/@id=$file-id][not(ancestor::xlf:unit[@id=$unit-id])]
        [ancestor::xlf:segment | ancestor::xlf:ignorable][@canReorder='no'])">
        'canReorder' attribute is not matching with the 'canReorder'
        attribute of the corresponding 'ec' element .
    </sch:report>
    <sch:report test="(@canReorder='yes' or not(@canReorder)) and
        not(following::xlf:ec[@id=$id][ancestor::xlf:target]
        [ancestor::xlf:file/@id=$file-id][not(ancestor::xlf:unit
        [@id=$unit-id])]
        [ancestor::xlf:segment | ancestor::xlf:ignorable]
        [@canReorder='yes']) and
        not(following::xlf:ec[@id=$id][ancestor::xlf:target]
        [ancestor::xlf:file/@id=$file-id][not(ancestor::xlf:unit
        [@id=$unit-id])]
        [ancestor::xlf:segment | ancestor::xlf:ignorable]
        [not(@canReorder)])">
        'canReorder' attribute is not matching with the 'canReorder'
        attribute of the corresponding 'ec' element.
    </sch:report>
</sch:rule>
```

The final part of Schematron optimisation is on producing an advanced and informative error reporting. We used Schematron “rich” features to enrich the error log with a web link of the Specification and, where possible, the XLIFF FragId notation of the node which failed the validation process. An example of such rule is shown in Listing 4.15 for FK4 in Table 3.5. Notably, the advanced error reporting is the main reason for avoiding “abstract patterns” in Schematron, which like in RelaxNG enable rules to be reused. It is also possible to automate the process of troubleshooting and error fixing with the knowledge of FragId of the element.

Listing 4.15: A Schematron Rules with Advanced Error Reporting

```
<sch:pattern id="FD29">
    <sch:title>dataRefStart attribute must point to a data element
        within the same unit</sch:title>
    <sch:rule context="xlf:*[@dataRefStart][ancestor::xlf:source]"
        see="http://docs.oasis-open.org/xliff/xliff-core/v2.0/
        xliff-core-v2.0.html#datarefstart">
        <sch:let name="fragid" value="concat(ancestor::xlf:file/@id,
            '/u=',ancestor::xlf:unit/@id,'/',current()/@id)"/>
        <sch:assert test="ancestor::xlf:unit//xlf:data
            [@id=current()/@dataRefStart]" diagnostics="fragid-info">
            'dataRefStart' attribute must point to a 'data'
            element within the same 'unit'.
        </sch:assert>
    </sch:rule>
</sch:pattern>
```

4.6 NVDL Mapping Script

We have shown so far that all of the XLIFF 2.1 integrity constraints can be expressed using a combination of W3C XML Schema or RelaxNG with Schematron. Although the first two schema languages allow integration of the latter for advanced functionality, we will avoid such approach for three reasons; first, this method is not standardised and, therefore, is in controversy with one of our main objectives- platform neutrality. Secondly, the schema language provides a suitable solution for management of namespaces in the multimodal XML document. Finally, if NVDL is selected as the mapping script, XLIFF users can easily append additional schemas for the custom namespaces, embedded within XLIFF 2.1 documents at extension pints.

We developed the NVDL script which would validate the specified namespaces (XLIFF 2.1 Core and Modules) of an XLIFF document against the appropriate schema or number of schemas. For those constraints which require cross namespace check (e.g. mtc:ref, FK10 in Table 3.5), we had to implement an advanced feature of the schema language, which would run the schema against the entire document, instead of the “detached block” of the target namespace. This mechanism is called “modes” in NVDL and a fraction of it is provided within Listing 4.16, which provides the Schematron rules access to all of namespaces the document contains.

Listing 4.16: Usage of NVDL “modes” in XLIFF 2.0 Validation

```
 <mode name="initial">
    <namespace ns="urn:oasis:names:tc:xliff:document:2.0">
        <validate schema="xliff_core_2.0.xsd" useMode="attach-all"/>
        <validate schema="xliff_core_2.sch" useMode="attach-all"/>
        <validate schema="resource_data.sch" useMode="attach-all"/>
    </namespace>
</mode>
<mode name="attach-all">
    <anyNamespace>
        <attach/>
    </anyNamespace>
</mode>
```

The other feature of NVDL that we used in our platform is the “match’ attribute which can be set to ‘elements’ or ‘attributes’ value to apply the schema to the appropriate nodes of a namespace. The Format Style Module, for instance, defines only attributes within its namespace which would not be validated if the NVDL script does not specify the target nodes (i.e. attributes). Listing 4.17 demonstrates how the FS module namespace is validated in our platform.

Listing 4.17: Matching attributes of the Format Style Module in NVDL

```
<namespace ns="urn:oasis:names:tc:xliff:fs:2.0" match="attributes">
    <validate schema="fs.xsd"/>
</namespace>
```

4.7 Conclusion

In this chapter we addressed the sub-questions of our research and discussed our way through the development and effective design of the validation platform this research aims to deliver. We reviewed the existing validation schema for XLIFF 2.1, XSD, and argued that however it could be improved, but it cannot be modified due to administrative restrictions and, therefore, those XLIFF 2.1 integrity constraints that were identified as "expressible in XSD" in the previous chapter must be developed in an alternative language. Further we explained and reviewed the process and logic of expressing the constraints in RelaxNG and Schematron, as well as creating the NVDL mapping script. Finally, we showed that the Relax NG was not included in the final platform because of the de facto duplicating role it was performing and, therefore, would cause redundant functionality of the validation platform.

This chapter presented a valuable and broad investigation of Schematron rules design and development and examined its limitations. These aspects of the language, as we argued, lacks research and study in the literature. The detailed explanation and step-by-step review of the strategy we used for Schematron rules of XLIFF could provide a reliable reference for further research or similar work. We demonstrated the important role of applying detailed analyses to the integrity constraints to obtain a clear understanding of the entire set of dependencies a business rule might imply, like the XLIFF 2.1 Validation Module.

In the next chapter we will discuss the last step of our research- evaluation and implementation of the developed validation platform.

Chapter 5

Evaluation and Implementation

5.1 Introduction

This chapter will present the final step of our research which is evaluation of the validation artefacts for XLIFF 2.1. This step is in line with our main research methodology, design and development (see chapter 1), where evaluation of the developed artefacts is considered a vital component of the research (see chapter 4).

We will start the chapter from describing the hired research methodology for the task of evaluation. Further, we will explore existing methods for implementation and deployment of the artefact and will present the chosen approach for our work. The designed and developed validation platform for XLIFF 2.1 then will be examined in practice and evaluated in accordance with the chapter's methodology. Finally, we will discuss the results of this assessment and and provide a comparison of our platform with existing validation applications of XLIFF 2.

