XHydro

Erläuterung der zugrundeliegenden Entwurfsentscheidungen

INHALTSVERZEICHNIS

1 XHydro Allgemeine Entwurfsentscheidungen

„XHydro“ ist ein XML-Schema [1], welches für die Belange der Datenübertragung im hydrologischen Messnetz der Wasser- und Schifffahrtsverwaltung (WSV) konzipiert wurde.

Zentrale Eigenschaften dieses XML-Formats sind:

- Anwendungs-, Organisations- und Länderneutralität
- Offenheit für Erweiterungen um beliebige betreiberspezifische Dateninhalte
- Leichtgewichtig
- Modularer Aufbau
- Interoperabilität
- Möglichkeit zur Signierung, Verschlüsselung und Komprimierung

Aufbau und Struktur des XHydro-Schemas werden anhand eines UML-Modells in [2] dokumentiert.

Im Folgenden werden die grundlegenden Entwurfsentscheidungen dargelegt, die bei der Entwicklung des XML-Schemas getroffen wurden und dessen konkrete Ausgestaltung maßgeblich prägen. Dabei werden auch mögliche Alternativen zu den getroffenen Entscheidungen mit ihren jeweiligen Vor- bzw. Nachteilen diskutiert. Die Darlegung der Entwurfsentscheidungen dient vornehmlich der nachvollziehbaren Dokumentation des Entwicklungsprozesses, gibt aber auch Empfehlungen zur Nutzung bzw. Implementierung von XHydro--basierten Web Services.

1.1 IT-Anforderungen hinsichtlich Sicherheit und Datenvolumen

Das XML-Schema beinhaltet ausschließlich Datenstrukturen, die zur Erfüllung fachspezifischer Anforderungen benötigt werden. Hinsichtlich darüber hinausgehender, IT-technischer Belange, wie Datenkomprimierung oder Sicherheitsaspekten (Verschlüsselung, Authentifizierung, Authentizität) beinhaltet es keine implizite Unterstützung, insbesondere um diesbezüglich keine Vorgaben bzw. Einschränkungen vorzunehmen.

Dieses Vorgehen bietet mehrere Vorteile: Das XML-Schema bleibt vergleichsweise einfach und beschränkt sich auf die rein fachlichen Anforderungen. Es entsteht keine Abhängigkeit hinsichtlich der zur Gewährleistung der IT-Sicherheitsanforderungen eingesetzten Verfahren. Diese sind somit jederzeit austauschbar; es können und sollen die jeweilig dem aktuellen Stand der Technik entsprechenden und verfügbaren Standardtechnologien bei Übertragung bzw. Speicherung zum Einsatz kommen. Zudem lassen sich beispielsweise Komprimierung, Signierung bzw. Verschlüsselung jeweils optional, oder der fachliche Teil auch gänzlich ohne den Overhead von Signierung und Verschlüsselung verwenden. Darüber hinaus können Serialisierungs- und Deserialisierungsfunktionalitäten existierender Web Service-Rahmenwerke eingesetzt werden, um eine Objekt-Struktur automatisch in XML zu serialisieren bzw. um ein XML-Dokument zu parsen.

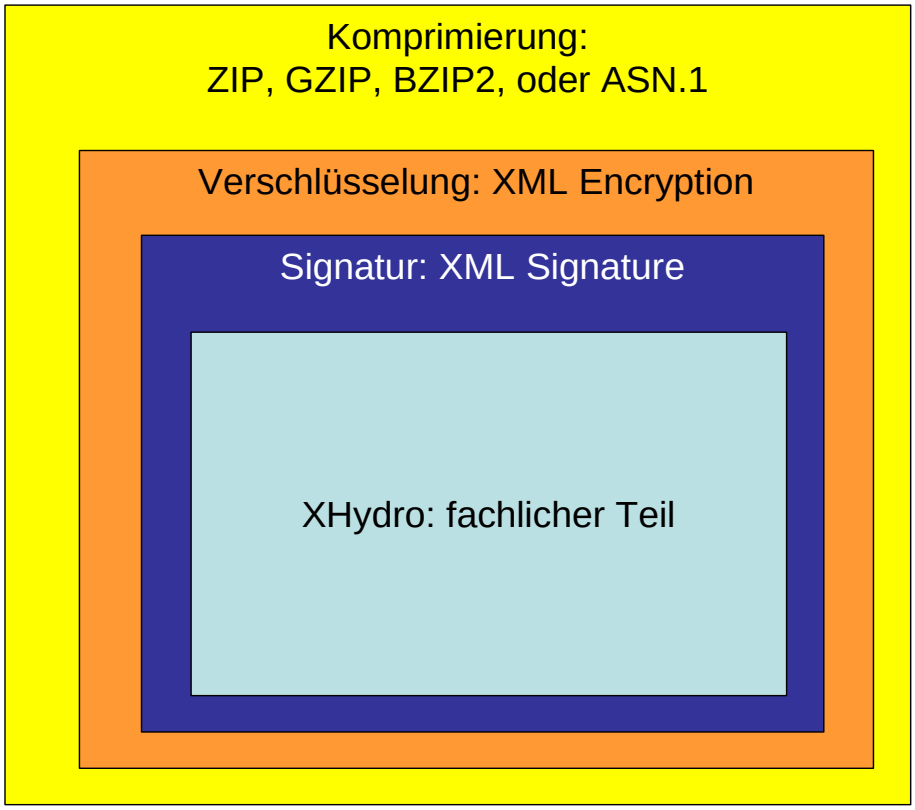

Abbildung 1: Beispiel einer XHydro-Datenübertragung

Im Folgenden wird auf die einzelnen Techniken nochmals detaillierter eingegangen.

1.1.1 Signatur

Um die Authentizität eines Dokuments sicherzustellen, bietet es sich an diese zu signieren. Für XML-Dokumente hat das W3C hierzu die Empfehlung „XML Signature“ [3] verabschiedet. Der Standard beschreibt drei verschiedene Signaturtypen, mit denen beliebige XML-Dokumente, oder ggf. auch nur Teile davon, signiert werden können:

- „Detached“-Signatur: Die Signatur eines XML-Dokuments wird unabhängig vom eigentlichen Inhalt, entweder in einer separaten Datei oder im gleichen XML-Dokument ausgelagert.
- „Enveloped“-Signatur: Hier ist die Signatur ein Teil des ursprünglichen XML-Inhalts.
- „Enveloping“-Signatur: Der ursprüngliche Inhalt des XML-Dokuments wird innerhalb des „`Object`“-Elements der Signatur gespeichert.

Eine wichtige Entwurfs-Anforderung war, dass XHydro-Dokumente möglichst selbstbeschreibend sein sollen. Diese Forderung lässt sich mit allen drei Typen erfüllen. Allerdings sollte die Signatur in derselben Datei gespeichert sein, in der auch der fachliche Inhalt abgelegt ist, d.h. bei einer „detached“-Signatur sollte diese nicht in eine externe Datei ausgelagert werden.

Empfehlung: Es sollte die „Enveloping“-Methode verwendet werden, denn der fachliche Inhalt kann dadurch entweder ohne Signatur, oder mit Signatur verschickt werden, ohne ihn dabei zu verändern.

Da der fachliche Teil bei dieser Vorgehensweise in jedem Fall im Originalzustand verbleibt, vereinfacht sich die Verarbeitung auf der Empfängerseite. Beispielsweise ließe er sich in beiden Fällen unproblematisch durch ein und denselben XSLT-Prozess [4] extrahieren.

1.1.2 Verschlüsselung

Für die Verschlüsselung von XML-Dokumenten hat das W3C den Standard „XML Encryption“ [5] definiert. Mit diesem Standard können

- gesamte Dokumente, oder
- beliebige XML-Elemente inklusive ihrer Tag-Namen, bzw.
- der Inhalt beliebiger XML-Elemente verschlüsselt werden,
  wobei hier die Tag-Namen im Klartext verbleiben.

