Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Entwicklung eines XML-basierten Konfigurationsframeworks einer verteilten Multi-Physik Kopplungssoftware

Georg Abrams

Studiengang: Softwaretechnik

Prüfer/in: Prof. Dr. Miriam Mehl

Betreuer/in: Dipl.-Ing. Florian Lindner

Beginn am: 06. Juni 2017

Beendet am: 06. Dezember 2017

CR-Nummer: D.2.9, E.1

Abstract

preCICE ist eine Multi-Physik Kopplungssoftware und dient der Kopplung von Single-Physik Lösern. XML ist eine Markup Language, die der Beschreibung von hierarchisch aufgebauten Dokumenten dient. Der Zusammenhang zwischen diesen beiden Technologien ist, dass XML Dokumente bei preCICE zur Laufzeit eingebunden und zur Konfiguration der Software verwendet werden. Da der aktuell eingebaute XML Parser jedoch veraltet ist und einige Fehler mit sich bringt, soll dieser ersetzt werden. Was allgemein XML ist, welche XML Parser überhaupt existieren und wie ein neuer XML Parser in preCICE eingebaut wird, wird in diesem Dokument näher beschrieben.

Inhaltsverzeichnis

Abstract 0

Abkürzungsverzeichnis 4

Abbildungsverzeichnis 5

1 Einleitung 6

1.1 Beschreibung 7

1.2 Ziele 7

1.3 Methodik/Vorgehen 8

2 preCICE 9

3 Tools und Hilfsmittel 10

3.1 SCons 10

4 XML 11

4.1 Markup Language 11

4.2 Grundlagen 11

4.3 Geschichte 13

4.4 XML Parsing 13

4.4.1 XML Parser 13

4.4.2 DOM 13

4.4.3 SAX 15

4.5 XSL(T) 16

4.6 DTD 18

4.7 XPath 18

4.8 XML Verwaltungsbibliotheken 20

4.8.1 Xerces-C++ 20

4.8.2 Libxml2 21

4.8.3 LibXML++ 22

4.8.4 Expat 23

4.8.5 RapidXml 24

4.8.6 PugiXML 25
4.8.7 TinyXML 26
4.8.8 TinyXML2 26
4.9 irrXML 27
4.10 Wahl der XML Bibliothek 27
4.10.1 Definition der Kriterien 27
4.10.2 Benchmark Memory und Performance 30
4.10.3 Zusammenfassung der Eigenschaften 32
4.10.4 Auswertung der Kriterien und Ergebnisse 34
5 Entwicklung 36
5.1 Bisherige Umsetzung 36
5.2 Testprogramm 38
5.3 Der neue Parser 38
5.4 Automatische DTD Beschreibung 40
6 Fazit 41
7 Aussicht 42
8 Literaturverzeichnis 43
9 Selbstständigkeitserklärung 45

Abkürzungsverzeichnis

preCICE: Precise Code Interaction Coupling Environment

XML: Extensible Markup Language

RTTI: Runtime Type Information

STL: Standard Template Library

W3C: World Wide Web Consortium

API: Application Programming Interface

DTD: Document Type Definition

MPI: Message Passing Interface

HTML: Hypertext Markup Language

SGML: Standard Generalized Markup Language

DOM: Document Object Model

SAX: Simple API for XML

XSL: Extensible Stylesheet Language

XSLT: Extensible Stylesheet Language Transformation

RAM: Random Access Memory

GUI: Graphical User Interface

Abbildungsverzeichnis

Abbildung 4.1: XML Dokument 12
Abbildung 4.2: Baumstruktur eines XML Dokuments 14
Abbildung 4.3: Umwandlung von XML zum Result Tree 16
Abbildung 4.4: Umwandlung von Result Tree zu Darstellung 17
Abbildung 4.5: Beispiel XML für XPath 19
Abbildung 4.6: Parsing Zeit (x86), relativ zu pugixml. Die Farben repräsentieren die verschiedenen Dateien. Die X-Achse beinhaltet den Faktor, um den die anderen Parser schneller oder langsamer die Dateien verarbeiten konnten. 30
Abbildung 4.7: Parsing Speicherbedarf (x86), relativ zu pugixml Die Farben repräsentieren die verschiedenen Dateien. Die X-Achse beinhaltet den Faktor, um den die anderen Parser mehr oder weniger Speicher beim Verarbeiten der Dateien benötigen. 31
Abbildung 4.8: XML Parser im Überblick (funktionale Anforderungen) 32
Abbildung 4.9: XML Parser im Überblick (nichtfunktionale Anforderungen 1) 32
Abbildung 4.10: XML Parser im Überblick (nichtfunktionale Anforderungen 2) 33
Abbildung 5.1: Bisheriger Konfigurationsvorgang 37
Abbildung 5.2: Beispielhafter Aufruf von „connectTags(…)“ 39

1 Einleitung

Da große und komplexe Software oft vielseitig einsetzbar ist, muss diese auch an den speziellen Use-Case abgestimmt werden. Dies geschieht mithilfe der richtigen Konfiguration. Bei der Frage, welche Konfigurationstechnologie nun die jeweils richtige ist, spielen gewisse Kriterien eine wichtige Rolle. Je nachdem, welche Kriterien relevant sind, fällt die Entscheidung auf eine andere Technologie. Ein relevanter Aspekt ist zum Beispiel die Performance. Findet die Konfiguration zur Laufzeit statt und müssen Teile der Software mehrfach neukonfiguriert werden, so stellen Technologien, die weniger performant sind, ein Problem dar. Da vor allem im Bereich der Simulation, Berechnungen in Echtzeit stattfinden und diese auch visuell dargestellt werden, darf bei der Neukonfiguration so wenig Zeit, wie möglich verloren gehen. Ein anderer Faktor ist die Lesbarkeit für den Menschen. Wird die Konfiguration von Hand erstellt, so muss diese auch lesbar und wartbar sein. Konfigurationen im Binärformat, die zwar performant, jedoch nicht für den Menschen lesbar sind, entfallen somit. Zusätzlich muss für die Lesbarkeit auch eine Struktur gewährleistet werden. Bei kleineren Konfigurationsdateien reicht es aus, wenn die Einstellungen als Key-Value-Paare untereinandergeschrieben sind. Bei großen hingegen oder wenn diese sogar verteilt sind, bietet es sich an, eine Baumstruktur zu verwenden.

Diese Arbeit behandelt das XML-basierte Konfigurationsframework der Plug-and-Play Multi-Physik Kopplungssoftware preCICE (siehe Kapitel 2).

1.1 Beschreibung

preCICE (Precise Code Interaction Coupling Environment) ist eine Software zur Kopplung von Single-Physik Lösern zu komplexen Multi-Physik Szenarien. Die Software wird zur Laufzeit durch eine Konfigurationsdatei in XML konfiguriert. Dabei können einzelne Module der Software Callbacks für bestimmte Tags registrieren. Somit kann eine dezentralisierte Konfiguration, sowie eine Entkopplung der Module realisiert werden.

Das Softwaremodul zur Konfiguration setzt jedoch auf eine veraltete Bibliothek zur Verarbeitung von XML, welche u.a. Probleme bei der Verarbeitung von Kommentaren oder Namensräumen aufweist.

Im Rahmen dieser Bachelorarbeit soll das Konfigurationsmodul von preCICE ersetzt werden. Dabei soll die neue Implementierung weitgehend als ein Drop-In-Ersatz durch Beibehaltung der API fungieren, sodass keine weiteren, aufwendigen Änderungen im restlichen Teil des Codes entstehen.

1.2 Ziele

Das Ziel dieser Bachelorarbeit ist die Analyse und der Vergleich von existierenden XML Bibliotheken, sowie die Analyse der veralteten, bereits eingebauten XML Bibliothek. Die Auswertung dieser Ergebnisse soll dann entscheidend für die Wahl einer neuen Bibliothek sein, welche anschließend die veraltete Bibliothek ersetzt. Die Implementierung soll bei weitestgehender Beibehaltung der API erfolgen. Zusätzlich zur Implementierung sollen noch Unit-Tests im bereits vorhandenen Testing-Framework erstellt werden. Die letzte Anforderung ist optional: automatisches Erstellen eines XML Schemas, DTD- oder RelaxNG Beschreibung.

1.3 Methodik/Vorgehen

Das Vorgehen lässt sich in zwei Teile unterteilen: Den Theorie- und den Praxisteil. Der Theorieteil beschäftigt sich mit der Analyse von XML Themen und XML Bibliotheken. Der Praxisteil besteht aus dem Implementierungsprozess eines neuen XML Parsers, basierend auf dem Ergebnis aus dem Theorieteil.

Es werden die meisten Themen über XML in dieser Arbeit behandelt, die nicht zu tief ins Detail gehen. Vor Allem Themen, die direkt etwas mit XML Bibliotheken zu tun haben. Zum Beispiel Funktionen und Features von den Parsern. Oder allgemein: Themen, die relevant für das Konzept von XML sind.

Die Basis für zu untersuchende XML Parser ist Google und Stackoverflow. Gute Software hat meistens auch eine große Community. Sind viele Suchergebnisse bei Google oder viele Empfehlungen auf Stackoverflow zu einem XML Parser vorhanden, so kann davon ausgegangen werden, dass die Bibliothek potentiell als neuer Parser für preCICE in Frage kommt. Einige XML Bibliotheken werden von vorne herein ausgefiltert: Die Lizenz des neuen Parsers muss mit der Lizenz von preCICE kompatibel sein. Dabei entfallen alle kostenpflichtigen XML Parser. Außerdem kommen nur C und C++ Parser in Frage. Da Plattformunabhängigkeit ebenfalls gewährleistet werden muss, entfallen alle Parser, die nur eine kleine Menge an Systemen unterstützen.

Sind mehrere XML Parser zusammengekommen, werden diese einzeln beschrieben und anhand von definierten Kriterien untersucht.

Der Parser, der am ehesten, für preCICE geeignet ist, wird anschließend implementiert und ersetzt den aktuellen XML Parser, der einige Schwächen aufweist.