5.2 Methodology

We designed this stage of our research, evaluation, on the basis of qualitative research methodology by testing the platform in practice and validating XLIFF 2.1 files against it. We will follow a common practice of evaluation in software development known as *test suite* which contains sets of test cases such as test levels and test types (Nishi 2012). Our test suite will represent the entire valid and invalid cases of XLIFF 2 data model at lower levels (not abstract), also known as *executable test suite*. For this purpose, we reanalysed the integrity constraints which were gathered in chapter 3 to elaborate all the possible violations of each constraint and also to define the valid scenarios in relation to the rule. The appropriate XLIFF 2.1 files were then generated to be validated against the platform, which is required to report all invalid files and pass all valid instance. We consider this approach for the evaluation as reliable for the following reason: the test files are produced based directly on the integrity constraints and not on intentional violation of the artefacts and Schematron rules in particular. Additionally, the validation artefacts, as was discussed in chapter 4, were developed by grouping, reorganizing or merging the collected constraints and, therefore, the generated files in fact can adequately measure the performance of our platform.

After the design and development of DSDL-based validation artefacts and their evaluation, we decided to create an open API in order to make the platform accessible through the web. Such decision was made in agreement with Savourel's (2007) suggestion that hiring open API enables interoperability among tools and technologies. We chose to design a RESTful web service as a container for the developed artefacts using the Software as a Service (SaaS) model. This selection is supported by the observable tendency of modelling and proposing usage of Service-Oriented Architectures and web services in the localisation industry (Wasala et al 2011; Lewis et al 2010; Lewis et al 2009b) and will enable our *service* to be integrated in workflows for validation purposes.

5.3 Implementation

We used oXygen XML Editor as the development platform to deliver the artefacts presented within this thesis. For API and other forms of implementations (e.g. as a web service), however, the task is to adopt Java implementations of the schema languages used in our artefacts. Unfortunately, NVDL has fairly weak, simplified and rather incomplete implementations, especially when an NVDL script invokes Schematron rules.[1] In contrast, W3C XML Schema and RelaxNG enjoy full implementations in the form of libraries for common programming platforms (e.g. Java, .NET). On the other hand, Schematron developers have followed a completely different approach, where the only implementation [2] , used in "oXygen" as well, compiles (converts) a Schematron file to an XSLT stylesheet. The output then can be applied to XML instances using XSLT processors. This way might have been preferred due to the "XML pathfinder" the two languages share- XPath. As none of the NVDL implementations actually perform this intermediary task (compile embedded Schematron rules and apply the XSLT stylesheet), and also because improvement of the existing implementation is out of the scope of this research, we decided to, at least for the first release, build the validator (API) without the developed NVDL script (the oXygen Editor uses a specifically customised version of an NVDL implementation). This decision is also in line with our goal to follow the "oXygen" logic as much as possible to guarantee the same results we received during the design and development. Therefore, we compiled the Schematron files to XSLT stylesheets (by

[1]By "Schematron" we refer to the ISO version of the language. In fact, Schematron 1.5 has reliable and complete implementations.

[2]This implementation is recommended on the official web site of Schematron, http://www.schematron.com/implementation.html [accessed 26 Sep 2016]

oXygen Editor) to be used alongside with the XSD schemas as the resources of our web service. The resource files are implemented in Java using the native `javax.xml.validation.Validator` library for W3C Schema and Saxon processor for XSLT stylesheets. In the following section we will evaluate the platform by using oXygen Editor to run the NVDL script for testing alongside with the XSLT stylesheet for performance and runtime measurement.

5.4 Evaluation of Validation Artefacts

The XLIFF 2.1 standard, like any other XML vocabulary, is defined within its specification which declares a set of elements and attributes, their namespaces and a number of dependencies, structural and functional, over them in plain text and natural language (human-readable). Throughout the research we showed how the specification was studied in detail and the data model was analysed to truly identify and extract the entire set of the standard's integrity constraints. Our validation platform is, in fact, an encoded and machine-readable representation of the XLIFF 2.1 specification and this section is dedicated to evaluate this claim.

The firs step of the evaluation is to determine the minimum, but sufficient suite of XLIFF 2.1 files to be validated by the platform. The files will be generated based on the integrity constraints of the standard and in two major groups of valid and invalid instances, which in combination represent all the errors to be caught for each rule as well as the cases where the constraints are satisfied and, therefore, validation must be passed. We will recall the XLIFF 2.1 integrity constraints from chapter 3 and discuss the development of our test suite for each group. We identify evaluation of structural and datatype parts of the platform out of scope of this work as these tasks are performed by the XSD schema and our invalid instances do not violate these types of constraints and will contain the minimum required nodes and structure. The test suite (92 files in total) is stored in the CD which is attached to this thesis.

The testing is conducted using the oXygen XML Editor v. 17.0 which is believed to be the most common and advanced tool in the XML world and, as a matter of fact, offers the best interface for running Schematron and NVDL schemas. The results of the evaluation will be presented in separate sections for each type of integrity constraints. Since "oXygen" does not provide any accurate information on the execution time of each validation process, it was decided to measure this factor by applying the aforementioned XSLT stylesheet. Following the XSLT runtime measurement conventions, each transformations was executed 10 times using Saxon-HE 9.7.0.7J (Saxonica) engine and the corresponding average run-time (in milliseconds) for each file is presented within the A.R. (Average Run-time) columns of tables 5.1, 5.2, 5.3 and 5.4. This factor will be discussed in the Performance and Comparison section.

5.4.1 Primary Keys

We descried the importance of key constraints throughout this work and argued that they are prerequisite for other functional dependencies to be defined properly. It was also highlighted in chapter 4 that as a part of effective design, the keys are validated at the very first step in our platform.

For XLIFF 2.1 key constraints to be satisfied, the value of specific attributes must be unique among the host elements in a certain scope and **a** duplication in the same scope is necessary and sufficient for violating the constraint. Thus, each key constraint (Table 3.4) must be represented in the suite by one invalid, whit duplication(s) in the scope, and one valid instances. The instances were designed to contain all the possible errors that must be reported to ensure that all logical levels of the data model are protected. The results of the validation for invalid instances are presented in Table 5.1, where number of the *Errors Expected (E.E.)*

is equal to or greater than the *Errors Reported (E.R.)* for each file (e.g. 'K1-IV' is the *Invalid* file for K1 in Table 3.4).

Table 5.1: Evaluation of the Platform for XLIFF 2.1 Primary Keys

File Name	E. E.	E. R.	A.R.	Notes
K1-IV	1	1	4.15	'file' level is protected
K2-IV	3	4	5.83	The extra report is for the first occurrence. 'group' is protected
K3-IV	1	2	5.19	The extra report is for the first occurrence. 'unit' is protected
K4-IV	3	3	7.32	'note' elements are protected at all levels
K5-IV	2	2	6.88	'data' is protected for both Core and MTC module
K6-IV	20	22	18.46	The extra reports represent the first occurrences. Large number is due to drawback of 'sc'/'ec'. Inline content is fully protected in 'unit'
K7-IV	2	3	5.68	The extra report is for the first occurrence, FD for 'order' is also triggered. 'target' is protected in 'unit'

File Name	E. E.	E. R.	A.R.	Notes
K8-IV	1	1	5.98	MTC module is protected
K9-IV	3	4	4.72	The extra report is for the first occurrence. GLS module is protected
K10-IV	2	2	5.03	MDA module is protected
K11-IV	4	6	13.73	The extra reports represent the first occurrences. RES module is protected
K12-IV	3	4	6.12	The extra report is for the first occurrence. CTR module is protected for 'property' attribute
K13-IV	6	6	7.14	CTR module is protected for 'id'
K14-IV	1	2	4.23	The extra report is for the first occurrence. CTR module is protected for 'version' attribute
K15-IV	5	6	9.43	The extra report is for the first occurrence. ITS module is protected

As Table 5.1 suggests, the platform completely protects the key constraints for XLIFF 2.1 and since *all* the valid instances successfully passed the validation, we can conclude that the platform is stable in terms of keys and we can proceed to the next set of files.