Nur bei der ersten Methode, bei der das gesamte Dokument verschlüsselt wird, besteht keine Erfordernis das fachliche Schema zu modifizieren.
Bei Verwendung einer der beiden Element-weisen Verschlüsselungsvarianten hätte das fachliche Schema so angepasst werden müssen, dass für *jedes* ggf. zu verschlüsselnde XHydro-Haupt- bzw. Kind-Element ein optionales (im XML-Encryption-Standard definiertes) `EncryptedData`-Kindelement eingefügt wird. Eine solche Vorgehensweise hätte das fachliche Schema offensichtlich unnötig verkompliziert und seine Erweiterbarkeit erschwert. Außerdem müsste bei dieser Methode *vor* einer fachlichen Validierung jeweils geprüft werden, ob sich im Dokument evtl. `EncryptedData`-Elemente befinden, denn diese müssen zur Überprüfung selbstverständlich unverschlüsselt vorliegen. Wird das gesamte Dokument verschlüsselt, so ist natürlich auch vor der Validierung eine Entschlüsselung erforderlich, allerdings besteht hier kaum die Gefahr diesen Umstand zu übersehen. Nachdem auch bei der Datenübertragung im hydrologischen Messnetz der WSV ohnehin keine Notwendigkeit einer nur *teilweisen* Verschlüsselung existiert, wurde auf diese Option zugunsten eines einfacheren und leichter erweiterbaren Schemas verzichtet.

Empfehlung: Falls ein XHydro-Dokument vor Speicherung oder Datenübertragung explizit verschlüsselt werden soll, wird hierzu die Nutzung des XML-Encryption - Standards empfohlen. Vor der fachlichen Validierung eines Dokuments muss ggf. zunächst eine Entschlüsselung erfolgen. Auf eine gesonderte Verschlüsselung kann natürlich verzichtet werden, z.B. falls bei der Datenübertragung ein kryptographisch gesichertes Transportprotokoll, wie z.B. TLS, zum Einsatz kommt.

1.1.3 Dokument-Größe und Komprimierung

Während XML bei seiner Flexibilität viele Vorteile bietet (z.B. Plattformunabhängigkeit, Sprachunabhängigkeit, Menschenlesbarkeit), so hat es allerdings den Nachteil, deutlich „voluminöser“ als (im Extremfall) maßgeschneiderte, anwendungsspezifische Binär-Formate zu sein. Bei Einsatzszenarien, in denen die Größe der auszutauschenden Dokumente eine entscheidende Rolle spielt, kann dies dann zu Problemen führen bzw. ein Ausschlusskriterium

für den Einsatz von XML darstellen.

Bei der Schema-Entwicklung hat man darauf Wert gelegt, ohne Verzicht auf Flexibilität und Lesbarkeit, die Dokumentgröße möglichst klein zu halten.
Dazu wurde sich der folgenden Techniken bedient:

Unnötige Wiederholung von Elementen vermeiden: Das Schema bietet mehrere Möglichkeiten um Elemente einzusparen:
Liegt beispielsweise eine isochrone Zeitreihe vor, d.h. die Messwerte wurden in regelmäßigen Zeitabständen erhoben, so ist es möglich statt einer individuellen Zeitzuordnung zu jedem einzelnen Messwert, nur den Beginn und den zeitlichen Abstand der Messungen zu spezifizieren.
Bei Multiparameter-Zeitreihen, d.h. es wurden jeweils mehrere Messgrößen zur gleichen Zeit aufgenommen, lässt sich für diese Größen ein gemeinsamer Zeitstempel vergeben.
Qualitätsbezeichner, welche die Messunsicherheit beschreiben, lassen sich jedem einzelnem Messwert, aber auch pauschal -bzw. als überschreibbarer Standard-Wert- einer gesamten Zeitreihe zuordnen.

Kurze Elementnamen: Ein gewisser Grad an platzsparender Kodierung lässt sich auch durch die Verwendung kurzer XML-Elementnamen erreichen. Allerdings erkauft man sich diesen Vorteil auf Kosten der Lesbarkeit. Bei XHydro fiel die Entscheidung zugunsten kurzer Elementnamen, da dem Vorteil einer geringeren Dokumentgröße höhere Priorität beigemessen wurde. Im Bedarfsfall lassen sich XHydro-Dokumente mittels eines einfachen XSLT-Prozesses in eine besser lesbare Form mit langen, sprechenden Elementnamen überführen.

Namensraum (namespace) Präfixe vermeiden: In einem XML-Dokument können Elemente und Attribute aus unterschiedlichen Namensräumen genutzt werden, wobei dann entsprechend häufig auch die Präfixe für diese verschiedenen Namensräume angegeben werden müssen.
Dies kann vermieden werden, indem im XML-Schema die `elementFormDefault`- und `attributeFormDefault`-Optionen global auf „`unqualified`“ gesetzt, und möglichst viele Elemente und Attribute lokal im Schema verwendet werden. In diesem Fall ist das Namensraum-Präfix dann ausschließlich für global definierte Elemente und Attribute zu setzen.
Der wesentliche Nachteil dieses Ansatzes ist, dass lokale Elemente zu einem späteren Zeitpunkt nicht mehr verändert werden können, was der Anforderung nach einfacher Schema-Erweiterung widerspricht. Daher wurden in XHydro die wichtigsten wieder zu verwendenden Elemente global definiert. Zudem definiert das Schema nur *einen* Namensraum und gibt diesen (default namespace) vor, damit die wiederholte Angabe der Namensraum-Präfixe für Elemente unterbleiben kann.

Anzumerken ist, dass der Vorteil kurzer Elementnamen und eingesparter Namensraum-Präfixe zwar nur noch marginal ist, wenn längere Zeitreihen in (z.B. ZIP-) komprimierter Form zu übertragen sind. Allerdings hat im Messnetz der WSV der zeitnahe Versand von Messwerten überwiegend den früher üblichen täglichen Abruf abgelöst. Gerade in diesem mittlerweile dominierenden Anwendungsfall, bei dem nur sehr wenige Messwerte übertragen werden, ist der Vorteil kurzer Tag-Namen und vermiedener namespace-Präfixe, auch bei eingeschalteter Kompression, relevant.

1.1.4 Komprimierung des XML-Dokuments

Gerade Textdokumente mit sich häufig wiederholenden Textbestandteilen, wie sie XML-Dokumente typischerweise darstellen, lassen sich effektiv mittels Standard-Kompressionsverfahren (wie die lizenzfreien ZIP, GZIP, BZIP2 oder deflate) komprimieren. Im konkreten Fall von XHydro-Dokumenten lässt sich deren Größe bei sehr kurzen Zeitreihen auf 50% und bei langen Zeitreihen auf 5% ihrer Originalgröße reduzieren.
Oben genannte Verfahren sind praktisch für jede erdenkliche Plattform und Programmiersprache vorhanden und leicht in Web Service-Rahmenwerke integrierbar. Bei der Netzwerkübertragung stehen sie normalerweise ohnehin als Option zur Verfügung: Ausgetauscht werden XML-Dokumente i.d.R. mittels Web Services über das SOAP-Protokoll, welches wiederum auf dem Transportprotokoll HTTP basiert. Seit Version 1.1 bietet HTTP eine für Endgeräte transparente (ZIP oder deflate) Komprimierung bzw. Dekomprimierung, die sich beim Abruf wie auch beim Versand anfordern lässt.
Vor allem ist der Einsatz dieser Komprimierungsverfahren unabhängig vom XML-Schema; d.h. sie sind optional verwendbar bzw. austauschbar, so dass problemlos die jeweils dem Stand der Technik entsprechende Verfahren genutzt werden können.