2 preCICE

Die, in C++ geschriebene, Multi-Physik Kopplungssoftware preCICE wurde entwickelt, um die Funktionalität von einzelnen Single-Physik Lösern als Plug-and-Play Multi-Physik Simulationssoftware zu vereinen. Das Tool beschäftigt sich mit einer Vielzahl an Simulationen von Strömungen um einen Körper. Aus diesem Grund ist eine hohe Flexibilität beim Einsatz dieser Software erforderlich. Das Ziel ist schnelles Prototyping mit bereits vorhandenen Lösern zu verbessern, indem die Kopplungsmethoden zwischen den Lösern verbessert werden. Die Kommunikation findet über MPI Ports (MPI-2.0) oder über TCP/IP Pakete statt und liegt damit bei der Software und nicht mehr bei den Entwicklern der einzelnen Löser. Einsatzgebiete sind Löser, die zum Beispiel auch auf verschiedenen Rechnern oder Supercomputern laufen. Die Kommunikation zwischen zwei einfachen Programmen kann ebenfalls von preCICE übernommen werden. Damit jeder Löser eine stabile Kommunikation erhält, müssen auch verschiedene Parallelisierungsstrategien unterstützt werden. Die Kommunikation findet Peer-to-Peer statt. Das heißt, es werden nur die Verknüpfungen zwischen einzelnen Knoten geschaffen, die auch notwendig sind.

Damit auch Closed-Source-Software preCICE als Schnittstelle nutzen kann, wurde preCICE unter der LGPL3 Lizenz veröffentlicht. Da diese Simulationssoftware als Bibliothek existiert, kann sie von jedem Programm, das Funktionsaufrufe im C-Stil unterstützt, benutzt werden. (Bungartz)

3 Tools und Hilfsmittel

3.1 SCons

SCons ist ein Open-Source Tool zum Bauen von Software. Gemacht, um das Make Tool zu ersetzen und mit einer ähnlichen Funktionalität wie autoconf/automake, Software leichter, zuverlässiger und schneller zu bauen. Das Tool wird mit Python konfiguriert und unterstützt eine automatische Analyse von Abhängigkeiten für C, C++ und Fortran. Weitere Sprachenunterstützung kann vom User ergänzt werden. Die Build-Unterstützung ist für viele Sprachen schon integriert und lässt sich ebenfalls erweitern.[1]

Dieses Tool wird für das automatische Erzeugen von preCICE verwendet.

[1] http://scons.org/

4 XML

4.1 Markup Language

Eine Markup Language ist eine Sprache, die aus Schlüsselwörtern, Namen oder Tags besteht und mithilfe von diesen die Visualisierung von Daten/Information ermöglicht. Bekannte Auszeichnungssprachen sind zum Beispiel XML, HTML und SGML. Im Vergleich zur Programmiersprache, ist die Markup Language nur für die Darstellung zuständig, während Programmiersprachen auch für die Verarbeitung und Manipulation von Daten zuständig sind.[2]

4.2 Grundlagen

XML steht für „Extensible Markup Language“ und beschreibt einen Weg, Informationen in einem Dokument strukturiert darzustellen. Zu den Informationen zählen sowohl eingebettete Bilder, als auch Text. Die Informationen werden als Text in einer hierarchisch aufgebauten Baumstruktur gespeichert. Der Baum besteht aus Elementen. Diese können entweder Text oder weitere Elemente enthalten.

XML ist eine wohldefinierte Sprache. Damit ein XML Dokument wohldefiniert ist, muss es alle Regeln erfüllen. Dazu gehört, dass ein XML Dokument genau ein Wurzelelement enthält. Alle anderen Elemente müssen dementsprechend in dem Wurzelelement liegen. Diese Elemente bestehen aus einem Start- und einem End-Tag. Ein Start-Tag enthält „<“, dann den Namen des Elements und anschließend „>“. Das End-Tag ist genauso aufgebaut. Nur, dass es am Anfang ein „</“ enthält. Hat ein Element keinen Text, so kann es direkt geschlossen werden: „<Name/>“. Wird ein Element in einem anderen geöffnet, so muss es auch innerhalb des gleichen Elements wieder geschlossen werden. Das heißt, dass ein Kind-Element nicht innerhalb eines anderen Elements, das nicht das jeweilige Elternelement ist, geschlossen werden darf. Der Text, der in einem XML Dokument verwendet wird, ist ebenfalls definiert. Damit dieser fehlerfrei ausgelesen werden kann, müssen alle XML Parser die UTF-8 und UTF-16 Kodierung unterstützen. Ein Element kann Attribute enthalten. Diese bestehen aus einem Namen und einem Wert, welcher in Anführungszeichen gesetzt wird. Enthält ein Element mehrere Attribute mit dem gleichen

[2] http://deacademic.com/dic.nsf/dewiki/116858

Namen, so ist dieses ungültig. Beim Start- und End-Tag spielt die Groß- und Kleinschreibung eine Rolle.

Außerdem können in einem XML Dokument CDATA Sektionen auftreten. Steht ein Text in dieser Sektion, wird dieser nicht als Markup erkannt. Das hat den Vorteil, dass XML Tags, als Text in diesen Sektionen dargestellt werden können, ohne dass Fehler bei der Visualisierung dieses Dokuments oder beim Parsen entstehen. CDATA Sektionen beginnen mit „<![CDATA[“ und enden mit „]]>“.

Kommentare sind bei XML ebenfalls erlaubt und dürfen überall außerhalb von anderen Markups vorkommen. Ziel davon ist, dass der dort enthaltene Text von XML Parsern nicht beachtet oder ignoriert wird. Ein Kommentar beginnt mit „<!--“ und endet mit „-->“. Die Zeichenfolge „--“ ist aufgrund von Kompatibilität zu SGML im Kommentarbereich nicht erlaubt. (Tim Bray, 2008)

```
<?xml version="1.0"?>
<books>
     <book>
          <author>Wiesel</author>
          <title lang="de">Der Pinguin</title>
     </book>
     <pubinfo>
          <date>03.12.2017</date>>
     </pubinfo>
</books>

<!-- Zuletzt geändert am 04.12. -->
```

Abbildung 4.1: XML Dokument

Die Abbildung zeigt ein XML Dokument, bestehend aus Elementen, einem Attribut und Textsektionen. „<?“, dann Text, „?>“ ist Verarbeitungsanweisung mit Informationen für eine bestimmte Anwendung. Auch wenn die erste Zeile einer Verarbeitungsanweisung ähnelt, handelt es sich um die XML Deklaration. (Henning Behme, 2012) Bei dieser Deklaration ist die Angabe der XML Version notwendig, damit ein Parser das Dokument entsprechend der Version verarbeiten kann. Der Rest, wie zum Beispiel die Angabe der Kodierung, ist optional. (selfhtml, XML/Regeln/XML-Deklaration, 2017)

4.3 Geschichte

Der Beginn von XML liegt in SGML (Standardised Generalised Markup Language), welche in den 1970ern von Charles Goldfarb, Ed Mosher und Ray Lorie bei IBM entwickelt wurde. Dabei handelt es sich nicht, wie der Name sagt, um eine Markup Language, sondern um eine Sprache, die diese Sprachen definiert. In den 1990ern entwickelten Jon Bosak, Tim Bray, James Clark und einige andere XML, da HTML zu eingeschränkt und SGML zu komplex war, um durch Menschen umgesetzt zu werden. (Anderson, 2004)

4.4 XML Parsing

4.4.1 XML Parser

Ein XML Parser ist ein Programm, welches die physikalische Darstellung eines XML Dokuments in eine In-Memory Form umwandelt. Liegt das XML Dokument in dieser Form vor, so kann es von anderen Programmen benutzt werden. Damit ein Programm XML auslesen kann, muss entweder der Parser in dieses Programm implementiert oder eine externe Bibliothek aufgerufen werden.[3]

4.4.2 DOM

Das DOM (Document Object Model) ist eine In-Memory Repräsentation von hierarchischen Dokumenten, welche sowohl ein plattform-, als auch ein sprachenneutrales Interface darstellt. Die W3C Schnittstelle ermöglicht Programmen dynamischen Zugriff auf die Daten, welche in Form von Knoten vorliegen, um die Struktur, den Inhalt und Stil eines Dokuments zu lesen und zu bearbeiten.[4]

Die Anfänge von DOM haben mit dem Beginn von JavaScript begonnen. Nachdem die Browser Internet Explorer und Netscape ihre eigene Version des DOMs hatten, „Dynamic HTML“ genannt, hat das W3C 1997 eine bessere Version zur Darstellung von HTML erschaffen: das DOM. Die DOM Level 1 Spezifikation erschien 1998, gefolgt von der DOM Level 2 Core Spezifikation im Jahr 2000. Die aktuelle DOM Core Spezifikation ist Level 3 und ist 2004 erschienen. (Hégaret, 2002)

[3] http://www.stylusstudio.com/xml/parser.html
[4] https://www.w3schools.com/xml/dom_intro.asp

Die Definition aller DOM Interfaces ist mit der Interface Description Language Notation durch die Object Management Group Organisation umgesetzt.

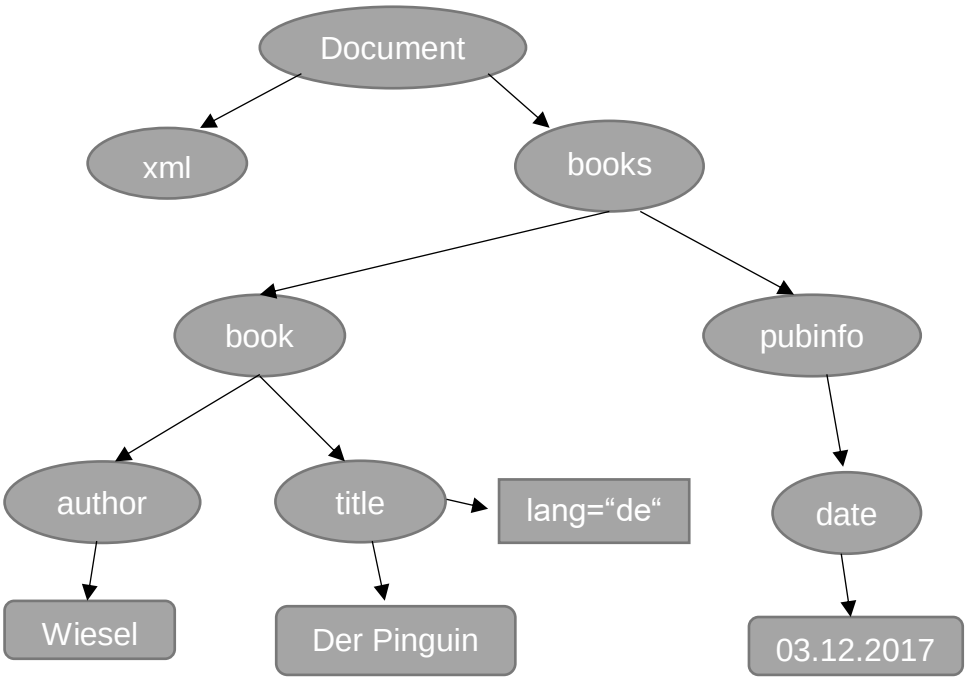

Abbildung 4.2: Baumstruktur eines XML Dokuments

Die Abbildung stellt das Dokument aus Abbildung 4.1 dar und zeigt, dass der Wurzelknoten Referenzen auf weitere Knoten hat. Und diese Knoten haben wiederum weitere Referenzen auf Knoten, sodass im Speicher ein verknüpfter Baum entsteht. Die Knoten repräsentieren die einzelnen Elemente. Außerdem haben die Knoten Attribute oder Textsektionen. Der Vorteil von DOM ist, dass aufgrund der Referenzen zwischen den Knoten, der Zugriff auf beliebige Elemente und ihre Attribute, sowie Textsektionen jederzeit möglich ist. Gleichzeitig heißt das aber auch, dass vor der Manipulation oder Validierung durch das Programm, das gesamte Dokument erst eingelesen werden muss. Dies bedeutet wiederum, dass der Speicher im RAM groß genug für ein vielfaches der eigentlichen Dokumentgröße, aufgrund des Overheads, sein muss, damit das gesamte Dokument als Baum aufgenommen werden kann. (Harold & Means, Document Object Model (DOM), 2002)

4.4.3 SAX

SAX (Simple API for XML) ist eine eventbasierte API zum parsen von XML Dokumenten. Eigentlich wurde das SAX Interface nur für Java entwickelt, jedoch unterstützen heutzutage viele Parser in unterschiedlichen Sprachen diese Schnittstelle. Zum Beispiel Libxml2, Xerces, Crimson und so weiter.