5.4.2 Foreign Keys

Attributes which are identified as foreign keys in Table 3.5 reference identifiers of other elements, than they are defined in, in a specific scope. In other words, to satisfy this type of constraints, it is necessary and sufficient that at least one instance of the referenced element (or elements in case of `IDREFS`, i.e. FK7-FK9) is present in the declared scope. Consequently, absence of the referenced element would break the rule and, therefore, foreign keys require one valid and one invalid instance for the evaluation. Like for shared keys, the foreign keys which are used in multiple elements must be represented by all of the host elements at least once. Table 5.2, organised as for keys, demonstrates the results of the evaluation for this type of constraints. It was decided to group FK3-4 and FK 8-9 as these attributes must appear together.

Table 5.2: Evaluation of the Platform for XLIFF 2.1 Foreign Keys

File Name	E. E.	E. R.	A. R.	Notes
FK1-IV	4	9	16.46	Extra reports because of ‘sc/ec’ violation. ‘copyOf’ is protected for all elements at ‘unit’
FK2-IV	3	3	9.65	‘dataRef’ is protected for all elements at ‘unit’

File Name	E. E.	E. R.	A. R.	Notes
FK3-4-IV	2	2	6.40	Data referencing for ‘pc’ is protected at ‘unit’
FK5-IV	2	2	9.22	Comment Annotation (referencing ‘note’) is protected for ‘mrk’ and ‘sm’ at ‘unit’
FK6-IV	7	7	10.34	Large number is due to drawback of ‘sc’/‘ec’. ‘startRef’ is protected for ‘ec’ and ‘em’ at ‘unit’
FK7-IV	3	3	11.23	‘subFlows’ is protected for all elements at ‘file’
FK8-9-IV	2	2	10.63	Sub-flows mechanism is protected for ‘pc’ at ‘file’
FK10-IV	1	1	8.17	Referencing for MTC module is protected at‘unit’
FK11-IV	2	2	12.16	Referencing for GLS module is protected at‘unit’
FK12-IV	1	1	10.39	Referencing for RES module is protected at‘file’
FK13-IV	1	1	8.06	Referencing for Quality Issues of ITS module is protected at‘unit’

File Name	E. E.	E. R.	A. R.	Notes
FK14-IV	1	1	6.54	Referencing for Provenance Records of ITS module is protected at‘unit’

With respect to the valid instances of foreign keys, *all* of the files passed the validation without errors and, therefore, it can be concluded that the validation platform implements the XLIFF 2.1 specification in terms of primary keys and foreign keys and we can now conduct the evaluation for other integrity constraints.

5.4.3 Co-occurrence Constraints

In order to address this group of constraint we must first distinguish conditional and biconditional co-occurrence constraints. To represent the valid cases, conditional statements would require only one instance for each of the valid and invalid cases, whilst biconditional constraints define 2 invalid cases, for failure of each condition, and one valid. Since the logical levels are secured by keys and also because these constraints are rather independent, it was decided to represent all of the constraints in one file for each namespace (i.e. the Core and Modules separately) but using different stable levels of the data model, e.g. ‘unit’, to ensure that within each level only one error is presented. As result we generated 7 valid files, which *passed* the validation, and 7 invalid files which are presented in Table 5.3.

Table 5.3: Evaluation of the Platform for XLIFF 2.1 Co-Constraints

File Name	E. E.	E. R.	A.R	Notes
CCs-Core-IV	23	23	26.94	All Core co-occurrence constraints are protected
CCs-MTC-IV	1	1	5.38	CC15 is protected at the MTC module
CCs-RES-IV	5	5	8.36	All Core co-occurrence constraints are protected for RES module
CCs-FS-IV	2	4	10.23	The extra report is for inevitable illegal use of ‘ec’. Constraints for usage of FS attributes are protected
CCs-SLR-IV	1	1	9.12	Constraints for the SLR module and its attributes are protected
CCs-VAL-IV	2	2	6.17	All Core co-occurrence constraints are protected for VAL module

File Name	E. E.	E. R.	A.R	Notes
CCs-ITSM-IV	13	13	16.24	All Core co-occurrence constraints are protected for usage of ITS module attributes

As the table suggests, all of co-occurrence constraints of the standard is covered by the platform.

5.4.4 Business Rules

In chapter 3 we presented and reviewed a small group of business rules separately and it was decided to target these constraints by regular expressions. The developed expressions were further discussed and demonstrated in chapter 4 and since they imply an extremely large range of valid and invalid cases, it is not reasonable to evaluate them on the same logic as other constraints. The expressions represent the exact pattern for the targeted values as declared by the specification and can be considered the adequate transformation of the constraint into regular expressions. However, we will include a valid and an invalid file (random cases) for each expression in the suite, so all of the constraints would be represented. We followed the same logic of gathering all the rules within one valid and one invalid instance for each namespace and use different levels of XLIFF for each rule. The results of this test is represented in Table 5.4 and it can be seen that the drawbacks of some constraints can have a broad range and invoke other errors as well, but overall and taking into account that *all* valid files successfully passed the validation, the validation platform proved its performance in practice.

Table 5.4: Evaluation of the Platform for XLIFF 2.1 Business Rules

File Name	E. E.	E. R.	A.R.	Notes
BRs-Core-IV	18	28	33.45	Extra reports are due to drawbacks of illegal ‘sc’/‘ec’ usage; Includes RegEx for ‘cp’; All BRs are protected for Core
BRs-GLS-IV	1	1	5.10	Minimum content is protected for GLS module
BRs-CTR-IV	11	11	16.87	All BRs are protected for CTR module
BRs-RES-IV	2	2	4.24	All BRs are protected for RES module
BRs-VAL-IV	12	12	16.15	All BRs are protected for VAL module

File Name	E. E.	E. R.	A.R.	Notes
BRs-ITSM-IV	6	24	31.71	The large number of errors is due to drawbacks of illegal attributes combinations; The functionality of ITSM is protected

Finally and with regard to progressive constraints, it was decided to generate test files for each group of affected agents; Writers, Modifiers and Enrichers. Valid and invalid cases are represented within pairs of files and they all showed the exact expected result in practice. The full package of the files can be found on the attached CD.