Neben den o.g. Standardmethoden existieren auch spezialisierte Verfahren, mit denen sich eine noch stärkere Komprimierung von XML-Dokumenten erzielen lässt, wobei eine Konvertierung in ein binäres Format, wie z.B. XMill [6] erfolgt. Beispielsweise sei hier das ASN.1-Format [7] genannt, welches auch in der Telekom-Branche häufig zum Einsatz kommt. Software-Werkzeuge zur Transformation zwischen XML- und diesen Binär-Formaten sind aber nicht längst nicht in dem Maße verfügbar wie Standardmethoden, zudem fallen üblicherweise auch Lizenzkosten an. Weiterhin würden zu viele Vorteile wie Lesbarkeit, einfache Verwendung von Standard-Werkzeugen -z.B. zur Validierung- verloren gehen.

Empfehlung: Es sollten Standardkomprimierungsverfahren -insbesondere beim Einsatz von Web Services und HTTP- genutzt werden, vor allem da hierbei keine Anpassungen am Schema erforderlich werden.

1.2 Fähigkeit zur nahtlosen Zusammenarbeit

1.2.1 Web Service - Interoperabilität

Eines, wenn nicht *das* übergeordnete Ziel bei Web Services ist die sog. „Lose Kopplung“, wobei hierunter ein möglichst geringer Grad gegenseitiger Abhängigkeit der involvierten Hard- bzw. Software-Komponenten zu verstehen ist. D.h., alle beteiligten Instanzen sollen ad hoc miteinander kommunizieren können, also interoperabel sein. Zu Beginn der Entwicklung von Web Services galt die Interoperabilität zwischen Web-Diensten noch als ein zentrales Problem, da die Web Service-Standards (SOAP und WSDL) Fragen offen ließen und z.T. mehrdeutig spezifiziert waren. Dies führte dazu, dass Web-Dienste, welche mit verschiedenen Werkzeugen auf verschiedenen Plattformen erstellt wurden, nicht immer ohne Anpassungen miteinander kommunizieren konnten. Die Web Services Interoperability Organization (WS-I) [8], in der wesentliche Unternehmen vertreten sind, wurde mit dem Ziel gegründet, eindeutige Spezifikationen für Web Service-Standards festzuschreiben, damit die Interoperabilität WS-I-kompatibler Web-Dienste gewährleistet ist. Für XHydro war, zum Zeitpunkt seiner Entwicklung, der relevante Standard des WS-I das „Basic Profile“ in der Version 1.1.

Dessen Spezifikation fordert in Kapitel 4.8 (Use of XML-Schema [9]):

- R2800 A DESCRIPTION MAY use any construct from XML-Schema 1.0.
- R2801 A DESCRIPTION MUST use XML-Schema 1.0 Recommendation as the basis of user defined datatypes and structures.

Bei XHydro wurde auf WS-I-Kompatibilität geachtet, demgemäß basiert es wie gefordert ausschließlich auf standardisierten Konstrukten von XML-Schema, um die Interoperabilitätsanforderung zu erfüllen. Eine inhaltliche Einschränkung war dadurch nicht gegeben.

1.2.2 MS-Excel - Kompatibilität

Bei der Entwicklung wurde zudem darauf Wert gelegt, dass sich XHydro-Dokumente unmittelbar mit Microsoft-Excel laden und verarbeiten lassen.

1.3 Erweiterbarkeit, Modularität und Modellierungstechnik

Eine zentrale Anforderung an das XHydro-Schema war, darauf vorbereitet zu sein nachträglich möglichst beliebige Erweiterungen vornehmen zu können, ohne das dies bei bereits existierender Software zu Verarbeitungsschwierigkeiten führt.
Unter dieser Prämisse und der o.g. Excel-Kompatibilität konnten einige XML-Schema Konstrukte, u.a. abstrakte Typen, abstrakte Elemente und substitution-groups, nicht verwendet werden.

Die genannten Anforderungen wurden durch modularen Aufbau und Verwendung geeigneter Modellierungstechniken erreicht. Folgender Abschnitt beschreibt diese Aspekte.

1.3.1 Nutzungsmöglichkeit in anderen Anwendungsdomänen

XHydro wurde mit Blick auf Wiederverwendbarkeit in anderen Anwendungsdomänen, in denen Zeitreihen eine wichtige Rolle spielen, entwickelt. Zunächst wurde daher ein generisches, und aufbauend darauf ein fachspezifisches Schema erzeugt.

Dabei sollte es möglich ein, das Inhaltsmodell eines Elements oder Attributs im konkretisierten Schema wiederzuverwenden, wobei Erweiterungen bzw. Beschränkungen der Elemente und Attribute im konkretisierten Schema problemlos möglich sein sollten.
Einzelne wichtigere Elemente des allgemeinen Schemas sollten auch direkt referenziert werden können, damit sich darauf aufbauend im konkretisierten Schema im Bedarfsfall neue Elemente definieren lassen.

1.3.2 Modularität

Verschiedene fachliche Aspekte (z.B. Zeitdefinition, Maßeinheiten usw.) wurden in separaten Modulen behandelt, damit sich bestehende Standards leichter integrieren lassen.
In den generischen Schema-Modulen wurde kein Namensraum (`targetNamespace` XML-Schema-Attribut) definiert, um diese Module ohne Beschränkung in anderen, konkretisierten Modulen nutzen zu können.
Der Hintergrund ist, dass sich in einem XML-Schema nur solche Schema-Elemente neu de-

finieren lassen, die den gleichen Namensraum verwenden. Nachdem das allgemeine Zeitreihenschema auch für unterschiedliche Anwendungsdomänen wiederverwendbar sein, und die Schemas der verschiedenen Domänen natürlich auch ihre jeweils eigenen Namensräume haben sollten, müsste das generische Schema gleichzeitig mehrere Namensräume definieren. Daher blieb als einzig technisch machbare Lösung, im generischen Schema *keinen* Namensraum zu definieren. Man spricht hier von einem Chamäleon-Schema: Elemente von Schemas, die selbst keinen Namensraum definieren, verwenden automatisch den Namensraum des Schemas in den sie importiert werden. Somit können anwendungsspezifische Schemas die Elemente des importierten generischen Schemas neu definieren.

Die Entscheidung über den Grad der Modularisierung, d.h. wie viele Module konkret definiert werden sollten, leitete sich von allgemeinen aber auch von sich aus der Praxis ergebenden Randbedingungen ab: Grundsätzlich sollten alle Module eines anwendungsspezifischen Schemas denselben Namensraum verwenden, um die XML-Dokument-Instanzen kompakt zu halten. Natürlich sollten die jeweiligen, für spezifische Anwendungsfälle konstruierten Schemata (z.B. Medizin, Temperaturmessungen usw.) in separate Schemadateien ausgelagert werden, welche ihre jeweiligen Namensräume definieren.
Leider traten in der Praxis Probleme im Umgang mit XML-Schema-Konstrukten auf, welche diese Modularität (per include, redefine und import) unterstützen. Insbesondere bei längeren Inklusionsketten arbeiteten die (zum Zeitpunkt der Entwicklung von XHydro) verfügbaren XML-Schema-Prozessoren nicht immer fehlerfrei. Daher erschien es ratsam, zugunsten einer Verarbeitbarkeit mit möglichst vielen XML-Schema-Tools, den Grad der Modularisierung praktikabel einzuschränken. Daher wurden lediglich zwei Module definiert: ein Modul für das generische Zeitreihen-Schema (ohne Namensraum), und ein Modul für das XHydro-Schema (mit Namensraum).
Das Verhältnis zwischen generischem und anwendungsspezifischen Schemas zeigt folgende Abbildung.

