Untersuchung der Auswirkungen der Verwendung von String.Latin auf die Performance der Validierung in XÖV-Datenübermittlungen

Abschlussbericht im Rahmen des Werkvertrags 2010/52/TUI

zwischen

OSCI-Leitstelle, Bremen

und

Daniel Gent

18.08.2010

Internes Zeichen der FHB: 01-0805-01 - 1/2009 - 29

Management Summary

Im Rahmen dieses Berichts wird untersucht, inwieweit die Befürchtung zutrifft, dass eine Umsetzung der Verpflichtung auf Lateinische Zeichen in XÖV-Datenübermittlungen durch Validierung gegen XML-Schemata, die den Datentyp String.Latin verwenden, zu Performanceproblemen führt.

Dazu wurde eine Performanceanalyse durchgeführt und für alle relevanten Softwareprodukte der durch die Verwendung von String.Latin entstehende Mehraufwand ermittelt.

Die Analyse dieser Ergebnisse ergab, dass dieser Mehraufwand gering ist. Performanceprobleme auf Grund einer technischen Umsetzung von String.Latin sind nicht zu befürchten.

Es wird daher die Empfehlung ausgesprochen, die technische Umsetzung der Verpflichtung auf die „Lateinischen Zeichen in UNICODE“ durchzuführen.

Da Zeichen auf unterschiedliche Art in Unicode dargestellt werden können, kommt dieser Bericht darüber hinaus zu dem Ergebnis, dass für einen reibungslosen Verlauf der XÖV-Datenübermittlungen die bestehenden Regelungen aus dem XÖV-Handbuch verschärft werden sollten. Zeichenketten sollten durch die erzeugenden Systeme NFC-normalisiert versendet werden, um eine einheitliche Repräsentation der übermittelten Zeichen zu gewährleisten.

Inhaltsverzeichnis

1 Einleitung

1.1 Ausgangssituation

Die OSCI-Leitstelle hat im Rahmen des Projektes „Deutschland Online Standardisierung“ die Menge der „Lateinischen Zeichen in UNICODE“ [1] identifiziert und auf ihrer Webseite veröffentlicht. Sie hat außerdem den Basisdatentyp String.Latin bereitgestellt. Dieser schränkt den Datentyp xs:string mit Hilfe eines Patterns auf die abgestimmte Menge der lateinischen Zeichen ein.

Im Rahmen einer Befragung von Herstellern einschlägiger IT-Verfahren waren Bedenken gegen eine technische Umsetzung geäußert worden. Insbesondere werden Performanceprobleme befürchtet, die sich dann ergeben könnten wenn in jeder XÖV-konformen Datenübermittlung eine Validierung gemäß des String.Latin-Patterns[1] durchgeführt wird.

1.2 Zielstellung

Im Rahmen dieses Berichtes wird untersucht, inwieweit die geäußerten Bedenken zutreffend sind, und darauf basierend eine Empfehlung ausgesprochen, ob eine technische Umsetzung der Verpflichtung auf die „Lateinischen Zeichen in UNICODE“ durchgeführt werden sollte.

1.3 Aufbau des Berichts

In Kapitel 2 wird zunächst die Testmethodik beschrieben, die verwendet wurde, um Daten zu erhalten, auf deren Basis eine Empfehlung ausgesprochen werden kann.

In Kapitel 3 werden die Ergebnisse dieser Tests dargestellt.

In Kapitel 4 wird eine Analyse der Ergebnisse durchgeführt und auf deren Basis eine Empfehlung ausgesprochen.

In Kapitel 5 wird eine Problematik behandelt, die im Zuge dieser Untersuchung auffiel. Es werden verschiedene Möglichkeiten beschrieben, wie diese Problematik zu lösen ist, und eine Empfehlung ausgesprochen, welche dieser Möglichkeiten zu wählen ist.

[1] das im Rahmen dieser Untersuchung verwendete String.Latin-Pattern ist in Anhang A) dargestellt

2 Konzeption der Untersuchung

2.1 Grundsätzliches

Ziel der Untersuchung ist es, Daten zu erhalten anhand derer zu beurteilen ist, inwieweit die Verwendung von String.Latin anstelle von xs:string Einfluss auf die Laufzeitperformance der Validierung hat.

Dazu mussten zunächst einmal die Softwareprodukte identifiziert werden, mit denen eine Validierung durchführbar ist. Eine Aufstellung dieser Produkte, die im weiteren Verlauf der Untersuchung als Testkandidaten dienten, findet sich in Abschnitt 2.2.

Weiterhin musste ein Testverfahren entwickelt werden, dass einerseits sicherstellt, dass ein Testkandidat technisch in der Lage ist, die Validierung von XML-Instanzen gegen XML-Schemata, die das String.Latin-Pattern verwenden, korrekt durchzuführen und andererseits den durch die Verwendungen von String.Latin anstelle von xs:string bei XÖV-Datenübermittlungen anfallenden Mehraufwand zu quantifizieren. Die Beschreibung dieses Testverfahrens findet sich in Abschnitt 2.3.

2.2 Bestandsaufnahme der marktüblichen Softwareprodukte

Um eine möglichst große Aussagekraft der Untersuchung zu gewährleisten, wurden alle marktüblichen Softwareprodukte aus dem Java- und dem .NET-Umfeld untersucht, die XML-Instanzen gegen XML-Schemata validieren können. Dies sind im Einzelnen:

- **Java**
    - Apache Xerces 2 Java 2.10.0 [2]
    - Saxon-EE 9.2 [3]
    - jaxb 2.2.1 [4]
    - Apache XMLBeans 2.5.0 [5]
    - Woodstox 4.0.8 [6]
    - dom4j 1.6.1 [7]
    - Sun Multi-Schema XML Validator (MSV) 20090415 [8]
    - Oracle XDK 11
    - AltovaXML 2010 [9]
- **.NET**
    - .NET Framework Class Library 3.5 (System.XML Namensraum) [10]
    - AltovaXML 2010 [9]

Da die folgenden Softwareprodukte zur Validierung externe Komponenten verwenden, die bereits separat getestet werden, wird auf ihre Untersuchung verzichtet.