Die Entwicklung hat im Dezember 1997 mit Peter Murray-Rust begonnen. Er ist Entwickler der Java-basierten XML Browsers „JUMBO“ und musste drei verschiedene XML Parser mit jeweils anderen Technologien unterstützen. Danach initiierte er den Vorschlag, eine gemeinsame eventbasierte API zum Lesen von XML Dokumenten für Entwickler zu erstellen. Diese nannte er „YAXPAPI“ (Yet another XML Parser API). Nach weiteren Diskussionen mit anderen Entwicklern über diese gemeinsame Schnittstelle und vielen Kommentaren, Vorschlägen und Kritik erlaubte der XML Erfinder Jon Bosak die Nutzung der xml.org Domain für diese Schnittstelle. Das Release der ersten Version mit den Frontend-Treibern für die 4 wichtigsten Java XML Parser entstand 1998, nachdem David Megginson die Vorschläge für das Interface aufgeschrieben und umgesetzt hat. Nach weiteren Diskussionen und Verbesserungsvorschlägen mit der XML-DEV Community kam der SAX1 Release fünf Monate später. SAX2 erschien im Jahr 2000 und unterstützt Namespaces.[5]

Während DOM ein baumbasiertes Pull-Modell ist, ist SAX ein eventbasiertes Push-Modell. Das heißt, dass während das XML Dokument gelesen wird, gleichzeitig das Programm, das den SAX Parser implementiert hat, mit den notwendigen Daten versorgt wird. Dazu gehören der Start-Tag mit den Attributen, Präfixen und Namensräumen, dann Zeichenketten und das End-Tag. Einige Parser unterstützen sogar Kommentare. Wird eines der Bestandteile erkannt, wird sofort per Callback die Information an das übergeordnete Programm übermittelt und kann dort verarbeitet werden. So kann das Programm dann die relevanten Daten filtern und gegebenenfalls speichern. Das bietet sich vor allem bei großen Dokumenten an, welche nicht in den Speicher des Programms passen, da diese nicht vollständig gelesen werden müssen, sondern Schritt für Schritt abgearbeitet werden. Der Nachteil hingegen ist, dass auf diese Weise, Fehler, wie zum Beispiel ein fehlendes End-Tag erst spät erkannt werden. Zudem ist eine Manipulation des Dokuments nicht vorgesehen. (Harold & Means, Die Simple API for XML (SAX), 2005)

[5] http://www.saxproject.org/sax1-history.html

4.5 XSL(T)

XSL (Extensible Stylesheet Language) ist eine Formatsprache und ermöglicht das Beschreiben von XML Dokumenten. Diese enthält zwei Komponenten, die die Funktion der Formatierung und Transformation übernehmen. Für die Formatierung ist XML-FO (XSL Formatting Objects) zuständig, für die Transformation XSLT (XSL Transformation). (selfhtml, XML/XSL/XSLT, 2017)

Einsatzgebiet für diese Sprachen ist vor Allem folgender Use-Case: Eine XML Datei soll visuell dargestellt werden, so, dass sie von einem Menschen gut lesbar ist. Bei der Darstellung spezifiziert das XSL Dokument das Layout, die Position, bzw. allgemein sämtliche Darstellungsoptionen für ein Medium. Das Medium kann ein Smartphone, Browserfenster, PDF, und so weiter sein. Der Weg von XML zur Darstellung besteht aus zwei Schritten: Zuerst wird der XML Baum mithilfe von XSLT zu einem anderen Baum, dem Result Tree umgewandelt. Dabei werden auf das ursprüngliche XML eine Reihe von XSL Regeln angewendet. Danach wird aus diesem Result Tree eine Darstellung generiert. Die Anzahl an Knoten in den beiden Bäumen können komplett unterschiedlich sein.

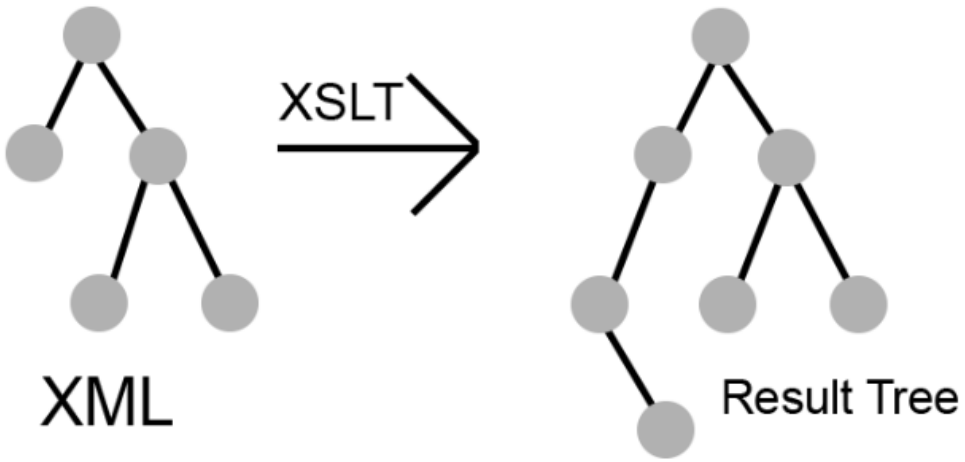

Abbildung 4.3: Umwandlung von XML zum Result Tree

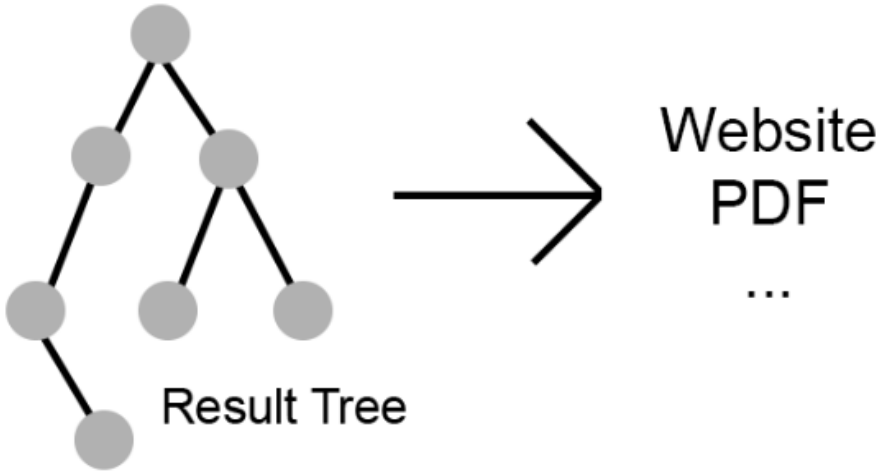

Abbildung 4.4: Umwandlung von Result Tree zu Darstellung

Eine XSL Regel besteht aus zwei Teilen: Pattern und Template. Zuerst wird das Pattern auf die XML Elemente angewendet. Findet eine Übereinstimmung statt, so wird mit Hilfe des Template ein neuer Teilbaum generiert. Dieses Konzept erlaubt es auf eine Vielzahl an Dokumenten angewendet zu werden, sofern alle eine ähnliche Baumstruktur haben. Der Result Tree kann wieder ein XML Dokument, HTML Dokument oder etwas Anderes sein. (Berglund, 2006)

4.6 DTD

Ein Parser kann mit der DTD Beschreibung ein XML Dokument validieren. Das heißt, dass geprüft wird ob die Elemente und der Inhalt korrekt ist. Dabei findet nicht nur eine syntaktische, sondern auch teilweise eine semantische Überprüfung statt. Zum Beispiel kann die richtige Reihenfolge der Elemente überprüft werden. Ist ein XML Dokument korrekt, so ist es nach der Validierung sichergestellt, dass es sowohl wohlgeformt, als auch valide ist. Eine DTD Beschreibung kann sowohl im Dokument, als auch außerhalb eines Dokuments in einer externen Datei stehen. Der Beginn der Beschreibung wird mit „!DOCTYPE“ und dem Wurzelnamen gekennzeichnet. Darauf folgt „[“ und wird ganz am Ende mit „]>“ wieder geschlossen. XML Elemente werden „<!ELEMENT“ und dem Namen definiert und anschließend mit „>“ wieder geschlossen. In der Definition stehen mögliche Kindknoten, die Anzahl oder welche Art von Text zugelassen ist.[6] Attribute werden mit „<!ATTLIST“ eingeleitet, gefolgt von dem Elementnamen, dem Attributnamen, dem Attributtyp und dem Attributwert. Das Schließen erfolgt wieder mit „>“. Der Attributtyp kann zum Beispiel ein einfacher Text sein. Der Attributwert gibt an, ob der Wert optional, fix oder notwendig ist. Es kann auch der Standardwert angegeben werden.[7]

4.7 XPath

XPath steht für XML Path Language und beschreibt eine Methode zur Navigation durch XML Dokumente. Mit über 200 eingebauten Funktionen, gehört XPath zum XSLT Standard. Die Anfänge waren 1999 mit XPath 1.0, danach XPath 2.0 im Jahr 2007. Die aktuelle Version ist XPath 3.0 (seit 2004). Mittlerweile ist XPath in vielen Programmiersprachen nutzbar. Darunter befinden sich JavaScript, Java, C, C++ und so weiter. XPath behandelt ein XML Dokument in Baumform.[8]

Dabei ist die Navigation ähnlich mit der, von einer Ordnerstruktur. Die Navigation startet mit dem Wurzelelement. Einzelne Knoten werden mit einem Slash getrennt.