5.4.5 Performance and Comparison

In this section we will provide information on the efficiency of the validation platform and also compare its functionality with existing methods of XLIFF validation. Although, the main characteristic and advantage of our validation solution is its transparency and platform neutrality.

To start with the platform efficiency and describing the execution runtime, we would first explore and discuss those numbers, which are not presented within the previous sections, i.e. runtime for valid instances of the test suite. Table 5.5

provides this information where each valid file is given and average execution runtime (A.R.) in milliseconds.

Table 5.5: Execution Runtime for Valid Instances of the Test Suite

File Name	A.R.	File Name	A.R.
K1-V	9.30	K2-V	11.55
K3-V	8.13	K4-V	14.68
K5-V	27.65	K6-V	23.85
K7-V	15.67	K8-V	18.38
K9-V	24.41	K10-V	24.91
K11-V	17.61	K12-V	26.02
K13-V	29.79	K14-V	31.93
K15-V	15.15	FK1-V	11.22
FK2-V	16.26	FK3-4-V	12.48
FK5-V	15.60	FK6-V	13.88
FK7-V	10.84	FK8-9V	12.41
FK10-V	13.11	FK11-V	14.58
FK12-V	10.64	FK13	11.76
FK14-V	8.97	CCs-Core-V	28.87

File Name	A.R.	File Name	A.R.
CCs-FS-V	10.58	CCs-ITSM-V	19.57
CCs-MTC-V	19.16	CCs-RES-V	15.17
CCs-SLR-V	19.04	CCs-VAL-V	24.51
BRs-Core-V	37.69	BRs-CTR-V	21.29
BRs-GLS-V	18.16	BRs-ITSM-V	32.18
BRs-RES-V	23.26	BRs-VAL-V	19.78

It can be seen from the table that valid instances generally take more time to pass the validation process and this is due to fact, that they contain larger number of elements and attributes to provide sufficient valid case at all levels of the XLIFF data model. Comparing these files with the invalid ones with larger runtime, i.e. K6-IV and K11-IV in Table 5.1, FK1-IV in Table 5.2, CCs-Core-IV and CCs-ITSM-IV in Table 5.3 and members of Table 5.4 (except for Glossary and Resource Data modules), we could conclude that the validation process runtime is directly affected by the file size and, more importantly, by the number of those elements and attributes which qualify for the specified XPath expressions and have to pass the check, regardless of their validity.

Finally, in this section we would provide a comparison of our validation platform with the existing solutions for XLIFF 2. To conduct this pilot study, we validated our test suite against Lynx, a part of Okapi framework for XLIFF, and XMarker, an XLIFF 2 marker. The results of this comparison is presented in Table 5.6, where the platforms are first identified by name and the core components, and then tested for validation performance in the following categories: structure and datatype (S&D), keys (K), foreign keys (F.K.), co-occurrence constraints (C.C.), business rules (B.R.), progressive constraints (P.C.) and their ability to recognize and handle XLIFF fragment identification (F.I.).

Name	Core	S&D	K	F.K.	C.C.	B.R.	P.C.	F.I.
XLIFF 2 Normative Validation	NVDL (XSD and Schematron)	Yes	Yes	Yes	Yes	Yes	Yes	Yes
XMarker	XSD	Yes	No	No	No	No	Yes	Yes
Lynx	XSD and Java	Yes	Yes	p[3]	p	p	No	Yes

Table 5.6: Comparison of XLIFF 2 Validation Platforms

[3]partially

As the table suggests, all the validation applications rely on XSD for structural and datatype validation and, therefore, they all deliver the same functionality with this regard. While XMarker validation is solely based on W3C Schema, it does not report any of advanced validation tasks, however, it provides facility for fragment identification and processing requirements for other functionalities of the application. Lynx, on the other hand, performs a Java-based validation for those constraints not covered by XSD. Although, Fk5 (Table 3.5), CC12, CC13 and CC14 (Table 3.6) and BR2 and BR9 (Table 3.7) are not implemented in the platform and it does not provide validation for progressive constraints.

5.5 Conclusion

The objectives of this chapter were set to finalize this research by providing overall analyses and conducting evaluation of the output of our work- Advanced validation platform for XLIFF 2.1. Within this chapter we argued that our platform is rather an instance of the standard's specification which can be processed and consumed by the automated nodes of an XLIFF-based workflow. To prove this claim, we validated the designed test suite against our platform and the results of the evaluation stage demonstrated that the artefacts indeed represent the machine-readable specification of XLIFF 2.1 and are capable of expressing and applying the entire set of the integrity constraints. We also highlighted the role of the effective design and optimisation in providing a reliable and sound platform which was tested for efficiency and compared to the current solutions as well.

Finally, we discussed the implementation of our platform and described that lack of open, accessible and complete implementations of the target schema languages led us to seek alternative solutions for the deployment of the XLIFF 2.1 validation platform.

The validator is also soon to be embedded in the XLIFF Framework of oXygen Editor and also to be accessible as a part of FREME project, a digital content enricher engine which is based on XLIFF roundtrip.

Chapter 6

Conclusion and Future Directions

This chapter represents overall conclusions of our research as well as the recommendations and suggestions this work revealed for the further research, however detailed conclusions were provided in appropriate sections for each chapter. We will first have a brief review of the objectives of our work, the research questions it aims to target and the methodology we used to investigate the answers. Then, we will summarize the process our research which is represented within chapters 3, 4 and 5 to restate the achievements and findings of this thesis. Finally, we will provide appropriate recommendations, for every aspect of our research, which might be used as guidelines and directions for future research.

6.1 Summary

The problem this research attempts to solve is the semantic interoperability issues of XLIFF 2-based workflows. These issues span three major areas in the literature: Localisation, Interoperability and XML. In chapter 1 we identified the scope our work in the underlying domains and introduced the sub-areas of the literature, in which this work is defined. We concluded that although adoption of XML-based open standards addressed the syntactic interoperability issues of localisation workflows, but problems at the semantic level soon started to occur, mainly due to the comprehensive nature of common and contemporary standards. Particularly for XLIFF, the most widely-adopted localisation standard, the interoperability issues occur at the semantic level, where the users must conform to the specified rules and have the same interpretation of functionalities the standard offers. Further we showed that a standardised and transparent validation platform, based on declarative methods, which can express all of the XLIFF 2 conformance requirements throughout the specification is capable of enabling semantic interoperability and guarantee compliance among all of "XLIFF Agents". We conducted a detailed review in the related literature for the aforementioned topics to investigate the areas, to which the specified research questions are related. Our literature review helped us to learn and understand the nature of semantic interoperability problems for XML environments and to explore and evaluate the existing solutions and current tendencies. In fact, our review attempts to provide enough fundamental and conceptual information, needed to adequately target the research question. The review also helped us to identify the gaps and subjects which lack research in the literature, most importantly the industry-driven data models from the XML point of view. Our literature review is presented in chapter 2.