- jaxb
  - verwendet für die Validierung die „Java API for XML Processing“ (jaxp), welche beispielsweise von Apache Xerces 2 Java oder Saxon implementiert wird
- Woodstox
  - verwendet für Validierung die Sun Multi-Schema XML Validator Bibliothek
- dom4j
  - verwendet für die Validierung Apache Xerces oder die Sun Multi-Schema XML Validator Bibliothek

Die für die Untersuchung verwendeten Testkandidaten sind somit:

- **Apache Xerces 2 Java 2.10.0** (im Folgenden: Xerces)
- **Saxon-EE 9.2** (im Folgenden: Saxon)
- **Apache XMLBeans 2.5.0** (im Folgenden: XMLBeans)
- **Sun Multi-Schema XML Validator 20090415** (im Folgenden: MSV)
- **Oracle XDK 11** (im Folgenden: XDK)
- **AltovaXML 2010 für Java** (im Folgenden: AltovaXML Java)
- **.NET Framework Class Library 3.5** (im Folgenden .NET FCL)
- **AltovaXML 2010 für .NET** (im Folgenden AltovaXML .NET)

2.3 Durchzuführende Tests

Die Testkandidaten werden jeweils zwei Tests unterzogen:

- Funktionalitätstest
- Laufzeittest

Durch den Funktionalitätstest soll überprüft werden, ob der jeweilige Testkandidat in der Lage ist, eine Validierung von XML-Instanzen gegen XML-Schemata, insbesondere gegen solche, die das String.Latin Pattern zur Definition von Datentypen verwenden, korrekt durchzuführen.

Die Laufzeittests sollen Informationen liefern, wie groß für den jeweiligen Kandidaten der zu erwartende Mehraufwand ist, der durch die Verwendung von String.Latin anstelle von xs:string anfällt.

2.3.1 Funktionalitätstests

Da in XÖV-Datenübermittlungen bereits die Validierung gegen XML-Schemata erfolgt, wird davon ausgegangen, dass eine korrekte Validierung gegen diese Schemata möglich ist. Aus diesem Grund wird nicht getestet, ob die Kandidaten das Validieren gegen beliebige XML-Schemata oder gegen die in XÖV-Datenübermittlungen verwendeten Schemata korrekt durchführt. Es wird lediglich getestet, ob prinzipiell eine Validierung stattfindet und ob bei der Validierung die Einschränkung auf die Lateinischen Zeichen berücksichtigt wird.

Folglich ist es zur Durchführung der Funktionalitätstests nicht notwendig, die in der XÖV-Datenübermittlung verwendeten Schemata zu benutzen. Stattdessen werden folgende XML-Schemata verwendet, die an dieser Stelle in verkürzter Form dargestellt sind. Die vollständigen XML-Schemata sind in Anhang B) zu finden.

1. XML-Schema **xs**

```
<xs:schema>
  <xs:element name=”data” type=”xs:string”/>
</xs:schema>
```

2. XML-Schema **sl**

```
<xs:schema>
  <xs:element name=”data”>
    <xs:simpleType>
      <xs:restriction base=”xs:normalizedString”>
        <xs:pattern value=”String.Latin-Pattern”/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
</xs:schema>
```

Beide XML-Schemata legen fest, dass eine valide XML-Instanz nur das Element `data` enthalten darf. Außerdem definieren die Schemata den Typ des Elementinhaltes. In Schema xs ist dieser als xs:string definiert, die Typdefinition in Schema sl entspricht der des String.Latin-Datentyps.

Diese Schemata werden zur Validierung folgender XML-Instanzen verwendet:

1. XML-Instanz **vv**

Diese Instanz enthält ein <data> Element, dessen Inhalt alle Zeichen der Menge der Lateinischen Zeichen sind. Sie ist damit valide gegen beide Schemata.

2. XML-Instanz **vi**

Diese Instanz enthält ein <data> Element, dessen Inhalt ein einzelnes Zeichen ist, welches nicht Element der Menge der Lateinischen Zeichen ist. Sie ist damit valide gegen das Schema xs und invalide gegen das Schema sl.

3. XML-Instanz **ii**

Die Instanz enthält ein <datax> Element, dessen Inhalt ein beliebiger Text ist. Sie ist damit invalide gegen beide Schemata.

Die vollständigen XML-Instanzen sind im Anhang C) zu finden.

Um zu gewährleisten, dass ein Testkandidat generell in der Lage ist, eine Validierung korrekt durchzuführen, muss zweierlei geprüft werden: Eine valide Instanz muss als valide erkannt werden, eine invalide Instanz als invalide.

Ersteres wird durch die Validierung von Instanz vv gegen beide Schemata getestet. In beiden Fällen muss die Validierung durch den Testkandidaten ergeben, dass die Instanz valide ist.

Letzteres wird durch die Validierung von Instanz ii gegen beide Schemata getestet. In beiden Fällen muss die Validierung durch den Testkandidaten ergeben, dass die Instanz invalide ist.

Um zu überprüfen, ob ein Testkandidat die Einschränkung auf die Menge der Lateinischen Zeichen berücksichtigt, muss getestet werden, ob ausschließlich Zeichen dieser Menge im Zuge der Validierung als zu dieser zugehörig angesehen werden.

Dazu muss zunächst einmal festgestellt werden, ob alle Zeichen der Menge der Lateinischen Zeichen durch das String.Latin Pattern erkannt werden. Dies geschieht durch die Validierung der Instanz vv gegen das Schema sl, welche die Instanz als valide klassifizieren muss.

Weiterhin muss für jedes Zeichen, das Element der durch xs:string definierten Menge, aber nicht Element der Menge der Lateinischen Zeichen ist, getestet werden, ob es nicht mittels Restriktion auf String.Latin als unzulässig erachtet wird. Da die Anzahl der zu testenden Zeichen über 1.000.000 beträgt, und jedes dieser Zeichen im Kontext einer einzelnen XML-Instanz getestet werden müsste, wurde der Test darauf reduziert, dass nur eines dieser Zeichen für einen Test verwendet wird. Mit diesem Test wird damit nur nachgewiesen, dass nicht jedes beliebige Zeichen als gültig bezüglich des String.Latin-Patterns bewertet wird. Bei diesem Test wird Instanz vi gegen Schema sl validiert, und erwartet, dass diese als invalide klassifiziert wird. Um nachzuweisen, dass das verwendete Zeichen tatsächlich Teil der durch xs:string definierten Menge ist, wird die Instanz auch gegen das Schema xs validiert und erwartet, dass sie als valide klassifiziert wird.