[6] https://www.w3schools.com/xml/xml_dtd.asp
[7] https://www.w3schools.com/xml/xml_dtd_attributes.asp
[8] https://www.w3schools.com/xml/xpath_intro.asp

```xml
<?xml version="1.0" encoding="UTF-8"?>

<bookstore>

<book>
  <title lang="en">Harry Potter</title>
  <price>29.99</price>
</book>

<book>
  <title lang="en">Learning XML</title>
  <price>39.95</price>
</book>

</bookstore>
```

Abbildung 4.5: Beispiel XML für XPath[9]

Am Beispiel der Abbildung soll das erste Buch („book“ Knoten) aus dem Knoten „bookstore“ ausgewählt werden: „/bookstore/book[1]“. Die Knoten werden mit Slash getrennt und „[1]“ gibt das n-te Element an. Statt einer Zahl können auch weitere Filter angegeben werden. Zum Beispiel ermöglicht „last()“ das Auswählen von dem letzten Knoten. „position()<3“ ermöglicht die ersten beiden „book“ Knoten abzufragen. Mit dem Befehl „//title[@lang]“ lassen sich alle Knoten abfragen, die sich irgendwo befinden (dafür steht das „//“), den Namen „title“ haben und das Attribut „lang“ besitzen. Zusätzlich kann mit „//title[@lang='en']“ die Auswahl auf alle Elemente reduziert werden, die beim „lang“ Attribut den Wert „en“ haben. Wird ein „|“ zwischen den Pfaden verwendet, so können mehrere Pfade gleichzeitig abgefragt werden. Zum Beispiel liefert „//title | // price“ alle „title“ und „price“ Elemente.[10]

[9] https://www.w3schools.com/xml/xpath_syntax.asp
[10] https://www.w3schools.com/xml/xpath_syntax.asp

4.8 XML Verwaltungsbibliotheken

Um das Ziel zu erfüllen preCICE mit einem neuen XML Parser zu versehen, werden einige XML Bibliotheken näher untersucht. Das erste Auswahlkriterium ist, dass die neue Bibliothek mit C beziehungsweise C++ arbeiten können muss. Der Aufwand XML Parser aus anderen Sprachen einzubinden ist zwar technisch möglich, jedoch aufgrund der Vielzahl an C/C++ XML Parsern nicht notwendig. Des Weiteren soll die neue Bibliothek plattformunabhängig einsetzbar sein. XML Parser, die zum Beispiel rein auf Windows Systemen laufen, entfallen somit.

4.8.1 Xerces-C++

Xerces-C++ ist ein in C++ geschriebener XML Parser. Dieser Parser ist aktuell in der Version 3.2.0 verfügbar und nutzt nur einen Teil von in C++ vorhanden Technologien, um die Portabilität dieser Software zu erhöhen. Beispielsweise wird die Benutzung von Templates und „#ifdefs“ auf ein Minimum reduziert und auf RTTI vollständig verzichtet. Dieser Parser wird noch aktiv entwickelt. Jedoch werden vorherige Versionen (< 2.8.0) nicht mehr unterstützt. Die Shared Library erlaubt nicht nur Lesen und Schreiben von XML Dokumenten, sondern auch sie zu erzeugen und zu modifizieren. Außerdem gehört die Validierung des XML Dokuments zu den Hauptfunktionalitäten dieses Parsers. Zum Lesen des Dokuments steht dem Entwickler außer dem DOM auch die SAX und SAX2 API zur Verfügung. Xerces-C++ basiert auf den W3C Empfehlungen für XML 1.0 und vielen damit verbundenen Standards. Die Kernstärke liegt in der Performance, Modularität und Einsatz bei großen Projekten, sowie großen XML Dokumenten. Neben dem Source-Code, sind Codebeispiele und eine umfangreiche API Dokumentation vorhanden. Das alles ist unter der Apache Software License, Version 2.0 verfügbar. Mögliche Einsatzgebiete für den Parser sind zum Beispiel Web-Server, die viel Wert auf Sicherheit im Zusammenhang mit XML legen. Sowie in-Echtzeit-Validierung von XML Dokumenten für XML Editoren. Zusätzlich ist es vorgesehen, dass die Fehlerbehandlung von dem Entwickler angepasst werden kann.[11]

[11] http://xerces.apache.org/xerces-c/index.html

4.8.2 Libxml2

Libxml2 ist ein in C für das GNOME Projekt entwickelter XML Parser und Toolkit. Dieser Parser ist unter der MIT Lizenz veröffentlicht und wird nach wie vor noch aktiv weiterentwickelt. Libxml2 Version 2.9.7 ist das zurzeit neuste Release und es wird stark davon abgeraten Libxml1 zu verwenden. Wie auch Xerces-C++, setzt Libxml2 auf ein Minimum an zusätzlichen Abhängigkeiten um die maximale Portabilität zu gewährleisten. Für einfaches einbetten, wird ausschließlich ANSI C/POSIX verwendet. Der Parser funktioniert nicht nur auf Linux, Unix und Windows Systemen, sondern auch auf einer Vielzahl von anderen Plattformen. Sämtliche Features sind gut dokumentiert und ausführliche Code Beispiele erleichtern die Nutzung der jeweiligen Schnittstellen und Features. Es wird das Parsing von XML und HTML unterstützt. Dafür hat der Entwickler die Wahl zwischen einem Push- und einem Pull-basierten Parser Interface, da jeweils beide von Libxml2 unterstützt werden. Die Validierung erfolgt mit DTD, während des Parsing Vorgangs. Diese kann jedoch auch mit RelaxNG durchgeführt werden. Die internen Strukturen von Libxml2 werden technisch so nah wie möglich am DOM Interface orientierend, gehalten. Zusätzlich bietet der Parser eine minimalistische Version eines HTTP und FTP Clients, mit denen der Entwickler Zugriff auf Remote Ressourcen, wie zum Beispiel externe DTD Schemen hat. Dieses Feature bietet jedoch noch nicht allzu viele Funktionen und wird in Zukunft ausgebaut. Wie auch bei Xerces-C++ ist die Modularität hier eine Stärke: Features, die nicht benötigt werden, können beim kompilieren entfernt werden. Auf diese Weise kann Speicher gespart werden, der besonders bei eingebetteten Systemen wichtig ist. Falls abfragen gegen das XML System erwünscht sind, kann XPath verwendet werden, da es nach vollständigen Spezifikationen in Libxml2 integriert ist. (Veillard, Introduction, kein Datum) [12]

Neben dem hauptsächlichen DOM Interface und einer einfachen Callbackbasierten SAX Schnittstelle, existiert auch eine SAX-ähnliche Schnittstelle: TextReader. Da keine genauen Vorgaben zu einer SAX Schnittstelle in C existieren, wurde hier eine Hybridlösung gewählt. Diese Lösung entstand aus der SAX Implementierung von einer der ersten Versionen von Expat. Da eben SAX für C nicht wohldefiniert ist und das ganze Konzept recht kompliziert ist und zusätzlich die Validierung nicht von vorne herein möglich ist, wird noch auf ein Konzept aus der XmlTextReader API

[12] http://xmlsoft.org/intro.html

(C#) zurückgegriffen. Dieses Konzept wird als weitaus einfacheres Prinzip beschrieben, indem ein Cursor, streambasiert voranschreitet und bei jedem Knoten anhält. Das ganze passiert, indem die Funktion „Read()“ wiederholt aufgerufen wird. An diesem Punkt kann der Entwickler auf die Knoten und ihre Attribute und sogar die Namensräume zugreifen. Eine Validierung mit DTD durchzuführen, ist nun ebenfalls möglich. Dieses Konzept ist vom Prinzip her, sehr stark an die DOM Spezifikation angelehnt und erlaubt einen eher standardisierten Zugriff auf die XML Dokumente, als die vorhandene SAX Schnittstelle. (Veillard, Libxml2 XmlTextReader Interface tutorial, kein Datum) [13]

4.8.3 LibXML++

LibXml++ ist zwar kein eigener XML Parser, ermöglicht jedoch, den in C entwickelten Libxml2 Parser mit C++ Code zu nutzen. Dieser Wrapper ist unter der Lizenz LGPL Version 2 oder höher verfügbar und stellt die wichtigsten Funktionalitäten zur Nutzung der DOM, SAX und TextReader Schnittstelle bereit. Alle drei Schnittstellen sind mit einem Beispiel Code zur Veranschaulichung und Nutzung versehen. Im Gegensatz zu anderen C++ Parsern, strebt dieser Wrapper an, möglichst viele Vorteile von C++ zu nutzen und keineswegs auf Grund von Portabilität auf Technologien zu verzichten. Dabei wird auch kaum Wert auf die für Java vorgesehenen Spezifikationen für SAX, gelegt. Denn Ziel ist es, die dennoch recht komplizierten Vorgänge von Libxml2 mit C++ zu vereinfachen. Deswegen wird auch ein moderner Compiler g++ 4.9 oder höher benötigt, sowie zusätzliche Bibliotheken: Libxml2 und das GUI Toolkit glibmm-2.4 oder ein Teil davon, der „Glib::ustring“ enthält.[14]

[13] http://xmlsoft.org/xmlreader.html
[14] https://developer.gnome.org/libxml++-tutorial/stable/chapter-introduction.html

4.8.4 Expat

Expat ist eine, in C entwickelte, low level XML Parsing Bibliothek, die für den Einsatz in anderer Software oder als Basis für größere XML Parser und Editoren dient. Die Ziele sind Zuverlässigkeit, Geschwindigkeit und die Richtigkeit. Das Haupteinsatzgebiet ist das Parsing von sehr großen Dateien. Denn die Datei wird Stück für Stück eingelesen und geparst. Sind gewisse Callbacks für bestimmte XML Bausteine, wie zum Beispiel Start- oder End-Tags gesetzt, wird der Callback beim Lesen getriggert. Das ganze Verfahren ist streambasiert und ermöglicht auch das Einlesen von Teilen des XML Dokuments. So können auch Dokumente verarbeitet werden, die zu groß für den Speicher sind. Expat wird aktiv entwickelt und ist unter der MIT Lizenz verfügbar. Die aktuelle Version ist 2.2.5. Die Webseite ist noch nicht vollständig und verweist auf einen Artikel von 1999, der den Einstieg in Expat ermöglichen soll. Weder Validierung, noch richtige Namespaces werden unterstützt. Der Quellcode ist auf Github mit einigen Beispielen zur Nutzung der Bibliothek verfügbar. (Cooper, 1999)[15]