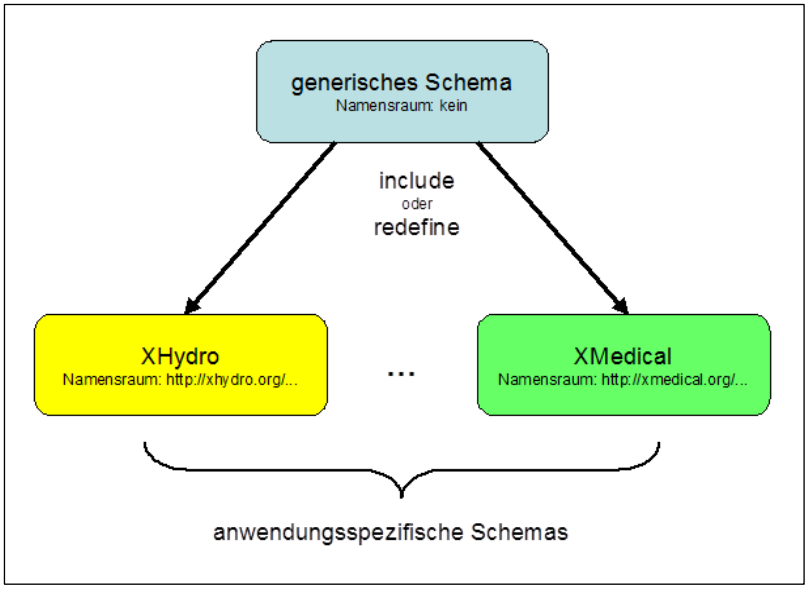

Abbildung 2: Beziehung zwischen generischem und spezifischen Schemas

1.3.3 Schema-Organisation

Neben der Modularisierung ist auch die richtige Schema-Organisation wichtig, um die Anforderungen hinsichtlich Erweiterbarkeit zu erfüllen. Grundsätzlich muss hierbei beachtet werden, dass lokale Elemente, Attribute und Inhaltsmodelle in anderen Modulen nicht referenziert werden können, d. h., sie können nicht direkt wiederverwendet werden.
Die üblichen Schema-Organisations-Strategien werden im Folgenden kurz erläutert.

Russian Doll: Alle Typen und Elemente sind lokal, mit Ausnahme des root-Elements. Die Namensgebung leitet sich von der Schema-Eigenschaft ab, im Aufbau verschachtelt wie eine russische Matroschka-Puppe zu sein. Diese Strategie ist nicht geeignet für wiederverwendbare Schemas.

Salami Slice: Alle Typen sind lokal, aber alle Elemente sind global. Das heißt, die unbenannten Typ-Definitionen sind in den Element-Definitionen eingebettet und nicht separat modelliert.

```
<xs:element name="XHydro">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="timeSeries" />
            <xs:group ref="otherAnyGroup" />
        </xs:sequence>
        <xs:attributeGroup ref="otherAnyAttributeGroup" />
    </xs:complexType>
</xs:element>
```

Abbildung 3: Beispiel für einen „Salami Slice“-artigen Schemaaufbau
Hier ist es möglich, die Elemente in neuen Modellen zu kombinieren, aber nicht die Inhaltsmodelle wiederzuverwenden bzw. zu verändern. Globale Elemente haben den Nachteil, dass für das gleiche Elemente immer das gleiche Inhaltsmodell verwendet werden muss, z.B. kann man keine verschiedenen `<name>`- Elemente für Zeitreihen *und* Personen verwenden.

Venetian Blind: Alle Elemente sind lokal definiert, sämtliche Typen sind global. Dadurch wird es z.B. möglich, verwendete Namensräume in den XML-Dokument-Instanzen zu minimieren, weil die Namensräume für lokal definierte Elemente versteckt werden können (`elementFormDefault`-Parameter im XML-Schema). Anderseits ist es nicht möglich, die Elemente direkt zu referenzieren und dadurch diese Elemente in anderen Schemas in anderer Kombination zu verwenden.

Garden of Eden: Elemente und Inhaltsmodelle sind beide global definiert. Es ist möglich, die Inhaltsmodelle zu verwenden und gleichzeitig auch die Elemente neu zu kombinieren. Der Ansatz hat die gleichen Nachteile wie die Salami-Slice-Strategie, und außerdem werden mit dieser Strategie die Schemas am komplexesten.

Um die an den Schemaaufbau gestellten Anforderungen zu erfüllen, wurde daher die Strate-

gie *Venetian Blind* gewählt. Alle Typen werden hierbei global, und die meisten Elemente lokal definiert. Ausnahmen bilden nur Elemente, die als Wurzelelemente im Dokument vorkommen können, diese müssen global definiert werden.

Attribute werden immer lokal definiert, weil der Default-Namensraum für Attribute nicht gültig ist. Daher lassen sich Namensraum-Präfixe nur vermeiden, indem alle Attribute lokal sind. Lokale Attribute sind in der Regel, mit ganz wenigen Ausnahmen (wie z.B. im RDF [10] Standard), in fast allen XML-Schemas die übliche Praxis.

1.3.4 Wiederverwendbarkeit

Grundsätzlich können Elemente eines Schemas auf drei Arten wiederverwendet werden:

Erstens lassen sich durch Komposition von existierenden Elementen und Gruppen eines Schemas neue Elemente bzw. Gruppen erstellen. Diese Art der Wiederverwendung ist durch die globale Definition der wichtigeren Elemente gewährleistet.

Zweitens kann man die global definierten, einfachen und komplexen Typen bzw. Element-Gruppen in neuen Typdefinitionen bzw. Gruppen-Definitionen erweitern bzw. einschränken. Diese Art der Wiederverwendung ist durch die globale Typdefinition gegeben.

Drittens besteht die Möglichkeit Typ- und Elementdefinitionen durch das XML-Schema-`redefine`-Konstrukt zu ersetzen. Durch diese Methode lässt sich in bestimmtem Rahmen das Aussehen eines Schemas verändern. Dies aber gilt nur für Schemas, welche den gleichen Namensraum definieren. Bei generischen Schemas, die keinen Namensraum definieren, gilt, was in Abschnitt 1.3.2 beschrieben wurde. Diese Art der Wiederverwendung spielt eine sehr große Rolle im XHydro-Projekt, weil es dadurch möglich wird, im anwendungsspezifischen Schema die Konstrukte des allgemeinen Schemas anzupassen.

Teilkomponenten eines Schemas lassen sich in veränderter Form an anderer Stelle wiederverwenden, jedoch gelten hier folgende bedeutsame Restriktionen:
Es können zwar Typen und Gruppen neu definiert werden, nicht aber Elemente; zudem dürfen einfache Typen auch nur eingeschränkt werden. Erweiterungen von komplexen Typen durch XML-Schema-Extensions sind nur durch Hinzufügen von neuen Subelementen am *Ende* eines Typs erlaubt, d.h., es ist nicht möglich Typdefinitionen an beliebigen Stellen zu erweitern. Gruppen-Definitionen dagegen lassen sich ganz flexibel verändern.
Daher wurden in XHydro, an Stellen an denen diese Flexibilität auch benötigt wird, Element-Gruppen definiert.

1.3.5 Joker-Elemente

Neben der Anforderung, das XHydro-Schema möglichst leicht abändern zu können, sollte es auch darauf vorbereitet sein, sich ohne Schemaänderungen um neue, bislang noch nicht vorgesehene Informationen ergänzen zu lassen. Beispielsweise könnte sich die Notwendigkeit ergeben, Zeitreihen in XHydro-Dokumenten Gerätehersteller-spezifische Messparameter als Metadaten anzuheften.
Diese Anforderung lässt sich durch die Verwendung sogenannter Joker-Elemente und -Attribute (engl. wildcard) erfüllen. Im XHydro-Schema wurde daher am Ende *jedes* komple-

xen Elements ein <any>-Schema-Konstrukt mit namespace="##other"- und processContent="lax"-Parametern, sowie bei *jedem* (komplexen oder einfachen) Element das <anyAttribute>-Schema-Konstrukt mit namespace="##other"- und processContent="lax"-Parametern verwendet.
Benötigt wird der namespace="##other"-Parameter hier, um nichtdeterministische Content-Modelle zu vermeiden, d.h. um zu erzwingen, dass Erweiterungen in einem anderen Namensraum definiert werden.
Mit der Vorgabe processContent="lax" wird spezifiziert, dass Schema-Definitionen für evtl. Erweiterungen optional sind. Damit ist sichergestellt, dass der offizielle Teil des XML-Dokuments (d.h. der Kernbestandteil ohne nachträgliche Erweiterungen) immer validiert und eingelesen werden kann, unabhängig davon, ob für evtl. Erweiterungen Schema-Dateien verfügbar sind.