Zusammenfassend noch einmal die bei den Funktionalitätstests durchgeführten Validierungen und die zu erwartenden Ergebnisse:

	Instanz vv	**Instanz vi**	**Instanz ii**
Schema xs	valide	valide	invalide
Schema sl	valide	invalide	invalide

2.3.2 Laufzeittests

Der durch die Verwendung von String.Latin anstelle von xs:string anfallende Mehraufwand wird durch Laufzeittests quantifiziert. Dazu wird eine XML-Instanz gegen ein XML-Schema, das eine Restriktion auf den Typ xs:string enthält, validiert. Anschließen wird dieselbe Instanz gegen ein XML-Schema validiert, das abgesehen von der Ersetzung von xs:string durch String.Latin identisch ist. Der Unterschied der ermittelten Laufzeiten dient dann als Grundlage für die Einschätzung des Mehraufwands.

Um aus den Ergebnissen der Laufzeittests Rückschlüsse auf die tatsächlichen Auswirkungen der Verwendung von String.Latin im XÖV-Datenverkehr ziehen zu können, muss der Laufzeittest dem real stattfindenden Validierungsprozess weitestgehend nachempfunden werden.

Dazu muss zunächst einmal die Beschaffenheit der verwendeten XML-Instanzen beachtet werden. Würden beispielsweise Instanzen verwendet werden, die nur zu einem sehr geringen Teil aus Daten bestehen, die der Restriktion auf xs:string respektive String.Latin unterliegen, würden daraus völlig andere Ergebnisse resultieren als bei der Verwendung von Instanzen, die nahezu ausschließlich aus solchen Daten bestehen.

Aus diesem Grund werden für die Laufzeittests stellvertretend für alle XÖV-Standards die Testnachrichten von XMeld verwendet (XMeld-Testsuite in der Fassung vom 22.03.2010). Es handelt sich dabei um 407 XML-Instanzen mit einer Gesamtgröße von 4,23 MB. Diese Instanzen sind valide gegen die XMeld-Schemata. Um einen Laufzeitvergleich anstellen zu können, wurde eine modifizierte Version dieser Schemata erstellt, bei der jede Restriktion auf xs:string durch eine Restriktion auf String.Latin ersetzt wurde. Dazu wurde das auf [11] erhältliche XML-Schema[2], welches die Definition für den String.Latin Typ enthält, in die Hierarchie der XMeld-Schemata integriert

Ferner muss definiert werden welcher Vorgang zeitlich erfasst werden soll. Eine Zeitmessung von Programmstart bis Programmende, würde bezüglich des relativen Laufzeitunterschieds zu gänzlich anderen Ergebnissen führen als eine Zeitmessung, die nur die Laufzeit der Funktion, die letztendlich die Validierung durchführt, erfasst. Der relative Laufzeitunterschied ist aber für die Abschätzung des zu erwartenden Mehraufwands ein wichtiges Maß, da er zu einem gewissen Grad eine Abstraktion von der verwendeten Hardware zulässt.

[2] Das in diesem XML-Schema verwendete String.Latin-Pattern ist fehlerhaft, da die Zeichen 50, 52, 84 und 86 der Lateinischen Zeichen nicht erfasst werden. Aus diesem Grund wurde es durch das String.Latin-Pattern aus Anhang A) ersetzt.

Da nicht im Einzelnen bekannt ist, wie eine Validierung in der Realität durchgeführt wird, werden Annahmen gemacht, die auf der Prämisse basieren, dass eine möglichst hohe Effizienz angestrebt wird. Basierend auf diesen Annahmen werden die Laufzeittests gestaltet.

Annahme 1) Für die Validierung einer XML-Instanz wird kein neuer Prozess gestartet, sondern es läuft bereits ein Prozess, der beispielsweise via Interprozesskommunikation verwendet wird.

Basierend auf dieser Annahme beinhaltet der bei den Laufzeittests gemessene Vorgang nicht das Starten oder Beenden des Validierungsprozesses. Weiterhin erlaubt diese Annahme, dass bei einem Laufzeittest nicht eine einzelne XMeld-Testnachricht, sondern sämtliche Testnachrichten der Testsuite validiert werden.

Annahme 2) XML-Instanzen werden nur gegen vorab bekannte XML-Schemata validiert (hier XMeld-Nachrichten gegen XMeld-Schemata).

Werden XML-Instanzen nur gegen vorab bekannte XML-Schemata validiert, kann unter der Voraussetzung, dass Annahme 1) zutrifft, eine Vorverarbeitung dieser XML-Schemata durchgeführt werden, das heißt diese können bereits in den Speicher geladen und gegebenenfalls kompiliert werden.

Bei den Laufzeittests werden zwei Szenarien getestet: Im Szenario „nothing prebuilt“ wird Annahme 2) nicht berücksichtigt. Das bedeutet, dass der gemessene Vorgang neben dem eigentlichen Validieren auch das Laden und Verarbeiten der Instanzen und Schemata beinhaltet. Im Szenario „schema prebuilt“ wird Annahme 2) berücksichtig. Es werden vor Beginn der Laufzeitmessung die Schemata, soweit es für den jeweiligen Testkandidaten möglich ist, vorverarbeitet.

Um bei den Laufzeittests verlässliche Werte zu erhalten, muss sichergestellt werden, dass die Zeitmessungen nicht durch äußere Einflüsse verfälscht werden. Zu diesen Einflüssen zählen beispielsweise Aktivität von Hintergrundprozessen oder Optimierungsvorgänge, die zur Laufzeit von Java oder .NET Programmen auftreten können. Dazu wird vor Beginn der eigentlichen Tests eine Anzahl von Vorläufen durchgeführt, die sicherstellen sollen, dass die Laufzeitumgebungen von Java und .NET alle Optimierungen vorgenommen haben. Außerdem wird jeder Testlauf mehrmals wiederholt und die gemessene Laufzeit im Anschluss gemittelt.