Expat ist in vielen Paketen enthalten und wird von einer Vielzahl an Software verwendet. Bekannte Software, die Expat einsetzt ist zum Beispiel Firefox, VirtualBox und WinSCP. Zwar ist Expat streambasiert, jedoch gibt es viele Wrapper, die auch die Nutzung von SAX und DOM ermöglichen.[16]

[15] https://www.xml.com/pub/1999/09/expat/index.html
[16] https://libexpat.github.io/doc/users/

4.8.5 RapidXml

RapidXml ist ein, in modernem C++ geschriebener, XML Parser. Wie der Name schon sagt, ist dieser mit dem Versuch, den schnellstmöglichen Parser zu entwickeln entstanden. Dabei ähnele die Geschwindigkeit des Parsers auf eine Datenmenge, der, von der „strlen()“ Funktion auf die gleiche Datenmenge. Die hohe Geschwindigkeit sei auf Code Optimierungen und Templates zurück zu führen, die bereits einen kleinen Teil zur Compile-Zeit erledigen. Außerdem werden keine String Kopien erzeugt, sondern Pointer in dem zu parsenden Text gespeichert. Der Vorteil ist, dass der gesamte Parser in einer einzigen Header-Datei ist und somit keine zusätzlichen Abhängigkeiten oder Konfigurationen benötigt werden. Für den Einsatz ist nur eine Include-Anweisung auf die Header-Datei notwendig. Die verwendete Lizenz ist Boost Software License 1.0 oder die MIT Lizenz. Die aktuelle Version ist 1.13 und wird seit 2009 nicht mehr weiterentwickelt. Das Error Handling findet über die C++ Exceptions statt, kann jedoch bei Bedarf auf Funktionen umgeleitet werden. Um die maximale Performance zu erreichen wird statt dem C++ new Operator ein Speicher Pool Objekt verwendet. Trotz der nicht Beachtung von Teilen der W3C Spezifikationen wird stets ein korrekter Baum aus XML Dokumenten erzeugt. Sonstige Korrektheit wird mit einer Reihe von Unit Tests sichergestellt. Außerdem verspricht das Tutorial mit einigen wenigen Code Beispielen die Einarbeitung in nur 2 Minuten. RapidXml unterstützt nur das DOM Interface. Damit Namensräume unterstützt werden, ist ein Patch erforderlich. (Kalicinski, 2009)[17]

[17] http://rapidxml.sourceforge.net/manual.html

4.8.6 PugiXML

Der C++ Parser pugixml ist DOM-basiert und kann den Baum entweder aus einem XML Dokument oder einem Text-Puffer bzw. IOStream erzeugen. Das Projekt ist aus dem pugxml Parser entstanden. Die aktuelle Version ist 1.8.1 und wird noch weiterentwickelt. Veröffentlicht unter der MIT Lizenz, kann pugixml genutzt werden, indem die 3 Dateien dem Projekt hinzugefügt werden. Alternativ lässt sich pugixml als statische Bibliothek erstellen. Die Stärken liegen bei der Portabilität, Speichereffizienz und der Geschwindigkeit. Modularität ist ebenfalls gegeben, da zum Beispiel, nicht nur das unterstützte XPath 1.0 deaktiviert werden kann, sondern auch STL und Exceptions. Zusätzlich zum normalen Modus, existiert auch ein kompakter Modus, der eine etwas andere Speicheraufteilung hat. Damit erhöht sich die Parsing Zeit und der Speicherbedarf wird verringert. Da kein architekturspezifischer Code verwendet wird, funktioniert pugixml auf vielen Architekturen und Plattformen, unter anderem auch auf Spielekonsolen.

Nachdem das XML Dokument in einen Baum umgewandelt wurde, kann durch diesen entweder mit der API oder mit XPath navigiert werden. So kann das Dokument auch bearbeitet werden. Und anschließend als ganzes oder nur teilweise gespeichert werden. Neben den normalen XML Bausteinen, werden auch Kommentare, Deklarationen und Processing Instructions erkannt. Um Speichereffizienz zu ermöglichen, werden bei Allokationen größere Speicherblöcke angefordert und nach und nach befüllt. Die Dokumentation ist umfangreich und übersichtlich und bietet vor Allem sehr viele Code-Beispiele mit Erklärungen. (Kapoulkine, 2016)[18]

[18] https://pugixml.org/docs/manual.html

4.8.7 TinyXML

TinyXML ist ein unter der zLib Lizenz veröffentlichter XML Parser, der in C++ entwickelt wurde. Das Ziel ist der schnelle Einstieg in den Code und die einfache Einbindung in Projekte, da keine besonderen C++ Konstrukte verwendet werden und auf STL, RTTI und Exceptions verzichtet werden kann. Mit der Erweiterung TinyXML++, die eine überarbeitete API bietet, können viele C++ Features genutzt werden. Zudem ist da das Error-Handling besser. Die aktuelle Version von TinyXML ist 2.6.0 und wird weder empfohlen, noch weiterentwickelt. TinyXML ermöglicht es DOM-basiert XML Dokumente zu parsen, bearbeiten und zu erzeugen. Eine eingeschränkte XPath Unterstützung wird mit der Erweiterung TinyXPath ermöglicht. Die Dokumentation ist ausführlich und gut beschrieben. Außerdem sind einige Code-Beispiele vorhanden. (Lee Thomason, 2010)[19]

4.8.8 TinyXML2

TinyXML2 ist komplett überarbeiteter XML Parser, basierend auf TinyXML. Dieser ist in C++ geschrieben und arbeitet ebenfalls mit dem DOM Interface. Die Funktionalität und die Ziele sind ähnlich, jedoch ist TinyXML2 schneller und verbraucht weniger Speicher. Außerdem sind weniger Dateien vorhanden, die zum Einbinden benötigt werden. Die Dokumentation und die Code-Beispiele sind ebenfalls ähnlich aufgebaut. Die aktuelle Version ist 5.0.1 und ist ebenfalls unter der zLib Lizenz verfügbar. (Thomason, 2017)[20]

[19] http://www.grinninglizard.com/tinyxmldocs/index.html
[20] http://leethomason.github.io/tinyxml2/

4.9 irrXML

Der zurzeit eingesetzte Parser irrXML ist in C++ geschrieben und unter der zLib Lizenz veröffentlicht, stammt ursprünglich aus der Irrlicht Engine und wird nicht mehr weiterentwickelt. Ziel ist es einen schnellen, speichersparenden XML Parser, vor Allem für Spiele bereit zu stellen. Mit einer Größe von 60 Kb, ist dieser Parser sehr klein. Die Stärken sind die Plattformunabhängigkeit und Möglichkeit, XML Daten aus vielen verschiedenen Quellen zu verarbeiten. Zum Beispiel: Dateien, Speicher, Netzwerk. Da keine Validierung stattfindet, muss sichergestellt, dass die Input-Daten korrekt sind. Dieser streambasierte Parser benötigt keine weiteren Abhängigkeiten. XML Daten können nur vorwärts geparst werden. Modifikationen sind nicht möglich. Die aktuelle Version (1.2) ist etwas dokumentiert und es existieren einige, wenige Code-Beispiele. (Gebhardt, 2005)[21]

4.10 Wahl der XML Bibliothek

4.10.1 Definition der Kriterien

Die neue Bibliothek soll noch aktiv entwickelt werden. Bibliotheken, die kein Support mehr erhalten, entfallen als potentieller Ersatz. Obwohl preCICE keine XML Dokumente von Außerhalb nutzt, müssen dennoch Sicherheitslücken vermieden werden, da preCICE auf vielen Rechnern verteilt agiert. Findet keine aktive Entwicklung von der Bibliothek mehr statt, so können kritische Fehler oder Bugs nicht mehr behoben werden. Zusätzlich ändern sich Teile der XML Spezifikation. Beziehungsweise es werden neue Spezifikationen ergänzt. Da die preCICE Konfiguration durch den Nutzer vorgenommen wird, müssen neue XML Features stets unterstützt werden.

Die Performance ist nicht besonders ausschlaggebend, da Dateien eine Länge von 100 Zeilen nicht überschreiten, aber dennoch interessant. Zurzeit findet die Konfiguration am Anfang statt. Es tritt also nie der Fall auf, dass zeitkritische Konfiguration erforderlich ist. Dennoch ist die Usability für den Nutzer angenehmer, wenn ein XML Dokument schnell verarbeitet wird.

[21] http://www.ambiera.com/irrxml/docu/index.html

Bei preCICE werden Namensräume und Präfixe verwendet. Die aktuelle Bibliothek weist Probleme mit Namensräumen auf: Der Parser fügt den Namen des Elements und das Präfix zusammen zu einem „Fullname“ und arbeitet mit diesem Namen weiter. Die neue Bibliothek muss dementsprechend Namensräume beherrschen, um die volle Funktionalität zu bieten.

Da die aktuelle Bibliothek keine richtige Validierung der Dokumente vornimmt, sondern eher davon ausgeht, dass diese korrekt sind, muss die neue Bibliothek Dokumente validieren können. Zumal auch eine der Anforderungen dieser Arbeit, das automatische Erstellen eines XML Schemas, beziehungsweise einer DTD/RelaxNG Beschreibung ist.

Der Parser ist nicht Hauptbestandteil von preCICE. Da noch andere Komponenten vorhanden sind, muss die Anzahl von zusätzlichen Abhängigkeiten, vor Allem dem Nutzer zuliebe, möglichst gering gehalten werden. Zusätzliche Abhängigkeiten erfordern auch eine Anleitung, damit diese vom Nutzer installiert und eingerichtet werden können. Erstrebenswert ist ein Parser, der auch in möglichst einfachem C++ geschrieben ist, also möglichst wenige zusätzliche C++ Features benutzt. So können mehr Systeme und Architekturen unterstützt werden und es wird kein zu moderner Compiler benötigt.

Verwendete Bibliotheken müssen von Seiten der Lizenz auf jeden Fall mit der aktuellen Lizenz von preCICE (LGPL3) kompatibel sein, damit die Nutzung von preCICE rechtlich gewährleistet ist. Die Bibliotheken wurden bereits so ausgewählt, dass alle über eine Open-Source-Lizenz verfügen.