Im Ergebnis wurde somit ermöglicht, dass sich bei XHydro-Dokumenten *jederzeit* (d.h. z.B. sogar nach Versand durch den Datensammler) an praktisch *allen Stellen* spezifische Erweiterungen (in Form von Elementen oder Attributen) vornehmen lassen, ohne dabei Probleme bei der späteren Verarbeitung (Validieren oder Parsen) zu verursachen. Anwendungen, die ausschließlich an den XHydro-Kerninformationen interessiert sind, können die angefügten Erweiterungen einfach ignorieren.

Wie oben beschrieben, ist es bei nachträglichen Erweiterungen von Typdefinitionen nur möglich, neue Elemente am *Ende* von existierenden Typen hinzuzufügen. Dann aber würden hierbei die Joker-Elemente nicht mehr (wie gewohnt) am Ende eines komplexen XHydro-Elementes stehen, was evtl. der Übersichtlichkeit abträglich sein könnte. Daher wurden die Joker-Elemente, der für die Erweiterung vorgesehenen Elementen, in Element-Gruppen eingebettet (beispielsweise timeSeriesExtensionGroup). Somit besteht die Möglichkeit, diese *Gruppen* anstatt der Typen neu zu definieren und dadurch neue Subelemente *vor* den Joker-Elementen zu platzieren.

1.3.6 Elemente mit variablem Inhalt

Es ist in vielen Fällen nicht sinnvoll oder gar unmöglich, die Bedeutung eines Elements vorab genau zu spezifizieren. Zum Beispiel sollte zu jeder Messung eine Ortsangabe geliefert werden. Aber je nach Art der Messung könnte der Messwert für einen Punkt, eine Linie, eine Fläche oder einen dreidimensionalen Raumabschnitt gelten. In derartigen Fällen wird ein Element benötigt, das abhängig vom Anwendungsfall verschiedene Inhalte haben kann.

XML-Schema bietet grundsätzlich vier verschiedene Möglichkeiten um solche Elemente zu definieren [11]:

Substitution group: Elemente, die zur gleichen substitution-Gruppe gehören, können ausgetauscht werden. So ließe sich z.B. ein abstraktes location-Element definieren, mit point- und line-Elementen, die zur substitution-Gruppe des location-Elements gehören. Nachteil dieses Ansatzes ist die umständliche Verarbeitung mit XSLT/XPath, da alle möglichen Elementnamen zu prozessieren sind. Außerdem ergibt sich noch das Problem, dass *sämtliche* Elemente in der substitution-Gruppe den Typ des abstrakten Elements erweitern müssen. Letzteres Problem ließe sich zwar lösen, indem beim abstrakten Element anyType als Typ verwendet wird. Nachteilig wäre allerdings, dass diese Lösung erzwingt sämtliche Elemente als global deklarieren zu müssen. Ein Vorteil des Ansatzes wäre jedoch,

dass man für alle Elemente einfache, sprechende Namen wählen kann, außerdem wäre die Menge der Elemente dynamisch erweiterbar.

Type substitution: In einem XML-Dokument ist es grundsätzlich immer möglich, anstatt eines vorgegebenen Typen einen anderen Typen zu verwenden, wenn dieser eine Erweiterung des ursprünglichen Typs darstellt. (D.h., solange diese Möglichkeit nicht explizit im Schema verboten wurde.) Dazu muss der aktuelle Element-Typ mit dem `xsi:type`-Attribut signalisiert werden. So lässt sich im Schema ein abstrakter Typ definieren (z.B. `locationType`) und später im XML-Dokument einer beliebiger *Sub*-Typ (z.B. `pointType`) verwenden. Dieser Ansatz beinhaltet den gleichen Nachteil wie bei der `substitutionGroup`-Lösung, d.h. *alle* Subtypen müssen eine Erweiterung des abstrakten Typen sein.
Außerdem werden XML-Dokumente dadurch etwas länger, da Subtypen immer explizit anzugeben sind (z.B. **`<location xsi:type="pointType" />`**). Andererseits lassen sich Dokumente gemäß dieser Lösung besser per XSLT verarbeiten, da sich der Elementname nicht ändert.

Choice: Das `choice`-Konstrukt kann verwendet werden, um im Schema eine beliebige Anzahl möglicher Elemente zu definieren, von denen im XML-Dokument dann eines davon benutzt werden kann. Allerdings weist diese Lösung den Nachteil auf, dass die Liste der Elemente sich später nicht mehr erweitern lässt. Somit wäre dieser Ansatz nur dann sinnvoll, wenn bereits vorab (d.h. zum Zeitpunkt der Schemadefinition) die später zu verwendenden Elemente genau bekannt sind.

Dangling Type: Schließlich lässt sich auch die Typdefinition eines Elements gänzlich offen lassen, indem im Schema hierzu nur Namensraum und Namen der Typdefinition spezifiziert werden. Später kann man im XML-Dokument angeben, welche Schemadatei aktuell für die Typdefinition verwendet werden soll. Nachteilig an diesem Lösungsansatz ist allerdings die damit getroffene Festschreibung des URI-Namensraums, was z.B. im Falle des generischen Schemas nicht akzeptabel wäre.
Sämtliche Ansätze weisen ihre individuellen Vor und Nachteile auf. Aufgrund der Anforderung, dass das Schema möglichst nur *einen* Namensraum benutzen soll, scheidet aber die Möglichkeit dangling type aus. Die geforderte Excel-Kompatibilität verbietet substitutiongroup- und type-substitution- Lösungen. Erstere benötigt abstrakte Elemente sowie substitution groups, letztere abstrakte Typen; MS-Excel (in Version 2007) unterstützt diese Konstrukte jedoch nicht.
XHydro verwendet daher den choice-Ansatz.

1.3.7 Codelisten

Um, z.B. für die Bezeichner von Messgrößen und deren Dimensionsangaben, eine einheitliche Nomenklatur zu verwenden, bietet es sich an, hierzu jeweils entsprechende Codelisten zu definieren. Diese Codelisten lassen sich im Nachhinein leicht um evtl. zusätzlich benötigte Einträge ergänzen; auch werden zudem bei solch einer minor-Schemaänderung (siehe Absatz 1.4) keine bestehenden Dokumente invalidiert. Aber ein Zwang zur ausschließlichen Verwendung vordefinierter Elemente würde im Bedarfsfall keinen ad-hoc - Gebrauch nicht definierter Codes erlauben. Im Rahmen eines den Entwicklungsprozess begleitenden Anwender-Workshops wurde u.a. dieses Defizit erkannt. Daher sind sämtliche Codelisten so modelliert, dass grundsätzlich zwei Elemente angeboten werden: Es lässt sich entweder ein

Codelisten-Eintrag oder ein beliebiger String verwenden. Optional lassen sich auch Metadaten (z.B. `codeList`, `codeListAgency`) zu bestimmten Attributen angegeben.