3 Testergebnisse

3.1 Testumgebung

Alle Tests wurden jeweils auf zwei unterschiedlichen Systemen durchgeführt:

- **Testsystem A**
  - Prozessor: AMD Athlon X2 QL-62 2GHz
  - Hauptspeicher: 2GB
  - Betriebssystem: Windows 7 (32 Bit)

- **Testsystem B**
  - Prozessor: Intel Core2Duo E6700 2,66GHz
  - Hauptspeicher: 2GB
  - Betriebssystem: Windows XP

3.2 Testergebnisse für Testsystem A

Testkandidat	Funktionalitätstest	
	bestanden	nicht bestanden
Xerces	X	
Saxon	X	
XMLBeans	X	
MSV	X	
XDK	X	
AltovaXML Java	X	
.NET FCL	X	
AltovaXML .NET	X	

Ergebnisse der Funktionalitätstests für Testsystem A

Testkandidat	Laufzeit xs:string (s)	Laufzeit String.Latin (s)	Differenz absolut (s)	Differenz relativ
Xerces	1,218	1,635	0,354	27,65%
Saxon	1,226	1,449	0,224	18,27%
XMLBeans	0,940	1,156	0,216	22,98%
MSV	0,557	0,813	0,256	46,01%
XDK	1,044	1,600	0,556	53,26%
AltovaXML Java	133,142	136,462	3,320	2,49%
.NET FCL	0,730	1,306	0,576	78,90%
AltovaXML .NET	257,136	260,534	3,398	1,32%

Laufzeiten für Szenario "schema prebuilt" für Testsystem A

Testkandidat	Laufzeit xs:string (s)	Laufzeit String.Latin (s)	Differenz absolut (s)	Differenz relativ
Xerces	16,256	16,855	0,598	3,68%
Saxon	17,196	17,541	0,372	2,17%
XMLBeans	51,493	52,498	1,005	1,95%
MSV	12,605	13,224	0,620	4,92%
XDK	16,284	17,087	0,804	4,94%
AltovaXML Java	138,194	138,590	0,396	0,29%
.NET FCL	26,657	27,722	1,065	4,00%
AltovaXML .NET	268,554	270,032	1,478	0,55%

Laufzeiten für Szenario "nothing prebuilt" für Testsystem A

3.3 Testergebnisse für Testsystem B

Testkandidat	Funktionalitätstest	
	bestanden	nicht bestanden
Xerces	X	
Saxon	X	
XMLBeans	X	
MSV	X	
XDK	X	
AltovaXML Java	X	
.NET FCL	X	
AltovaXML .NET	X	

Ergebnisse der Funktionalitätstests für Testsystem B

Testkandidat	Laufzeit xs:string (s)	Laufzeit String.Latin (s)	Differenz absolut (s)	Differenz relativ
Xerces	0,530	0,746	0,216	40,47%
Saxon	0,522	0,639	0,117	22,46%
XMLBeans	0,448	0,573	0,125	27,90%
MSV	0,225	0,362	0,137	60,61%
XDK	0,452	0,527	0,273	60,31%
AltovaXML Java	77,342	78,537	1,195	1,54%
.NET FCL	0,309	0,597	0,288	93,20%
AltovaXML .NET	151,230	152,602	1,372	0,91%

Laufzeiten für Szenario "schema prebuilt" für Testsystem B

Testkandidat	Laufzeit xs:string (s)	Laufzeit String.Latin (s)	Differenz absolut (s)	Differenz relativ
Xerces	8,280	8,599	0,319	3,85%
Saxon	8,326	8,500	0,174	2,09%
XMLBeans	29,081	29,541	0,460	1,58%
MSV	6,589	6,736	0,147	2,24%
XDK	7,998	8,342	0,343	4,29%
AltovaXML Java	78,112	79,049	0,973	1,20%
.NET FCL	12,868	13,284	0,416	3,23%
AltovaXML .NET	151,270	152,578	1,308	0,86%

Laufzeiten für Szenario "nothing prebuilt" für Testsystem B

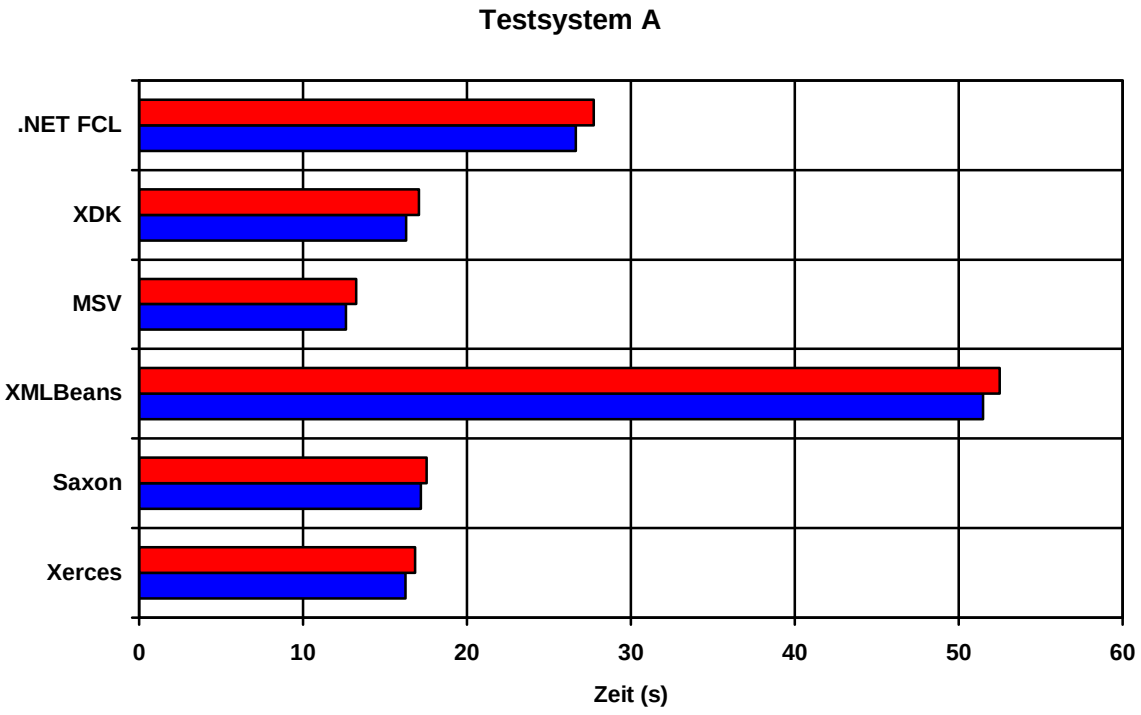
3.4 Bemerkungen zu den Testergebnissen der AltovaXML-Testkandidaten