Es spielt keine Rolle, welche API von der Bibliothek angeboten wird. Ob DOM-, SAX- oder streambasiert, die XML Konfiguration stattfindet, ändert nichts am Resultat. Streambasierte Bibliotheken, wären leichter zu implementieren, da die aktuelle Bibliothek ebenfalls streambasiert funktioniert. Der Implementierungsaufwand wäre mit dieser Methode minimal. Allerdings liegt die Hauptfunktionalität in den Callbacks, die für die Konfiguration entscheidend sind. Da wäre es naheliegend, eine SAX-basierte Bibliothek zu verwenden. Im Moment verwendet preCICE eine große Klasse (XMLAttribute.hpp), die dafür vollständig umgeschrieben werden müsste. Diese Klasse wird in vielen anderen Dateien verwendet, weswegen diese ebenfalls angepasst werden müssten. Diese Anpassungen wären möglich und sinnvoll, da so

im Endeffekt, Code gespart werden würde, da eine Zwischeninstanz (XMLAttribute.hpp) nicht mehr benötigt werden würde. Jedoch wären diese Maßnahmen im Rahmen dieser Arbeit vom Aufwand her, unangemessen. Die SAX Schnittstelle wäre am sinnvollsten, jedoch auch komplexer als die anderen Schnittstellen. Dafür spart diese Speicher während der Ausführung. Dies findet Relevanz auf kleineren Systemen. Eine DOM Struktur der Konfiguration ist auch möglich, da bei preCICE mögliche XML Konfigurationselemente sich bereits in einer Baumstruktur befinden. Das Abgleichen der Soll- und Ist-Elemente wäre somit vereinfacht. DOM hat den Vorteil, dass im XML Baum eine Vorwärts- und Rückwärtsnavigation möglich wäre. Diese wird jedoch nicht benötigt.

Die Plattformunabhängigkeit ist ebenfalls sehr wichtig, da preCICE auf vielen verschiedenen Rechnern läuft und vor Allem Linux, Windows und OS X kompatibel ist. Dieses Kriterium trifft jedoch auf alle untersuchten XML Parser zu und wird deshalb nicht weiter beachtet.

Das Kriterium, dass der Parser in C/C++ geschrieben sein muss, trifft ebenfalls auf alle XML Bibliotheken zu.

Zusätzlich zu diesen Kriterien kommen noch einige andere Kriterien, die nur auf die engere Auswahl angewendet werden:

- Wie einfach ist es diese Bibliothek als Entwickler einzurichten?
- Wie gut ist die Dokumentation?
- Existieren Code-Beispiele für das benötigte Szenario?

4.10.2 Benchmark Memory und Performance

Die folgenden Abbildungen zeigen den Speicherverbrauch und Zeitaufwand beim Parsing von 9 verschiedenen Dateien. Die Dateigröße beträgt zwischen 1 und 20 Mb und der Inhalt variiert beim Verhältnis von Markup zu Text. Außerdem befindet sich die 10 Mb große XMark-Test-Datei, die speziell für XML Benchmarking ausgelegt ist.

Die Auswertung erfolgt im Vergleich zu pugixml. Parser, die weiter rechts in der Grafik sind, benötigen mehr Zeit bzw. Speicher als pugixml und Parser, die weiter links benötigen dementsprechend weniger Ressourcen. Die horizontale Skala ist logarithmisch aufgebaut.[22]

Abbildung 4.6: Parsing Zeit (x86), relativ zu pugixml. Die Farben repräsentieren die verschiedenen Dateien. Die X-Achse beinhaltet den Faktor, um den die anderen Parser schneller oder langsamer die Dateien verarbeiten konnten.[23]

[22] https://pugixml.org/benchmark.html
[23] https://pugixml.org/benchmark.html

Die Tests wurden auf einem Intel Core i7 @ 2.67 GHz durchgeführt und dabei zum einen die Zeit gemessen,

- die DOM-basierte Parser zum erzeugen des Baumes benötigen,
- SAX-basiert Parser zum vollständigen Parsen mit dazugehörigen Dummy Callbacks
- Stream-basierte Parser zum einfachen durchlaufen des Dokuments[24]

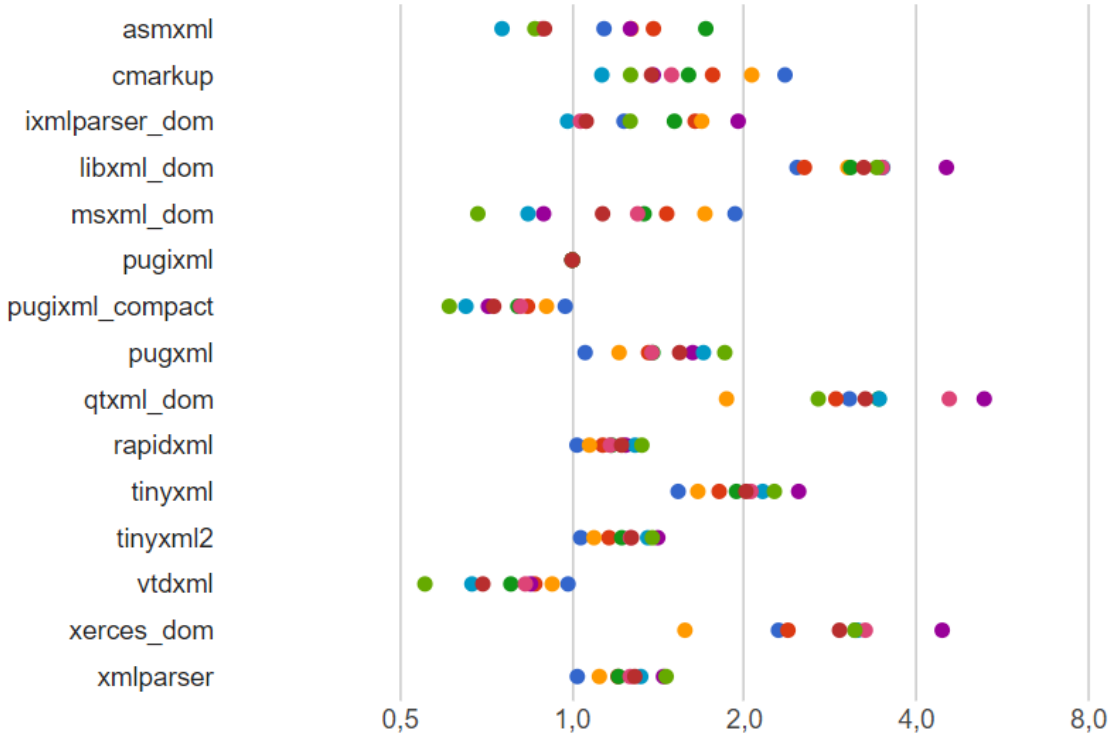

Abbildung 4.7: Parsing Speicherbedarf (x86), relativ zu pugixml Die Farben repräsentieren die verschiedenen Dateien. Die X-Achse beinhaltet den Faktor, um den die anderen Parser mehr oder weniger Speicher beim Verarbeiten der Dateien benötigen.[25]

Zum anderen wurde der Maximalwert der Speicherbeanspruchung bei DOM-basierten Parsern gemessen, da SAX- und streambasierte Parser kontinuierlich einen ähnlichen Speicherbedarf haben.[26]

[24] https://pugixml.org/benchmark.html
[25] https://pugixml.org/benchmark.html
[26] https://pugixml.org/benchmark.html

4.10.3 Zusammenfassung der Eigenschaften

	Lizenz	Valid.	Namesp.	Interface	XPath
Xerces-C++	Apache L. 2.0	Ja	Ja	DOM, SAX	XPath 1.0
Libxml2	MIT	Ja	Ja	DOM, SAX, Str.	Ja
Libxml++	LGPL v2 o.h.	Ja	Ja	DOM, SAX, Str.	Ja
Expat	MIT	Nein	Nein	Stream	Nein
RapidXml	MIT	Nein	m. Patch	DOM	Nein
PugiXml	MIT	Nein	Nein	DOM	Ja
TinyXML	zLib	Nein	Nein	DOM	m. Erw.
TinyXML2	zLib	Nein	Nein	DOM	m. Erw.
irrXML	zLib	Nein	Nein	Stream	Nein

Abbildung 4.8: XML Parser im Überblick (funktionale Anforderungen)

Erklärung der gelben Felder:

- Xerces-C++ unterstützt nur eine nicht vollständig den Spezifikationen entsprechende Version von XPath 1.0
- RapidXml unterstützt Namensräume nur mit einem Patch, der bei Github erhältlich ist.
- TinyXML und TinyXML2 unterstützen XPath nur mit der Erweiterung TinyXPath

	Aktiv	Sprache	Portabilität	Dokumentation	Code-Bsp.	Verbreitung
Xerces-C++	Ja	C++	+	++	++	+
Libxml2	Ja	C	++	++	++	++
Libxml++	Ja	C++	o	o	o	-
Expat	Ja	C	+	o	o	++
RapidXml	Nein	C++	o	+	o	+
PugiXml	Ja	C++	++	+	++	o
TinyXML	Nein	C++	+	+	o	-
TinyXML2	Ja	C++	o	+	o	-
irrXML	Nein	C++	+	o	o	-

Abbildung 4.9: XML Parser im Überblick (nichtfunktionale Anforderungen 1)

	Performance	Korrektheit	Skalierbarkeit	Sicherheit	Flexibilität
Xerces-C++	-	+	++	++	++
Libxml2	-	+	++	+	++
Libxml++	-	+	o	+	-
Expat	o	+	-	-	+
RapidXml	++	+	-	o	-
PugiXml	++	+	-	o	o
TinyXML	o	+	-	o	o
TinyXML2	+	+	-	o	o
irrXML	o	-	-	o	o

Abbildung 4.10: XML Parser im Überblick (nichtfunktionale Anforderungen 2)

4.10.4 Auswertung der Kriterien und Ergebnisse

Zusammenfassend lassen sich die zuvor genannten Kriterien wie folgt kategorisieren:

Hartes Kriterium (Kriterium muss erfüllt sein):

- Aktive Entwicklung: Wird die Bibliothek noch weiterentwickelt?
- Unterstützung von Namensräumen: Werden Namensräume und Präfixe als solche erkannt oder wird das Präfix nur als Name gewertet?
- Validierung: Werden Validierungen von XML Schemen oder RelaxNG-, DTD-Beschreibungen unterstützt?
- (Sprache)
- (Plattformunabhängigkeit)
- (Lizenz)

Weiches Kriterium (Erfüllung wünschenswert):

- Zusätzliche Abhängigkeiten: Werden weitere Abhängigkeiten/Bibliotheken benötigt?
- Performance: Wie schnell/speichereffizient findet das Verarbeiten der XML Dokumente statt?

Irrelevantes Kriterium:

- DOM/SAX/Stream: Welches Interface wird unterstützt?
- XPath: Wird XPath unterstützt?

Zusätzlich findet eine Aufteilung in funktionale und nichtfunktionale Anforderungen statt. Die nichtfunktionalen Anforderungen beschreiben weitere Kriterien, die über die eigentliche Funktion des Parsers hinausgehen:

- Portabilität: Lassen sich die verwendeten Sprachenkonstrukte auch auf anderen Architekturen verwenden?
- Dokumentation: Ist die Dokumentation gut beschrieben und welche Lernkurve ist zu erwarten?
- Code-Beispiele: Gibt es Code-Beispiele, die den Einstieg in den Parser erleichtern?
- Verbreitung: Wie viele Projekte nutzen diese Bibliothek?
- Korrektheit: Arbeitet der Parser immer richtig?