1.3.8 Namenskonventionen

Es empfiehlt sich Namenskonventionen einzuführen, um im gesamten Schema einen einheitlichen Stil zu verwenden, was der Lesbarkeit und Verständlichkeit zugutekommt. Entscheidend ist überhaupt eine Namenskonvention zu verwenden, die Details ihrer Ausgestaltung sind von sekundärer Bedeutung.
In XHydro wurde der Namenskonvention gefolgt, die auch in den meisten W3C-Standards benutzt wird. Für alle Element- und Attribut-Namen wird der sogenannte „lower camel case“ verwendet, z.B. `timeSeries` oder `dataValue`. Für XML-Schema-Typdefinitionen und für andere spezielle XML-Schema-Konstrukte wie Attributgruppen, werden konsistente Suffixe wie „Type“ oder „Group“ verwendet, um die Definitionen für Menschen leichter interpretierbar zu halten. Z.B. heißt daher die *Typ*-Definition des `timeSeries`-Elements „`timeSeries-Type`“.

1.3.9 Schema-Dokumentation

Insbesondere aufgrund der Entscheidung zur kompakten Namensgebung bei Element- und Attributnamen ist es wichtig, die Schemaelemente gründlich zu dokumentieren. Sonst bestünde die Gefahr, dass die exakte Semantik einiger Elemente für Endanwender nicht mehr nachvollziehbar ist. Deshalb wurde jedes Element und Attribut mit Hilfe von XML-Schema-`documentation`-Elementen kurz in natürlicher Sprache beschrieben. Dadurch beinhaltet die Schemadatei selbst eine Basis-Dokumentation, die für jeden Anwender des Schemas automatisch immer mitgeliefert wird.
Natürlich wird durch die eingebettete Dokumentation die Schemadatei größer, was u.U. zu Problemen führen kann, wenn das Schema häufig bei Endgeräten aktualisiert werden muss. Das kann aber nur dann vorkommen, wenn das Schema sich häufig ändert. Normalerweise werden die Schemas lokal gespeichert und es lässt sich z.B. durch „XML Catalog“ [12] spezifizieren, wo die lokale Schemadatei für einen bestimmten Namensraum liegt. Selbstverständlich muss man die Schemadatei immer noch lokal laden, wobei die Größe für eingebettete Geräte eine Rolle spielen kann. In diesen Fällen lässt sich eine Schema-Version bereitstellen, in der die `xs:annotation`-Teile, z.B. durch ein einfaches XSLT-Skript, ausgefiltert werden.

```
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
    <xsl:template match="@*|*">
        <xsl:copy>
            <xsl:apply-templates select="*|@*|text()"/>
        </xsl:copy>
    </xsl:template>
    <!-- Filter out documentation elements -->
    <xsl:template match="xs:annotation">
        <!-- Do nothing -->
    </xsl:template>
</xsl:stylesheet>
```

Abbildung 4: Beispiel für ein XSLT-Skript zum Entfernen der `annotation`-Elemente

Außerdem ist es wichtig zu erwähnen, dass eine Validierung von XML-Dokumenten gegen das Schema optional ist. Das heißt, ein Gerät kann auch auf die Validierung verzichten und das XML-Dokument ganz ohne Schema-Unterstützung parsen, womit sich dann auch die Notwendigkeit zum Vorhalten der Schemadatei erübrigt.

1.4 Versionierung

Typischerweise ergeben sich im Laufe der Zeit Veränderungen an den technischen oder organisatorischen Rahmenbedingungen, die für einen Web Service gelten. Daher ist es von großer Bedeutung, die Behandlung verschiedener Schema-Versionen bereits im Vorfeld klar zu regeln, um mögliche Probleme im operationellen Betrieb auszuschließen.

Es wurde sich dazu entschieden, den Empfehlungen [13] [14] zu folgen, die von der XML-Schema-Community als „best practices“ angesehen werden:

Bei Schema-Versionierungen wird üblicherweise zwischen kleineren (minor) und größeren (major) Versionssprüngen unterschieden:

Kleinere Versionssprünge enthalten ausschließlich Änderungen, welche weder existierende gültige Dokumente invalidieren, noch existierende ungültige Dokumente gültig machen. Mit anderen Worten: Der existierende Datenbestand muss bei solchen kleineren Änderungen nicht angepasst werden.

Auf Grund dieses Kriteriums gelten die folgenden Änderungen als kleine Änderungen:

- Hinzufügen eines neuen optionalen Elements
- Hinzufügen eines neuen optionalen Attributs
- Hinzufügen eines Joker-Subelements zu einem komplexen Element (`<any>`)

- Hinzufügen eines Joker-Attributs zu einem komplexen Element (`<anyAttribute>`)

Die Aufzählung beinhaltet das Hinzufügen der Joker-Elemente bzw. -Attribute nur der Vollständigkeit halber. Denn nachdem alle Elemente des generischen Schemas bereits Joker-Konstrukte besitzen, wäre ein weiteres Hinzufügen nicht sinnvoll.

Große Versionssprünge stellen sämtliche sonstigen Modifikationen dar, da hierdurch die Semantik der XML-Dokumente, welche dem Schema folgen, verändert wird.

U.a. gilt dies für folgende Änderungen:

- Löschen eines Elements
- Löschen eines Attributs
- Änderung des Datentyps eines Attributes bzw. eines einfachen Elements
- Hinzufügen eines obligatorischen Elements bzw. Attributs
- Änderung der Reihenfolge der Subelemente eines komplexen Elements

Versionsangaben werden durch das XML-Schema-Standardattribut „`version`“ in Form von „`x.y`“ repräsentiert, wobei „`x`“ für die major-, und „`y`“ für die minor-Versionsnummer steht. Beide Versionsnummern sind immer ganzzahlig und beginnen mit „0“. Gültige Versionsbezeichnungen wären demnach z.B. „`0.1`“, „`1.0`“ oder „`1.5`“.

Zudem enthält auch der jeweilige Namensraum des XML-Schemas eine Angabe über die major-Versionsnummer, indem dieser die monatsgenaue Information über das Datum beinhaltet, an dem der jeweilige Versionssprung erfolgte. Außerdem stellt der Bezeichner für den Namensraum einen gültigen URI mit einer existierenden Webadresse dar, von der das Schema und zugehörige Beschreibungen heruntergeladen werden können. Die Webseiten mit URIs von älteren Schema-Versionen sollten außerdem auch Links auf die neueste Schema-Version enthalten. Ein gemäß dieser Definition gültiger Namensraum, eines im Oktober 2006 verabschiedeten Schemas, könnte demnach beispielsweise `http://xhydro.org/schemaname/2006/10` lauten. Auf der Schema-Webseite für eine bestimmte Hauptversion sollte immer auch deren späteste kleinere Version erreichbar sein. Dadurch wird gewährleistet, dass neue Dokumente immer die neueste kleinere Version verwenden. Gleichzeitig schadet dieses Verfahren nicht bei der Verarbeitung älterer Dokumente, da neue minor-Schemaversionen, wie beschrieben, benutzt werden können. Nachteilig an dieser Art von Namensraum-Änderung ist, dass hierbei formal (d.h. quasi „für Computerprogramme“) nicht grundsätzlich nachvollziehbar ist, dass zwischen den beiden beispielhaften Namensräumen `http://xhydro.org/schemaname/2006/10` und `http://xhydro.org/schemaname/2006/12` eine Verbindung besteht. Nachdem aber für Menschen diese Verbindung immer noch leicht nachvollziehbar bleibt, gilt dieses kleine Manko, angesichts der Vorteile gegenüber anderen Lösungen, als leicht tolerierbar.