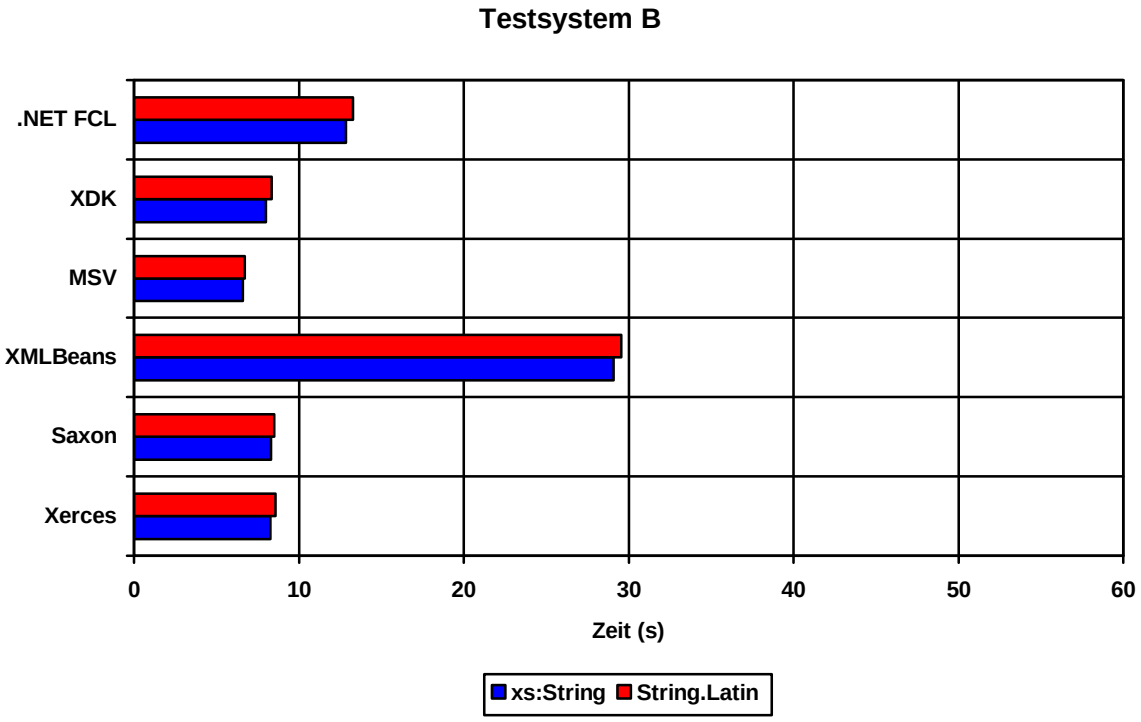
Bei den Testresultaten fällt auf, dass die Laufzeiten der beiden AltovaXML-Kandidaten sehr viel länger ausfallen, als die der übrigen Testkandidaten und kaum ein Unterschied zwischen den beiden getesteten Szenarios besteht. Dies ist darin begründet, dass es sich sowohl bei der Java- als auch der .NET-Variante, nicht um eigenständige Bibliotheken handelt, sondern lediglich um ein Interface, dass Zugriff auf einen externen Prozess gewährt. Dieses Interface ist extrem simpel aufgebaut und erlaubt keinerlei Vorverarbeitung von XML-Instanzen oder XML-Schemata, was dazu führt, dass sich die Laufzeiten für die beiden Szenarios sehr ähneln.

Da die AltovaXML-Kandidaten dadurch für eine Validierung von XML-Instanzen in einem performancekritischen Umfeld nicht geeignet sind, und der Einfluss von String.Latin durch die extrem hohen Laufzeit ohnehin nur minimal ist, werden die Ergebnisse der Laufzeittests für AltovaXML Java und AltovaXML .NET nicht weiter betrachtet.