- Skalierbarkeit: Wie viele weitere Features existieren und lässt sich damit ein neuer Aufgabenbereich abdecken?
- Sicherheit: Ist der Parser für kritische Anwendungen geeignet?
- Flexibilität: Entspricht der Parser den W3C Standards?

Aktive Entwicklung, Unterstützung von Namensräumen und Validierung treffen nur auf Xerces-C++, Libxml2 und Libxml++ zu. Somit stehen die anderen XML Bibliotheken nicht mehr zur Auswahl. Libxml++ enthält zusätzliche Abhängigkeiten und ist nur zusammen mit Libxml2 verfügbar. Somit entfällt auch diese Bibliothek. Für Libxml2 und Xerces-C++ existiert eine gute und ausführliche Dokumentation mit passendem Code. Die Performance ist bei beiden Parsern ähnlich, jedoch ist Libxml2 SAX minimal schneller als der SAX Parser von Xerces-C++. Das DOM ist hingegen bei Xerces-C++ minimal platzsparender als das DOM bei Libxml2. Da Libxml2 in C geschrieben ist, benötigt es weniger Programmierfeatures als Xerces-C++ und ist somit in diesem Kriterium besser. Außerdem bietet Libxml2 sowohl DOM-, als auch SAX-, als auch eine streambasierte API, während Xerces-C++ nur DOM und SAX Schnittstellen hat. Eine vollständige XPath Unterstützung ist bei Libxml2 gegeben. Die Lizenz von Libxml2 enthält dazu weniger Restriktionen, als die von Xerces-C++. Zwischen Installation und erstem XML Parsing verging bei Libxml2 weniger Zeit, da die Installation weniger Konfigurationsschritte enthielt. Xerces-C++ hingegen bietet die Möglichkeit mehr Einstellungen vorzunehmen und somit das Setup an die Bedürfnisse des Benutzers anzupassen.

Nach Auswertung der Kriterien fällt die Entscheidung auf Libxml2 als neue XML Bibliothek für preCICE.

Für die Schnittstelle wird SAX gewählt, da es Potential hat, in Zukunft, Code und Speicher zu sparen und der Konfigurationsidee von preCICE mit den Tag Callbacks näherkommt, als die DOM- oder streambasierte Schnittstelle.

5 Entwicklung

5.1 Bisherige Umsetzung

Der aktuelle Parser irrXML besteht aus 6 Dateien, von denen 3 relevant sind:

- XML.cpp
- XML.h
- CXMLReaderImpl.h

XML.cpp enthält einige Methoden zum Erzeugen des streambasierten XML Readers aus CXMLReaderImpl.h und Lesen des XML Dokuments.

XML.h stellt die Enumeration von verschiedenen Knotentypen bereit:

- EXN_NONE: Initialzustand, kein Knoten ausgewählt
- EXN_ELEMENT: Aktueller Knoten ist ein Start-Tag
- EXN_ELEMENT_END: Aktueller Knoten ist ein End-Tag
- EXN_TEXT: Aktueller Knoten ist Text zwischen einem XML Element
- EXN_COMMENT: Aktueller Knoten ist ein Kommentar oder DTD
- EXN_CDATA: Aktueller Knoten ist eine CDATA Sektion
- EXN_UNKNOWN: Aktueller Knoten hat einen unbekannten Typ

Sowie einige Konvertierungsfunktionen und Getter-Methoden für die XML Attribute. Zum Beispiel für das Abrufen eines Attributs als Vektor oder als Bool.

CXMLReaderImpl.h implementiert die „read()“ und „getNodeType()“ Methoden, sowie das gesamte Lesen und Verarbeiten des XML Inhalts. „read()“ wird in einer Schleife aufgerufen. Dort, wo die XML Daten benötigt werden. Diese Funktion iteriert durch das Dokument bis zum nächsten Knotenpunkt. Dann wird die Funktion „getNodeType()“ aufgerufen um den Typ mit der Knoten-Enumeration aus XML.h abzugleichen. Handelt es sich um einen Start-Tag, so werden die Attribute verarbeitet.

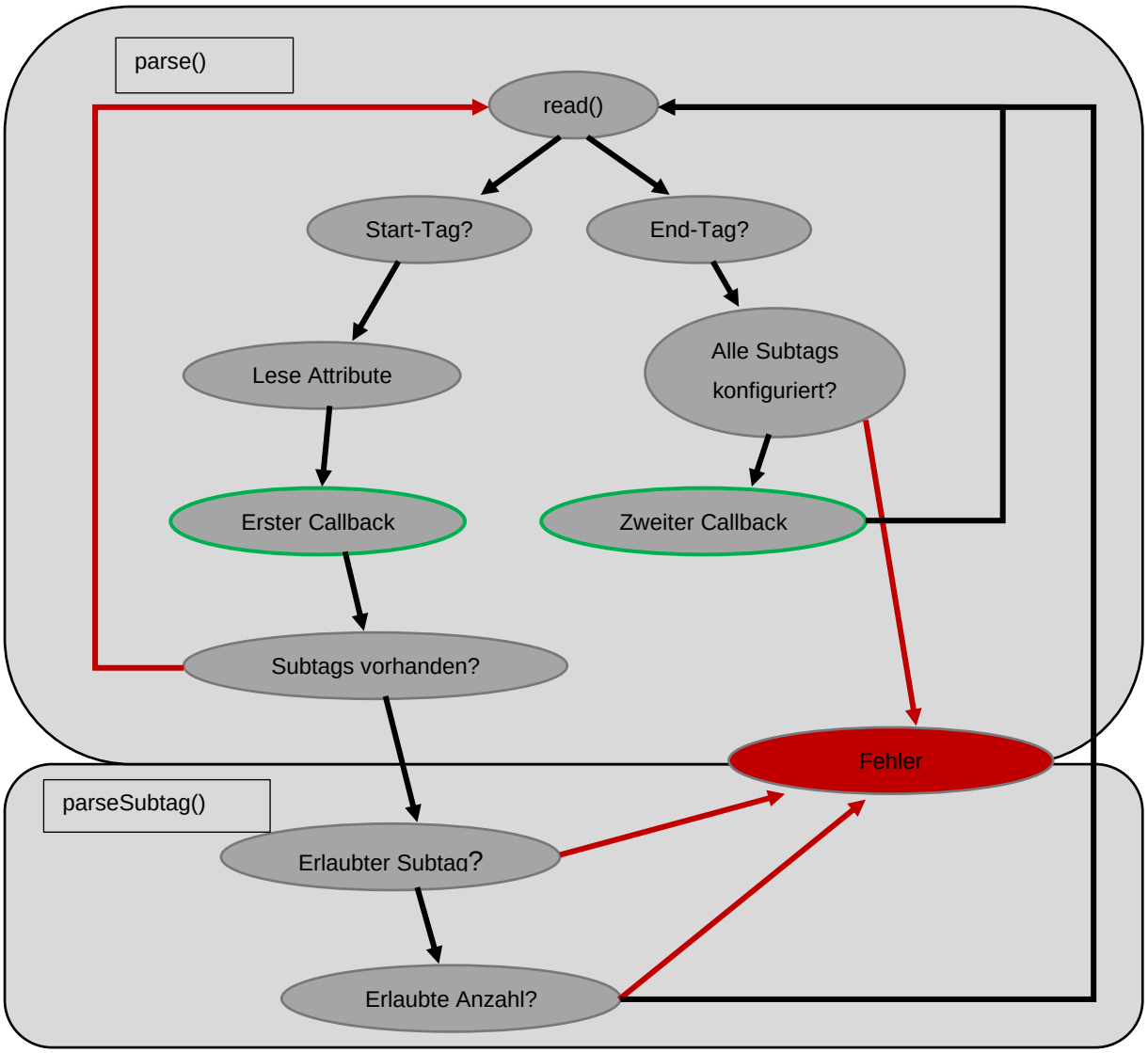

Abbildung 5.1: Bisheriger Konfigurationsvorgang

Die Abbildung zeigt den bisherigen Konfigurationsvorgang mit den beiden Funktionen „parse()“ und „parseSubtag()“ aus der Klasse „XMLTag“ (utils/xml/XMLTag.cpp, .hpp). Der rote Pfeil ist der Weg für „Nein“ und der schwarze für „Ja“ bzw. ein Folgepfeil. Der Vorgang basiert auf Rekursion: „parse()“ wird aufgerufen. Es wird der nächste Knoten gesucht (sobald „read()“ „false“ zurück gibt, wird die Funktion verlassen). Beim Start-Tag wird der erste Callback getriggert. Zu diesem Zeitpunkt sind die Attribute bereits vorhanden und können genutzt werden. Sofern Subtags vorhanden sind, werden diese rekursiv mit „parse()“ gelesen. Kommt irgendwann dann das End-Tag, so wird der zweite Callback getriggert. Zu diesem Zeitpunkt sind alle Subtags konfiguriert und können verwendet werden. Mögliche Tags, Subtags und Attribute sind bereits in einer Baum Struktur vorhanden. Die Bindung von den Elementen aus der XML an die bereits vorhandenen findet während der Callbacks statt. Wenn die Namen gleich sind, so gehören die zwei Elemente zusammen.

5.2 Testprogramm

Da der Kompiliervorgang von preCICE bei der kleinsten Änderung in der XML-Tag.cpp Klasse 10 Minuten dauerte, wurde ein Testprogramm geschrieben, welches die ersten Schritte des neuen Parsers simuliert. Es wird ein Dokument von dem neuen Parser geöffnet und ausgelesen. Anschließend werden Handler gesetzt, die auf Start-Tags und End-Tags warten und Callbacks triggern. Der neue Parser unterstützt ebenfalls Text zwischen den Elementen für zukünftige Zwecke, jedoch wird diese Funktion bei der Konfiguration im Moment noch nicht benötigt, da nur Start-Tags mit oder ohne Attribute vorhanden sind, sowie End-Tags. Mit jedem Callback wird dann eine Baumstruktur (wir definieren den Namen „Ist-Baum“) erstellt, die der Baumstruktur von den vordefinierten Konfigurationselementen (wir definieren den Namen „Soll-Baum“) ähnelt. Das, mit dem Ziel die zwei Bäume leichter abgleichen zu können, um die Bindung von den XML Elementen an die vorhandenen Elemente zu erzeugen.