We defined the main research question (RQ) of this thesis in chapter 1 as: **Are standardised declarative methods expressive enough to define XLIFF 2.1 a business-driven XML vocabulary?**

The RQ was targeted within chapter 3, where an extended study of the XLIFF 2.1 specification helped us to first gather all integrity constraints and further examine them against the selected schema languages and demonstrate the possibility of such validation platform. The extracted constraints and rules are presented throughout chapter 3 and they are all accompanied by appropriate analytic information for development purposes. Detailed analyses of the constraints revealed the complex nature of the functional dependencies and helped us to indicate the scope of each one as well as the interrelations among them.

Finding the answer to the RQ leads a number of other sub-questions, such as investigating available XML schema languages and delivering appropriate validation artefacts using them. We generalised the secondary questions, research sub-question, as the following:

What is the optimal set of standardised XML validation methods to express the XLIFF 2.1 specification thoroughly?

We aim to address the sub-questions within chapter 4, where it is demonstrated how we designed and developed the validation artefacts based on the findings of chapter 3. We also explain the process of optimisation and final selection of schema languages in Chapter 4. Special attention was given to Schematron development and we showed the process of mapping the functional dependencies to appropriate rules within the schema language and presented the core of this thesis.

Finally, in chapter 5 we evaluated the performance of our validation platform by running specially designed valid and invalid files. This test suite was discussed within the chapter, where the presented results proved reliability and robustness of the delivered validation platform.

With respect to our research methodology, we identified *Design and Development* as the main methodology for this work (see chapter 1). The secondary research methodologies for different stages of the work are presented in chapters 3, 4 and 5 separately.

6.2 Conclusions

The result of this piece of research proved that the expressivity power of available declarative methods for XML validation is sufficient for the XLIFF 2 standard, a complex data model intended to facilitate a comprehensive exchange format for the localisation industry. We investigated our schema languages, especially Schematron, from the fundamental XML point of view and showed that expressivity of a schema language is in direct relation with its "navigational capabilities", i.e. how many levels of relativity the schema language can distinguish and how advanced can the search patterns be defined (i.e. XPath predicates and functions). Our conclusion of studying XML schema languages is that the available technologies offer a very wide range of possibilities for all of the know validation tasks and challenges. The DSDL platform enables an interoperable framework for the supported languages to perform, and also possibility of further development. However, these methods are not well recognised nor commonly used in the industry and, consequently, lack sample and educational resources as well as reliable implementations. This leads to the next conclusion, that while more data models are not examined against the DSDL methods, there would not be enough information and knowledge to evaluate their limitations. As the XLIFF 2.1 example illustrates, the effectively designed validation platform could significantly increase the standard conformance in the workflow.

The other important conclusion of our work is around XML. We argued that some of the mechanisms are not in consistency with the current requirements of modern industry-driven data models, such as the `xml:id` concept which is not suitable for XLIFF 2. This could be an essential disadvantage for XML, especially because it is becoming the predominant exchange format in information systems. Further analyses of common XML vocabularies (from different fields) could help to identify other possible issues and propose improvements for XML data model designing.

6.3 Contribution

This work produced a standardised validation infrastructure for XLIFF 2.x and a ready validation solution for XLIFF 2.1 in particular. This research also helped to identify other issues which could affect interoperability of XLIFF users, such as ambiguities of the specification and errors of the XLIFF 2.0 test-suit. These findings are to be modified and corrected in the next release of the standard, 2.1, which will also contain our artefacts as the recommended validation platform.

This research and its result essentially attempt to promote DSDL methods and demonstrate how their role and potentials are being underestimated. We gathered our detailed study of the schema languages and provided a ”Best Practice” for DSDL-based validation within the following publication:
Saadatfar, S., Filip, D. (2016) ‘Best Practice for DSDL-based Validation’, in XML London, Presented at the XML London, London, 64–70.

However, this subject is at early stages of research and our aim is to improve the state-of-the-art of the platform and courage other XML standards to choose declarative validation to overcome semantic interoperability issues.

6.4 Future Work

We identify two major directions for the future work in the context of our research. The first and the most important topic is further investigation and systematic research on XML integrity constraints. The final goal of such research could be considered as defining and expressing XML specific constraints by mathematical and generalised notations, which then could be hired for mapping of the constraints and appropriate DSDL-based validation artefacts. Improving the state-of-the-art of current DSDL best practice might pave the way towards automatic or semi-automatic schema creation.

The second direction of further work is related to the test files for evaluation of developed schema. Within chapter 5 we pointed out the importance of both valid and invalid test suite to examine the validation platform and also to promote the proper usage of the target XML vocabulary. An automated process could be developed to produce the appropriate files based on the analyses of the provided schemas. Such algorithm would be very useful for developers to ensure that all the possible errors, expressed within schemas, are actually being reported as such and no constraint has been left without attention.

Bibliography

Abrao, M., Bouchou, B., Alves, M., Laurent, D. & Musicante, M. (2004), 'Incremental constraint checking for XML documents', *XSym* .

Alavi, M. & Carlson, P. (1992), 'A review of MIS research and disciplinary development', *Journal of Management Information Systems* **8**(4), 45–62.

Almeida, F., Oliveira, J. & Cruz, J. (2010), 'Open Standards And Open Source: Enabling Interoperability', *International Journal of Software Engineering and Applications* **1**(2), 1–11.

Anastasiou, D. (2011), 'The Impact of Localisation on Semantic Web Standards', *European Journal of ePractice* **March/April**(12), 42–52.

Apparao, V., Byrne, S., Champion, M., Isaacs, S., Jacobs, I., Le Hors, A., Nicol, G., Robie, J., Sutor, R. & Wilson, C Wood, L. (1998a), '(Eds.) Document object model (DOM) level 1 specification', [online], ed, W3C Recommendation, available: http://www.w3.org/TR/REC-DOM-Level-1 [accessed 18 Aug 2016] .

Arenas, M., Fan, W. & Libkin, L. (2002), What's Hard about XML Schema Constraints?, *in* 'Database and Expert Systems Applications', Springer Berlin Heidelberg, pp. 269–278.

Arenas, M. & Libkin, L. (2004), 'A Normal Form for XML Documents', *ACM Transactions on Database Systems* **29**(1), 195–232.

Asuncion, C. H. & van Sinderen, M. (2010), Pragmatic Interoperability: A Systematic Review of Published Definitions, *in* 'Proceedings of the IFIP TC5 International Conference on Enterprise Architecture, Integration and Interoperability, EAI2N 2010, held in conjunction with the World Computer Congress 2010, 20-23 Sep', Springer, Brisbane, Australia, pp. 164–175.

Asuncion, C. H. & van Sinderen, M. (2011), Towards pragmatic interoperability in the new enterprise—a survey of approaches, *in* 'Enterprise Interoperability', Springer, pp. 132–145.

Benedikt, M., Brun, G., Gibson, J., Kuss, R. & Ng, A. (2002), 'Automated update management for XML integrity constraints', *PLANX* .

Berges, I., Bermudez, J., Goni, A. & Illarramendi, A. (2010), Semantic interoperability of clinical data, *in* 'Proceedings of the First International Workshop on Model-Driven Interoperability', ACM, Oslo, Norway.