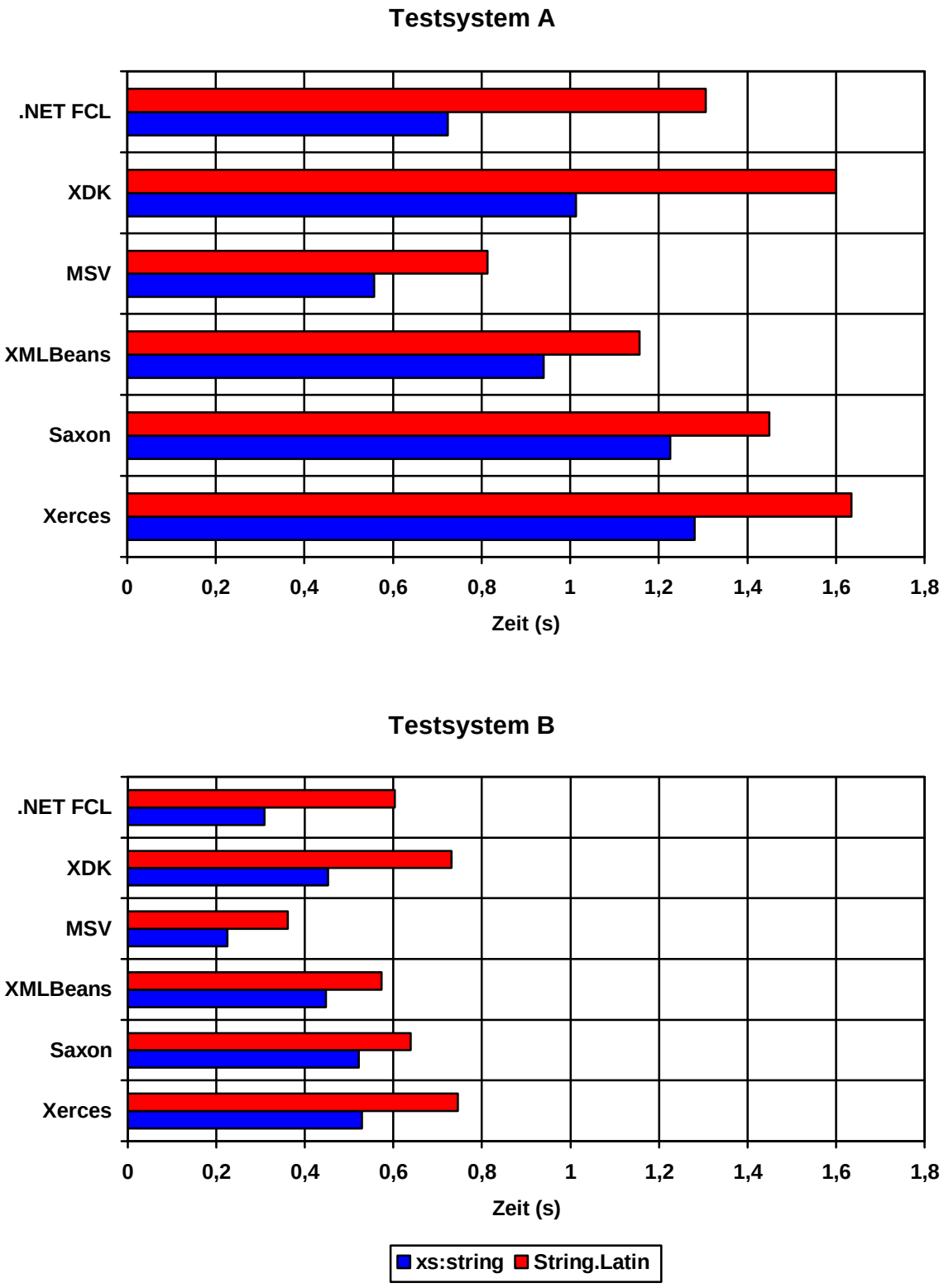
3.5 Gegenüberstellung der getesteten Szenarios

3.5.1 Szenario „nothing prebuilt“





3.5.2 Szenario „schema prebuilt“



4 Bewertung der Ergebnisse

4.1 Fazit

Die Ergebnisse der Funktionalitätstest zeigen, dass alle der getesteten Softwareprodukte eine Validierung gegen XML-Schemata, und insbesondere gegen XML-Schemata, die das String.Latin-Pattern enthalten, korrekt durchführen können.

Die Laufzeittests zeigen zunächst einmal, dass die Verwendung von String.Latin anstelle von xs:string definitiv eine Verlängerung der Laufzeit verursacht. Dies lässt sich bei beiden Testsystemen, beiden Szenarien und allen Testkandidaten beobachten. Es muss also betrachtet werden, ob es sich hierbei um einen signifikanten Anstieg handelt.

Bei den Laufzeitmessungen wurde sowohl der absolute, als auch der relative Laufzeitunterschied ermittelt. Wie man durch Vergleich der Ergebnisse für beide Testsysteme erkennen kann, wachsen die absoluten Unterschiede von Testsystem A zu Testsystem B in etwa um den Faktor 2 an, während die relativen Unterschiede auf vergleichbarem Niveau bleiben, insbesondere das Verhältnis der relativen Unterschiede unterhalb der Testkandidaten. Es wird daher angenommen, dass sich die Werte der relativen Unterschiede auch auf die zur Validierung von XÖV-Nachrichten verwendeten Systeme übertragen lassen. Die gemessenen relativen Unterschiede stellen also die aussagekräftigeren Werte dar.

Geht man davon aus, dass bei der Validierung von XÖV-Nachrichten die für das Szenario „nothing prebuilt“ getroffenen Annahmen zutreffend sind, ergibt sich ein relativer Mehraufwand von weniger als 5%. Dies kann eindeutig nicht als signifikanter Anstieg der Laufzeit bewertet werden.

Geht man hingegen davon aus, dass die von Szenario „schema prebuilt“ getroffenen Annahmen zutreffend sind, ergibt sich im schlechtesten Fall ein relativer Mehraufwand von fast 100%. Hierbei kann erst einmal nicht angenommen werden, dass es sich um einen vertretbaren Mehraufwand handelt. Um dieses zu überprüfen, müssen auch die absoluten Laufzeitunterschiede betrachtet werden.

Die größte absolute Differenz im Szenario „schema prebuilt“ tritt bei der Validierung mit .NET FCL auf Testsystem A auf und beträgt in etwa eine Sekunde. Da bei den Testläufen 407 XMeld-Nachrichten validiert werden, bedeutet das einen durchschnittlicher Mehrzeitverbrauch von ungefähr 0,0025 Sekunden pro XMeld-Nachricht. Zieht man in Betracht, dass die DSRV als einer der größten zentralen Datenempfänger von XMeld-Nachrichten im Januar 2010 ca. 1.400.000 XMeld-Nachrichten, also rund 70.000 Nachrichten pro Arbeitstag erhalten hat, ergibt dies einen täglichen Mehraufwand an Laufzeit von 175 Sekunden, also in etwa drei Minuten.

Dieser Mehraufwand wird nicht als signifikant angesehen, insbesondere deshalb nicht, da es sich bei Testsystem A um ein System der unteren Leistungskategorie handelt und zu erwarten ist, dass die im XÖV-Datenverkehr verwendeten Systeme um vieles leistungsfähiger sind und der absolute Mehraufwand damit noch geringer ausfällt.