Bei generischen Modulen, die selbst keinen Namensraum besitzen, werden deren Versionsangaben natürlich nur im Attribut `version` geführt. Schließlich wird noch die letzte Änderung eines Moduls in der Schemadatei in einem Dublin-Core - `modified`-Element gespeichert.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:dcterms="http://purl.org/dc/terms/"
           elementFormDefault="qualified" version="0.1">
    <xs:annotation>
        <xs:documentation>
            Generic XHydro time series model.
            This model describes a generic view
            of a time series that should be reusable
            across many domains.
        </xs:documentation>
        <xs:appinfo>
            <dcterms:modified xsi:type="dcterms:W3CDTF">
                2006-10-26
            </dcterms:modified>
        </xs:appinfo>
    </xs:annotation>
</xs:schema>
```

Abbildung 5: Beispiel zu Schema-Versionsangaben eines generischen Moduls (ohne Namensraum)

Wichtig ist zu beachten, dass ein größerer Versionssprung in einem generischen Modul auch einen größeren Sprung in allen anderen Modulen, welche das veränderte Modul verwenden, nach sich zieht.

1.4.1 Alternative Versionierungsverfahren

Neben der bei XHydro favorisierten Lösung sind natürlich auch andere Verfahren zur Versionierung bekannt. Einige dieser alternativen Lösungen werden im Folgenden der Vollständigkeit halber kurz erläutert, insbesondere um darzulegen, weshalb sich für die oben dargestellte Vorgehensweise entschieden wurde.

Eine Alternative wäre, bei Versionssprüngen nicht den Namensraum selbst, sondern lediglich den Speicherort des Schemas zu verändern. Die jeweils gültigen Versionen würden dann mit dem Standard-Attribut `schemaLocation` in den XML-Instanzen referenziert werden. Dem Vorteil des unveränderten Namensraums stünde der Nachteil entgegen, dass das `schemaLocation`-Attribut nur als *Hinweis* für den XML-Parser zu verstehen ist, und entsprechend grundsätzlich auch eine andere XML-Schemadatei für den Namensraum lokal verwendet werden könnte. Letztlich wäre dann die Verwaltung der verschiedenen Schema-Versionen nicht deterministisch, sondern vielmehr abhängig von der jeweiligen Parser-Implementierung bzw. Konfiguration.

Eine weitere gängige Möglichkeit besteht darin, Versionsinformationen explizit durch ein neues Attribut (z.B. könnte man dies „`schemaVersion`“ nennen) auch in den XML-Dokument-Instanzen auszuweisen. Auch in diesem Fall ergäbe sich der Vorteil, dass keine

Anpassungen des Namensraums bei Versionssprüngen nötig würden. Allerdings liegt die Auswahl der richtigen Schemadatei in diesem Fall dann in der Verantwortung des Empfängers, was wiederum die Verwaltung von Schema-Versionen unnötig kompliziert und fehleranfällig gestalten würde: Diese Logik, d.h. die korrekte Auflösung der jeweils gültigen Schemadatei entsprechend des Spezial-Attributs, müsste auf jeder Empfängerseite implementieren werden.

2 Abbildung XML-Schema-Elementname zur UML-Bezeichnung

XML-Schema-Element	UML-Bezeichnung	Bedeutung
ag	aggregation	Informationen über den Aggregationsprozess
b	content *(Kindelement von dBinaryValue)*	Binär kodierter Wert
c	category	Hauptkategorie des Messwertes (z.B. „Temperatur“ für die parameter „Wassertemperatur“ bzw. „Lufttemperatur“ etc.) (Freitext)
d	data	Enthält alle timedDataElement / <tde> - Elemente der Zeitreihe
dd	dDescription	Beschreibung (textlich, informell) des Messgeräts/Sensors oder der Systemuhr
dl	location	Ortsbezug für das Messgerät (Messgerät/Sensor oder Systemuhr)
dn	dName	Bezeichnung des Messgeräts/Sensors oder der Systemuhr
dst	distance	Zeitlicher Abstand zwischen den Messwerte (bei isochronen Zeitreihen)
dt	dataType	Information ob und ggf. welche aggregierte Größe vorliegt
dtc	dataTypeCode	Informationen wie der Messwert gebildet wurde (Freitext)
ext	*(nicht in UML modelliert)*	Erweiterungselement, bietet an vielen Stellen die Möglichkeit beliebige Zusatzinformationen hinzuzufügen
f	frequency	Frequenz mit der (Zwischen-) Messwerte ermittelt werden, angegeben wird der zeitliche Abstand
iso	isochron	Informationen zu Messzeitpunkten mit Startzeitpunkt und zeitl. Abstand (bei isochronen Zeitreihen)
it	interval	Aggregationsintervall, d.h. die Zeitspanne für die der Messwert repräsentativ ist
l	level	Schwell- oder Deltawert dessen Über- bzw. Unterschreitung (im Falle einer ereignisge-

		steuerten Zeitreihe) eine Messwertspeicherung triggert.
ld	lDescription	Ortsbeschreibung (informell, textlich)
ldn	locationDescription	Informationen zur Ortsangabe mit Name und informeller Beschreibung
ln	lName	Ortsbezeichnung (informeller Name)
mt	mimetype	Datenformat einer binären Größe (z.B. für JPEG-Bild oder PDF-Datei)
od	oDescription	Beschreibung (textlich, informell) der Organisation die die Zeitreihe ermittelt hat
on	oName	Name der Organisation, welche die Zeitreihe ermittelt hat
org	organization	Informationen zur Organisation/Institution welche die Zeitreihe ermittelt hat
ot	offset	Zeitlicher Offset (zur jeweils nächstgrößeren Zeiteinheit) mit der das Messintervall beginnt. Z.B. bedeutet ein Offset von 5min bei einer Intervalllänge von 30min: Das Zeitintervall startet jeweils 5min und 35min nach jeder vollen Stunde.