Bly, M. (2010), XLIFF: Theory and Reality: Lessons Learned by Medtronic in 4 Years of Everyday XLIFF Use, Limerick, Ireland.

Boag, S., Chamberlin, D., Fernandez, M. F., Florescu, D., Robie, J. & Simeon, J. (2010), '(Eds.) XQuery 1.0: An XML Query Language', [online], ed, W3C Recommendation, available: https://www.w3.org/TR/xquery/[accessed 15 Sep 2016] .

Bray, T., Paoli, J. & Sperberb-McQueen, C. (1998b), '(Eds.) Extensible Markup Language (XML) 1.0', [online], ed, W3C Recommendation, available: https://www.w3.org/TR/1998/REC-xml-19980210 [accessed 25 Aug 2016] .

Buneman, P., Davidson, S., Fan, W., Hara, C. & Tan, W. (2000), 'Reasoning about Keys in XML'.

Buneman, P., Davidson, S., Fan, W., Hara, C. & W C, T. (2002), 'Keys for XML', *Computer networks* **39**(5), 473–487.

Chen, H., Liao, H. & Gao, Z. (2010), 'On Defining Functional Dependencies in XML Schema', *Communications in Computer and Information Science* **118**, 120–131.

Clarck, J. & DeRose, S. (1999), '(Eds.) XML Path Language (XPath) Version 1.0', [online], ed, W3C Recommendation, available: https://www.w3.org/TR/xpath/ [accessed 18 Sep 2016] .

Clarck, J. & Makoto, M. (2001), '(Eds.) RELAX NG Specification', [online], OASIS

Standard. ed, Standard, OASIS, available: http://relaxng.org/spec-20011203.html [accessed 05 Jun 2016] .

Coenen, F. (1999), 'TOPICS IN INFORMATION PROCESSING: DECLARATIVE LANGUAGES', [online], available: http://cgi.csc.liv.ac.uk/ frans/OldLectures/2CS24/declarative.html [accessed 28 Dec 2016] .

Comerford, T., Filip, D., Raya, R. M. & Savourel, Y. (2014), '(Eds.) XLIFF Version 2.0', [online], OASIS Standard. ed, Standard, OASIS, available: http://docs.oasis-open.org/xliff/v2.0/os/xliff- core.html [accessed 11 Sep 2016] .

Dodds, L. (2001), Schematron: Validating XML Using XSLT, *in* 'XSLT UK Conference', Keble College, Oxford, England.

Du, J., Roturier, J. & Way, A. (2010), TMX Markup: A challenge when adapting SMT to the Localisation Environment, *in* '14th Annual EAMT conference', Saint-Raphael, France, pp. 253–260.

Easterbook, S., Singer, J., Storey, M.-A. & Damian, D. (2008), Selecting Empirical Methods for Software Engineering Research, *in* 'Guide to Advanced Empirical Software Engineering', Springer London, pp. 285–311.

Ellis, T. J. & Levy, Y. (2008), 'Framework of Problem-Based Research: A Guide for

Novice Researchers on the Development of a Research-Worthy Problem', *International Journal of an Emerging Transdiscipline* **11**, 17–33.

Ellis, T. J. & Levy, Y. (2010), A Guide for Novice Researchers: Design and Development Research Methods, pp. 107–118.

Esselink, B. (2000), *A Practical Guide to Localization*, John Benjamins Publishing.

Esselink, B. (2003), 'The Evolution of Localization', *Multilingual* **57**, 21–29.

European Commission (2004), 'European Interoperability Framework for Pan-European e-Government Services', [online], available: ec.europa.eu/idabc/servlets/Docd552.pdf?id=19529 [accessed 20 Jun 2016] .

European Commission (2010), 'European Interoperability Framework (EIF) for European Public Services', [online], available: http://ec.europa.eu/isa/documents/isa-annex-ii-eif-en.pdf [accessed 23 Jun 2016] .

Fallside, D. C. (2001), '(Eds.) XML Schema Part 0: Primer', [online], ed, W3C Recommendation, available: https://www.w3.org/TR/2001/REC-xmlschema-0-20010502/ [accessed 18 Oct 2016] .

Fan, W. (2005), 'XML Constraints: Specificaion, Analysis and Aplications', *DEXA* .

Fan, W. & Simeon, J. (2000), Integrity constraints for XML, *in* 'Proc. ACM PODS Conference', pp. 23–34.

Fawcett, J., Quin, L. & Ayers, D. (2012), *Begining XML*, 5th edn, John Wiley & Sons, Inc., Indianapolis.

FEISGILTT (2016), *FEISGILTT 2016 XLIFF Public Info Sessions* [online], available: https://locworld.com/wp-content/uploads/2015/11/FEISGILTT-2016-XLIFF-Public-Info-sessions.pdf [accessed: 10 Sep 2016].

Felleisen, M. (1991), ‘On the expressive power of programming languages’, *Science of Computer Programming* **17**, 35–75.

Filip, D. (2016), ‘XLIFF 2.1 open for public review’, *Multilingual Insights* [online], available: https://multilingual.com/language-in-the-news/xliff-2-1-open-public-review/ [accessed: 20 Nov 2016] .

Filip, D., McCance, S., Lewis, D., Lieske, C., Lommel, A., Kosek, L., Sasaki, F. & Savourel, Y. (2013), ‘(Eds.) Internationalization Tag Set (ITS) Version 2.0’, [online], W3C Recommendation, W3C, available: http://www.w3.org/TR/its20/ [accessed 16 May 2016] .

Filip, D. & O’Conchuir, E. (2011), ‘An Argument for Business Process Management in Localisation’, *Localisation Focus* **10**(1), 4–17.

Filip, D. & Saadatfar, S. (2016), ITS Module in XLIFF 2.1: Less Extensions, More Interoperability, *in* ‘FIESGILTT [online], available: https://locworld.com/wp-content/uploads/2015/11/ITSM.pdf [accessed 20 Nov 2016]’, Dublin.

Filip, D. & Wasala, A. (2013), ‘Process and Agent Classification Based Interoperability in the Emerging XLIFF 2.0 Standard’, *Localisation Focus* **12**(1), 61–73.

Folmer, E. & Verhoosel, J. (2011), *State of the Art on Semantic IS Standardization, Interoperability & Quality*, Gildeprint Drukkerijen, Enschede.

Genesereth, M. R. & Ketchpel, S. P. (1994), ‘Software Agents’, *Centre for Integrated Facility Engineering* .

Goldman, R., McHugh, J. & Widom, J. (1999), From Semistructured Data to XML: Migrating the Lore Data Model and Query Language, *in* ‘ACM SIGMOD Workshop on The Web and Databases (WebDB 1999) , June 3-4’, Philadelphia, Pennsylvania.

Harold, E. R. (2004), *Effective XML: 50 Specific Ways to Improve Your XML*, Effective software development series, Addison-Wesley Professional.