4.2 Empfehlung

Die durchgeführte Untersuchung hat Folgendes ergeben:

- alle relevanten Softwareprodukte für Validierung von XML-Instanzen gegen XML-Schemata sind in der Lage, korrekt gegen XML-Schemata zu validieren, die das String.Latin-Pattern verwenden
- durch die Verwendung von String.Latin anstelle von xs:String in den Schemata im XÖV-Kontext entsteht bei der Validierung kein signifikanter Mehraufwand

Unabhängig von der Höhe des festgestellten Mehraufwandes ist zu bemerken, dass eine Einschränkung auf die Lateinischen Zeichen in irgendeiner Form ohnehin zu realisieren ist. Ohne die Verwendung des Datentyp String.Latin in XML-Schemata müsste eine Überprüfung von Daten auf Konformität bezüglich der Lateinischen Zeichen, innerhalb der Fachverfahren erfolgen. Dies würde bedeuten, dass der Mehraufwand lediglich an anderer Stelle anfällt.

Es wird daher empfohlen, die technische Umsetzung der Verpflichtung auf die „Lateinischen Zeichen in UNICODE“ durchzuführen.

5 Nebenprodukt der Untersuchung: Composed/Decomposed Problematik

Für viele Zeichen aus der Menge der Lateinischen Zeichen gibt es auf Codepoint-Ebene keine eindeutige Entsprechung. Als Beispiel wird das Zeichen Nr. 341 (ǟ) betrachtet.

Dieses Zeichen ist in Unicode durch den Codepoint 01DF repräsentiert. Es lässt sich aber auch in die Bestandteile 0061 (a), 0308 (¨) und 0304 (¯) zerlegen, die als Sequenz das gleiche Zeichen erzeugen. Eine weitere mögliche Aufteilung ist 00E4 (ä) + 0304 (¯)

Es stellte sich nun die Frage, ob das String.Latin Pattern ein ǟ in allen Varianten als zulässig erachtet. Ein Test ergab, dass dem nicht so ist. Enthält das Pattern ein vorgefertigtes Zeichen, erkennt es auch nur dieses, enthält es ein in seine Bestandteile aufgeteiltes Zeichen wird auch nur in dieses erkannt. Beim matching findet also offensichtlich nur ein Abgleich der Codepoints statt. Dieses Verhalten ist problematisch, da ein abstraktes Zeichen unabhängig von seiner Repräsentation auf Codepoint-Ebene entweder zur Menge der Lateinischen Zeichen gehören sollte, oder nicht.

In Unicode wird die Relation zwischen Zeichen mit dem gleichen Aussehen und Verhalten als „kanonische Äquivalenz“ (canonical equivalence) bezeichnet [12].

Darüber hinaus beschreibt [13] Eigenschaften und Funktionen, welche Softwarekomponenten für die Behandlung von regulären Ausdrücken erfüllen sollten. Das Erkennen von kanonischer Äquivalenz zählt dabei schon zum zweiten Level „Extended Unicode Support“ und wird anscheinend nicht weitreichend unterstützt – eine kurze Recherche ergab, dass lediglich die Pattern Klasse aus Java [14] einen Modus für die Berücksichtigung von kanonischer Äquivalenz hat.

Im gleichen Dokument werden Herangehensweisen zum Erkennen kanonisch äquivalenter Zeichen genannt: Eine Möglichkeit ist das Erweitern des Patterns, so dass für jedes erlaubte Zeichen alle möglichen Codepoints und Codepoint-Sequenzen im Pattern enthalten sind. Eine andere ist die Konvertierung von Pattern und zu prüfender Zeichensequenz in ein einheitliches Format vor Beginn der Validierung. Unicode definiert in [12] vier verschiedene Normalformen. Die am weitesten verbreitete ist NFC. In dieser Form werden, wenn möglich, vorgefertigte Zeichen erzeugt und bei Zeichen die nur durch Codepoint-Sequenzen erzeugt werden können ist die Reihenfolge der Codepoints definiert.

Für das Erkennen von Lateinischen Zeichen ist letztere Methode vorzuziehen. Sowohl Java [15] als auch .NET [16] beinhalten Komponenten, welche eine Normalisierung nach NFC durchführen können. Das Erweitern des String.Latin Patterns hingegen würde zum einen bedeuten, dass zunächst einmal für jedes Zeichen in Erfahrung gebracht werden müsste durch welche Codepoint-Sequenzen es erzeugt werden kann, zum anderen würde der Validierungsaufwand durch ein komplexeres Pattern zusätzlich ansteigen.

Da eine XÖV-Nachricht nur einmal erzeugt, aber möglicherweise viele Male validiert wird, sollte eine Normalisierung beim Erzeuger der Nachricht durchgeführt werden.

Es wird daher empfohlen, die Anforderungen an Nachrichten innerhalb des XÖV-Datenverkehrs zu erweitern und zu fordern, dass diese bei der Erzeugung der Daten gemäß NFC normalisiert werden.

Anhang A) Korrigiertes[3] String.Latin-Pattern

```
(([&#x9;-&#xa;
   &#xd;
   &#x20;-&#x7e;
   &#xa1;-&#xac;
   &#xae;-&#x131;
   &#x134;-&#x17f;
   &#x187;-&#x188;
   &#x18f;
   &#x1a0;-&#x1a1;
   &#x1af;-&#x1b0;
   &#x1b7;
   &#x1bf;
   &#x1cd;-&#x1d4;
   &#x1de;-&#x1df;
   &#x1e2;-&#x1ef;
   &#x1f4;-&#x1f5;
   &#x1f7;
   &#x1fa;-&#x21f;
   &#x22a;-&#x233;
   &#x259;
   &#x292;
   &#x1e02;-&#x1e03;
   &#x1e0a;-&#x1e0b;
   &#x1e10;-&#x1e11;
   &#x1e1e;-&#x1e21;
   &#x1e24;-&#x1e27;
   &#x1e30;-&#x1e31;
   &#x1e40;-&#x1e41;
   &#x1e44;-&#x1e45;
   &#x1e56;-&#x1e57;
   &#x1e60;-&#x1e63;
   &#x1e6a;-&#x1e6b;
   &#x1e80;-&#x1e85;
   &#x1e8c;-&#x1e93;
   &#x1e9b;
   &#x1e9e;
   &#x1ea0;-&#x1ea1;
   &#x1eaa;-&#x1eac;
   &#x1ebd;
   &#x1ec4;-&#x1ec5;
   &#x1eca;-&#x1ecf;
   &#x1ed6;-&#x1ed7;
   &#x1ee4;-&#x1ee5;
   &#x1ef2;-&#x1ef3;
   &#x1ef8;-&#x1ef9;
   &#x20ac;])|([&#x4d;&#x4e;&#x6d;&#x6e;]&#x302;))*
```

[3] Korrektur bezüglich des in [11] verwendeten Patterns, welches die in [1] aufgeführten Zeichen 50, 52, 84 und 86 nicht berücksichtigt und insofern die in der Tabelle A-1 des XÖV-Handbuches abschließend aufgezählten lateinischen Zeichen in UNICODE nicht vollständig umsetzt.