5.3 Der neue Parser

Nachdem das Testprogramm vollständig funktionsfähig war, wurde es, bestehend aus zwei Teilen: „parser.cpp“ und „parser.hpp“ in preCICE implementiert. Der Parser hat volle Kontrolle über die zwei Bäume. Nachdem der Parser das XML Dokument vollständig gelesen hat, wird die Funktion „connectTags(…)“ mit den beiden Bäumen als Parameter aufgerufen. Diese Funktion ist für die Bindung der jeweiligen Elemente aneinander zuständig. Dabei wird jede Schicht, von beiden Bäumen, als Parameter von „connectTags(…)“, rekursiv aufgerufen.

```
<first_layer>

     <second_layer_a attr_a="1">
          <third_layer_a attr_a="2"/>

          <third_layer_b attr_b="3"/>
     </second_layer_a>

     <second_layer_b attr_b="4"/>

</first_layer>
```

```
<!-- erster Aufruf -->
<first_layer>

     <second_layer_a attr_a="1">
          <third_layer_a attr_a="2"/>

          <third_layer_b attr_b="3"/>
     </second_layer_a>

     <second_layer_b attr_b="4"/>

</first_layer>
```

```
     < !-- zweiter Aufruf -->
     <second_layer_a attr_a="1">
          <third_layer_a attr_a="2"/>

          <third_layer_b attr_b="3"/>
     </second_layer_a>

     <second_layer_b attr_b="4"/>
```

```
          <!-- dritter Aufruf -->
          <third_layer_a attr_a="2"/>

          <third_layer_b attr_b="3"/>
```

Abbildung 5.2: Beispielhafter Aufruf von „connectTags(…)“

Die Abbildung zeigt wie ein hierarchisches Dokument von „connectTags(…)“ verarbeitet wird. Dieser Baum entspricht in etwa dem Soll- und Ist-Baum. Beim ersten Aufruf wird der gesamte Baum übergeben. Die Funktion enthält zwei Schleifen, die geschachtelt durch beide Bäume durch iterieren. Angefangen bei dem Ist-Baum, wird das erste Element gefunden. Dann wird im Soll-Baum in der gleichen Ebene

das gleiche Element gesucht. Wird dieses gefunden, so werden die Attribute ausgelesen. Dann wird der erste Callback getriggert und die Elemente sind verknüpft. Wenn nicht, erscheint ein Fehler und der Vorgang wird abgebrochen. Der nächste Aufruf erfolgt mit dem Teilbaum aus der nächsten Ebene. Wieder werden die Attribute gelesen und der erste Callback für das Element getriggert. Beim dritten Aufruf wird sowohl der erste, als auch der zweite Callback getriggert: für beide Elemente, da keine Subtags mehr vorhanden sind. Daraufhin löst sich die Rekursion auf. Der zweite Callback aus dem zweiten Aufruf wird getriggert, sowie beide Callbacks für „second_layer_b“ und anschließend der zweite Callback aus dem ersten Aufruf. Natürlich wird auch überprüft, ob alle Elemente konfiguriert wurden und das Vorkommen korrekt ist. Zum Schluss wurden alle Verweise auf den alten Parser entfernt.

5.4 Automatische DTD Beschreibung

Die automatische DTD Beschreibung wird anhand der bereits vorhandenen Regeln erzeugt. Es ist bereits in preCICE definiert, welche Elemente welche Attribute enthalten müssen beziehungsweise dürfen. Außerdem sind Standardwerte schon angegeben. Diese Regeln müssen nur noch in DTD umgewandelt werden. Diese Umwandlung findet rekursiv statt. Jedes Element erzeugt die eigene Beschreibung und ruft die Kindelemente auf, sodass diese ebenfalls die Beschreibung erzeugen können. Unter jedem Element werden Außerdem die Attribute definiert. Anschließend ist preCICE in der Lage die gesamte Beschreibung in der Konsole auszugeben. Diese Ausgabe kann dann auf eine Datei umgelenkt und somit dort gespeichert werden.

6 Fazit

Diese Arbeit hat gezeigt, dass XML sehr mächtig und verbreitet ist und dass es viele Möglichkeiten gibt XML nicht nur als Konfigurationstechnologie zu nutzen. Es existieren viele XML Parser auf dem Markt. Die Google Suche liefert fast ausschließlich Open-Source XML Bibliotheken. Dabei fällt auf, dass viele kleine Parser vor Allem im Bereich Performance ihre Stärken zeigen. Diese haben dementsprechend wenige zusätzliche Features und beschränken sich nur auf das Nötigste. Vollständig ausgereifte und fast jedes Feature unterstützende XML Bibliotheken gibt es hingegen nur wenige. Im Bereich Open-Source und C/C++ beschränkt sich die Auswahl auf Libxml2 und Xerces-C++. Soll ein XML Parser in einem großen Projekt mit vielen Nutzern eingesetzt werden, so bieten sich die großen Parser an. Wird hingegen Performance, zum Beispiel bei Computerspielen oder in 3D Engines benötigt, so bieten sich die kleinen Parser an, die gegebenenfalls auch an spezielle Anforderungen angepasst werden können.

In dieser Arbeit wurden die verschiedenen Bibliotheken verglichen und analysiert. Anhand der Kriterien wurde anschließende der neue XML Parser als Drop-In Ersatz in die aktuelle API implementiert, der vor Allem, was die Skalierbarkeit der Aufgaben betrifft, sehr mächtig ist. Zusätzlich wurden Unit-tests (Boost-Tests) erstellt und in das bereits vorhandene Testing-Framework integriert. Zusätzlich wurde das automatische Erzeugen einer DTD Beschreibung ergänzt.

Zusammenfassend lässt sich sagen, dass kein perfekter und vollständig ausgereifter Open-Source C/C++ XML Parser existiert, da jeder Parser mit einem speziellen Use-Case entwickelt wurde und selbst die ganz großen noch nicht alle Funktionen vollständig implementiert haben. Nichtsdestotrotz, waren alle XML Parser sehr gut dokumentiert und mit Code-Beispielen ausgestattet, die vor Allem den Einstieg und das erste Verständnis des jeweiligen Konzepts vereinfacht haben.

7 Aussicht

Der neue Parser funktioniert so weit fehlerfrei, jedoch nicht maximal effizient. Das temporäre Erzeugen der Baumstruktur kann vermieden werden, wenn die Klasse „XMLTag.cpp“ umgeschrieben, beziehungsweise entfernt wird. Das direkte Triggern der Callbacks würde nicht nur Zeit, sondern auch Speicher sparen. Dafür müsste jedoch die Bindung von Soll- und Ist-Tags so früh wie möglich geschehen. Ein mögliches Konzept dafür wäre zum Beispiel, dass jedes Callback in den Ort feuert, wo die Tags definiert sind und dort zum Beispiel mit XPath die Zugehörigkeit zugeordnet wird. Da ja die Location von einem Tag in der XML Datei bekannt ist, sollte dies kein Problem darstellen, eine Abgleichung der Locations mit XPath durchzuführen.

Statt einzelne Fehler in einer XML durch den Parser suchen zu lassen, biete es sich an, einmalig eine Validierung mit Hilfe der erzeugten DTD Dateien und Libxml2 durchzuführen. Die hätte den Vorteil, dass der Fehler zu Beginn angezeigt wird und nicht erst nach dem die Hälfte konfiguriert wird. Zwar werden grobe Syntax Fehler, wie fehlende Klammern oder doppelte Attribute direkt am Anfang des Parsing Vorgangs erkannt, falsche oder fehlende Tags jedoch nicht.

8 Literaturverzeichnis

Anderson, T. (2004). *Introducing XML*. Von http://www.itwriting.com/xmlintro.php (Abgerufen am 03.12.2017)

Berglund, A. (2006). *Extensible Stylesheet Language (XSL) Version 1.1*. Von https://www.w3.org/TR/xsl/ (Abgerufen am 03.12.2017)

H.-J. Bungartz (2016). *preCICE – A fully parallel library for multi-physics surface coupling, Computers and Fluids.* Von http://linkinghub.elsevier.com/retrieve/pii/S0045793016300974 (Abgerufen am 03.12.2017)

Cooper, C. (1999). *Using Expat*. Von https://www.xml.com/pub/1999/09/expat/index.html (Abgerufen am 03.12.2017)

Gebhardt, N. (2005). *irrXML 1.2 API documentation*. Von http://www.ambiera.com/irrxml/docu/index.html (Abgerufen am 03.12.2017)

Harold, E. R., & Means, W. S. (2002). *Document Object Model (DOM)*. Von https://docstore.mik.ua/orelly/xml/xmlnut/ch18_01.htm (Abgerufen am 03.12.2017)

Harold, E. R., & Means, W. S. (2005). *Die Simple API for XML (SAX)*. Von https://www.data2type.de/xml-xslt-xslfo/xml/xml-in-a-nutshell/simple-api-for-xml/ (Abgerufen am 03.12.2017)

Hégaret, P. L. (2002). *The W3C Document Object Model (DOM)*. Von https://www.w3.org/2002/07/26-dom-article.html#history (Abgerufen am 03.12.2017)

Henning Behme, S. M. (2012). *Processing Instructions*. Von http://www.linkwerk.com/pub/xmlidp/2000/pi.html (Abgerufen am 03.12.2017)

Kalicinski, M. (2009). *RAPIDXML Manual*. Von http://rapidxml.sourceforge.net/manual.html (Abgerufen am 03.12.2017)

Kapoulkine, A. (2016). *pugixml 1.8 manual*. Von https://pugixml.org/docs/manual.html (Abgerufen am 03.12.2017)

Lee Thomason, Y. B. (2010). *TinyXml Documentation*. Von http://www.grinninglizard.com/tinyxmldocs/index.html (Abgerufen am 03.12.2017)

selfhtml. (2017). *XML/Regeln/XML-Deklaration*. Von https://wiki.selfhtml.org/wiki/XML/Regeln/XML-Deklaration (Abgerufen am 03.12.2017)

selfhtml. (2017). *XML/XSL/XSLT*. Von https://wiki.selfhtml.org/wiki/XML/XSL/XSLT (Abgerufen am 03.12.2017)

Thomason, L. (2017). *TinyXML-2*. Von http://leethomason.github.io/tinyxml2/ (Abgerufen am 03.12.2017)

Tim Bray, J. P.-M. (Hrsg.). (2008). *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. Von https://www.w3.org/TR/REC-xml/ (Abgerufen am 03.12.2017)

Veillard, D. (kein Datum). *Introduction*. Von http://xmlsoft.org/intro.html (Abgerufen am 03.12.2017)

Veillard, D. (kein Datum). *Libxml2 XmlTextReader Interface tutorial*. Von http://xmlsoft.org/xmlreader.html (Abgerufen am 03.12.2017)

9 Selbstständigkeitserklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

.

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.