Hayes, J., Wright, S. E., Filip, D., Melby, A. & Reineke, D. (2015), ‘Interoperability of XLIFF 2.0 Glossary Module and TBX-Basic’, *Localisation Focus* **14**(1), 23–39.

Heiler, S. (1995), ‘Semantic interoperability’, *ACM Comput. Surv* **2**(27), 271–273.

Hevner, A. R., March, S. T., Park, J. & Ram, S. (2004), ‘Design science research in information systems’, *Management Information Systems Quarterly* **28**(1), 75–105.

Hines, B., Rasmussen, J., Ryan, J., Kapadia, S. & Brennan, J. (2008), *IBM WebSphere DataPower SOA Appliance Handbook*, Pearson Education.

Hollingsworth, D. (1995), ‘Workflow Management Coalition, The Workflow Reference Model’, *Workflow Management Coalition* (1.1).

Hu, J. & Tao, L. (2006), ‘Visual Modeling of XML Constraints Based on a New Extensible Constraint Markup Language’, *Engineering Letters* [online], available:http://www.engineeringletters.com/issues-v13/issue-3/EL-13-3-4.pdf [accessed 11 Jul 2016] **13**(3).

IEEE (1991), ‘’IEEE Standard Computer Dictionary. A Compilation of IEEE Standard Computer Glossaries’, IEEE Std 610, 1’.

ISO (1986), ‘SO 8879:1986Information processing - Text and office systems - Standard Generalized Markup Language (SGML)’.

ISO (2001), ‘ISO/IEC JTC 1/SC34 Information Technology - Document Description and Processing Languages’.

ISO (2003), ‘ISO/IEC 19757 Information Technology - Document Schema Definition Language (DSDL) — Part 2: Regular-grammar-based validation — RELAX NG.’.

ISO (2004), ‘ISO/IEC 19757-3 Document Schema Definition Languages (DSDL) — Part 3: Rule-Based Validation — Schematron’.

ISO (2006), ‘ISO/IEC 19757-4:2006(E) Information technology — Document Schema Definition Languages (DSDL) — Part 4: Namespace-based Validation Dispatching Language (NVDL)’.

ISO (2008), ‘ISO 30042:2008 Systems to manage terminology, knowledge and content - TermBase eXchange (TBX)’, [online], Standard, ISO, available: http://www.iso.org/iso/catalogue-detail.htm?csnumber=45797 [accessed 21 Jun 2016] .

Johnson, R. B., ., Onwuegbuzie, A. J. & Turner, L. A. (2007), ‘Toward a Definition of Mixed Methods Research’, *Journal of Mixed Methods Research* **1**(2), 112–133.

Kaplan, B. & Maxwell, J. (1994), ‘Qualitative Research Methods for Evaluating Computer Information Systems’, *Evaluating Health Care Information Systems* pp. 45–68.

Kothari, C. R. (2004), *Research Methodology: Methods and Techniques*, New Age International (P) Ltd., New Delhi.

Kubicek, H. & Cimander, R. (2009), ‘Three Dimensions of Organizational Interoperability’, *Eurpean Journal of ePractice* [online], available: http://www.dlorg.eu/uploads/External-Publications/6.1.pdf [accessed 22 Jun 2016] **6**(1).

Kubicek, H., Cimander, R. & Jochen Scholl, H. (2011), Organizational Interoperability in E-Government, *in* ‘Enterprise Interoperability II’, Springer-Verlag, Berlin Heidelberg.

Lee, D. & Chu, W. W. (2000), ‘Comparative Analysis of Six Schema Languages’, *ACM SIGMOD Record* **29**(3).

Lee, M.-L., Ling, T. W. & Low, W. L. (2002), Designing Functional Dependencies for XML, *in* ‘EDBT ’02 Proceedings of the 8th International Conference on Extending Database Technology: Advances in Database Technology’, Springer-Verlag, London, pp. 124–141.

Lenker, M., Anastasiou, D. & Buckley, J. (2010), ‘Workflow Specification for Enterprise Localisation’, *Localisation Focus* **9**(1), 26–35.

Lewis, D., Brennan, R., Meehan, A. & O’Sullivan, D. (2015), ‘Using Semantic Mapping to Manage Heterogeneity in XLIFF Interoperability’, *Localisation Focus* **14**(1), 40–42.

Lewis, D., Curran, S., Doherty, G., Feeney, K., Karamanis, N., Luz, S. & McAuley, J. (2009a), ‘Supporting Flexibility and Awareness in Localisation Workflows’, *Localisation Focus* **8**(1), 29–38.

Lewis, D., Curran, S., Feeney, K., Etzioni, Z., Keeney, J., Way, A. & Schaler, R. (2009b), Web service integration for next generation localisation, *in* ‘Workshop on Software Engineering, Testing, and Quality Assurance for Natural Language Processing', Boulder, Colorado, pp. 47–55.

Lewis, D., Curran, S., Jones, D., Moran, J. & Feeney, K. (2010), An Open Service Framework for Next Generation Localisation, Malta.

Lewis, G. A., Morris, E., Simanta, S. & Wrage, L. (2008), Proceedings of the Seventh International Conference on Composition-Based Software Systems (ICCBSS 2008), 1343630: IEEE Computer Society, *in* 'Proceedings of the Seventh International Conference on Composition-Based Software Systems (ICCBSS 2008), 1343630: IEEE Computer Society', pp. 164–173.

Libkin, L. (2007), Normalization Theory for XML, *in* 'Database and XMLTechnologies', Springer Berlin Heidelberg, pp. 1–13.

LISA (2004), 'Translation Memory eXchange- TMX', [online], Standard, LISA, available: https://www.gala-global.org/tmx-14b [accessed 18 Jun 2016] .

Lommel, A. (2006), 'Localization standards, knowledge- and information-centric business models, and the commoditization of linguistic information', *Perspectives on localization* pp. 223–239.

Maler, E. & El Andaloussi, J. (1995), *Developing SGML DTDs*, Prentice Hall PTR.

Maruyama, H. (2002), *XML and Java: Developing Web Applications*, Addison-Wesley Professional.

Marx, M. & de Rijke, M. (2004), Semantic Characterizations of Navigational XPath, *in* 'The First Twente Data Management Workshop TDM'04 on XML databases and information retrieval by V. Mihajlovic & D. Hiemstra', Enschede: Centre for Telematics and Information Technology (CTIT), University of Twente, pp. 67–73.

Milkowski, M. & Lipski, J. (2011), Using SRX standard for sentence segmentation, *in* 'Human Language Technology. Challenges for Computer Science and Linguistics', number 6562 *in* 'Lecture Notes in Computer Science', Springer Berlin Heidelberg, pp. 172–182.

Moorkens, J. (2013), The Role of Metadata in Translation Memories, *in* 'Text, Extratext, Metatext and Paratext in Translation', Cambridge Scholars Publishing, pp. 79–90.