Anhang B) XML-Schemata für die Funktionalitätstests

XML-Schema xs

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://test"
           targetNamespace="http://test"
           elementFormDefault="qualified">
  <xs:element name="data" type="xs:string"/>
</xs:schema>
```

XML-Schema sl

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://test"
           targetNamespace="http://test"
           elementFormDefault="qualified">
  <xs:element name="data">
    <xs:simpleType>
      <xs:restriction base="xs:normalizedString">
        <xs:pattern value="(([&#x9;-&#xa;&#xd;&#x20;-
&#x7e;&#xa1;-&#xac;&#xae;-&#x131;&#x134;-&#x17f;&#x187;-
&#x188;&#x18f;&#x1a0;-&#x1a1;&#x1af;-
&#x1b0;&#x1b7;&#x1bf;&#x1cd;-&#x1d4;&#x1de;-
&#x1df;&#x1e2;-&#x1ef;&#x1f4;-&#x1f5;&#x1f7;&#x1fa;-
&#x21f;&#x22a;-&#x233;&#x259;&#x292;&#x1e02;-
&#x1e03;&#x1e0a;-&#x1e0b;&#x1e10;-&#x1e11;&#x1e1e;-
&#x1e21;&#x1e24;-&#x1e27;&#x1e30;-&#x1e31;&#x1e40;-
&#x1e41;&#x1e44;-&#x1e45;&#x1e56;-&#x1e57;&#x1e60;-
&#x1e63;&#x1e6a;-&#x1e6b;&#x1e80;-&#x1e85;&#x1e8c;-
&#x1e93;&#x1e9b;&#x1e9e;&#x1ea0;-&#x1ea1;&#x1eaa;-
&#x1eac;&#x1ebd;&#x1ec4;-&#x1ec5;&#x1eca;-
&#x1ecf;&#x1ed6;-&#x1ed7;&#x1ee4;-&#x1ee5;&#x1ef2;-
&#x1ef3;&#x1ef8;-
&#x1ef9;&#x20ac;])|([&#x4d;&#x4e;&#x6d;&#x6e;]&#x302;))*"/
>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
</xs:schema>
```

Anhang C) XML-Instanzen für die Funtionalitätstests

XML-Instanz vv

```
<?xml version="1.0" encoding="utf-8"?>
<data xmlns="http://test"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
      xsi:schemaLocation="http://test test.xsd">
```

!"#$%&'()*+,-

./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abc

defghijklmnopqrstuvwxyz{|}~¡¢£¤¥¦§¨©ª«¬®¯°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇ

ÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþÿĀāĂă

ĄąĆćĈĉĊċČčĎďĐđĒēĔĕĖėĘęĚěĜĝĞğĠġĢģĤĥĦħĨĩĪīĬĭĮįİıĲĳĴĵĶķĸĹĺĻļĽľĿŀŁłŃńŅņŇ

ňŉŊŋŌōŎŏŐőŒœŔŕŖŗŘřŚśŜŝŞşŠšŢţŤťŦŧŨũŪūŬŭŮůŰűŲųŴŵŶŷŸŹźŻżŽžſƆƈ

ƏƠơƯư3pǍǎǏǐǑǒǓǔǞǟǢǣǤǥǦǧǨǩǪǫǬǭǮǯǰǴǵ□ǺǻǼǽǾǿȀȁȂȃȄȅȆȇȈȉȊȋȌȍ

ŘřŘřÙùÛû□□□□□□□□□□□□□□□□□□□□əʒḂḃḊḋḌḍḞḟḠḡḢḣḦḧḰḱṀṁṄṅṖṗṠṡṢṣṪṫẀ

ẁẂẃẄẅẊẋẎẏŽžẓſ□ẠạẢảẤấẦầẨẩẪẫẬậẮắẰằẲẳẴẵẶặẸẹẺẻẼẽẾếỀềỂểỄễỆệỈỉỊịỌọỎỏỐốỒồỔổỖỗỘộỚớỜờỞởỠỡỢợỤụỦủỨứỪừỬửỮữỰựỲỳỴỵỶỷỸỹ€</data>

XML-Instanz vi

```
<?xml version="1.0" encoding="utf-8"?>
<data xmlns="http://test">□</data>
```

XML-Instanz ii

```
<?xml version="1.0" encoding="utf-8"?>
<datax xmlns="http://test"></datax>
```

Quellen

[1] http://www.osci.de/materialien/latinchars.xml
[2] http://xerces.apache.org/xerces2-j/
[3] http://www.saxonica.com/
[4] https://jaxb.dev.java.net/
[5] http://xmlbeans.apache.org/
[6] http://woodstox.codehaus.org/
[7] http://sourceforge.net/projects/dom4j/
[8] https://msv.dev.java.net/
[9] http://www.altova.com/altovaxml.html
[10] http://msdn.microsoft.com/en-us/library/system.xml%28v=VS.80%29.aspx
[11] http://www.osci.de/xoev/basisdatentypen/xoev-basisdatentypen.xsd
[12] http://www.unicode.org/reports/tr15
[13] http://www.unicode.org/reports/tr18/#Canonical_Equivalents
[14] http://download.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html
[15] http://download.oracle.com/javase/6/docs/api/java/text/Normalizer.html
[16] http://msdn.microsoft.com/en-us/library/system.string.normalize.aspx