p	parameter	Art der Messgröße (Freitext)
pd	device	Informationen zum Messgerät/Sensor
pmd	parameterMetaData	Informationen zum Typ der Zeitreihe
pmdl	parameterMetaDataList	Liste mit Informationen zum Typ der Zeitreihen
pt	pt	2D-Ortsangabe
pt3d	point3D	3D-Ortsangabe
px	px	Rechtswert-Koordinate
py	py	Hochwert-Koordinate
pz	pz	Höhen-Koordinate
rs	referenceSystem	Koordinatenbezugssystem (Freitext)
sts	timeStamp	Startzeitpunkt der Zeitreihe
tde	timedDataElement	Enthält einen oder mehrere Messwerte und optional (falls keine 'isochron-Zeitreihe') einen Zeitstempel
tl	location	Ortsbezug für die Messgröße (für die gesamte Zeitreihe)
ts	timeStamp	Zeitstempel für einen Wert
tsd	device	Informationen zur System-Uhr
tse	timeSeries	Zeitreihe
tsel	timeSeriesList	Liste mit einer oder mehreren Zeitreihen
tsmd	timeStampMetaData	Qualitätsangaben zum Zeitstempel, mit Angabe zur Unsicherheit, Fehlercodes und allgemeinen Angaben zur Systemuhr
tsmi	measurementInaccuracy *(Kindelement von timeStampQuality)*	Unsicherheit des Zeitstempels

tsp	timeStampPosition	Position des Zeitstempels im Zeitintervall (Freitext)
tsq	timeStampQuality	Qualitätsangaben zum Zeitstempel, mit Angabe zur Unsicherheit und Fehlercode
tsqr	timeStampQualityRemark	Qualitätsbezeichner zum Zeitstempel (Freitext)
tsv	tsValue	Zeitstempel-Wert
u	unit	Maßeinheit der Messgröße (Freitext)
uuid	UUID	ID der Zeitreihe
v	dataValue	Enthält die Messgröße und die Qualitätsangaben
va	dAnyValue	Selbst definierter Datentyp (kann beliebig komplex strukturiert sein)
vb	dBoolValue	Boolscher Wert
vbin	dBinaryValue	Binär kodierter Wert (z.B. JPG-Bild oder PDF-Datei, das Unterelement „mime type“ enthält eine Angabe hinsichtl. des Datenformats)
vf	dFloatValue	Gleitkomma-Wert
vi	dIntegerValue	Ganzzahlen-Wert
vl	location	Ortsbezug für die Messgröße (für einen Messwert der Zeitreihe)
vls	values	Enthält einen oder mehrere Messwerte
vmi	measurementInaccuracy *(Kindelement von dataValueQuality)*	Unsicherheit des Messwerts
vq	dataValueQuality	Informationen zur Messunsicherheit und besonderen Mess-Umständen
vqr	dataValueQualityRemark	Informationen über besondere Mess-Umstände (Freitext)
vs	dStringValue	Freitext-Wert
xc	category	Hauptkategorie des Messwertes (z.B. „Temperatur“ für die parameter „Wassertemperatur“ bzw. „Lufttemperatur“ etc.) (Eintrag aus Codeliste)
xdtc	dataTypeCode	Informationen wie der Messwert gebildet wurde (Eintrag aus Codeliste)
xid	exchangeId	Zeitreihen-Bezeichner
xids	exchangeIds	Liste mit einem oder (bei Multiparameter-Zeitreihe) mehreren Zeitreihenbezeichnern
xp	parameter	Art der Messgröße (Eintrag aus Codeliste)
xrs	referenceSystem	Koordinatenbezugssystem (Eintrag aus Codeliste)
xtsp	timeStampPosition	Position des Zeitstempels im Zeitintervall (Eintrag aus Codeliste)
xtsqr	timeStampQualityRemark	Qualitätsbezeichner zum Zeitstempel (Eintrag aus Codeliste)

xu	unit	Maßeinheit der Messgröße (Eintrag aus Codeliste)
xvqr	dataValueQualityRemark	Informationen über besondere Mess-Umstände (Eintrag aus Codeliste)
(im Schema nicht als Element modelliert)	geoReference	Code für das verwendete Koordinatenbezugssystem, und textlich informeller Ortsbeschreibung
(im Schema nicht als Element modelliert)	point	Ortsangabe (als 2D- oder 3D-Koordinaten)
(im Schema nicht als Element modelliert)	value	Enthält einen der folgenden Werte-Objekte: dBoolValue *(ja/nein - Aussage)*, dIntegerValue *(Ganzzahl)*, dFloatValue *(Gleitkommazahl)*, dStringValue *(Text)*, dBinaryValue *(binär kodierte Daten)*, dAnyValue *(selbst definierter Datentyp)*

Abbildung 6: Gegenüberstellung XML-Schema-Elementname zur UML-Bezeichnung

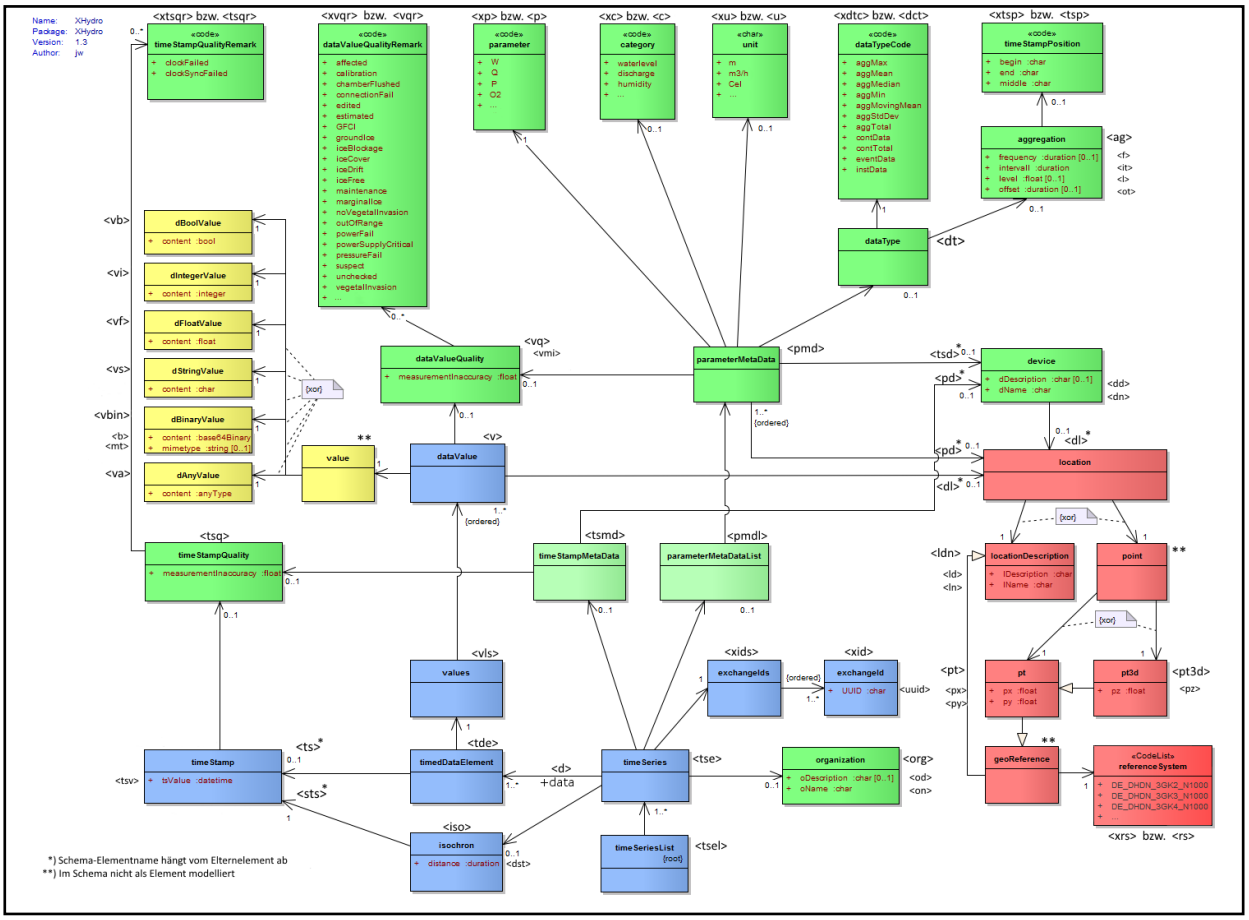

Abbildung 7: UML-Diagramm mit XML-Elementnamen

3 Literatur

[1] Wikipedia:
XML Schema
https://de.wikipedia.org/wiki/XML_Schema

[2] Bundesanstalt für Gewässerkunde:
XHydro, UML-Modell
http://www.xhydro.org/uml.html

[3] W3C:
XML Signature Syntax and Processing
http://www.w3.org/TR/xmldsig-core/

[4] Wikipedia:
XSL Transformation
https://de.wikipedia.org/wiki/XSL_Transformation

[5] W3C:
XML Encryption Syntax and Processing
http://www.w3.org/TR/xmlenc-core/

[6] Sourceforge:
XMill
http://sourceforge.net/projects/xmill

[7] International Telecommunication Union:
XML encoding rules (XER) for ASN.1
http://asn1.elibel.tm.fr/xml/

[8] Web Services Interoperability Organization:
Best Practices for Web services interoperability
http://www.ws-i.org/

[9] Web Services Interoperability Organization:
Basic Profile Version 1.1, Use of XML-Schema
http://www.ws-i.org/profiles/basicprofile-1.1-2004-08-24.html#WSDLSCHEMA

[10] Wikipedia:
Resource Description Framework
http://de.wikipedia.org/wiki/Resource_Description_Framework

[11] xFront:
Creating Variable Content Container Elements
http://www.xfront.com/VariableContentContainers.pdf

[12] OASIS:
XML Catalogs
https://www.oasis-open.org/committees/download.php/14809/xml-catalogs.html

[13] xFront:
Impact of XML Schema Versioning on System Design
http://www.xfront.com/SchemaVersioning.html

[14] XML Coverpages:
XML Schema best practices
http://xml.coverpages.org/HP-StephensonSchemaBestPractices.pdf