Murata, M., Lee, D., Mani, M. & Kawaguchi, K. (2005), 'Taxonomy of XML Schema Languages Using Formal Language Theory', *ACM Transactions on Internet Technology (ACM TOIT)* **5**(4), 660–704.

Myers, M. (1997), 'Qualitative Research in Information Systems', *MIS Quarterly* **2**(21).

Naudet, Y., Latour, T., Guedria, W. & Chen, D. (2010), 'Towards a systemic formalisation of interoperability', *Computers in Industry* **2**(61), 176–185.

Naudet, Y., Latour, T., Haussmann, K., Abels, S., Hahn, A. & Johannesonn, P. (2006), Describing Interoperability: the OoI Ontology, *in* 'Workshop on Enterprise Modelling and Ontologies for Interoperability (EMOI-INTEROP)', Dublin.

Neiva, F. W., David, J. M. N., Braga, R. & Campos, F. (2016), 'Towards pragmatic interoperability to support collaboration: A systematic review and mapping of the literature', *Information and Software Technology* **72**(2016), 137–150.

Neushul, J. D. & Ceruti, M. G. (2004), Using XML Schema as a Validation Mechanism to Provide Semantic Consistency For Dependable Information Exchange, Firenze, Italy.

Nishi, Y. (2012), Viewpoint-based Test Architecture Design, *in* 'IEEE Sixth International Conference on Software Security and Reliability Companion', pp. 194–197.

Orlikowski, W. & Baroudi, J. (1991), 'Studying Information Technology in Organizations: Research Approaches and Assumptions', *Information Systems Research* **2**, 1–28.

Ouksel, A. M. & Sheth, A. (1999), 'Semantic interoperability in global information systems', *SIGMOD Rec.* **1**(28), 5–12.

Park, J. & Ram, S. (2004), 'Information systems interoperability: What lies beneath?', *ACM Trans. Inf. Syst.* **4**(22), 595–632.

Pooley, D. & Raya, R. M. (2008), '(Eds.) Segmentation Rules eXchange- (SRX)', [online], ed, Standard, LISA, available:

http://web.archive.org/web/20080808140053/http://www.lisa.org/fileadmin/standards/srx20.pdf [accessed 18 Jun 2016] .

Pym, A. (2004), *The Moving Text Localization, translation, and distribution*, John Benjamins B.V., Amsterdam / Philadelphia.

Robie, J., Chamberlin, D., Dyck, M. & Snelson, J. (2014), '(Eds.) XML Path Language (XPath) Version 3.0', [online], ed, W3C Recommendation, available: https://www.w3.org/TR/xpath-30/ [accessed 24 Sep 2016] .

Ruokolainen, T., Naudet, Y. & Latour, T. (2007), An Ontology of Interoperability in Inter-Enterprise Communities, *in* 'Enterprise Interoperability II', Springer London, pp. 159–170.

Saadatfar, S. & Filip, D. (2016), Best Practice for DSDL-based Validation, *in* 'XML London', London, pp. 64–70.

Sartipi, K. & Yarmand, M. H. (2008), Standard-based data and service interoperability in eHealth systems, *in* 'ICSM 2008. IEEE International Conference on Software Maintenance 28 Sept-4 Oct', pp. 187–196.

Sasaki, F. (2011), Markup Languages and Internationalization, *in* 'Linguistic Modeling of Information and Markup Languages', 1 edn, number 40 *in* 'Text, Speech and Language Technology', Springer Netherlands, pp. 67–80.

Savourel, Y. (2007), 'Cat tools and standards: a brief summary', *Multilingual* (37).

Savourel, Y. (2013), *IDs - unit id unique per file* [online], available: https://lists.oasis-open.org/archives/xliff/201310/msg00055.html [accessed: 10 Oct 2016].

Savourel, Y. (2014), XLIFF 2.0 vs XLIFF 1.2, *in* 'FIESGILTT Dublin', Dublin.

Savourel, Y., Reid, J., Jewtushenko, T. & Raya, R. (2008), '(Eds.) XLIFF Version 1.2', [online], OASIS Standard. ed, Standard, OASIS, available: http://docs.oasis-open.org/xliff/v1.2/os/xliff- core.html [accessed 15 Jun 2016] .

Schaler, R. (2009), Localization, *in* 'Baker, M. and Saldanha, G., eds., Routledge Encyclopedia of Translation Studies', Abingdon: Routledge, pp. 157–161.

Schanbel, B., Hackos, J. T. & Raya, R. M. (2015), *A Practical Guide to XLIFF 2.0*, XML Press.

Seaman, C. B. (1999), 'Qualitative Methods in Empirical Studies of Software Engineering', *IEEE Transactionss On Software Engineering* **4**(25), 557–572.

Somers, H. (2003), *Computers and Translation: A translator's guide*, John Benjamins Publishing.

Tekli, J., Chbeir, R., Traina, A. & Traina, C. (2011), 'XML document-grammar comparison: related problems and applications', *Central European Journal of Computer Science, Springer* **14**(1), 117–136.

Thompson, H. S., Mendelsohn, N., Beech, D. & Muzmo, M. M. (2012), '(Eds.) W3C XML Schema Definition Language (XSD) 1.1', [online], ed, W3C Recommendation, available: https://www.w3.org/TR/xmlschema11-1/ [accessed 15 Oct 2016] .

Tingley, C. (2015), 'XLIFF 2.0 and the Evolution of a Standard', *Localisation Focus* **14**(1), 19–23.

Torres del Rey, J. & Morado-Vazquez, L. (2015), 'Xliff and the translator: Why does it matter?'.

Vaishnavi, V. & Kuechler, W. (2004), 'Design science research in information systems', last updated: November 15, 2015.[online], available: http://www.desrist.org/design-research-in-information-systems/ [accessed 05 Jul 2016] .

van der Vlist, E. (2002), *XML Schema*, O'Reilly Media.

van der Vlist, E. (2003), *RELAX NG, A Simpler Schema Language for XML*, O'Reilly Media.

van der Vlist, E. (2007), *Schematron*, O'Reilly Media.

Van Deursen, A., Klint, P. & Viser, J. (2000), 'Strong functional dependencies and their application to normal forms in XML', *ACM SIGPLAN Notices* **35**(6), 26–36.

Vergara-Niedermayr, C., Wang, F., Pan, T., Kurc, T. & Saltz, J. (2014), 'Semantically Interoperable XML Data', *Intenational Journal of Semantic Computing* .

Vincent, M., Liu, J. & Liu, C. (2004), 'Domain-Specific Languages; An Annotated Bibliography', *TODS* **29**, 445–462.

Wasala, A., O'Keeffe, I. & Schaler, R. (2011), 'LocConnect: Orchestrating Interoperability in a Service-oriented Localisation Architecture', *Localisation Focus* **10**(1), 29–43.

Yin, R. K. (1994), *Case Study Research, Design and Methods, 2nd ed.*, Sage Publications, Newbury Park.

Zhu, H. & Wu, H. (2011), Interoperability of Data Created using Extensible Data Standards.